TOSHIBA

32-Bit TX System RISC
TX19 Family
TX1940 Application Note

MIPS16, application Specific Extensions and R3000A are a trademark of MIPS Technologies, Inc.

The information contained herein is subject to change without notice.

The information contained herein is presented only as a guide for the applications of our products. No responsibility is assumed by TOSHIBA for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of TOSHIBA or others.

The products described in this document contain components made in the United States and subject to export control of the U.S. authorities. Diversion contrary to the U.S. law is prohibited.

TOSHIBA is continually working to improve the quality and reliability of its products. Nevertheless, semiconductor devices in general can malfunction or fail due to their inherent electrical sensitivity and vulnerability to physical stress.
It is the responsibility of the buyer, when utilizing TOSHIBA products, to comply with the standards of safety in making a safe design for the entire system, and to avoid situations in which a malfunction or failure of such TOSHIBA products could cause loss of human life, bodily injury or damage to property.
In developing your designs, please ensure that TOSHIBA products are used within specified operating ranges as set forth in the most recent TOSHIBA products specifications.
Also, please keep in mind the precautions and conditions set forth in the “Handling Guide for Semiconductor Devices,” or “TOSHIBA Semiconductor Reliability Handbook” etc..

The Toshiba products listed in this document are intended for usage in general electronics applications (computer, personal equipment, office equipment, measuring equipment, industrial robotics, domestic appliances, etc.).
These Toshiba products are neither intended nor warranted for usage in equipment that requires extraordinarily high quality and/or reliability or a malfunction or failure of which may cause loss of human life or bodily injury (“Unintended Usage”). Unintended Usage include atomic energy control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, combustion control instruments, medical instruments, all types of safety devices, etc.. Unintended Usage of Toshiba products listed in this document shall be made at the customer’s own risk.

The products described in this document may include products subject to the foreign exchange and foreign trade laws.

Contents

Handling Precautions

1. Using Toshiba Semiconductors Safely

TOSHIBA are continually working to improve the quality and the reliability of their products.

Nevertheless, semiconductor devices in general can malfunction or fail due to their inherent electrical sensitivity and vulnerability to physical stress. It is the responsibility of the buyer, when utilizing TOSHIBA products, to observe standards of safety, and to avoid situations in which a malfunction or failure of a TOSHIBA product could cause loss of human life, bodily injury or damage to property.

In developing your designs, please ensure that TOSHIBA products are used within specified operating ranges as set forth in the most recent products specifications. Also, please keep in mind the precautions and conditions set forth in the TOSHIBA Semiconductor Reliability Handbook.

2. Safety Precautions

This section lists important precautions which users of semiconductor devices (and anyone else) should observe in order to avoid injury and damage to property, and to ensure safe and correct use of devices.

Please be sure that you understand the meanings of the labels and the graphic symbol described below before you move on to the detailed descriptions of the precautions.

[Explanation of labels]

⚠DANGER	Indicates an imminently hazardous situation which will result in death or serious injury if you do not follow instructions.
⚠WARNING	Indicates a potentially hazardous situation which could result in death or serious injury if you do not follow instructions.
⚠CAUTION	Indicates a potentially hazardous situation which if not avoided, may result in minor injury or moderate injury.

[Explanation of graphic symbol]

Graphic symbol	Meaning
	Indicates that caution is required (laser beam is dangerous to eyes).

2.1 General Precautions regarding Semiconductor Devices

⚠CAUTION
Do not use devices under conditions exceeding their absolute maximum ratings (e.g. current, voltage, power dissipation or temperature). This may cause the device to break down, degrade its performance, or cause it to catch fire or explode resulting in injury. Do not insert devices in the wrong orientation. Make sure that the positive and negative terminals of power supplies are connected correctly. Otherwise the rated maximum current or power dissipation may be exceeded and the device may break down or undergo performance degradation, causing it to catch fire or explode and resulting in injury. When power to a device is on, do not touch the device's heat sink. Heat sinks become hot, so you may burn your hand. Do not touch the tips of device leads. Because some types of device have leads with pointed tips, you may prick your finger. When conducting any kind of evaluation, inspection or testing, be sure to connect the testing equipment's electrodes or probes to the pins of the device under test before powering it on. Otherwise, you may receive an electric shock causing injury. Before grounding an item of measuring equipment or a soldering iron, check that there is no electrical leakage from it. Electrical leakage may cause the device which you are testing or soldering to break down, or could give you an electric shock. Always wear protective glasses when cutting the leads of a device with clippers or a similar tool. If you do not, small bits of metal flying off the cut ends may damage your eyes.

2.2 Precautions Specific to Each Product Group

2.2.1 Optical semiconductor devices

⚠DANGER

When a visible semiconductor laser is operating, do not look directly into the laser beam or look through the optical system. This is highly likely to impair vision, and in the worst case may cause blindness.
If it is necessary to examine the laser apparatus, for example to inspect its optical characteristics, always wear the appropriate type of laser protective glasses as stipulated by IEC standard IEC825-1.

⚠WARNING

Ensure that the current flowing in an LED device does not exceed the device's maximum rated current.
This is particularly important for resin-packaged LED devices, as excessive current may cause the package resin to blow up, scattering resin fragments and causing injury.

When testing the dielectric strength of a photocoupler, use testing equipment which can shut off the supply voltage to the photocoupler. If you detect a leakage current of more than 100 μA, use the testing equipment to shut off the photocoupler's supply voltage; otherwise a large short-circuit current will flow continuously, and the device may break down or burst into flames, resulting in fire or injury.

When incorporating a visible semiconductor laser into a design, use the device's internal photodetector or a separate photodetector to stabilize the laser's radiant power so as to ensure that laser beams exceeding the laser's rated radiant power cannot be emitted.
If this stabilizing mechanism does not work and the rated radiant power is exceeded, the device may break down or the excessively powerful laser beams may cause injury.

2.2.2 Power devices

⚠DANGER

Never touch a power device while it is powered on. Also, after turning off a power device, do not touch it until it has thoroughly discharged all remaining electrical charge.
Touching a power device while it is powered on or still charged could cause a severe electric shock, resulting in death or serious injury.

When conducting any kind of evaluation, inspection or testing, be sure to connect the testing equipment's electrodes or probes to the device under test before powering it on.
When you have finished, discharge any electrical charge remaining in the device.
Connecting the electrodes or probes of testing equipment to a device while it is powered on may result in electric shock, causing injury.

⚠WARNING

Do not use devices under conditions which exceed their absolute maximum ratings (current, voltage, power dissipation, temperature etc.).
This may cause the device to break down, causing a large short-circuit current to flow, which may in turn cause it to catch fire or explode, resulting in fire or injury.

Use a unit which can detect short-circuit currents and which will shut off the power supply if a short-circuit occurs.
If the power supply is not shut off, a large short-circuit current will flow continuously, which may in turn cause the device to catch fire or explode, resulting in fire or injury.

When designing a case for enclosing your system, consider how best to protect the user from shrapnel in the event of the device catching fire or exploding.
Flying shrapnel can cause injury.

When conducting any kind of evaluation, inspection or testing, always use protective safety tools such as a cover for the device.
Otherwise you may sustain injury caused by the device catching fire or exploding.

Make sure that all metal casings in your design are grounded to earth.
Even in modules where a device's electrodes and metal casing are insulated, capacitance in the module may cause the electrostatic potential in the casing to rise.
Dielectric breakdown may cause a high voltage to be applied to the casing, causing electric shock and injury to anyone touching it.

When designing the heat radiation and safety features of a system incorporating high-speed rectifiers, remember to take the device's forward and reverse losses into account.
The leakage current in these devices is greater than that in ordinary rectifiers; as a result, if a high-speed rectifier is used in an extreme environment (e.g. at high temperature or high voltage), its reverse loss may increase, causing thermal runaway to occur.
This may in turn cause the device to explode and scatter shrapnel, resulting in injury to the user.

A design should ensure that, except when the main circuit of the device is active, reverse bias is applied to the device gate while electricity is conducted to control circuits, so that the main circuit will become inactive.
Malfunction of the device may cause serious accidents or injuries.

⚠CAUTION

When conducting any kind of evaluation, inspection or testing, either wear protective gloves or wait until the device has cooled properly before handling it.
Devices become hot when they are operated. Even after the power has been turned off, the device will retain residual heat which may cause a burn to anyone touching it.

2.2.3 Bipolar ICs (for use in automobiles)

⚠CAUTION

If your design includes an inductive load such as a motor coil, incorporate diodes or similar devices into the design to prevent negative current from flowing in.
The load current generated by powering the device on and off may cause it to function erratically or to break down, which could in turn cause injury.

Ensure that the power supply to any device which incorporates protective functions is stable.
If the power supply is unstable, the device may operate erratically, preventing the protective functions from working correctly. If protective functions fail, the device may break down causing injury to the user.

3. General Safety Precautions and Usage Considerations

This section is designed to help you gain a better understanding of semiconductor devices, so as to ensure the safety, quality and reliability of the devices which you incorporate into your designs.

3.1 From Incoming to Shipping

3.1.1 Electrostatic discharge (ESD)

When handling individual devices (which are not yet mounted on a printed circuit board), be sure that the environment is protected against electrostatic electricity. Operators should wear anti-static clothing, and containers and other objects which come into direct contact with devices should be made of anti-static materials and should be grounded to earth via an 0.5- to 1.0-MΩ protective resistor.

Please follow the precautions described below; this is particularly important for devices which are marked “Be careful of static.”.

(1) Work environment

- When humidity in the working environment decreases, the human body and other insulators can easily become charged with static electricity due to friction. Maintain the recommended humidity of 40% to 60% in the work environment, while also taking into account the fact that moisture-proof-packed products may absorb moisture after unpacking.

- Be sure that all equipment, jigs and tools in the working area are grounded to earth.

- Place a conductive mat over the floor of the work area, or take other appropriate measures, so that the floor surface is protected against static electricity and is grounded to earth. The surface resistivity should be 10^4 to 10^8 Ω/sq and the resistance between surface and ground, 7.5×10^5 to 10^8 Ω

- Cover the workbench surface also with a conductive mat (with a surface resistivity of 10^4 to 10^8 Ω/sq, for a resistance between surface and ground of 7.5×10^5 to 10^8 Ω) . The purpose of this is to disperse static electricity on the surface (through resistive components) and ground it to earth. Workbench surfaces must not be constructed of low-resistance metallic materials that allow rapid static discharge when a charged device touches them directly.

- Pay attention to the following points when using automatic equipment in your workplace:

  (a) When picking up ICs with a vacuum unit, use a conductive rubber fitting on the end of the pick-up wand to protect against electrostatic charge.

  (b) Minimize friction on IC package surfaces. If some rubbing is unavoidable due to the device’s mechanical structure, minimize the friction plane or use material with a small friction coefficient and low electrical resistance. Also, consider the use of an ionizer.

  (c) In sections which come into contact with device lead terminals, use a material which dissipates static electricity.

  (d) Ensure that no statically charged bodies (such as work clothes or the human body) touch the devices.

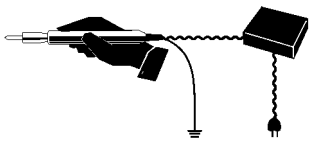
(e) Make sure that sections of the tape carrier which come into contact with installation devices or other electrical machinery are made of a low-resistance material.

(f) Make sure that jigs and tools used in the assembly process do not touch devices.

(g) In processes in which packages may retain an electrostatic charge, use an ionizer to neutralize the ions.

- Make sure that CRT displays in the working area are protected against static charge, for example by a VDT filter. As much as possible, avoid turning displays on and off. Doing so can cause electrostatic induction in devices.
- Keep track of charged potential in the working area by taking periodic measurements.
- Ensure that work chairs are protected by an anti-static textile cover and are grounded to the floor surface by a grounding chain. (Suggested resistance between the seat surface and grounding chain is 7.5×10^5 to $10^{12}\Omega$.)
- Install anti-static mats on storage shelf surfaces. (Suggested surface resistivity is 10^4 to 10^8 Ω/sq; suggested resistance between surface and ground is 7.5×10^5 to 10^8 Ω.)
- For transport and temporary storage of devices, use containers (boxes, jigs or bags) that are made of anti-static materials or materials which dissipate electrostatic charge.
- Make sure that cart surfaces which come into contact with device packaging are made of materials which will conduct static electricity, and verify that they are grounded to the floor surface via a grounding chain.
- In any location where the level of static electricity is to be closely controlled, the ground resistance level should be Class 3 or above. Use different ground wires for all items of equipment which may come into physical contact with devices.

(2) Operating environment



- Operators must wear anti-static clothing and conductive shoes (or a leg or heel strap).
- Operators must wear a wrist strap grounded to earth via a resistor of about 1 MΩ.
- Soldering irons must be grounded from iron tip to earth, and must be used only at low voltages (6 V to 24 V).
- If the tweezers you use are likely to touch the device terminals, use anti-static tweezers and in particular avoid metallic tweezers. If a charged device touches a low-resistance tool, rapid discharge can occur. When using vacuum tweezers, attach a conductive chucking pat to the tip, and connect it to a dedicated ground used especially for anti-static purposes (suggested resistance value: 10^4 to 10^8 Ω).
- Do not place devices or their containers near sources of strong electrical fields (such as above a CRT).

- When storing printed circuit boards which have devices mounted on them, use a board container or bag that is protected against static charge. To avoid the occurrence of static charge or discharge due to friction, keep the boards separate from one other and do not stack them directly on top of one another.

- Ensure, if possible, that any articles (such as clipboards) which are brought to any location where the level of static electricity must be closely controlled are constructed of anti-static materials.

- In cases where the human body comes into direct contact with a device, be sure to wear anti-static finger covers or gloves (suggested resistance value: 10^8 Ω or less).

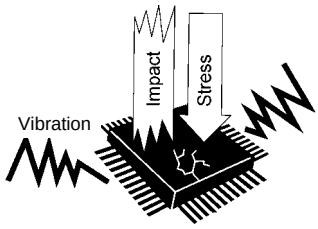
- Equipment safety covers installed near devices should have resistance ratings of 10^9 Ω or less.

- If a wrist strap cannot be used for some reason, and there is a possibility of imparting friction to devices, use an ionizer.

- The transport film used in TCP products is manufactured from materials in which static charges tend to build up. When using these products, install an ionizer to prevent the film from being charged with static electricity. Also, ensure that no static electricity will be applied to the product's copper foils by taking measures to prevent static occuring in the peripheral equipment.

3.1.2 Vibration, impact and stress

Handle devices and packaging materials with care. To avoid damage to devices, do not toss or drop packages. Ensure that devices are not subjected to mechanical vibration or shock during transportation. Ceramic package devices and devices in canister-type packages which have empty space inside them are subject to damage from vibration and shock because the bonding wires are secured only at their ends.



Plastic molded devices, on the other hand, have a relatively high level of resistance to vibration and mechanical shock because their bonding wires are enveloped and fixed in resin. However, when any device or package type is installed in target equipment, it is to some extent susceptible to wiring disconnections and other damage from vibration, shock and stressed solder junctions. Therefore when devices are incorporated into the design of equipment which will be subject to vibration, the structural design of the equipment must be thought out carefully.

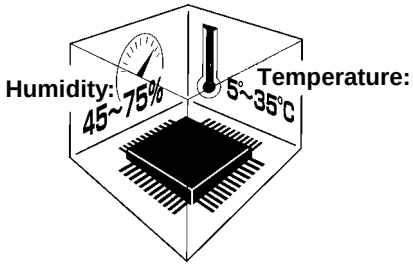
If a device is subjected to especially strong vibration, mechanical shock or stress, the package or the chip itself may crack. In products such as CCDs which incorporate window glass, this could cause surface flaws in the glass or cause the connection between the glass and the ceramic to separate.

Furthermore, it is known that stress applied to a semiconductor device through the package changes the resistance characteristics of the chip because of piezoelectric effects. In analog circuit design attention must be paid to the problem of package stress as well as to the dangers of vibration and shock as described above.

3.2 Storage

3.2.1 General storage

- Avoid storage locations where devices will be exposed to moisture or direct sunlight.
- Follow the instructions printed on the device cartons regarding transportation and storage.
- The storage area temperature should be kept within a temperature range of 5°C to 35°C, and relative humidity should be maintained at between 45% and 75%.



- Do not store devices in the presence of harmful (especially corrosive) gases, or in dusty conditions.
- Use storage areas where there is minimal temperature fluctuation. Rapid temperature changes can cause moisture to form on stored devices, resulting in lead oxidation or corrosion. As a result, the solderability of the leads will be degraded.
- When repacking devices, use anti-static containers.
- Do not allow external forces or loads to be applied to devices while they are in storage.
- If devices have been stored for more than two years, their electrical characteristics should be tested and their leads should be tested for ease of soldering before they are used.

3.2.2 Moisture-proof packing

Moisture-proof packing should be handled with care. The handling procedure specified for each packing type should be followed scrupulously. If the proper procedures are not followed, the quality and reliability of devices may be degraded. This section describes general precautions for handling moisture-proof packing. Since the details may differ from device to device, refer also to the relevant individual datasheets or databook.

(1) General precautions
Follow the instructions printed on the device cartons regarding transportation and storage.

- Do not drop or toss device packing. The laminated aluminum material in it can be rendered ineffective by rough handling.
- The storage area temperature should be kept within a temperature range of 5°C to 30°C, and relative humidity should be maintained at 90% (max). Use devices within 12 months of the date marked on the package seal.

- If the 12-month storage period has expired, or if the 30% humidity indicator shown in Figure 1 is pink when the packing is opened, it may be advisable, depending on the device and packing type, to back the devices at high temperature to remove any moisture. Please refer to the table below. After the pack has been opened, use the devices in a 5°C to 30°C. 60% RH environment and within the effective usage period listed on the moisture-proof package. If the effective usage period has expired, or if the packing has been stored in a high-humidity environment, bake the devices at high temperature.

Packing	Moisture removal
Tray	If the packing bears the “Heatproof” marking or indicates the maximum temperature which it can withstand, bake at 125°C for 20 hours. (Some devices require a different procedure.)
Tube	Transfer devices to trays bearing the “Heatproof” marking or indicating the temperature which they can withstand, or to aluminum tubes before baking at 125°C for 20 hours.
Tape	Deviced packed on tape cannot be baked and must be used within the effective usage period after unpacking, as specified on the packing.

- When baking devices, protect the devices from static electricity.

- Moisture indicators can detect the approximate humidity level at a standard temperature of 25°C. 6-point indicators and 3-point indicators are currently in use, but eventually all indicators will be 3-point indicators.

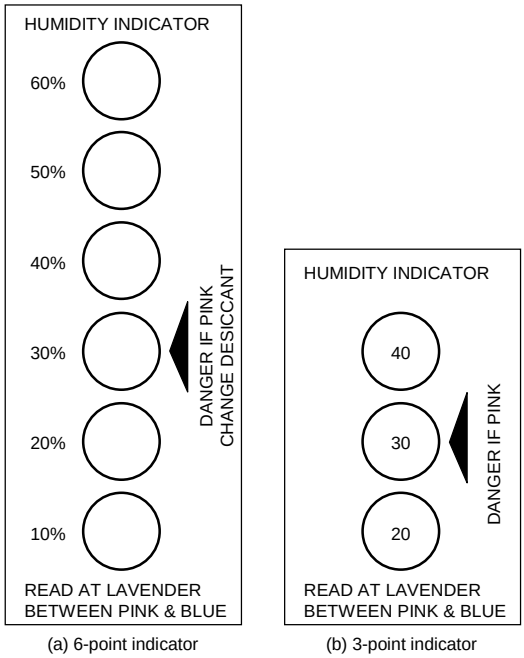


Figure 1 Humidity indicator

3.3 Design

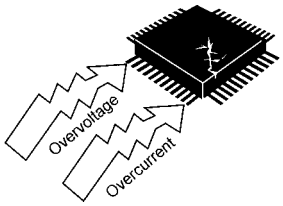
Care must be exercised in the design of electronic equipment to achieve the desired reliability. It is important not only to adhere to specifications concerning absolute maximum ratings and recommended operating conditions, it is also important to consider the overall environment in which equipment will be used, including factors such as the ambient temperature, transient noise and voltage and current surges, as well as mounting conditions which affect device reliability. This section describes some general precautions which you should observe when designing circuits and when mounting devices on printed circuit boards.

For more detailed information about each product family, refer to the relevant individual technical datasheets available from Toshiba.

3.3.1 Absolute maximum ratings

⚠CAUTION Do not use devices under conditions in which their absolute maximum ratings (e.g. current, voltage, power dissipation or temperature) will be exceeded. A device may break down or its performance may be degraded, causing it to catch fire or explode resulting in injury to the user.

The absolute maximum ratings are rated values which must not be exceeded during operation, even for an instant. Although absolute maximum ratings differ from product to product, they essentially concern the voltage and current at each pin, the allowable power dissipation, and the junction and storage temperatures.



If the voltage or current on any pin exceeds the absolute maximum rating, the device's internal circuitry can become degraded. In the worst case, heat generated in internal circuitry can fuse wiring or cause the semiconductor chip to break down.

If storage or operating temperatures exceed rated values, the package seal can deteriorate or the wires can become disconnected due to the differences between the thermal expansion coefficients of the materials from which the device is constructed.

3.3.2 Recommended operating conditions

The recommended operating conditions for each device are those necessary to guarantee that the device will operate as specified in the datasheet.
If greater reliability is required, derate the device's absolute maximum ratings for voltage, current, power and temperature before using it.

3.3.3 Derating

When incorporating a device into your design, reduce its rated absolute maximum voltage, current, power dissipation and operating temperature in order to ensure high reliability.
Since derating differs from application to application, refer to the technical datasheets available for the various devices used in your design.

3.3.4 Unused pins

If unused pins are left open, some devices can exhibit input instability problems, resulting in malfunctions such as abrupt increase in current flow. Similarly, if the unused output pins on a device are connected to the power supply pin, the ground pin or to other output pins, the IC may malfunction or break down.

Since the details regarding the handling of unused pins differ from device to device and from pin to pin, please follow the instructions given in the relevant individual datasheets or databook.

CMOS logic IC inputs, for example, have extremely high impedance. If an input pin is left open, it can easily pick up extraneous noise and become unstable. In this case, if the input voltage level reaches an intermediate level, it is possible that both the P-channel and N-channel transistors will be turned on, allowing unwanted supply current to flow. Therefore, ensure that the unused input pins of a device are connected to the power supply (Vcc) pin or ground (GND) pin of the same device. For details of what to do with the pins of heat sinks, refer to the relevant technical datasheet and databook.

3.3.5 Latch-up

Latch-up is an abnormal condition inherent in CMOS devices, in which Vcc gets shorted to ground. This happens when a parasitic PN-PN junction (thyristor structure) internal to the CMOS chip is turned on, causing a large current of the order of several hundred mA or more to flow between Vcc and GND, eventually causing the device to break down.

Latch-up occurs when the input or output voltage exceeds the rated value, causing a large current to flow in the internal chip, or when the voltage on the Vcc (Vdd) pin exceeds its rated value, forcing the internal chip into a breakdown condition. Once the chip falls into the latch-up state, even though the excess voltage may have been applied only for an instant, the large current continues to flow between Vcc (Vdd) and GND (Vss). This causes the device to heat up and, in extreme cases, to emit gas fumes as well. To avoid this problem, observe the following precautions:

(1) Do not allow voltage levels on the input and output pins either to rise above Vcc (Vdd) or to fall below GND (Vss). Also, follow any prescribed power-on sequence, so that power is applied gradually or in steps rather than abruptly.

(2) Do not allow any abnormal noise signals to be applied to the device.

(3) Set the voltage levels of unused input pins to Vcc (Vdd) or GND (Vss).

(4) Do not connect output pins to one another.

3.3.6 Input/Output protection

Wired-AND configurations, in which outputs are connected together, cannot be used, since this short-circuits the outputs. Outputs should, of course, never be connected to Vcc (Vdd) or GND (Vss).

Furthermore, ICs with tri-state outputs can undergo performance degradation if a shorted output current is allowed to flow for an extended period of time. Therefore, when designing circuits, make sure that tri-state outputs will not be enabled simultaneously.

3.3.7 Load capacitance

Some devices display increased delay times if the load capacitance is large. Also, large charging and discharging currents will flow in the device, causing noise. Furthermore, since outputs are shorted for a relatively long time, wiring can become fused.

Consult the technical information for the device being used to determine the recommended load capacitance.

3.3.8 Thermal design

The failure rate of semiconductor devices is greatly increased as operating temperatures increase. As shown in Figure 2, the internal thermal stress on a device is the sum of the ambient temperature and the temperature rise due to power dissipation in the device. Therefore, to achieve optimum reliability, observe the following precautions concerning thermal design:

(1) Keep the ambient temperature (Ta) as low as possible.

(2) If the device's dynamic power dissipation is relatively large, select the most appropriate circuit board material, and consider the use of heat sinks or of forced air cooling. Such measures will help lower the thermal resistance of the package.

(3) Derate the device's absolute maximum ratings to minimize thermal stress from power dissipation.

$\theta ja = \theta jc + \theta ca$

$\theta ja = (Tj - Ta) / P$

$\theta jc = (Tj - Tc) / P$

$\theta ca = (Tc - Ta) / P$

in which θja = thermal resistance between junction and surrounding air (°C/W)

θjc = thermal resistance between junction and package surface, or internal thermal resistance (°C/W)

θca = thermal resistance between package surface and surrounding air, or external thermal resistance (°C/W)

Tj = junction temperature or chip temperature (°C)

Tc = package surface temperature or case temperature (°C)

Ta = ambient temperature (°C)

P = power dissipation (W)

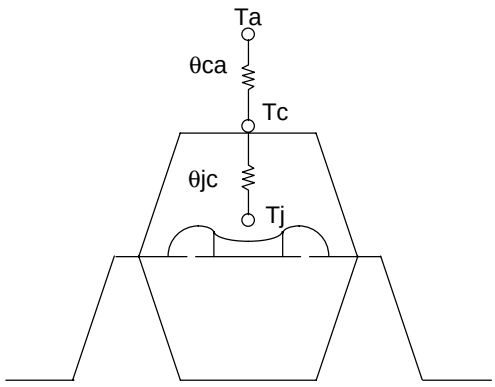


Figure 2 Thermal resistance of package

3.3.9 Interfacing

When connecting inputs and outputs between devices, make sure input voltage (V_{IL}/V_{IH}) and output voltage (V_{OL}/V_{OH}) levels are matched. Otherwise, the devices may malfunction. When connecting devices operating at different supply voltages, such as in a dual-power-supply system, be aware that erroneous power-on and power-off sequences can result in device breakdown. For details of how to interface particular devices, consult the relevant technical datasheets and databooks. If you have any questions or doubts about interfacing, contact your nearest Toshiba office or distributor.

3.3.10 Decoupling

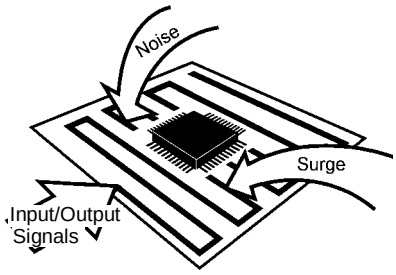
Spike currents generated during switching can cause Vcc (Vdd) and GND (Vss) voltage levels to fluctuate, causing ringing in the output waveform or a delay in response speed. (The power supply and GND wiring impedance is normally 50 Ω to 100 Ω.) For this reason, the impedance of power supply lines with respect to high frequencies must be kept low. This can be accomplished by using thick and short wiring for the Vcc (Vdd) and GND (Vss) lines and by installing decoupling capacitors (of approximately 0.01 μF to 1 μF capacitance) as high-frequency filters between Vcc (Vdd) and GND (Vss) at strategic locations on the printed circuit board.

For low-frequency filtering, it is a good idea to install a 10- to 100-μF capacitor on the printed circuit board (one capacitor will suffice). If the capacitance is excessively large, however, (e.g. several thousand μF) latch-up can be a problem. Be sure to choose an appropriate capacitance value.

An important point about wiring is that, in the case of high-speed logic ICs, noise is caused mainly by reflection and crosstalk, or by the power supply impedance. Reflections cause increased signal delay, ringing, overshoot and undershoot, thereby reducing the device's safety margins with respect to noise. To prevent reflections, reduce the wiring length by increasing the device mounting density so as to lower the inductance (L) and capacitance (C) in the wiring. Extreme care must be taken, however, when taking this corrective measure, since it tends to cause crosstalk between the wires. In practice, there must be a trade-off between these two factors.

3.3.11 External noise

Printed circuit boards with long I/O or signal pattern lines are vulnerable to induced noise or surges from outside sources. Consequently, malfunctions or breakdowns can result from overcurrent or overvoltage, depending on the types of device used. To protect against noise, lower the impedance of the pattern line or insert a noise-canceling circuit. Protective measures must also be taken against surges.



For details of the appropriate protective measures for a particular device, consult the relevant databook.

3.3.12 Electromagnetic interference

Widespread use of electrical and electronic equipment in recent years has brought with it radio and TV reception problems due to electromagnetic interference. To use the radio spectrum effectively and to maintain radio communications quality, each country has formulated regulations limiting the amount of electromagnetic interference which can be generated by individual products.

Electromagnetic interference includes conduction noise propagated through power supply and telephone lines, and noise from direct electromagnetic waves radiated by equipment. Different measurement methods and corrective measures are used to assess and counteract each specific type of noise.

Difficulties in controlling electromagnetic interference derive from the fact that there is no method available which allows designers to calculate, at the design stage, the strength of the electromagnetic waves which will emanate from each component in a piece of equipment. For this reason, it is only after the prototype equipment has been completed that the designer can take measurements using a dedicated instrument to determine the strength of electromagnetic interference waves. Yet it is possible during system design to incorporate some measures for the prevention of electromagnetic interference, which can facilitate taking corrective measures once the design has been completed. These include installing shields and noise filters, and increasing

the thickness of the power supply wiring patterns on the printed circuit board. One effective method, for example, is to devise several shielding options during design, and then select the most suitable shielding method based on the results of measurements taken after the prototype has been completed.

3.3.13 Peripheral circuits

In most cases semiconductor devices are used with peripheral circuits and components. The input and output signal voltages and currents in these circuits must be chosen to match the semiconductor device's specifications. The following factors must be taken into account.

(1) Inappropriate voltages or currents applied to a device's input pins may cause it to operate erratically. Some devices contain pull-up or pull-down resistors. When designing your system, remember to take the effect of this on the voltage and current levels into account.

(2) The output pins on a device have a predetermined external circuit drive capability. If this drive capability is greater than that required, either incorporate a compensating circuit into your design or carefully select suitable components for use in external circuits.

3.3.14 Safety standards

Each country has safety standards which must be observed. These safety standards include requirements for quality assurance systems and design of device insulation. Such requirements must be fully taken into account to ensure that your design conforms to the applicable safety standards.

3.3.15 Other precautions

(1) When designing a system, be sure to incorporate fail-safe and other appropriate measures according to the intended purpose of your system. Also, be sure to debug your system under actual board-mounted conditions.

(2) If a plastic-package device is placed in a strong electric field, surface leakage may occur due to the charge-up phenomenon, resulting in device malfunction. In such cases take appropriate measures to prevent this problem, for example by protecting the package surface with a conductive shield.

(3) With some microcomputers and MOS memory devices, caution is required when powering on or resetting the device. To ensure that your design does not violate device specifications, consult the relevant databook for each constituent device.

(4) Ensure that no conductive material or object (such as a metal pin) can drop onto and short the leads of a device mounted on a printed circuit board.

3.4 Inspection, Testing and Evaluation

3.4.1 Grounding

⚠CAUTION Ground all measuring instruments, jigs, tools and soldering irons to earth. Electrical leakage may cause a device to break down or may result in electric shock.

3.4.2 Inspection Sequence

⚠CAUTION

① Do not insert devices in the wrong orientation. Make sure that the positive and negative electrodes of the power supply are correctly connected. Otherwise, the rated maximum current or maximum power dissipation may be exceeded and the device may break down or undergo performance degradation, causing it to catch fire or explode, resulting in injury to the user.

② When conducting any kind of evaluation, inspection or testing using AC power with a peak voltage of 42.4 V or DC power exceeding 60 V, be sure to connect the electrodes or probes of the testing equipment to the device under test before powering it on. Connecting the electrodes or probes of testing equipment to a device while it is powered on may result in electric shock, causing injury.

(1) Apply voltage to the test jig only after inserting the device securely into it. When applying or removing power, observe the relevant precautions, if any.

(2) Make sure that the voltage applied to the device is off before removing the device from the test jig. Otherwise, the device may undergo performance degradation or be destroyed.

(3) Make sure that no surge voltages from the measuring equipment are applied to the device.

(4) The chips housed in tape carrier packages (TCPs) are bare chips and are therefore exposed. During inspection take care not to crack the chip or cause any flaws in it.
Electrical contact may also cause a chip to become faulty. Therefore make sure that nothing comes into electrical contact with the chip.

3.5 Mounting

There are essentially two main types of semiconductor device package: lead insertion and surface mount. During mounting on printed circuit boards, devices can become contaminated by flux or damaged by thermal stress from the soldering process. With surface-mount devices in particular, the most significant problem is thermal stress from solder reflow, when the entire package is subjected to heat. This section describes a recommended temperature profile for each mounting method, as well as general precautions which you should take when mounting devices on printed circuit boards. Note, however, that even for devices with the same package type, the appropriate mounting method varies according to the size of the chip and the size and shape of the lead frame. Therefore, please consult the relevant technical datasheet and databook.

3.5.1 Lead forming

⚠CAUTION

① Always wear protective glasses when cutting the leads of a device with clippers or a similar tool. If you do not, small bits of metal flying off the cut ends may damage your eyes.

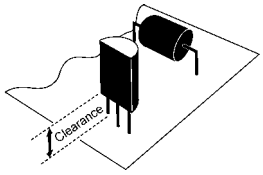
② Do not touch the tips of device leads. Because some types of device have leads with pointed tips, you may prick your finger.

Semiconductor devices must undergo a process in which the leads are cut and formed before the devices can be mounted on a printed circuit board. If undue stress is applied to the interior of a device during this process, mechanical breakdown or performance degradation can result. This is attributable primarily to differences between the stress on the device's external leads and the stress on the internal leads. If the relative difference is great enough, the device's internal leads, adhesive properties or sealant can be damaged. Observe these precautions during the lead-forming process (this does not apply to surface-mount devices):

(1) Lead insertion hole intervals on the printed circuit board should match the lead pitch of the device precisely.

(2) If lead insertion hole intervals on the printed circuit board do not precisely match the lead pitch of the device, do not attempt to forcibly insert devices by pressing on them or by pulling on their leads.

(3) For the minimum clearance specification between a device and a printed circuit board, refer to the relevant device's datasheet and databook. If necessary, achieve the required clearance by forming the device's leads appropriately. Do not use the spacers which are used to raise devices above the surface of the printed circuit board during soldering to achieve clearance. These spacers normally continue to expand due to heat, even after the solder has begun to solidify; this applies severe stress to the device.



(4) Observe the following precautions when forming the leads of a device prior to mounting.

- Use a tool or jig to secure the lead at its base (where the lead meets the device package) while bending so as to avoid mechanical stress to the device. Also avoid bending or stretching device leads repeatedly.
- Be careful not to damage the lead during lead forming.
- Follow any other precautions described in the individual datasheets and databooks for each device and package type.

3.5.2 Socket mounting

(1) When socket mounting devices on a printed circuit board, use sockets which match the inserted device's package.

(2) Use sockets whose contacts have the appropriate contact pressure. If the contact pressure is insufficient, the socket may not make a perfect contact when the device is repeatedly inserted and removed; if the pressure is excessively high, the device leads may be bent or damaged when they are inserted into or removed from the socket.

(3) When soldering sockets to the printed circuit board, use sockets whose construction prevents flux from penetrating into the contacts or which allows flux to be completely cleaned off.

(4) Make sure the coating agent applied to the printed circuit board for moisture-proofing purposes does not stick to the socket contacts.

(5) If the device leads are severely bent by a socket as it is inserted or removed and you wish to repair the leads so as to continue using the device, make sure that this lead correction is only performed once. Do not use devices whose leads have been corrected more than once.

(6) If the printed circuit board with the devices mounted on it will be subjected to vibration from external sources, use sockets which have a strong contact pressure so as to prevent the sockets and devices from vibrating relative to one another.

3.5.3 Soldering temperature profile

The soldering temperature and heating time vary from device to device. Therefore, when specifying the mounting conditions, refer to the individual datasheets and databooks for the devices used.

(1) Using a soldering iron

Complete soldering within ten seconds for lead temperatures of up to 260°C, or within three seconds for lead temperatures of up to 350°C.

(2) Using medium infrared ray reflow

- Heating top and bottom with long or medium infrared rays is recommended (see Figure 3).

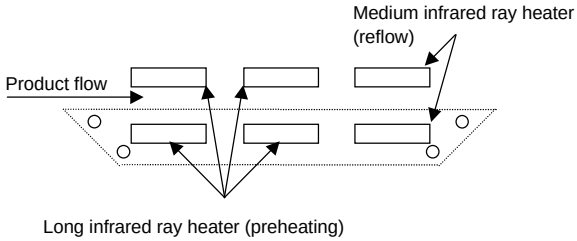


Figure 3 Heating top and bottom with long or medium infrared rays

- Complete the infrared ray reflow process within 30 seconds at a package surface temperature of between 210°C and 240°C.
- Refer to Figure 4 for an example of a good temperature profile for infrared or hot air reflow.

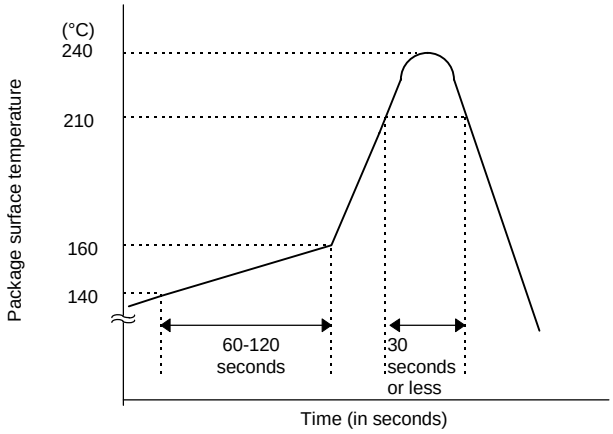


Figure 4 Sample temperature profile for infrared or hot air reflow

(3) Using hot air reflow

- Complete hot air reflow within 30 seconds at a package surface temperature of between 210°C and 240°C.
- For an example of a recommended temperature profile, refer to Figure 4 above.

(4) Using solder flow

- Apply preheating for 60 to 120 seconds at a temperature of 150°C.
- For lead insertion-type packages, complete solder flow within 10 seconds with the temperature at the stopper (or, if there is no stopper, at a location more than 1.5 mm from the body) which does not exceed 260°C.

- For surface-mount packages, complete soldering within 5 seconds at a temperature of 250°C or less in order to prevent thermal stress in the device.
- Figure 5 shows an example of a recommended temperature profile for surface-mount packages using solder flow.

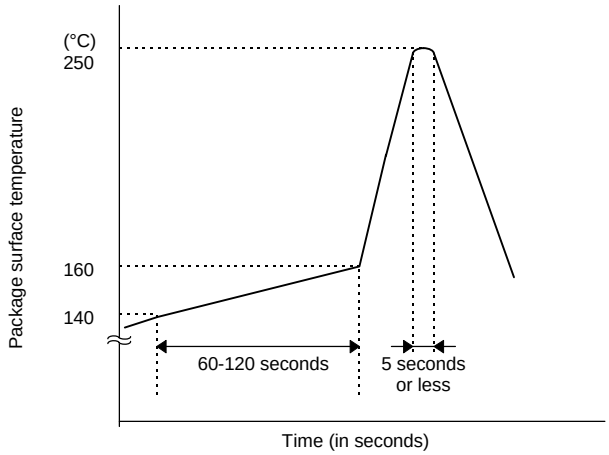


Figure 5 Sample temperature profile for solder flow

3.5.4 Flux cleaning and ultrasonic cleaning

(1) When cleaning circuit boards to remove flux, make sure that no residual reactive ions such as Na or Cl remain. Note that organic solvents react with water to generate hydrogen chloride and other corrosive gases which can degrade device performance.

(2) Washing devices with water will not cause any problems. However, make sure that no reactive ions such as sodium and chlorine are left as a residue. Also, be sure to dry devices sufficiently after washing.

(3) Do not rub device markings with a brush or with your hand during cleaning or while the devices are still wet from the cleaning agent. Doing so can rub off the markings.

(4) The dip cleaning, shower cleaning and steam cleaning processes all involve the chemical action of a solvent. Use only recommended solvents for these cleaning methods. When immersing devices in a solvent or steam bath, make sure that the temperature of the liquid is 50°C or below, and that the circuit board is removed from the bath within one minute.

(5) Ultrasonic cleaning should not be used with hermetically-sealed ceramic packages such as a leadless chip carrier (LCC), pin grid array (PGA) or charge-coupled device (CCD), because the bonding wires can become disconnected due to resonance during the cleaning process. Even if a device package allows ultrasonic cleaning, limit the duration of ultrasonic cleaning to as short a time as possible, since long hours of ultrasonic cleaning degrade the adhesion between the mold resin and the frame material. The following ultrasonic cleaning conditions are recommended:

Frequency: 27 kHz ~ 29 kHz

Ultrasonic output power: 300 W or less (0.25 W/cm^2 or less)

Cleaning time: 30 seconds or less

Suspend the circuit board in the solvent bath during ultrasonic cleaning in such a way that the ultrasonic vibrator does not come into direct contact with the circuit board or the device.

3.5.5 No cleaning

If analog devices or high-speed devices are used without being cleaned, flux residues may cause minute amounts of leakage between pins. Similarly, dew condensation, which occurs in environments containing residual chlorine when power to the device is on, may cause between-lead leakage or migration. Therefore, Toshiba recommends that these devices be cleaned. However, if the flux used contains only a small amount of halogen (0.05W% or less), the devices may be used without cleaning without any problems.

3.5.6 Mounting tape carrier packages (TCPs)

(1) When tape carrier packages (TCPs) are mounted, measures must be taken to prevent electrostatic breakdown of the devices.

(2) If devices are being picked up from tape, or outer lead bonding (OLB) mounting is being carried out, consult the manufacturer of the insertion machine which is being used, in order to establish the optimum mounting conditions in advance and to avoid any possible hazards.

(3) The base film, which is made of polyimide, is hard and thin. Be careful not to cut or scratch your hands or any objects while handling the tape.

(4) When punching tape, try not to scatter broken pieces of tape too much.

(5) Treat the extra film, reels and spacers left after punching as industrial waste, taking care not to destroy or pollute the environment.

(6) Chips housed in tape carrier packages (TCPs) are bare chips and therefore have their reverse side exposed. To ensure that the chip will not be cracked during mounting, ensure that no mechanical shock is applied to the reverse side of the chip. Electrical contact may also cause a chip to fail. Therefore, when mounting devices, make sure that nothing comes into electrical contact with the reverse side of the chip.
If your design requires connecting the reverse side of the chip to the circuit board, please consult Toshiba or a Toshiba distributor beforehand.

3.5.7 Mounting chips

Devices delivered in chip form tend to degrade or break under external forces much more easily than plastic-packaged devices. Therefore, caution is required when handling this type of device.

(1) Mount devices in a properly prepared environment so that chip surfaces will not be exposed to polluted ambient air or other polluted substances.

(2) When handling chips, be careful not to expose them to static electricity.
In particular, measures must be taken to prevent static damage during the mounting of chips. With this in mind, Toshiba recommend mounting all peripheral parts first and then mounting chips last (after all other components have been mounted).

(3) Make sure that PCBs (or any other kind of circuit board) on which chips are being mounted do not have any chemical residues on them (such as the chemicals which were used for etching the PCBs).

(4) When mounting chips on a board, use the method of assembly that is most suitable for maintaining the appropriate electrical, thermal and mechanical properties of the semiconductor devices used.

* For details of devices in chip form, refer to the relevant device's individual datasheets.

3.5.8 Circuit board coating

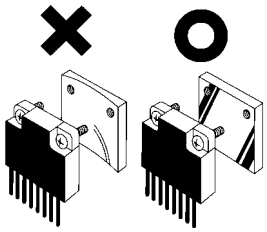
When devices are to be used in equipment requiring a high degree of reliability or in extreme environments (where moisture, corrosive gas or dust is present), circuit boards may be coated for protection. However, before doing so, you must carefully consider the possible stress and contamination effects that may result and then choose the coating resin which results in the minimum level of stress to the device.

3.5.9 Heat sinks

(1) When attaching a heat sink to a device, be careful not to apply excessive force to the device in the process.

(2) When attaching a device to a heat sink by fixing it at two or more locations, evenly tighten all the screws in stages (i.e. do not fully tighten one screw while the rest are still only loosely tightened). Finally, fully tighten all the screws up to the specified torque.

(3) Drill holes for screws in the heat sink exactly as specified. Smooth the surface by removing burrs and protrusions or indentations which might interfere with the installation of any part of the device.



(4) A coating of silicone compound can be applied between the heat sink and the device to improve heat conductivity. Be sure to apply the coating thinly and evenly; do not use too much. Also, be sure to use a non-volatile compound, as volatile compounds can crack after a time, causing the heat radiation properties of the heat sink to deteriorate.

(5) If the device is housed in a plastic package, use caution when selecting the type of silicone compound to be applied between the heat sink and the device. With some types, the base oil separates and penetrates the plastic package, significantly reducing the useful life of the device.
Two recommended silicone compounds in which base oil separation is not a problem are YG6260 from Toshiba Silicone.

(6) Heat-sink-equipped devices can become very hot during operation. Do not touch them, or you may sustain a burn.

3.5.10 Tightening torque

(1) Make sure the screws are tightened with fastening torques not exceeding the torque values stipulated in individual datasheets and databooks for the devices used.

(2) Do not allow a power screwdriver (electrical or air-driven) to touch devices.

3.5.11 Repeated device mounting and usage

Do not remount or re-use devices which fall into the categories listed below; these devices may cause significant problems relating to performance and reliability.

(1) Devices which have been removed from the board after soldering

(2) Devices which have been inserted in the wrong orientation or which have had reverse current applied

(3) Devices which have undergone lead forming more than once

3.6 Protecting Devices in the Field

3.6.1 Temperature

Semiconductor devices are generally more sensitive to temperature than are other electronic components. The various electrical characteristics of a semiconductor device are dependent on the ambient temperature at which the device is used. It is therefore necessary to understand the temperature characteristics of a device and to incorporate device derating into circuit design. Note also that if a device is used above its maximum temperature rating, device deterioration is more rapid and it will reach the end of its usable life sooner than expected.

3.6.2 Humidity

Resin-molded devices are sometimes improperly sealed. When these devices are used for an extended period of time in a high-humidity environment, moisture can penetrate into the device and cause chip degradation or malfunction. Furthermore, when devices are mounted on a regular printed circuit board, the impedance between wiring components can decrease under high-humidity conditions. In systems which require a high signal-source impedance, circuit board leakage or leakage between device lead pins can cause malfunctions. The application of a moisture-proof treatment to the device surface should be considered in this case. On the other hand, operation under low-humidity conditions can damage a device due to the occurrence of electrostatic discharge. Unless damp-proofing measures have been specifically taken, use devices only in environments with appropriate ambient moisture levels (i.e. within a relative humidity range of 40% to 60%).

3.6.3 Corrosive gases

Corrosive gases can cause chemical reactions in devices, degrading device characteristics. For example, sulphur-bearing corrosive gases emanating from rubber placed near a device (accompanied by condensation under high-humidity conditions) can corrode a device's leads. The resulting chemical reaction between leads forms foreign particles which can cause electrical leakage.

3.6.4 Radioactive and cosmic rays

Most industrial and consumer semiconductor devices are not designed with protection against radioactive and cosmic rays. Devices used in aerospace equipment or in radioactive environments must therefore be shielded.

3.6.5 Strong electrical and magnetic fields

Devices exposed to strong magnetic fields can undergo a polarization phenomenon in their plastic material, or within the chip, which gives rise to abnormal symptoms such as impedance changes or increased leakage current. Failures have been reported in LSIs mounted near malfunctioning deflection yokes in TV sets. In such cases the device's installation location must be changed or the device must be shielded against the electrical or magnetic field. Shielding against magnetism is especially necessary for devices used in an alternating magnetic field because of the electromotive forces generated in this type of environment.

3.6.6 Interference from light (ultraviolet rays, sunlight, fluorescent lamps and incandescent lamps)

Light striking a semiconductor device generates electromotive force due to photoelectric effects. In some cases the device can malfunction. This is especially true for devices in which the internal chip is exposed. When designing circuits, make sure that devices are protected against incident light from external sources. This problem is not limited to optical semiconductors and EPROMs. All types of device can be affected by light.

3.6.7 Dust and oil

Just like corrosive gases, dust and oil can cause chemical reactions in devices, which will adversely affect a device's electrical characteristics. To avoid this problem, do not use devices in dusty or oily environments. This is especially important for optical devices because dust and oil can affect a device's optical characteristics as well as its physical integrity and the electrical performance factors mentioned above.

3.6.8 Fire

Semiconductor devices are combustible; they can emit smoke and catch fire if heated sufficiently. When this happens, some devices may generate poisonous gases. Devices should therefore never be used in close proximity to an open flame or a heat-generating body, or near flammable or combustible materials.

3.7 Disposal of Devices and Packing Materials

When discarding unused devices and packing materials, follow all procedures specified by local regulations in order to protect the environment against contamination.

4. Precautions and Usage Considerations Specific to Each Product Group

This section describes matters specific to each product group which need to be taken into consideration when using devices. If the same item is described in Sections 3 and 4, the description in Section 4 takes precedence.

4.1 Microcontrollers

4.1.1 Design

(1) Using resonators which are not specifically recommended for use

Resonators recommended for use with Toshiba products in microcontroller oscillator applications are listed in Toshiba databooks along with information about oscillation conditions. If you use a resonator not included in this list, please consult Toshiba or the resonator manufacturer concerning the suitability of the device for your application.

(2) Undefined functions

In some microcontrollers certain instruction code values do not constitute valid processor instructions. Also, it is possible that the values of bits in registers will become undefined. Take care in your applications not to use invalid instructions or to let register bit values become undefined.

(3) Scratch and puncture wounds by the point of a probe

The tips of probes and adaptors used in development tools are individually designed to be compatible with particular devices. Probes for some devices have sharp points. When you handle them bare-handed, take care not to suffer a scratch or puncture wound.

4.1.2 Reliability predictions for microcontroller devices

For microcontroller devices, the following junction temperature range is used for reliability predictions:

Tj = 0°C ~ 85°C

An estimation of the chip junction temperature, Tj, can be obtained from the equation:

$Tj = Ta + Q \times \theta ja$

where:

Ta = ambient temperature (°C)

The assumption is that the ambient temperature is not affected by any heat transfers from the device.

Q = chip's average power dissipation (W)

θja = package thermal resistance (°C/W)

Note 1:	If you use a microcontroller device outside the 0 to 85°C range for long periods of time, contact your nearest Toshiba office or authorized Toshiba dealer.
Note 2:	For the θja value, contact your nearest Toshiba office or authorized Toshiba dealer.

TX1940 Application Note

Introductory Notes

Purpose

This application note is designed to help the user understand the TX1940. To this end, it illustrates the operation of a TX1940-based microcomputer system using detailed explanations of sample program running on the TX1940 Training Board (the TB1940).

* The programs presented in this application note are only sample programs and are not guaranteed to operate within any specific application.

Target audience

This application note is aimed at people with a general knowledge of electronics, logic circuits, microcomputers and the C programming language.

Composition of this document

This application note is divided into the following chapters:

Chapter 1 Overview of the TX1940

Chapter 2 Description of the Hardware

Explains the TX1940 hardware on which the sample programs are intended to run.

Chapter 3 Description of the Software

Explains the sample application programs which have been designed to run on the TB1940.

For details of the TX1940's hardware functions, instruction set and electrical characteristics, please refer to the databook for the TX1940CYAF and TX1940FDAF.

Related documentation

- Documents relating to the device
  32-Bit TX System RISC TX19 Family TX1940 (databook)

- Documents relating to development tools
  Microcomputer Development System Programming Tool Part (I)

Chapter 1 Overview of the TX1940

This chapter describes the features and characteristics of the TX1940FDBF. Throughout this application note, unless otherwise stated the name “TX1940” denotes the TX1940FDBF.

1.1 Features

The TX19 Family of 32-bit RISC processors are high-performance successors to the Toshiba-developed TX39 processor which was in turn based on the R3000A™ RISC microprocessor developed by the MIPS Group in the U.S.A. The high-code-efficiency MIPS16™ ASE (application-specific extension) extended instruction set makes the TX19 a high-performance 32-bit RISC processor family.

The 32-bit TX1940 RISC microprocessor is built around the TX19L core processor, incorporates various peripheral functions and is capable of low-voltage operation and low power consumption. The features of the TX1940 are summarized below:

(1) TX19L core processor

1) Provides increased code efficiency and arithmetic performance by virtue of the 16-Bit and 32-Bit ISA (instruction set architecture) Modes.
   - The 16-Bit ISA Mode instructions are object-compatible with the MIPS16™ ASE which exhibits excellent code efficiency.
   - The 32-Bit ISA Mode instructions are object-compatible with the TX39 which exhibits excellent arithmetic performance.
2) Realizes both high performance and low power consumption.

- High performance
  - Almost all instructions are executed in a single clock period.
  - 3-operand arithmetic instructions are used to increase performance.
  - 5-stage pipeline processing
  - Built-in high-speed ROM/RAM
  - DSP function: A 32-bit multiplier accumlator (MAC) operation takes only 1 clock cycle to execute.
- Low power consumption
  - Optimization achieved using low power consumption libraries
  - Standby function to stop operation of the core processor

3) Fast interrupt response suitable for real-time control
   - Independent entry address for each interrupt source
   - Automatically generates vector address for each interrupt source.
   - Automatically updates interrupt mask level.

(2) Internal RAM: 16 Kbytes
Internal ROM: 512-Kbyte flash memory

(3) External memory extension
- Can be expanded to 16 Mbytes (shared between programs and data)
- Coexisting 8-/16-bit external data buses

(4) DMA controller: 4 channels
- Activated by interrupt or by software

(5) 8-bit timer: 4 channels

(6) 16-bit timer: 4 channels

(7) Data timer: 1 channel

(8) General-purpose serial interface: 4 channels
UART and synchronous modes: 2 channels
UART-only: 2 channels

(9) Serial bus interface: 1 channel
Either I^2C Bus Mode or Clock-Synchronous Mode can be selected.

(10) 10-bit AD converter (with sample and hold circuit): 8 channels
Conversion time: 10.75 μs at 32 MHz

(11) Watchdog Timer

(12) Chip Select/Wait controller: 4 channels

(13) Interrupt sources
- CPU, 4 lines·····software interrupt instructions
- Internal, 31 lines·····up to 7 priority levels can be set (but not for the Watchdog Timer interrupt).
- External, 11 lines·····up to 7 priority levels can be set (but not for the NMI interrupt).

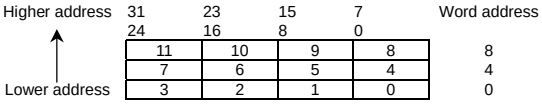
(14) Input/Output ports····· 77 pins

(15) Standby function
- Four Standby modes (IDLE (HALT, DOZE), SLEEP, STOP)

(16) Dual clocks
- Low power consumption: Low-speed clock (32.768 kHz)
- Timer for Real-Time Clock: Low-speed clock (32.768 kHz))

(17) Clock generator
- Built-in PLL (x4)
- Clock gear function: divides high-speed clock by 2, 4 or 8

(18) Only Little-endian is applied.
- Byte 0 is the least significant byte (bit 7~0).
- A word is addressed beginning with the most significant byte.



(19) Operating voltage: 2.7 V~3.6 V

(20) Operating frequency
- 32 MHz (Vcc ≥ 3.0 V) (when interleaving used)
- 27 MHz (Vcc ≥ 2.7 V) (when interleaving used)

(21) Package
- 100-pin QFP (14 × 14 × 1.4 (t) mm, 0.5-mm pitch)

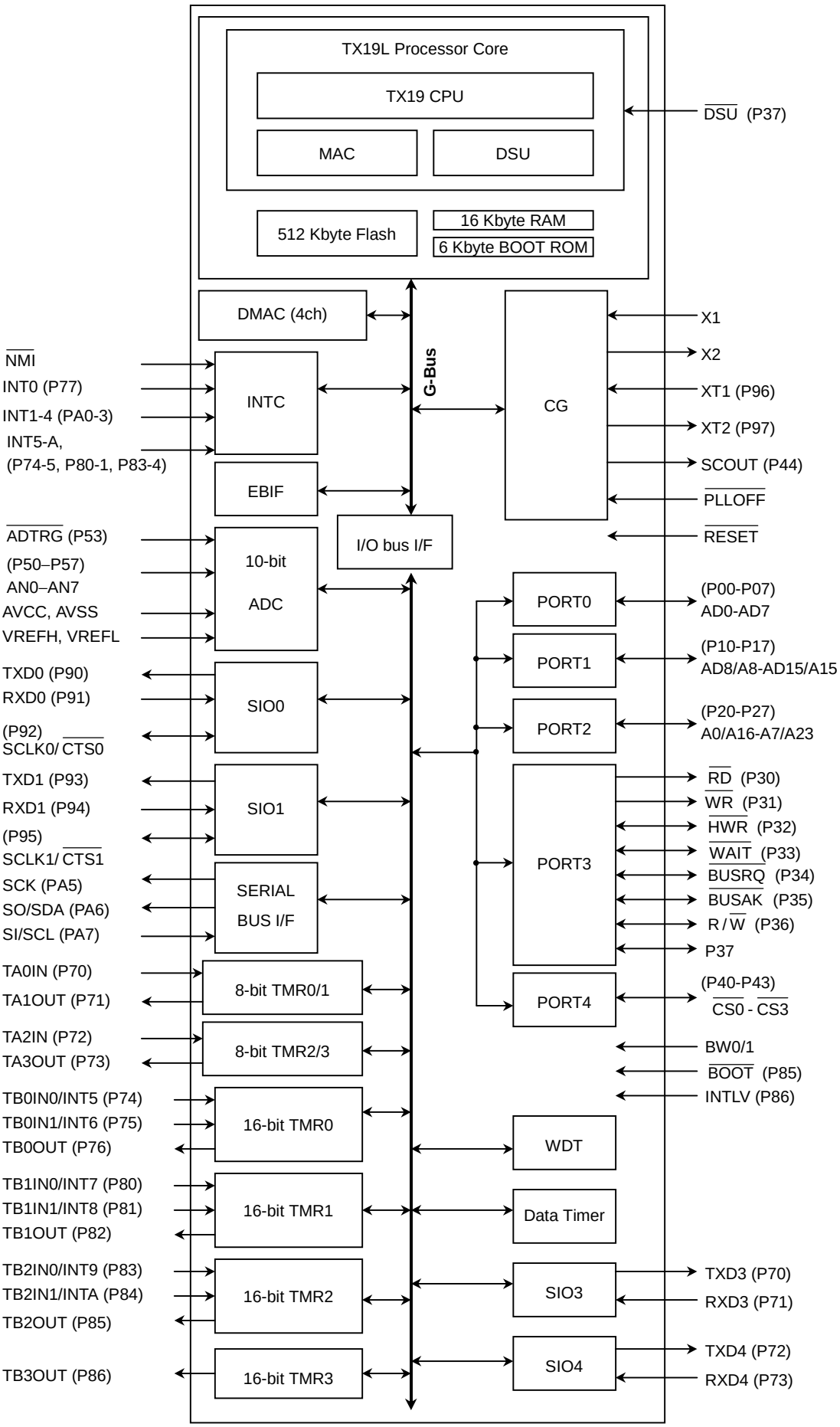


Figure 1.1 Block diagram of the TX1940FDBF ( ): Initial function after a Reset

1.2 Pin Assignment and Pin Functions

This section shows the pin assignment diagram for the TX1940FDBF and gives a brief description of the functions of each of its input and output pins.

1.2.1 Pin Assignment Diagram

Figure 1.2.1 below is the pin assignment diagram for the TX1940FDBF.

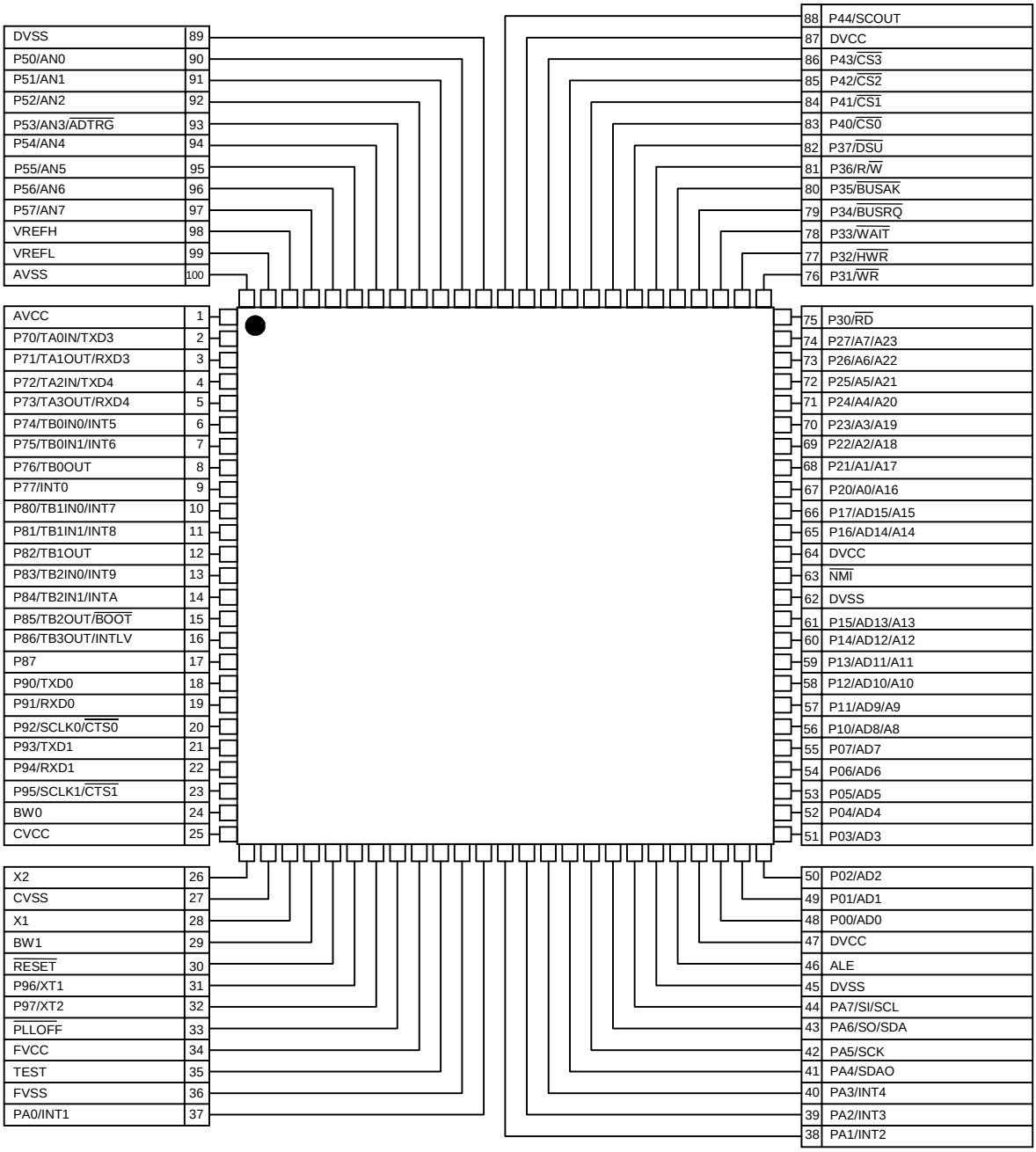


Figure 1.2.1 Pin assignment diagram (100-pin LQFP)

1.3 Pin Names and Functions

Table 1.3.1 below lists the names and functions of the TX1940FDBF's input/output pins.

Table 1.3.1 Pin names and functions

Pin Name	# of Pins	Type	Description
P00~P07 AD0~AD7	8	Input/output Input/output	Port 0: input/output port each bit of which can be set independently for input or output Address (lower): address/data bus 0~7
P10~P17 AD8~AD15 A8~A15	8	Input/output Input/output Output	Port 1: input/output port each bit of which can be set independently for input or output Address/Data (upper): address/data bus 8~15 Address: address bus 8~15
P20~P27 A0~A7 A16~A23	8	Input/output Output Output	Port 2: input/output port each bit of which can be set independently for input or output Address: address bus 0~7 Address: address bus 16~23
P30 $\overline{RD}$	1	Output Output	Port 30: output-only port Read: strobe signal for reading from external memory
P31 $\overline{WR}$	1	Output Output	Port 31: output-only port Write: strobe signal for writing data to pins D0~D7
P32 $\overline{HWR}$	1	Input/output Output	Port 32: input/output port (with internal pull-up resister) High-order write: strobe signal for writing data to pins D8~D15
P33 $\overline{WAIT}$	1	Input/output Input	Port 33: input/output port (with internal pull-up resister) Wait: Bus Wait request to the CPU
P34 $\overline{BUSRQ}$	1	Input/output Input	Port 34: input/output port (with internal pull-up resister) Bus request: signal used by external master to request the CPU for control of the bus
P35 $\overline{BUSAK}$	1	Input/output Output	Port 35: input/output port (with internal pull-up resister) Bus acknowledge: signal used to acknowledge that, having received a $\overline{BUSRQ}$ signal, the CPU has relinquished control of the bus
P36 $R/\overline{W}$	1	Input/output Output	Port 36: input/output port (with internal pull-up resister) Read/write: an output value of 1 indicates that the current cycle is a Read cycle or a dummy cycle; a value of 0 indicates that it is a Write cycle.
P37 $\overline{DSU}$	1	Input/output Input	Port 37: input/output port (with internal pull-up resister) DSU Enable signal: $\overline{DSU}$ is sampled with the rising edge of the $\overline{RESET}$ signal. If it is detected as asserted (low), the TX1940FDBF enters DSU mode. If it is detected as deasserted (high), the TX1940FDBF enters normal operating mode.
P40 $\overline{CS0}$	1	Input/output Output	Port 40: input/output port (with internal pull-up resister) Chip Select 0: outputs a 0 when the address is within a specified address area.
P41 $\overline{CS1}$	1	Input/output Output	Port 41: input/output port (with internal pull-up resister) Chip Select 1: outputs a 0 when the address is within a specified address area.
P42 $\overline{CS2}$	1	Input/output Output	Port 42: input/output port (with internal pull-up resister) Chip Select 2: outputs a 0 when the address is within a specified address area.
P43 $\overline{CS3}$	1	Input/output Output	Port 43: input/output port (with internal pull-up resister) Chip Select 3: outputs a 0 when the address is within a specified address area.
P44 SCOUT	1	Input/output Output	Port 44: input/output port System clock output: outputs the same high- or low-frequency clock as the CPU.
P50~P57 AN0~AN7 $\overline{ADTRG}$	8	Input Input Input	Port 5: input-only port Analog input: input to the AD converter AD trigger: external start request for the AD converter (shared with P53)
P70 TA0IN TXD3	1	Input/output Input Output	Port 70: input/output port 8-bit Timer 0 input: input to Timer 0 Serial transmission data 3: open-drain output as set by program
P71 TA1OUT RXD3	1	Input/output Output Input	Port 71: input/output port 8-bit Timer 1 output: output from Timer 0 or Timer 1 Serial receive data 3
P72 TA2IN TXD4	1	Input/output Input Output	Port 72: input/output port 8-bit Timer 2 input: input to Timer 2 Serial Transmission Data 4: open-drain output as set by program

Pin Name	# of Pins	Type	Description
P73 TA3OUT RXD4	1	Input/output Output Input	Port 73: input/output port 8-bit Timer 3 output: output from Timer 2 or Timer 3 Serial Receive Data 4
P74 TB0IN0 INT5	1	Input/output Input Input	Port 74: input/output port 16-bit Timer 0 input 0: count/capture trigger input to 16-bit Timer 0 Interrupt Request pin 5: High/Low level or rising/falling edge can be selected as trigger.
P75 TB0IN1 INT6	1	Input/output Input Input	Port 75: input/output port 16-bit Timer 0 input 1: capture trigger input to 16-bit Timer 0. Interrupt Request pin 6: High/Low level or rising/falling edge can be selected as trigger.
P76 TB0OUT	1	Input/output Output	Port 76: input/output port 16-bit Timer 0 output: output from 16-bit Timer 0
P77 INT0	1	Input/output Input	Port 77: input/output port Interrupt Request pin 0: High/Low level or rising/falling edge can be selected as trigger.
P80 TB1IN0 INT7	1	Input/output Input Input	Port 80: input/output port 16-bit Timer 1 input 0: count/capture trigger input to 16-bit Timer 1 Interrupt Request pin 7: High/Low level or rising/falling edge can be selected as trigger.
P81 TB1IN1 INT8	1	Input/output Input Input	Port 81: input/output port 16-bit Timer 1 input 1: capture trigger input to 16-bit Timer 1 Interrupt Request pin 8: High/Low level or rising/falling edge can be selected as trigger.
P82 TB1OUT	1	Input/output Output	Port 82: input/output port 16-bit Timer 1 output: output from 16-bit Timer 1
P83 TB2IN0 INT9	1	Input/output Input Input	Port 83: input/output port 16-bit Timer 2 input 0: count/capture trigger input to 16-bit Timer 2 Interrupt Request pin 9: High/Low level or rising/falling edge can be selected as trigger.
P84 TB2IN1 INTA	1	Input/output Input Input	Port 84: input/output port 16-bit Timer 2 input 1: capture trigger input to 16-bit Timer 2 Interrupt Request pin A: High/Low level or rising/falling edge can be selected as trigger.
P85 TB2OUT $\overline{\text{BOOT}}$	1 1	Input/output Output Input	Port 85: input/output port 16-bit Timer 2 output: output from 16-bit Timer 2 Single-Boot Mode set-up pin: $\overline{\text{BOOT}}$ is sampled with the rising edge of the $\overline{\text{RESET}}$ signal. If it is detected as asserted (low), the TX1940FDBF enters single boot mode, which enables the integrated flash memory to be reprogrammed. If it is detected as deasserted (high), the TX1940FDBF enters normal operating mode.
P86 TB3OUT INTLV	1	Input/output Output Input	Port 86: input/output port 16-bit Timer 3 output: output from 16-bit Timer 3 Interleave Mode set-up pin: INTLV is sampled with the rising edge of the $\overline{\text{RESET}}$ signal. If it is detected as asserted (high), the TX1940FDBF enters interleave mode. During reset, it must be pulled up when interleave mode is used and must be pulled down otherwise.
P87	1	Input/output	Port 87: input/output port This pin is used for mode setting. The device samples 0 on the rising edge of the reset signal. This pin must be pulled Low during reset sequence.
P90 TXD0	1	Input/output Output	Port 90: input/output port Serial Transmission Data 0: open-drain output as set by program
P91 RXD0	1	Input/output Input	Port 91: input/output port Serial Receive Data 0
P92 SCLK0 $\overline{\text{CTS0}}$	1	Input/output Input/output Input	Port 92: input/output port Serial Clock Input/Output 0 Serial data ready to send 0 (Clear to Send)
P93 TXD1	1	Input/output Output	Port 93: input/output port Serial Transmission Data 1: open-drain output as set by program
P94 RXD1	1	Input/output Input	Port 94: input/output port Serial Receive Data 1
P95 SCLK1 $\overline{\text{CTS1}}$	1	Input/output Input/output Input	Port 95: input/output port Serial Clock Input/Output 1 Serial data ready to send 1 (Clear to Send)
P96 XT1	1	Input/output Input	Port 96: input/output port (open-drain output) Low-frequency oscillator connecting pin

Pin Name	# of Pins	Type	Description
P97 XT2	1	Input/output Output	Port 97: input/output port (open-drain output) Low-frequency oscillator connecting pin
PA0~PA3 INT1~INT4	4	Input/output Input	Ports A0~A3: input/output ports Interrupt Request pins 1~4: High/Low level or rising/falling edge can be selected as trigger.
PA4	1	Input/output	Port A4: input/output port
PA5 SCK	1	Input/output Input/output	Port A5: input/output port Clock input/output pin in serial bus interface SIO Mode
PA6 SO SDA	1	Input/output Output Input/output	Port A6: input/output port Data transmission pin in serial bus interface SIO Mode Data transmission/receive pin in serial bus interface I2C Mode Open-drain output as set by program
PA7 SI SCL	1	Input/output Input Input/output	Port A7: input/output port Data receive pin in serial bus interface SIO Mode Clock input/output pin in serial bus interface I2C Mode Open-drain output as set by program
ALE	1	Output	Address Latch Enable. (This is only output when external memory is being accessed.)
$\overline{\text{NMI}}$	1	Input	Non-Maskable Interrupt Request pin: falling edge interrupt request pin
BW0~1	2	Input	Set both BW0 and BW1 to 1.
TEST	1	Input	Set TEST to 0.
$\overline{\text{PLLOFF}}$	1	Input	If the PLL multiplied clock is not being used, fix this input Low. If the PLL multiplied clock is used, fix this input high.
$\overline{\text{RESET}}$	1	Input	Reset (with internal pull-up resister): Initializes the LSI.
VREFH	1	Input	Reference voltage input pin (H) for the AD converter. It must be connected to AVCC when the A/D converter is not used.
VREFL	1	Input	Reference voltage input pin (L) for the AD converter. It must be connected to AVCC when the A/D converter is not used.
AVCC	1	—	AD converter power supply pin (*)
AVSS	1	—	AD converter GND pin (0 V) (**)
X1/X2	2	Input/output	Resonator/Oscillator connecting pin
DVCC, CVCC, FVCC	5	—	Power supply pin
DVSS, CVSS, FVSS	5	—	GND pin (0 V)

Note: When the DSU (debug support unit) is enabled, port A functions as an interface to the DSU tool irrespective of the Port A Function Register (PAFC) and Port A Control Register (PACR) settings. (However, INT1~INT4 and the serial bus interface cannot be used.)

DSU Tool Debug Interface

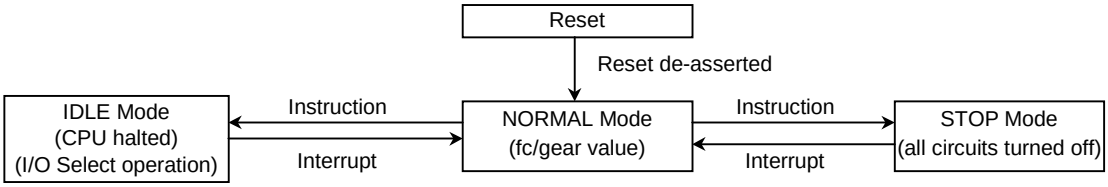
The pins listed below serve as interface signals for an external real-time debug system when $\overline{DSU}$ is detected as asserted (low) on the rising edge of $\overline{RESET}$. $\overline{DSU}$ has an internal pull-up resistor.		
$\overline{DRESET}$ (PA7)	I	Debug Reset $\overline{DRESET}$ DRESET signal for an external real-time debug system
DCLK (PA0)	O	Debug Clock DCLK signal for an external real-time debug system
$\overline{DBGE}$ (PA5)	I	Debugger Enable $\overline{DBGE}$ signal for an external real-time debug system
PCST[2] (PA1)	O	PC Trace Status [2] PCTS[2] signal for an external real-time debug system
PCST[1] (PA2)	O	PC Trace Status [1] PCST[1] signal for an external real-time debug system。
PCST[0] (PA3)	O	PC Trace Status [0] PCTS[0] signal for an external real-time debug system
SDI/ $\overline{DINT}$ (PA6)	I	Serial Data Input / Debug Interrupt SDI/ $\overline{DINT}$ signal for an external real-time debug system
SDAO/TPC (PA4)	O	Serial Data and Address Output/Target PC SDAO/TPC signal for an external real-time debug system

Note: P37, P85, P86 and P87 must remain high or low as specified for a system clock cycle before and after the rising edge of the $\overline{RESET}$ signal. The $\overline{RESET}$ pin must remain stable when it is both high and low.

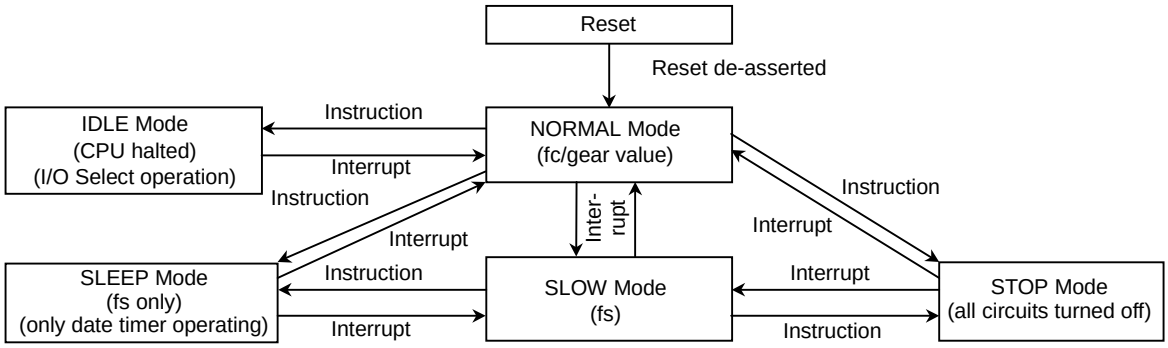
1.4 Clock/Standby Control

There are essentially two modes of clock operation: Single-Clock Mode (which uses only the X1 and X2 pins) and Dual-Clock Mode (which uses the pin pairings X1, X2 and XT1, XT2).

Figure 1.4.1 shows the state transition diagram for each operation mode.



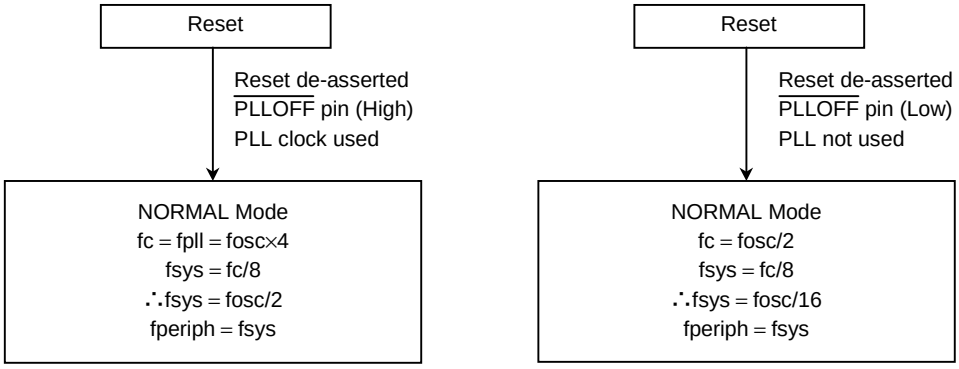
(a) State transitions in Single-Clock Mode



Note 1: Before transition to SLOW/SLEEP Mode can occur, the low-speed oscillator (fs) must be oscillating stably.
Note 2: When SLEEP Mode is terminated, the device returns to the state in which it was placed before entering SLEEP Mode.
Note 3: The state to which the device returns when STOP Mode is terminated can be specified using system control register SYSCR0.

(b) State transition in Dual-Clock Mode

Figure 1.4.1 State transition diagrams for each mode



A. When using a clock generated using the PLL

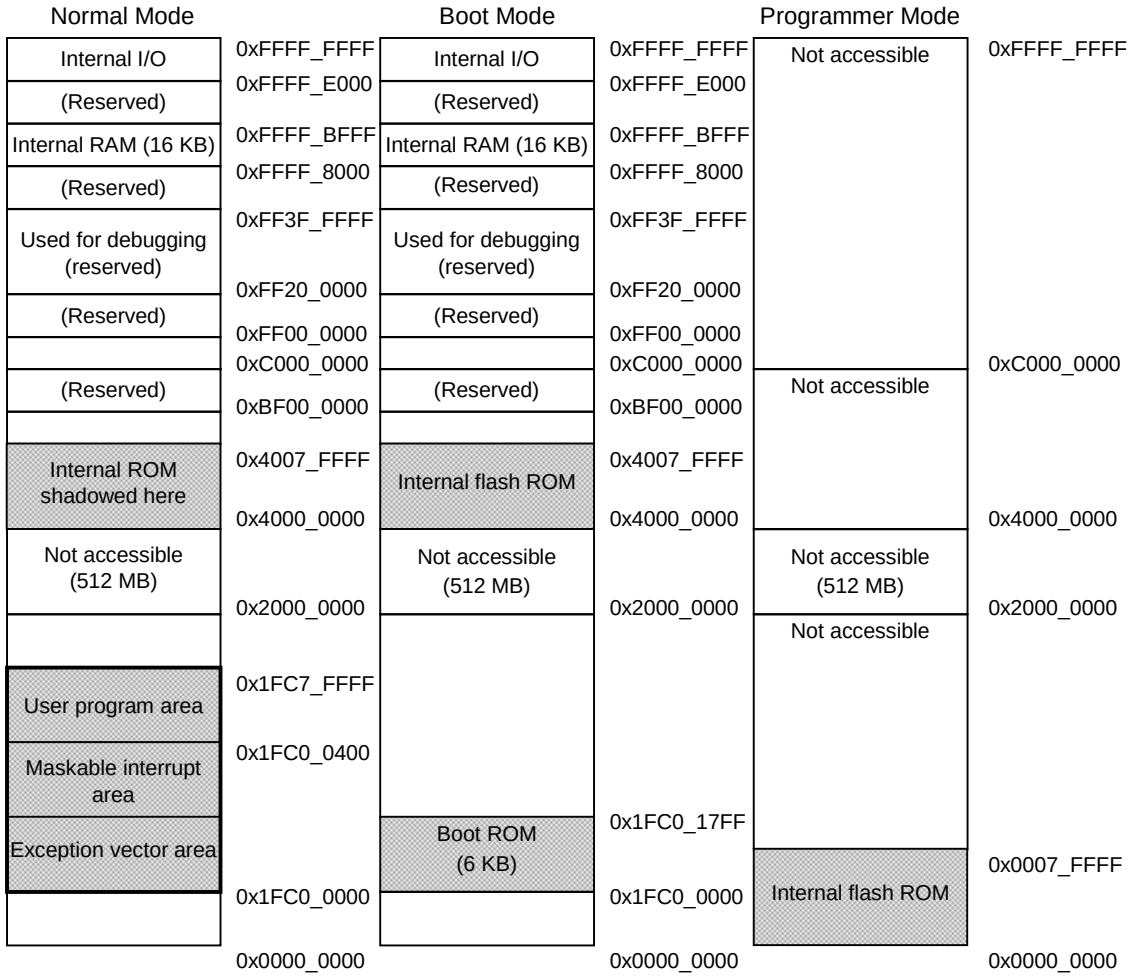
B. When not using the PLL

Figure 1.4.2 Default states both when using a PLL and when not using a PLL

fosc: Clock frequency input via X1 and X2 pins
fs: Clock frequency input via XT1 and XT2 pins
fpll: Clock frequency multiplied (×4) by PLL
fc: Clock frequency selected by setting of PLLOFF pin
fgear: Clock frequency selected by SYSCR1<GEAR1:GEAR0>
System clock fsys: Clock frequency selected by SYSCR1<SYSCK>
fperiph: Input clock for peripheral I/O prescaler

1.5 Memory Map

The memory map for the TX1940 is shown in Figure 1.5.1.



Note 1: The addresses shown above are physical addresses.

Figure 1.5.1 TX1940FDBF memory map for each mode

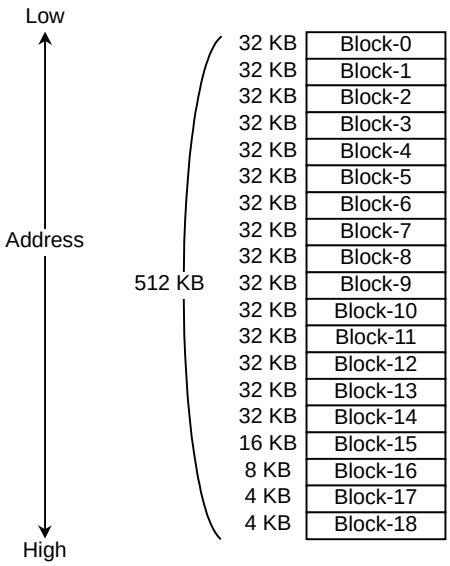


Figure 1.5.2 Block allocation

Table 1.5.1 Block address range by mode

	User Boot Mode	Boot Mode	Programmer Mode
Block-0	0x1FC0_0000 ~ 0x1FC0_7FFF (or 0x4000_0000 ~ 0x4000_7FFF)	0x1FC0_0000 ~ 0x1FC0_7FFF	0x0000_0000 ~ 0x0000_7FFF
Block-1	0x1FC0_8000 ~ 0x1FC0_FFFF (or 0x4000_8000 ~ 0x4000_FFFF)	0x1FC0_8000 ~ 0x1FC0_FFFF	0x0000_8000 ~ 0x0000_FFFF
Block-2	0x1FC1_0000 ~ 0x1FC1_7FFF (or 0x40010000 ~ 0x4001_7FFF)	0x1FC1_0000 ~ 0x1FC1_7FFF	0x0001_0000 ~ 0x0001_7FFF
Block-3	0x1FC1_8000 ~ 0x1FC1_FFFF (or 0x4001_8000 ~ 0x4001_FFFF)	0x1FC1_8000 ~ 0x1FC1_FFFF	0x0001_8000 ~ 0x0001_FFFF
Block-4	0x1FC2_0000 ~ 0x1FC2_7FFF (or 0x4002_0000 ~ 0x4002_7FFF)	0x1FC2_0000 ~ 0x1FC2_7FFF	0x0002_0000 ~ 0x0002_7FFF
Block-5	0x1FC2_8000 ~ 0x1FC2_FFFF (or 0x4002_8000 ~ 0x4002_FFFF)	0x1FC2_8000 ~ 0x1FC2_FFFF	0x0002_8000 ~ 0x0002_FFFF
Block-6	0x1FC3_0000 ~ 0x1FC3_7FFF (or 0x4003_0000 ~ 0x4003_7FFF)	0x1FC3_0000 ~ 0x1FC3_7FFF	0x0003_0000 ~ 0x0003_7FFF
Block-7	0x1FC3_8000 ~ 0x1FC3_FFFF (or 0x4003_8000 ~ 0x4003_FFFF)	0x1FC3_8000 ~ 0x1FC3_FFFF	0x0003_8000 ~ 0x0003_FFFF
Block-8	0x1FC4_0000 ~ 0x1FC4_7FFF (or 0x4004_0000 ~ 0x4004_7FFF)	0x1FC4_0000 ~ 0x1FC4_7FFF	0x0004_0000 ~ 0x0004_7FFF
Block-9	0x1FC4_8000 ~ 0x1FC4_FFFF (or 0x4004_8000 ~ 0x4004_FFFF)	0x1FC4_8000 ~ 0x1FC4_FFFF	0x0004_8000 ~ 0x000_4FFFF
Block-10	0x1FC5_0000 ~ 0x1FC5_7FFF (or 0x4005_0000 ~ 0x4005_7FFF)	0x1FC5_0000 ~ 0x1FC5_7FFF	0x0005_0000 ~ 0x0005_7FFF
Block-11	0x1FC5_8000 ~ 0x1FC5_FFFF (or 0x4005_8000 ~ 0x4005_FFFF)	0x1FC5_8000 ~ 0x1FC5_FFFF	0x0005_8000 ~ 0x0005_FFFF
Block-12	0x1FC6_0000 ~ 0x1FC6_7FFF (or 0x4006_0000 ~ 0x4006_7FFF)	0x1FC6_0000 ~ 0x1FC6_7FFF	0x0006_0000 ~ 0x0006_7FFF
Block-13	0x1FC6_8000 ~ 0x1FC6_FFFF (or 0x4006_8000 ~ 0x4006_FFFF)	0x1FC6_8000 ~ 0x1FC6_FFFF	0x0006_8000 ~ 0x0006_FFFF
Block-14	0x1FC7_0000 ~ 0x1FC7_7FFF (or 0x4007_0000 ~ 0x4007_7FFF)	0x1FC7_0000 ~ 0x1FC7_7FFF	0x0007_0000 ~ 0x0007_7FFF
Block-15	0x1FC7_8000 ~ 0x1FC7_BFFF (or 0x4007_8000 ~ 0x4007_BFFF)	0x1FC7_8000 ~ 0x1FC7_BFFF	0x0007_8000 ~ 0x0007_BFFF
Block-16	0x1FC7_C000 ~ 0x1FC7_DFFF (or 0x4007_C000 ~ 0x4007_DFFF)	0x1FC7_C000 ~ 0x1FC7_DFFF	0x0007_C000 ~ 0x0007_DFFF
Block-17	0x1FC7_E000 ~ 0x1FC7_EFFF (or 0x4007_E000 ~ 0x4007_EFFF)	0x1FC7_E000 ~ 0x1FC7_EFFF	0x0007_E000 ~ 0x0007_EFFF
Block-18	0x1FC7_F000 ~ 0x1FC7_FFFF (or 0x4007_F000 ~ 0x4007_FFFF)	0x1FC7_F000 ~ 0x1FC7_FFFF	0x0007_F000 ~ 0x0007_FFFF

1.6 Sample Circuit for Connecting External Memory

A sample circuit for connecting external devices using the TX1940's bus interface pins is shown below.

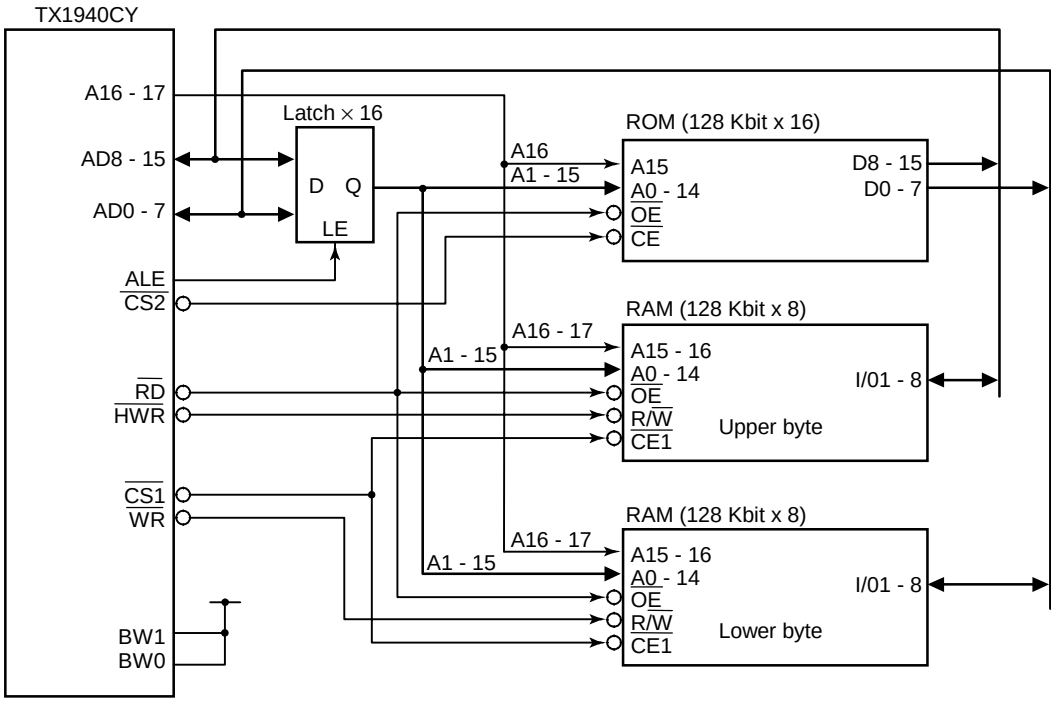


Figure 1.6.1 Example of external memory connection (ROM width = 16 bits, RAM width = 16 bits)

When the TX1940CYAF is reset, the Port 4 Control Register (P4CR) and Port 4 Function Register (P4FC) are both cleared to 0, so that the CS signal output is disabled. To output a CS signal from this port, set the corresponding bits in these registers to 1, first in P4FC and then in P4CR.

Chapter 2 Description of the Hardware

This chapter describes the hardware of the TX1940-based evaluation board used to run the applications described in Chapter 3, Description of the Software. In these application notes the hardware is referred to as the TB1940 (TX1940 Training Board).

2.1 Overview

The main features of the TB1940 are as follows:

(1) CPU: TX1940FDBF
(2) Operating frequencies: 32 MHz
32 kHz (generated by low-frequency oscillator)
(3) Power supply voltage: 3.3 V
(4) Internal ROM: 512 Kbytes (flash ROM)
(5) Internal RAM: 16 Kbytes
(6) RS-232C: 2 channels (internal SIO channels 0 and 1 used) (*2)
(7) Interrupt input switches: INT0, NMI

In addition, the TB1940 has the following circuits attached external to the chip.

(1) ROM: 512 Kbytes (*1)
(2) SRAM: 512 Kbytes (*1)
(3) Switches: AD conversion key, matrix key
(4) Light-emitting diodes: 4 digits, each comprising a 7-seg LED; 8 indicator LEDs
(5) Photosensor
(6) Voice input
(7) Volume control
(8) Motor
(9) I2C bus specification E2PROM
(10) Piezo-electric buzzer

(*1) In the sample programs presented in Chapter 3, only the internal ROM and RAM are used; the external ROM and RAM shown above are not used.

(*2) The sample program presented in Chapter 3 uses only Channel 1 of the SIO; Channel 0 is not used.

2.2 Board Configuration

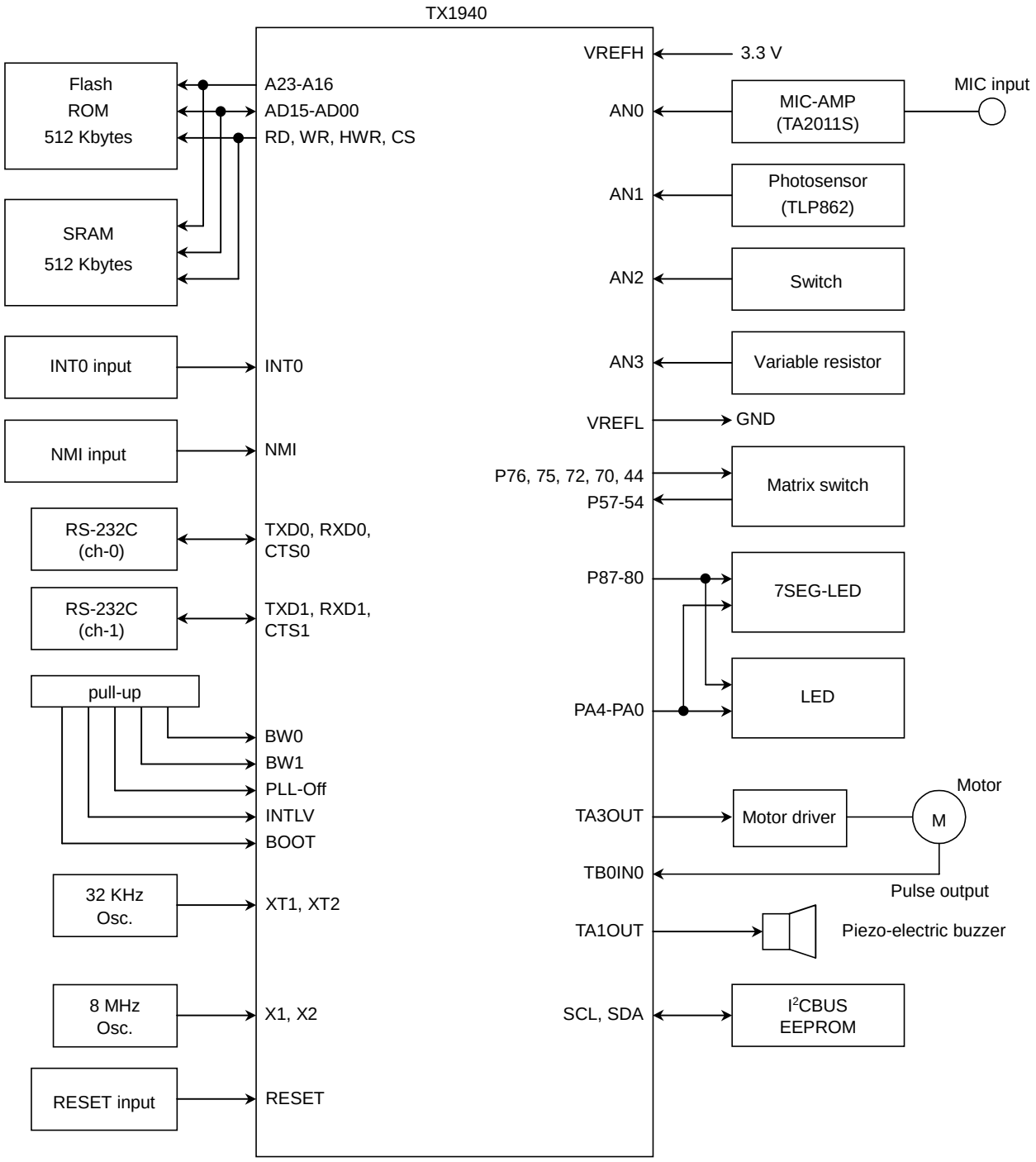


Figure 2.2.1 Configuration of the TX1940 Board

2.3 Peripheral Circuits

2.3.1 I^2C Bus Specification EEPROM

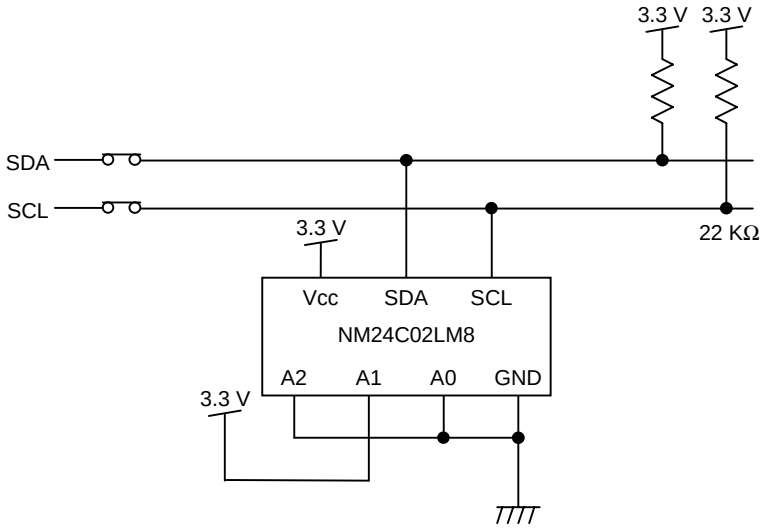


Figure 2.3.1 I^2C bus specification E^2PROM

2.3.2 Motor

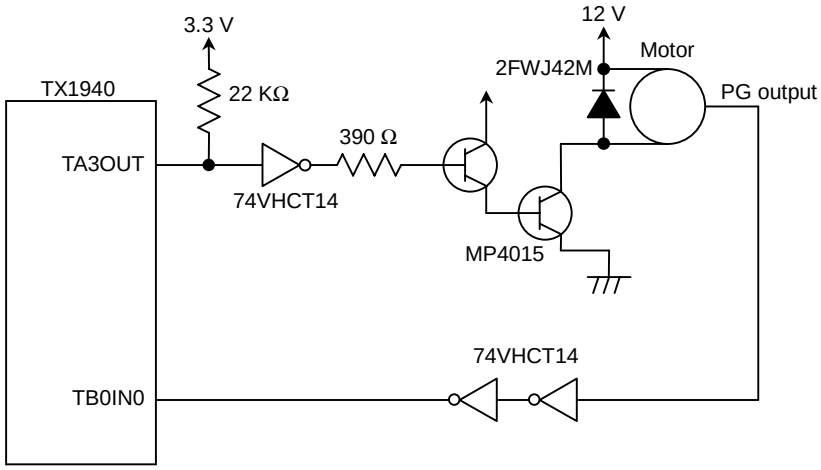


Figure 2.3.2 Motor

2.3.3 Photosensor

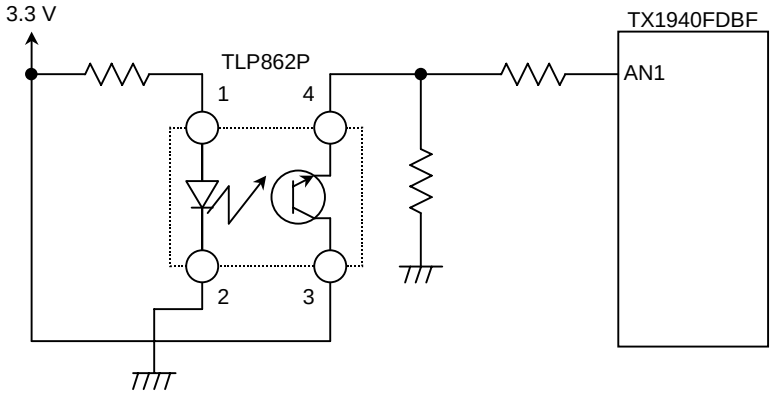


Figure 2.3.3 Photosensor

2.3.4 Volume control

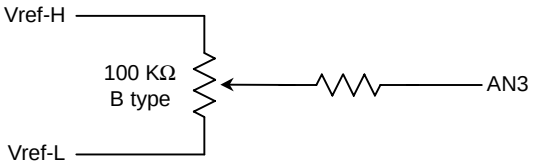


Figure 2.3.4 Volume control

2.3.5 AD conversion key switch

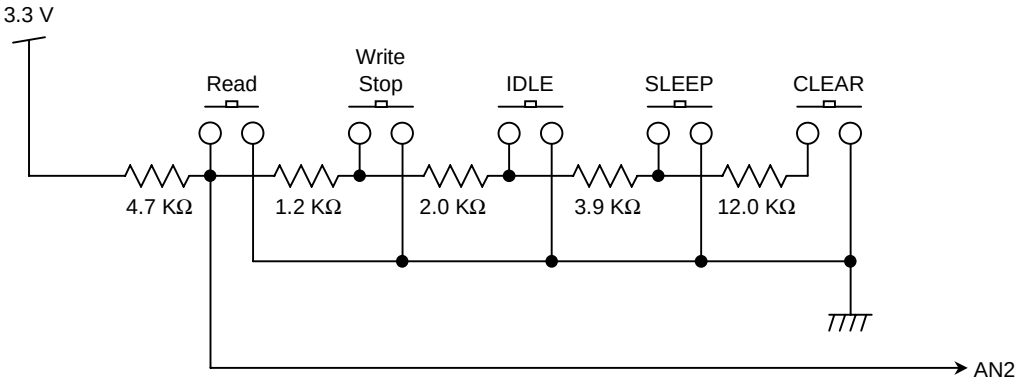


Figure 2.3.5 AD conversion key switch

2.3.6 Voice input

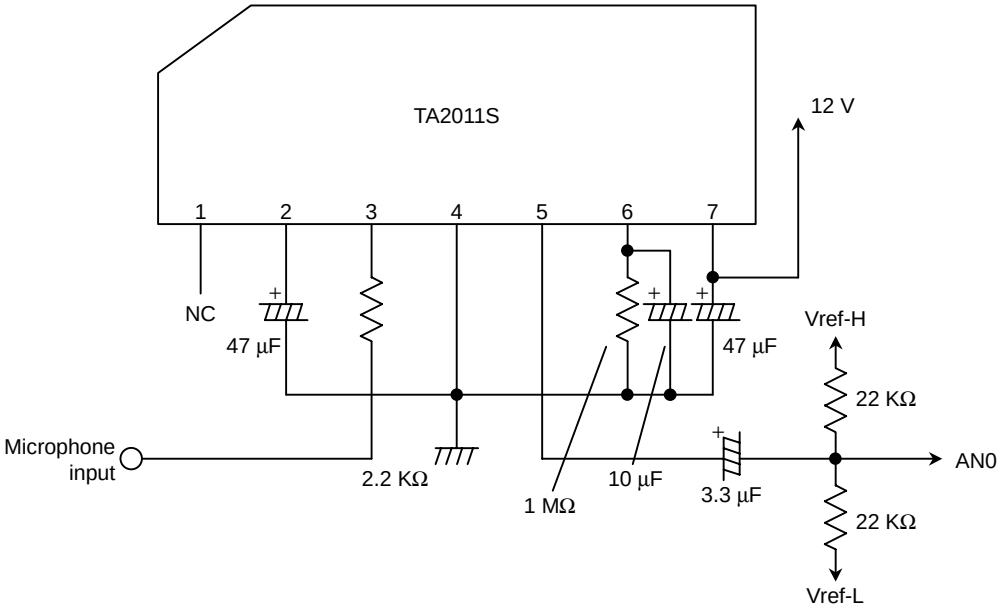


Figure 2.3.6 Voice input

2.3.7 RS-232C

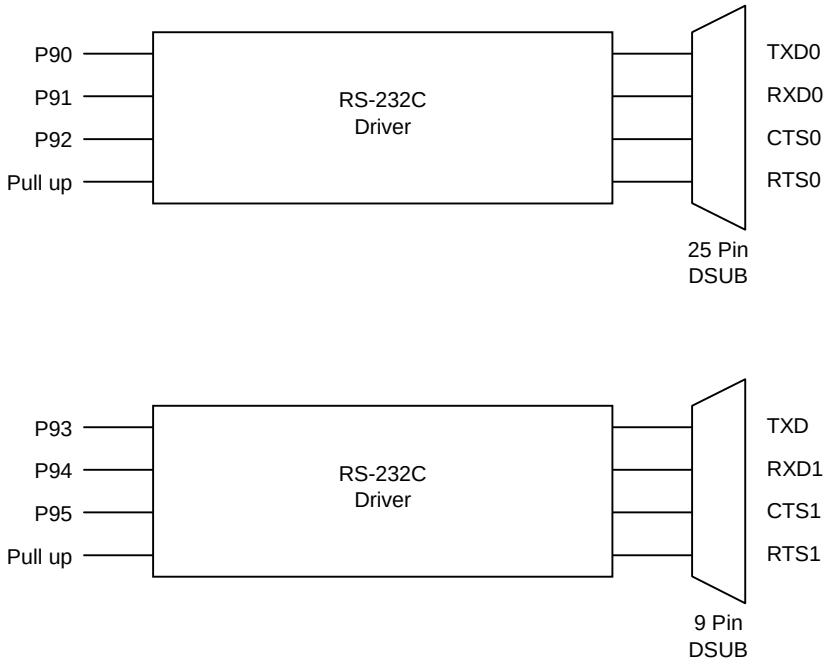


Figure 2.3.7 RS-232C

2.3.8 Light-emitting diodes

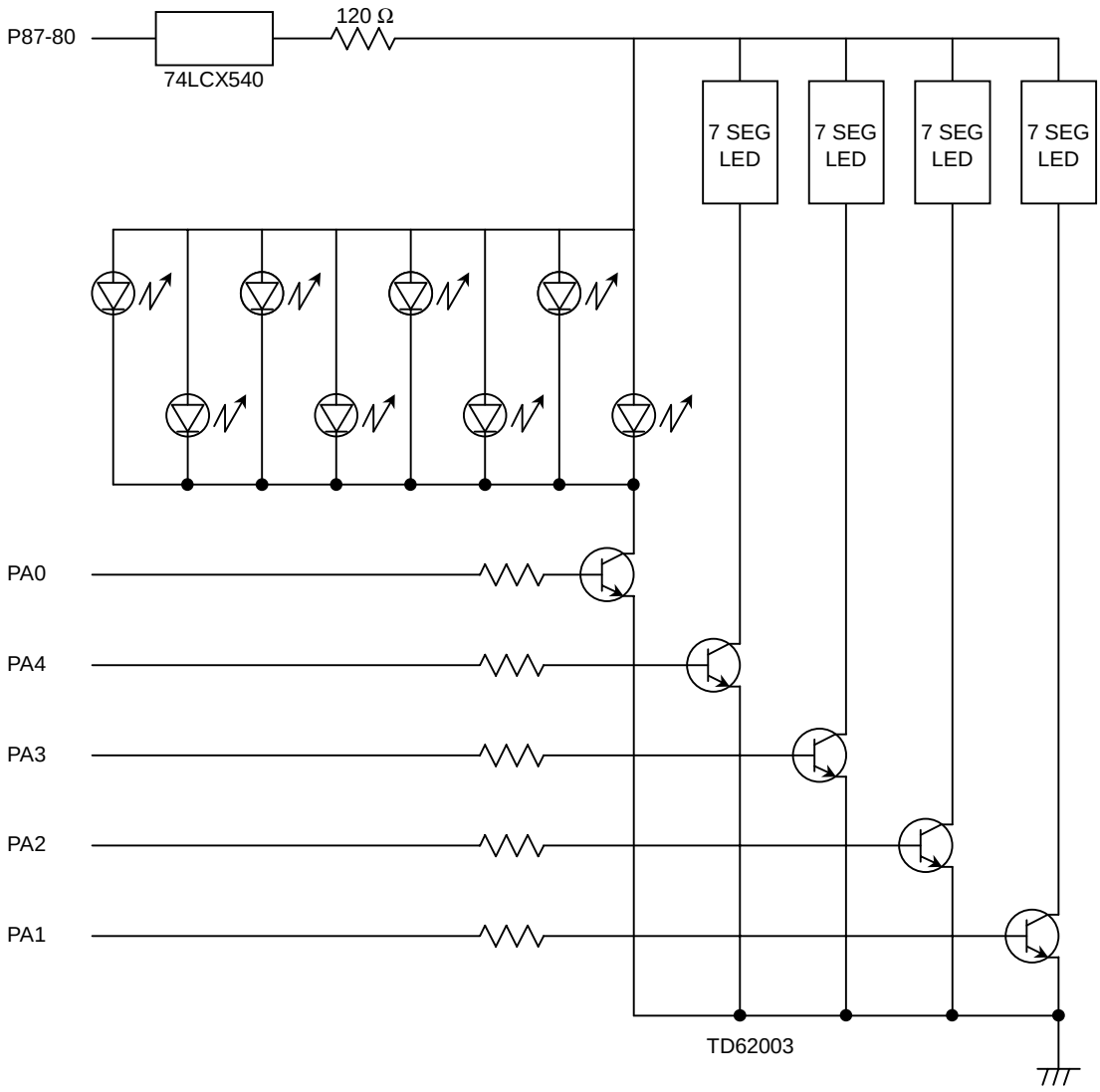


Figure 2.3.8 Light-emitting diodes

2.3.9 Diode matrix key switch

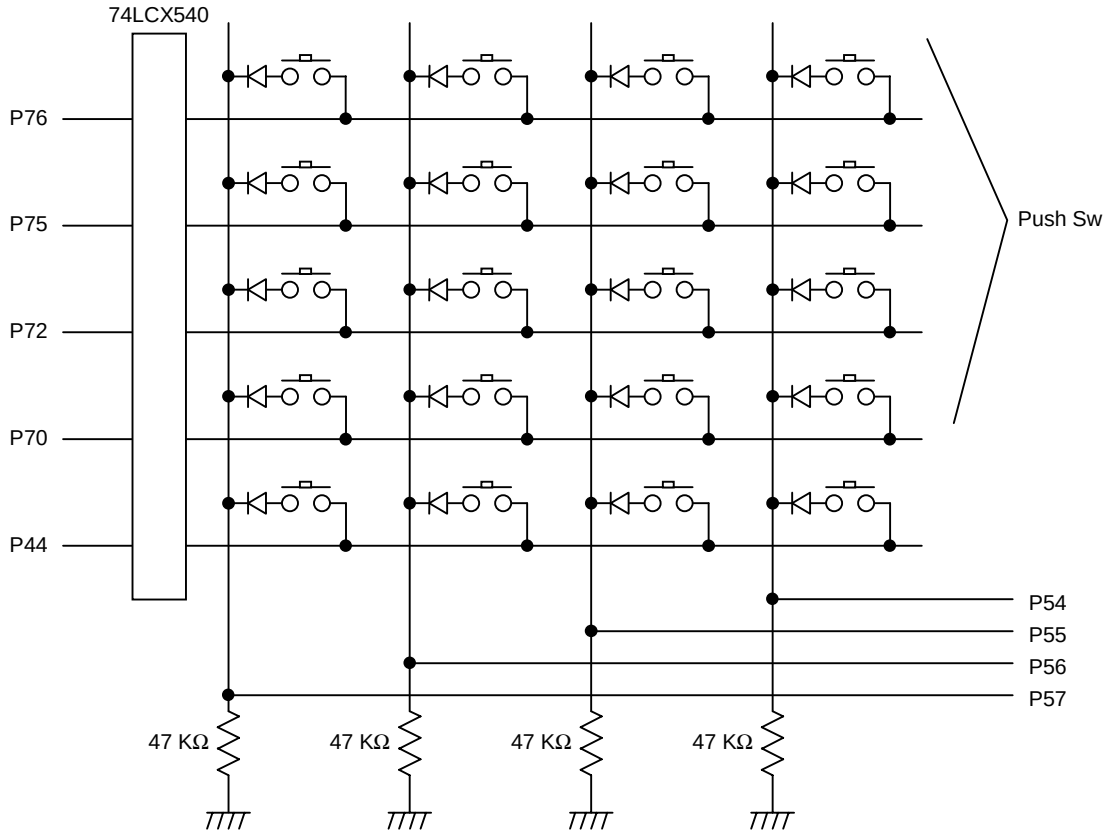


Figure 2.3.9 Diode matrix key switch

2.3.10 Piezo-electric buzzer

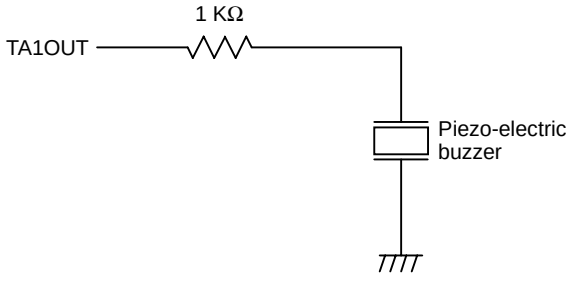


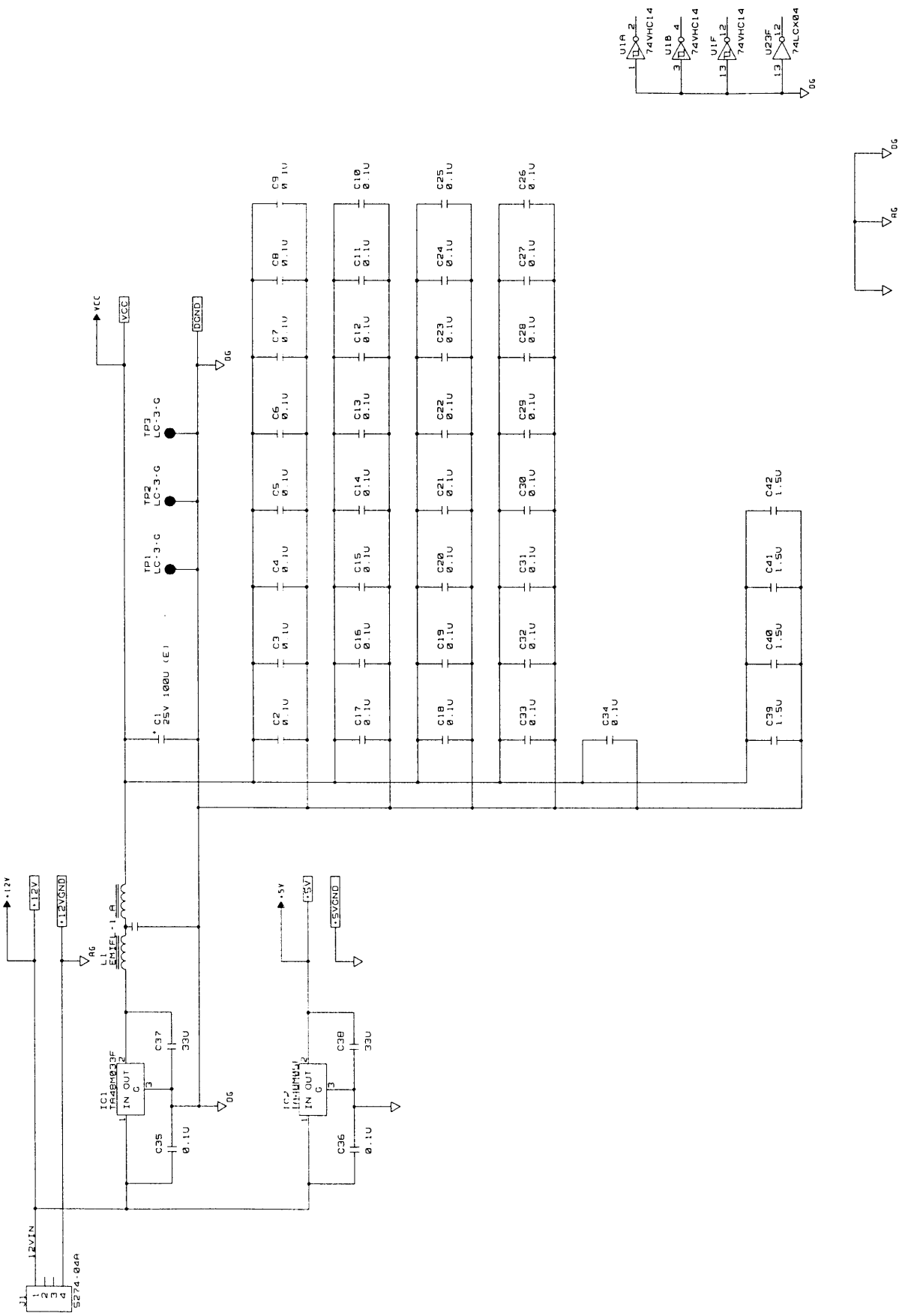
Figure 2.3.10 Piezo-electric buzzer

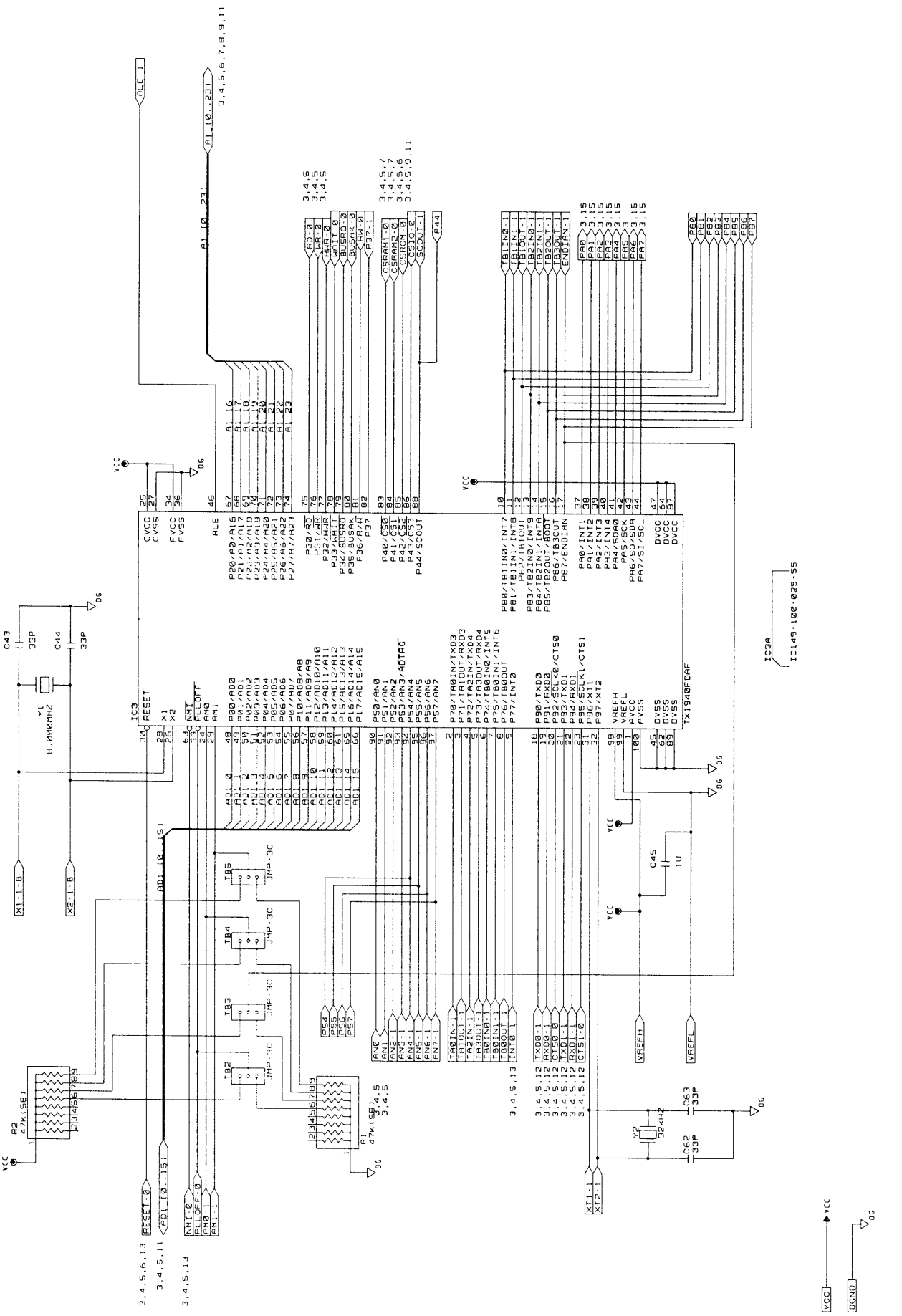
2.4 Parts List for TB1940

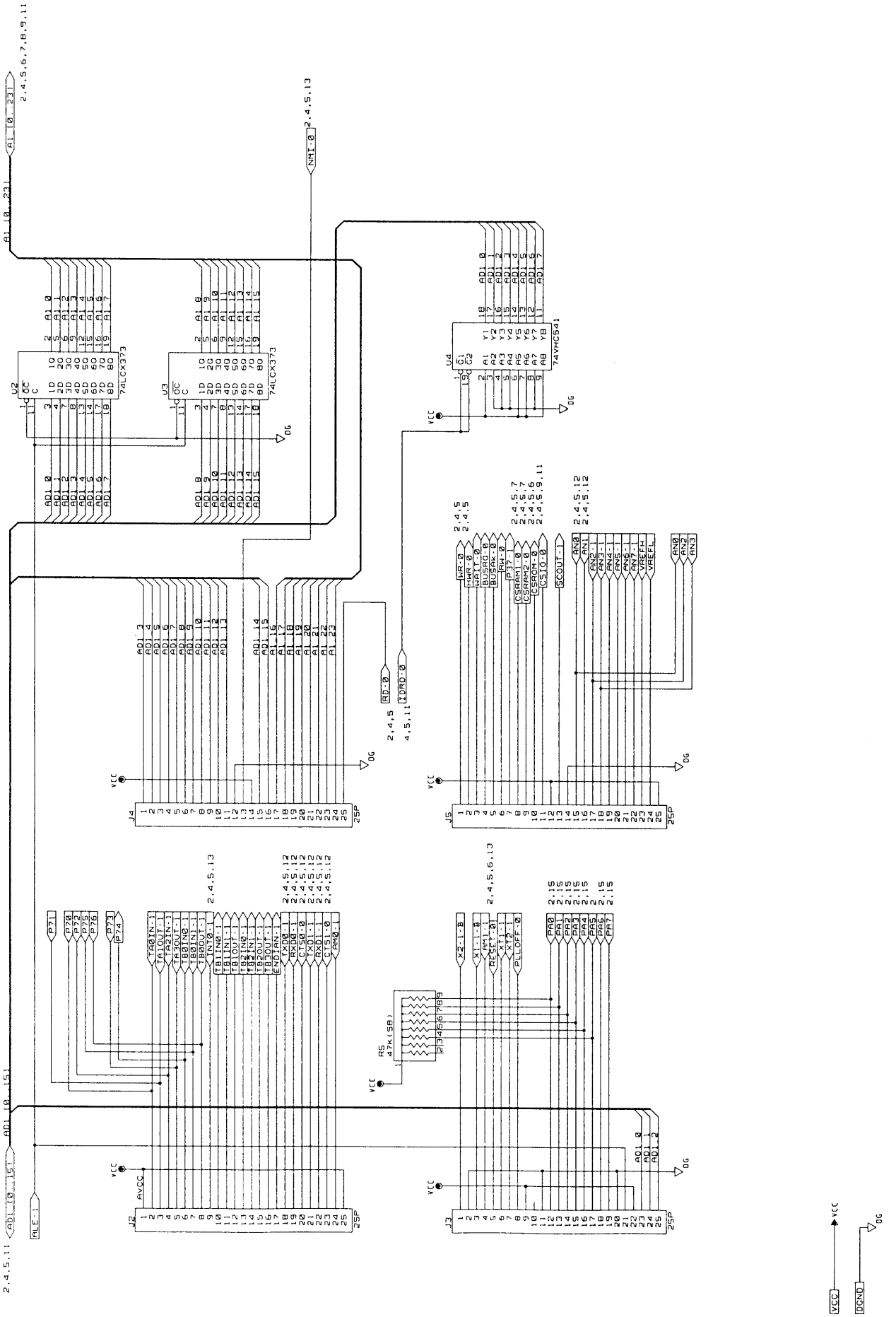
Part Name	Type Number	Rating	No. of Items
Buzzer	PKM11-4A0		1
Ceramic capacitors	DD104633CH330J50	33 pf	4
Ceramic capacitors	RPE114R155K50	1.5 μF	4
Ceramic capacitors	SR295F104Z	0.1 μF	44
Ceramic capacitor	SR305F105Z	1 μF	1
Electrolytic capacitor	KME25VB-100M	100 μF	1
Electrolytic capacitor	KME25VB-10M	10 μF	1
Electrolytic capacitors	KME25VB-33M	33 μF	2
Electrolytic capacitors	KME25VB-47M	47 μF	3
Electrolytic capacitor	KME50VB-3.3M	3.3 μF	1
Electrolytic capacitors	KME50VB-4.7M	4.7 μF	2
Coil	DSS310-55D223S		1
RS-232C connector (25-pin)	RDBD-25S-LNA		1
RS-232C connector (9-pin)	RDEB-9P-LNA		1
Diodes	1SS133		20
Diode	2FWJ42M		1
I^2C bus E^2PROM	NM24C02FLZEM		1
RS-232C drivers	MAX3243CAI		2
SRAM chips	TC55V1001AF-85		4
Flash ROM	TC58FVT400F-85		1
Microphone amp	TA2011S		1
Microcontroller	TMP1940FDBF		1
Regulator	TA48M033F		1
Regulator	TA48M05F		1
Logic IC	TC74LCX04F		1
Logic ICs	TC74LCX138F		3
Logic ICs	TC74LCX245F		2
Logic ICs	TC74LCX373F		2
Logic ICs	TC74VHC14F		2
Logic ICs	TC74VHC540F		2
Logic ICs	TC74VHC541F		6
Logic ICs	TC74VHC574F		2
7-segment LEDs	TLR336S		4
Light-emitting diodes	TLG102A	Green LED	16
Light-emitting diodes	TLR102A	Red LED	8
Variable resistor	RJ-6P100K		1
Resistors	RKLB4-221J	220 Ω × 4	4
Resistors	RKLB8-473J	47 KΩ × 8	6
Resistors	MFL1/4-101	100 Ω	3
Resistor	MFL1/4-102	1 KΩ	1
Resistors	MFL1/4-103	10 KΩ	7
Resistors	MFL1/4-105	1 MΩ	2
Resistors	MFL1/4-121	120 Ω	8
Resistor	MFL1/4-122	1.2 KΩ	1
Resistor	MFL1/4-123	12 KΩ	1
Resistors	MFL1/4-202	2 KΩ	3
Resistor	MFL1/4-222	2.2 KΩ	1
Resistors	MFL1/4-223	22 KΩ	2
Resistor	MFL1/4-391	390 Ω	1
Resistor	MFL1/4-392	3.9 KΩ	1

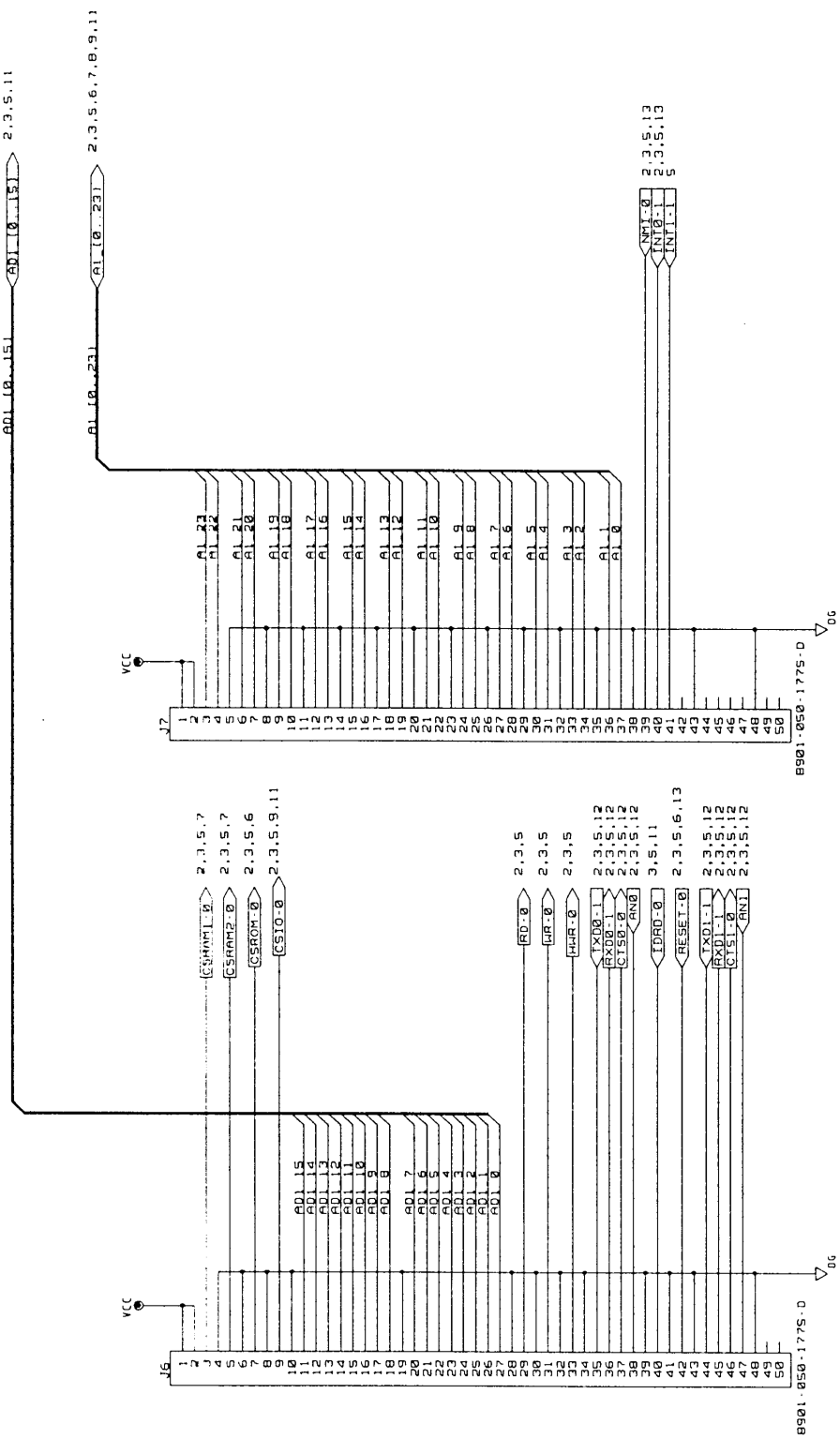
Part Name	Type Number	Rating	No. of Items
Resistors	MFL1/4-473	47 KΩ	7
Resistors	MFL1/8-203	20 KΩ	3
IC socket	IC149-100-025S5	For the CPU	1
IC socket	IC179-44600-110	For the F-ROM	1
Short pins	DIC-128A		4
Short terminals	DIC-149-3P		4
DIP switches	DSS104		2
Toggle switches	G-12AP		2
Push switches	DP1-120-K		20
Transistor array	MP4015		1
Transistor array	TD62503PA		1
Photo-interrupter	TLP862		1
Crystal resonator	C-2-TYPE-32K		1
Crystal resonator	CA-301-8M		1

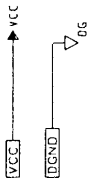
2.5 Circuit Diagram

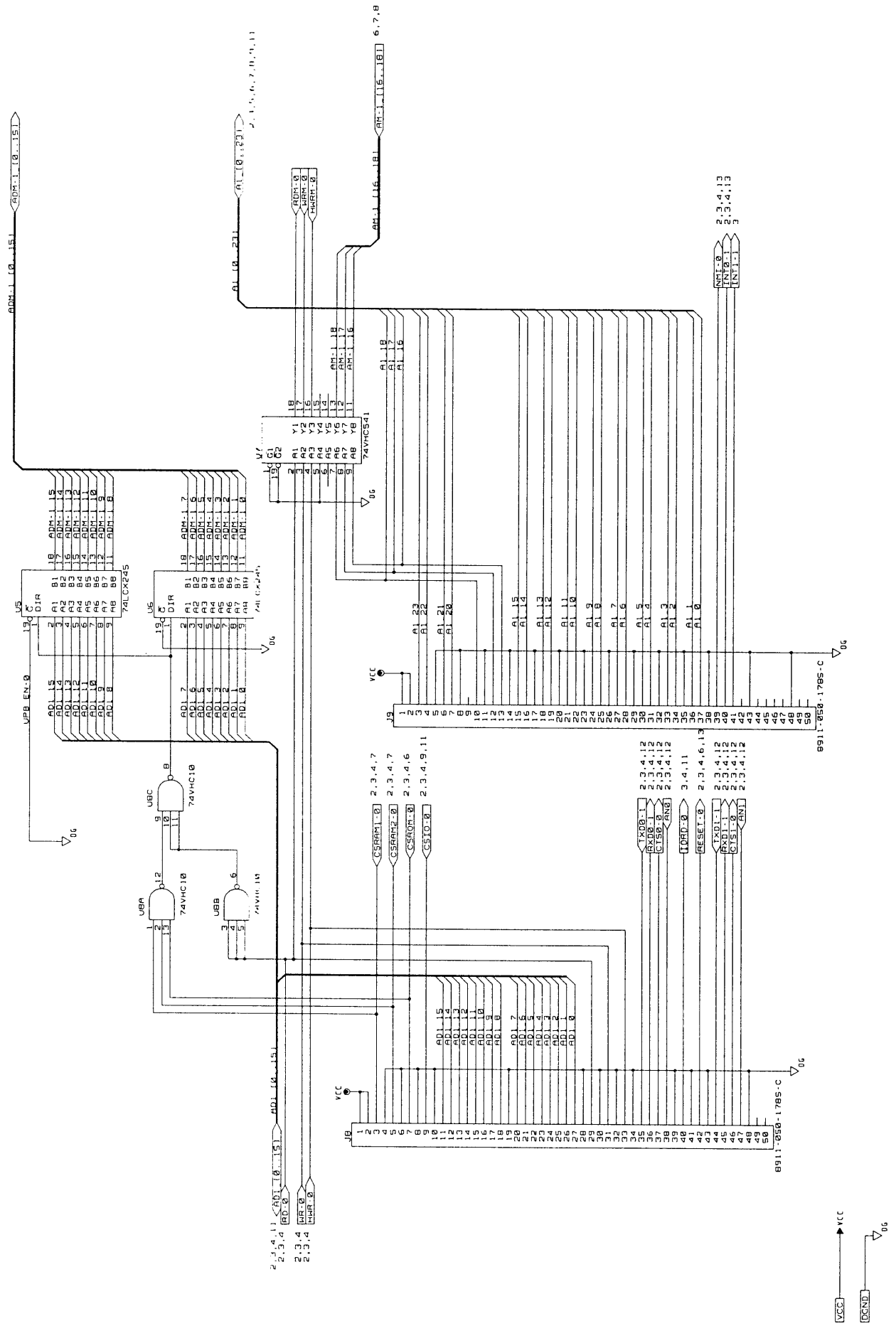


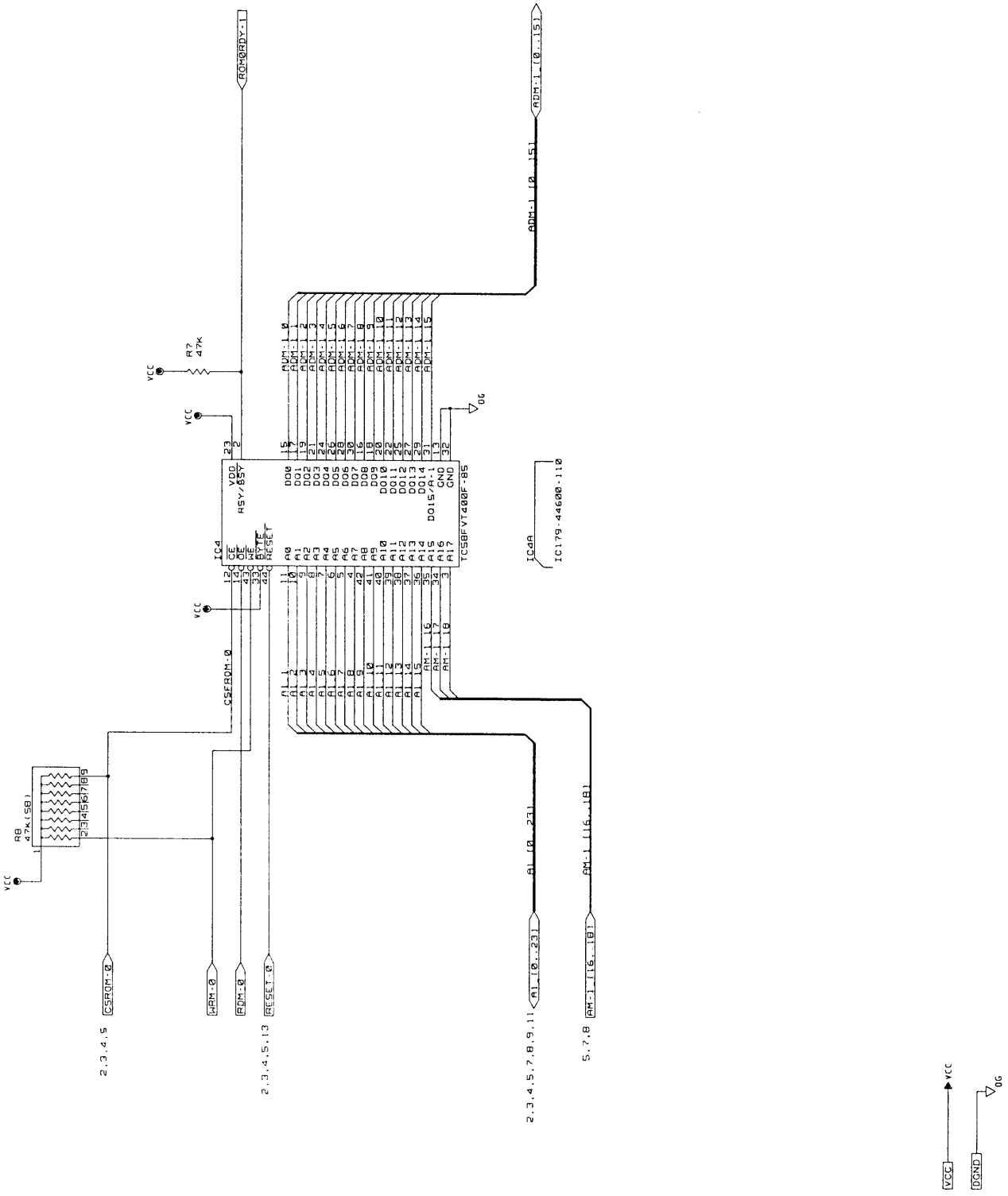

TX1940FDAF
IC3
IC3A
IC149-100-025-S5
Y1
8.000MHZ
Y2
32KHZ
C43
33P
C44
33P
C45
1U
C62
33P
C63
33P
R1
47K(SIP)
R2
47K(SIP)
JMP-3C
VCC
DG
RESET
PLLOFF
NMI
AM0
AM1
ALE
A1(0..23)
AD1(0..15)
VREFH
VREFL
DVSS
DVCC
AVSS
AVCC

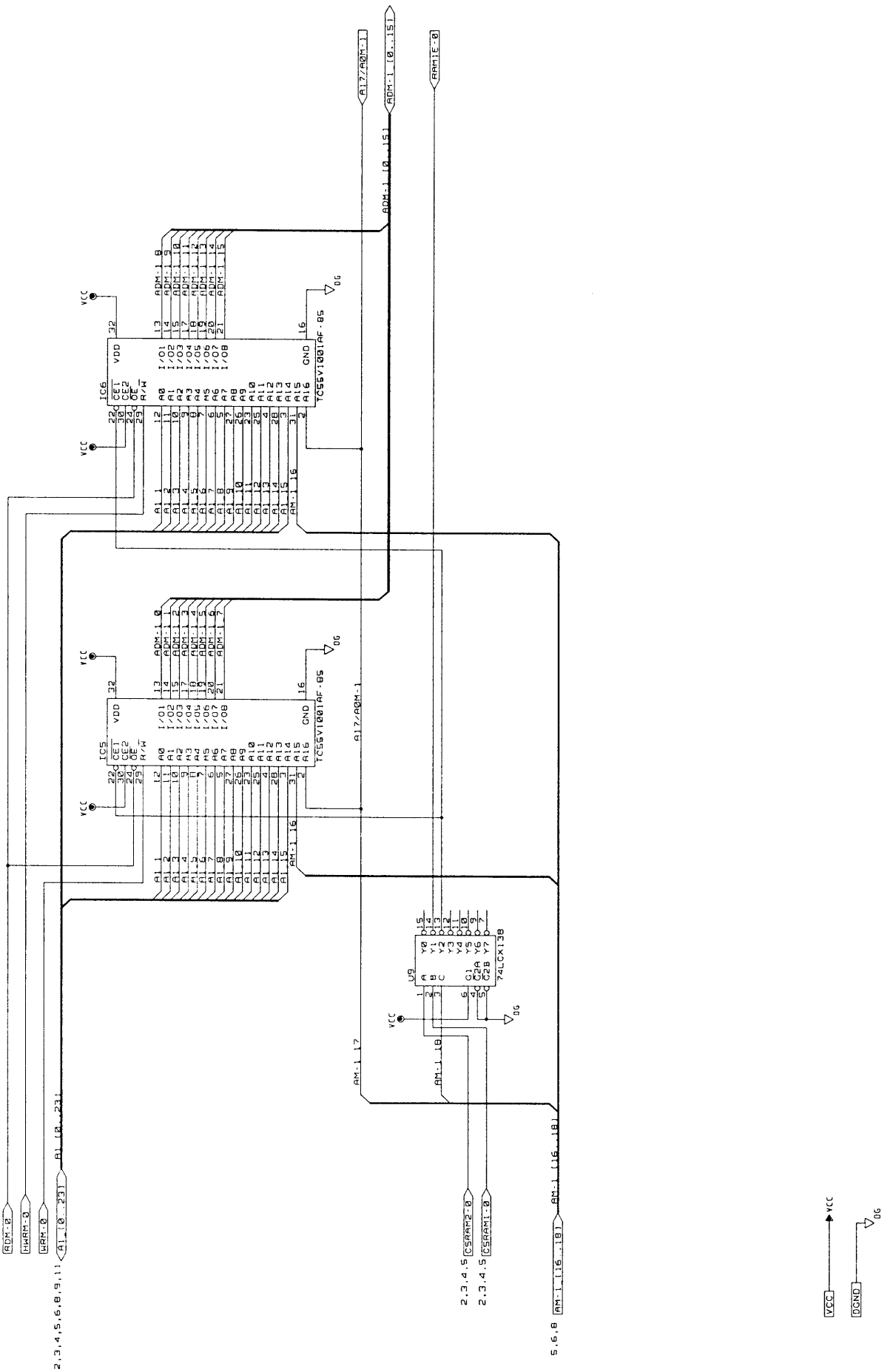

U2
74LCX373
U3
74LCX373
U4
74VHC541
J2
J3
J4
J5
R5
47K(SB)
NMI-0
ALE-1
AD1[0..15]
A1[0..23]
VCC
DGND

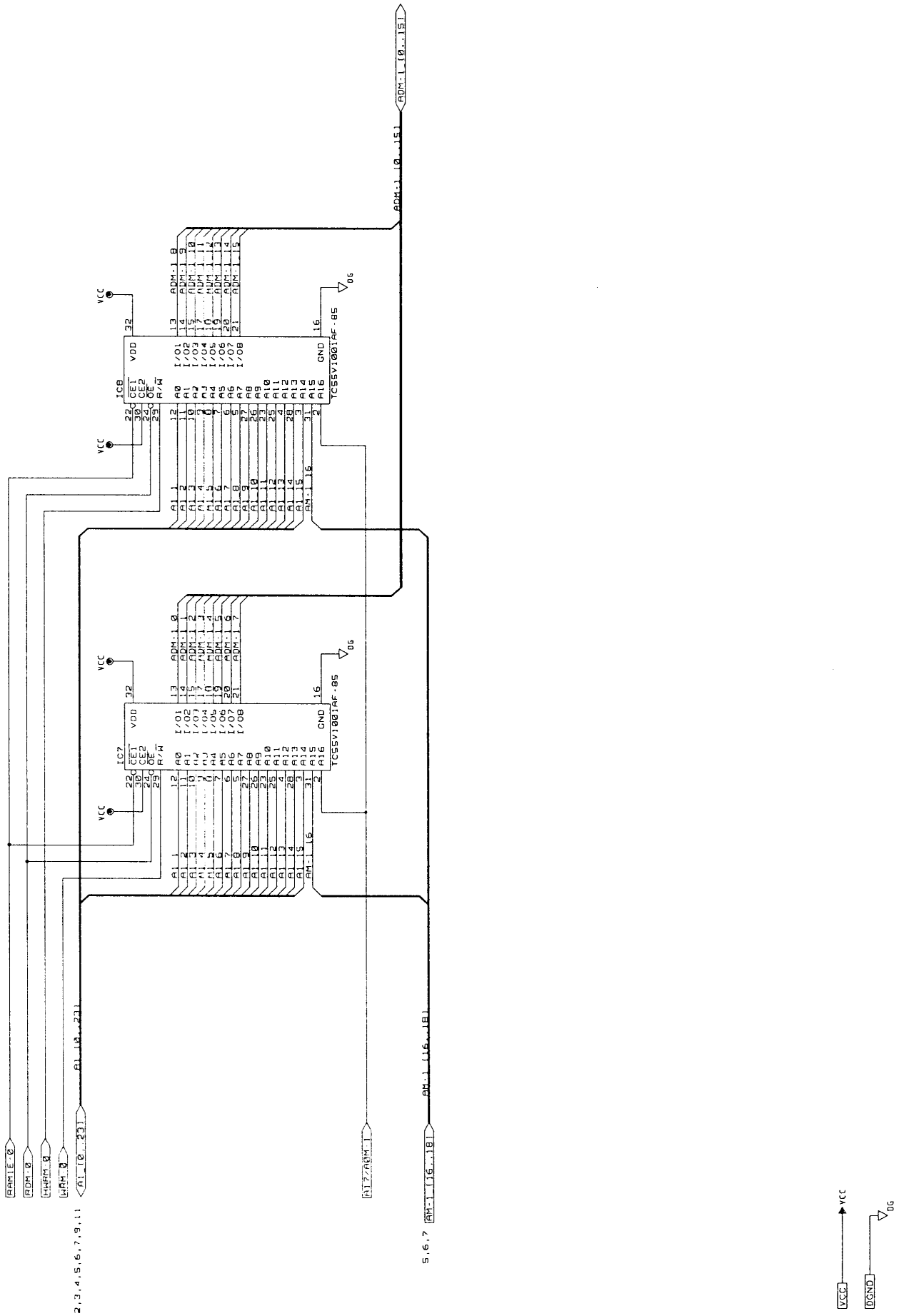

AD1[0..15]
2,3,5,11
A1[0..23]
2,3,5,6,7,8,9,11
INT2-0
INT1-1
2,3,5,13
VCC
DG
J7
B901-050-177S-D
CSNAND1-0
CSRAM2-0
CSROM-0
CSIO-0
RD-0
WR-0
HWR-0
TXD0-1
RXD0-1
CTS0-0
AN0
IDRD-0
RESET-0
TXD1-1
RXD1-1
CTS1-0
AN1
2,3,5,12
2,3,5,13
J6
B901-050-177S-D

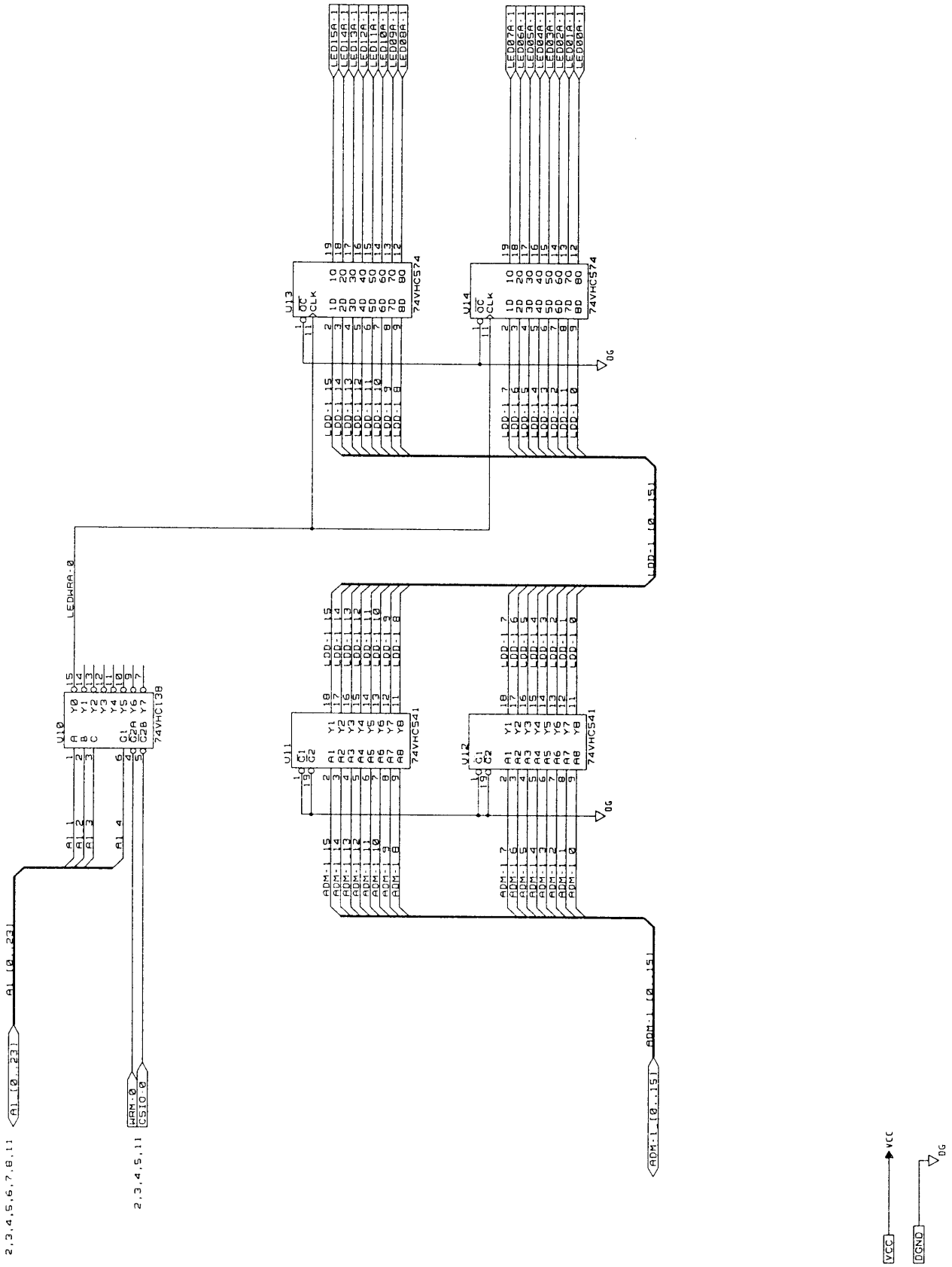

VCC
DGND

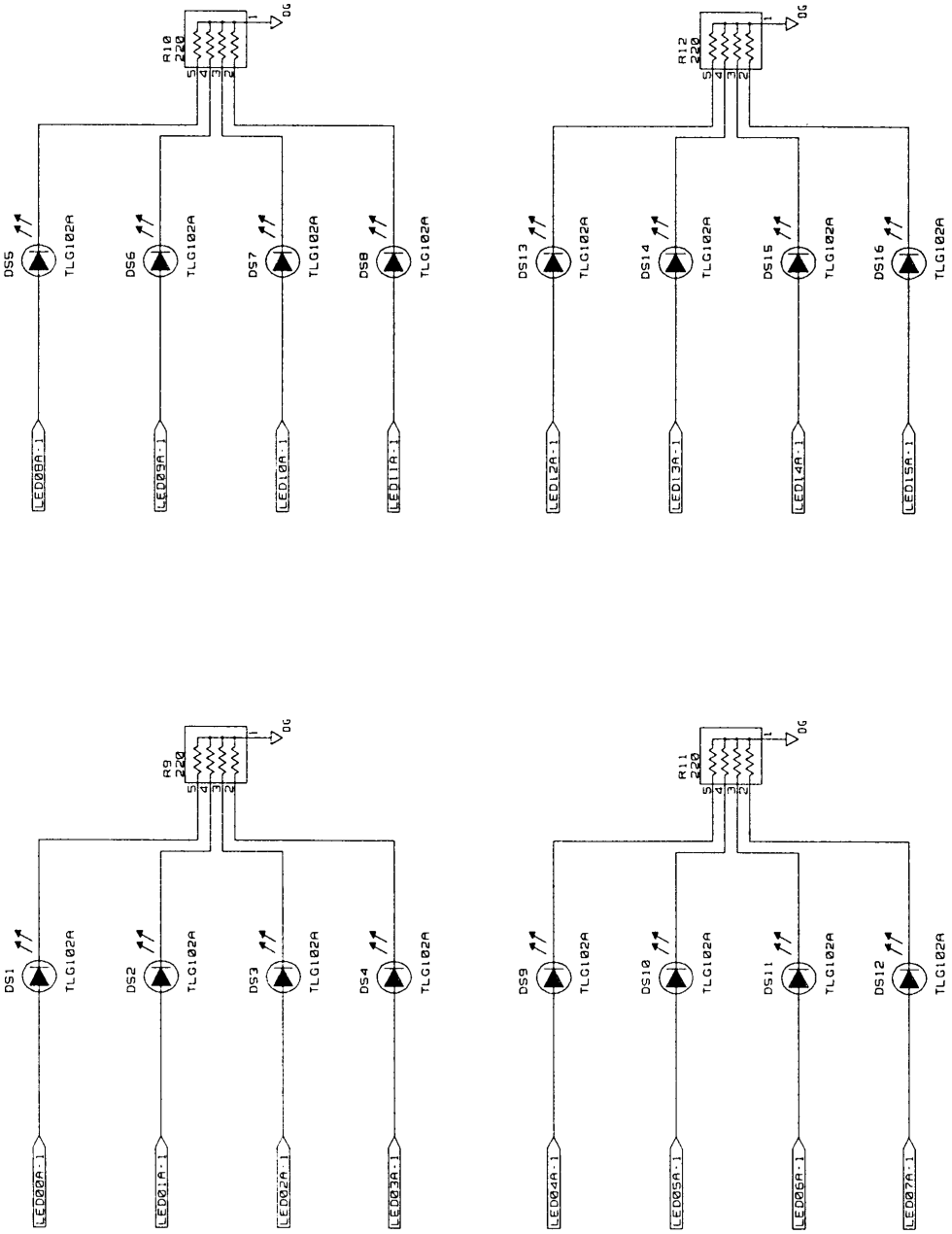

74LCX245
74VHC541
74VHC10
8911-050-178S-C
VCC
DGND

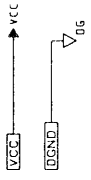

ROMRDY-1
R7
47K
VCC
DG
IC4
TC58FVT400F-85
IC4A
IC179-44600-110
ADM-1[0..15]
AM-1[16..18]
A1[0..23]
CSFROM-0
WAM-0
ADM-0
RESET-0
RP
47K(5B)
2,3,4,5
2,3,4,5,13
2,3,4,5,7,8,9,11
5,7,8
VCC
DGND

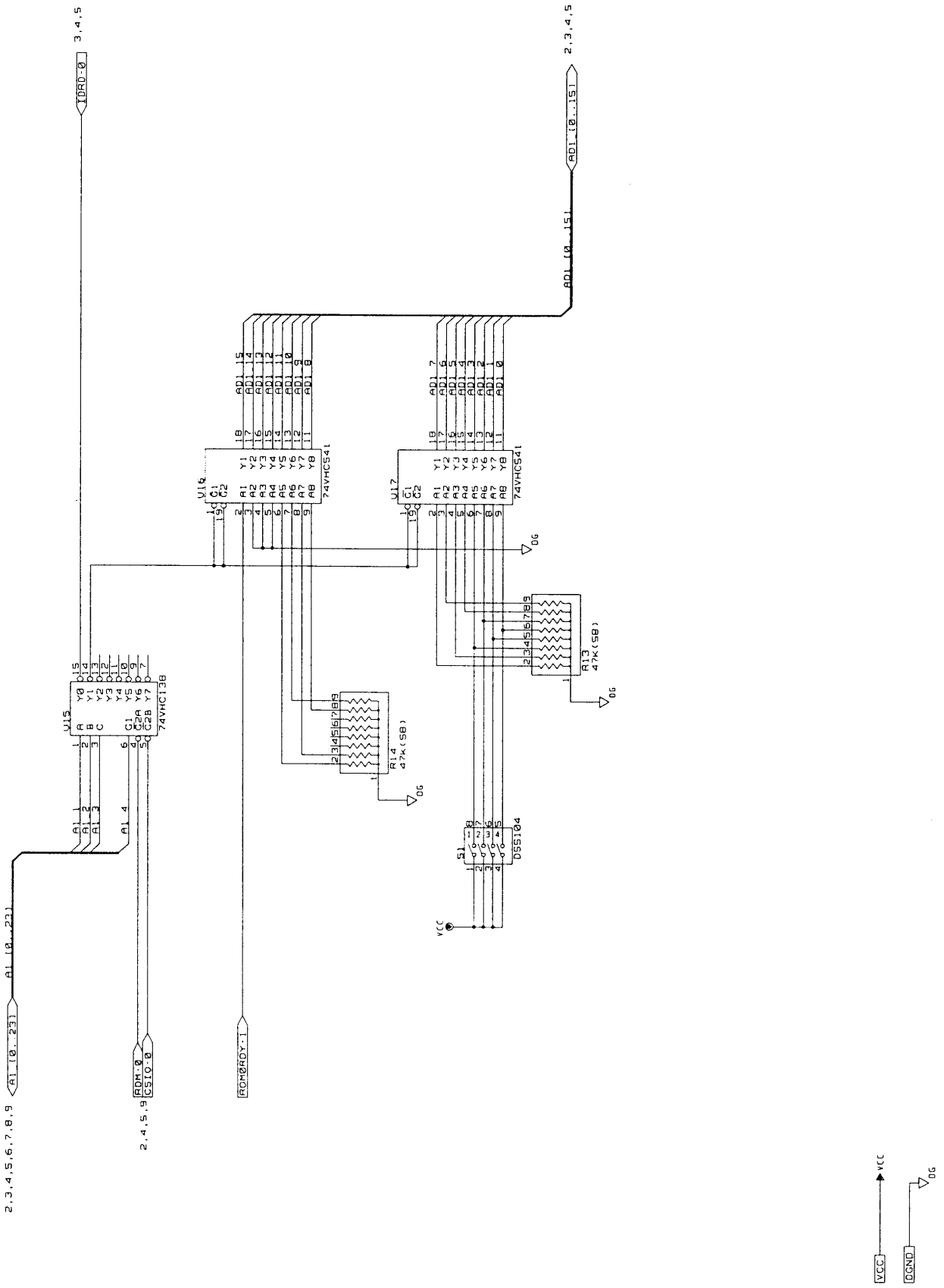

A17/ADM-1
ADM-1 [0..15]
RAM1E-0
TC55V1001AF-85
VDD
GND
VCC
DG
ADM-1 [0..15]
A17/ADM-1
AM-1 17
AM-1 18
AM-1 [16..18]
U9
74LCX138
CSRAM2-0
CSRAM1-0
ADM-0
HWRM-0
WRM-0
A1 [0..23]
2,3,4,5,6,8,9,11
2,3,4,5
2,3,4,5
5,6,8
VCC
DGND

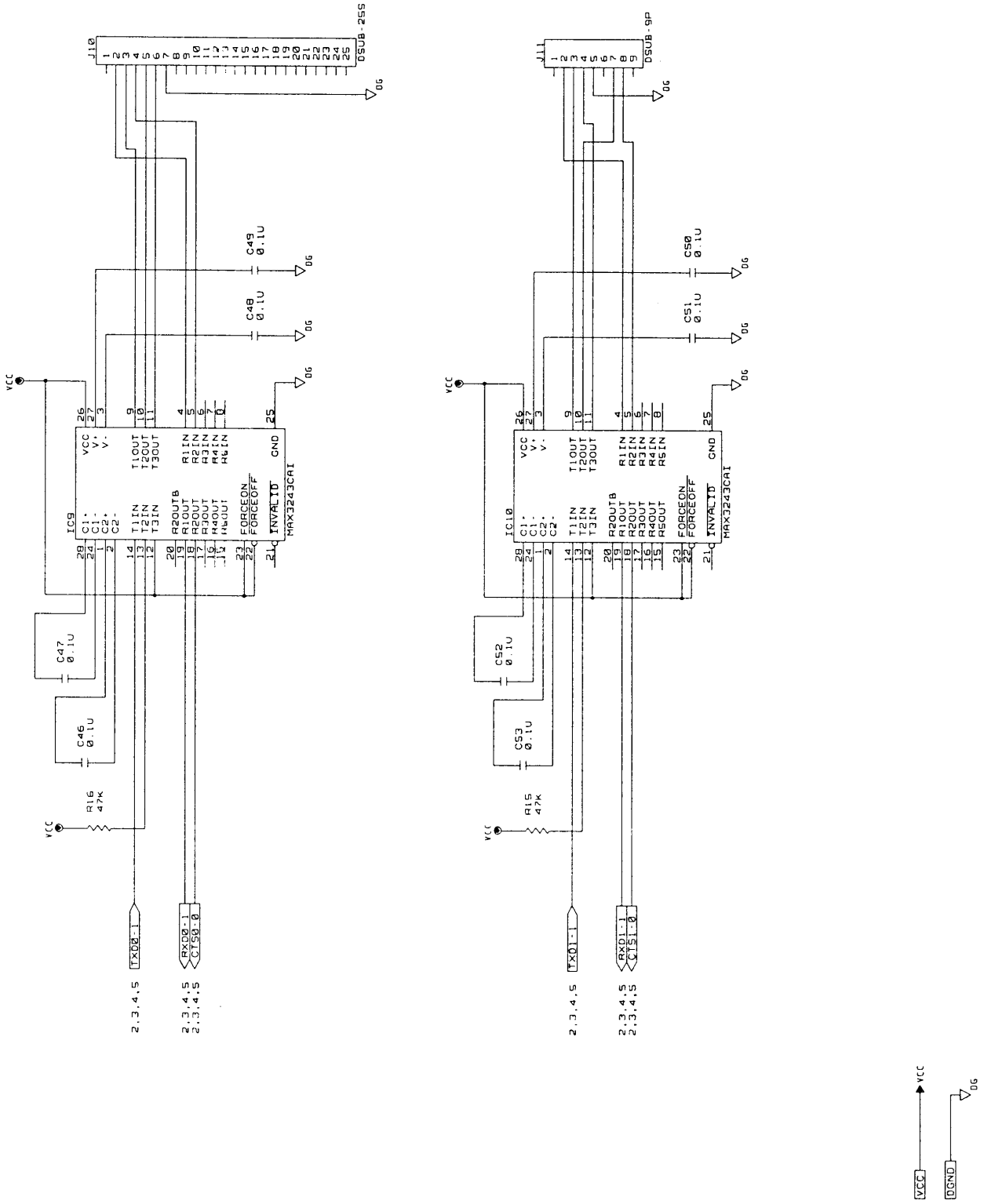

RAMIE_0
RDM_0
HWRM_0
WRM_0
A1 [0..23]
2,3,4,5,6,7,9,11
IC8
IC7
TC55V1001AF-85
VCC
DG
ADM-1 [0..15]
A17_ADM_1
AM-1 [16..18]
5,6,7
VCC
DGND

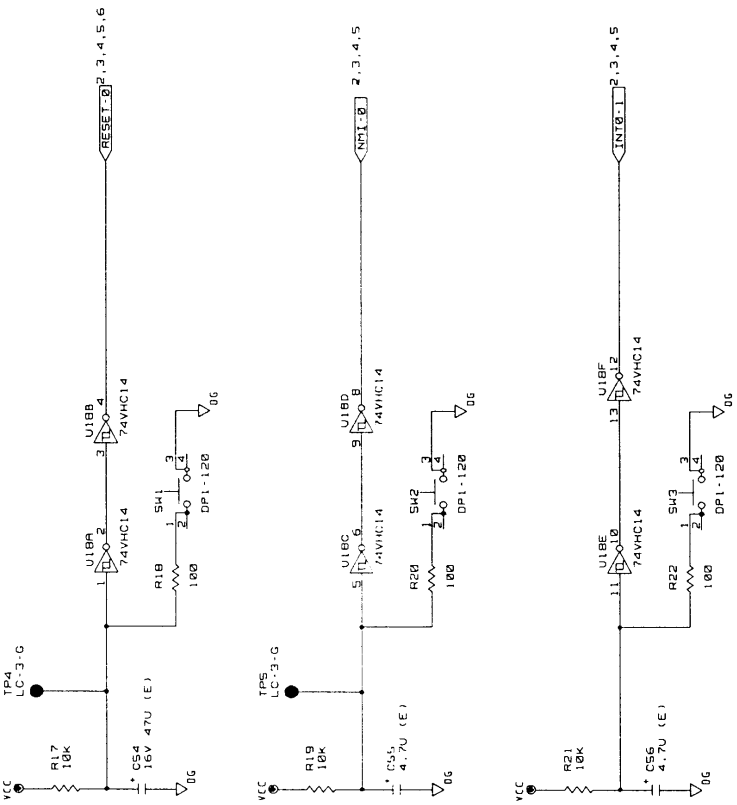

U13
U14
74VHC574
LED15A:1
LED14A:1
LED13A:1
LED12A:1
LED11A:1
LED10A:1
LED09A:1
LED08A:1
LED07A:1
LED06A:1
LED05A:1
LED04A:1
LED03A:1
LED02A:1
LED01A:1
LED00A:1
LDD:1[0..15]
DG
U10
74VHC138
LEDWRA:0
A1[0..23]
HRM:0
CSIO:0
U11
U12
74VHC541
ADM:1[0..15]
2,3,4,5,6,7,8,11
2,3,4,5,11
VCC
DGND

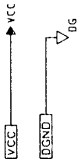

R10
220
R12
220
R9
220
R11
220
DG
DS1
DS2
DS3
DS4
DS5
DS6
DS7
DS8
DS9
DS10
DS11
DS12
DS13
DS14
DS15
DS16
TLG102A
LED00A-1
LED01A-1
LED02A-1
LED03A-1
LED04A-1
LED05A-1
LED06A-1
LED07A-1
LED08A-1
LED09A-1
LED10A-1
LED11A-1
LED12A-1
LED13A-1
LED14A-1
LED15A-1

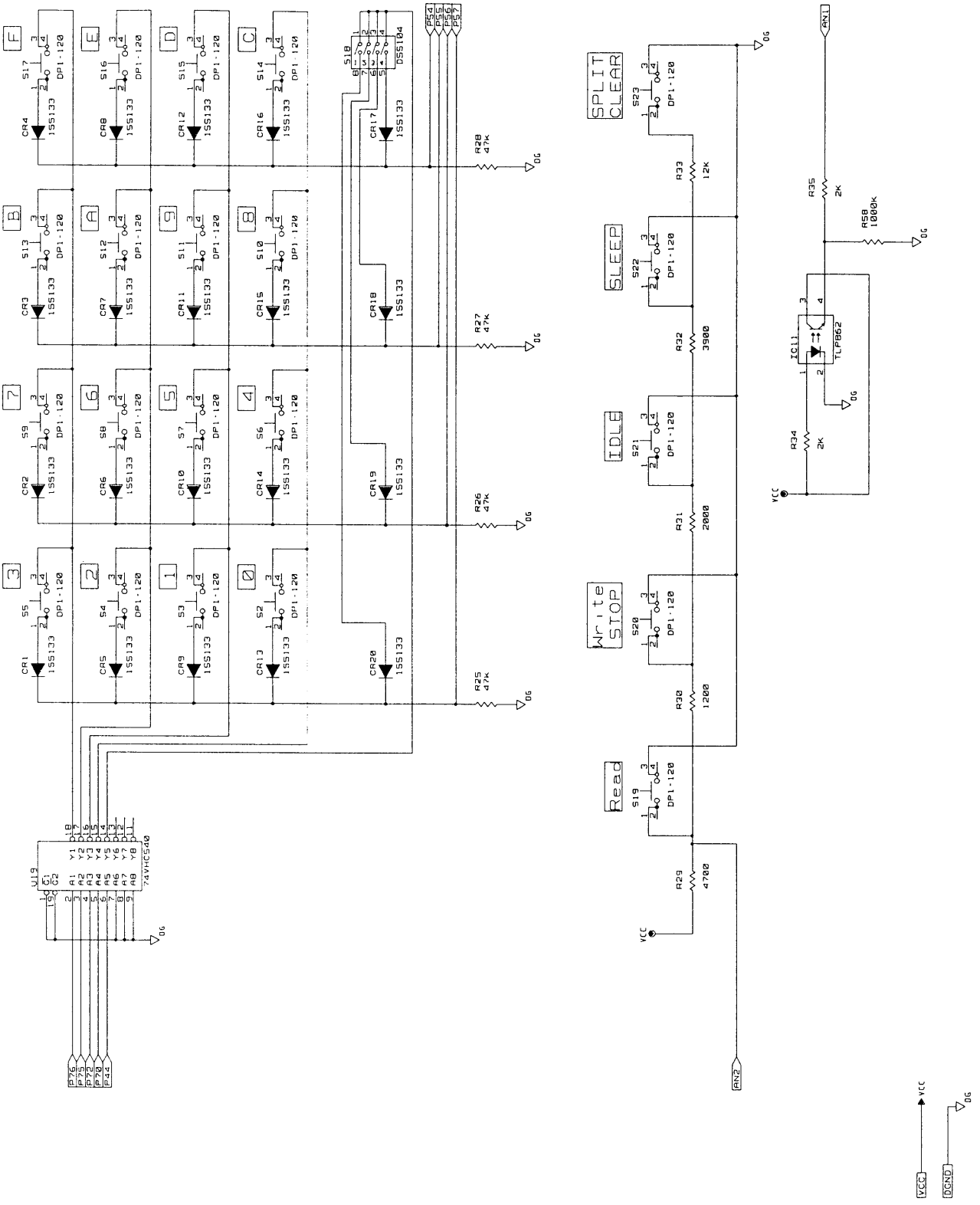

VCC
DG
VCC
DGND

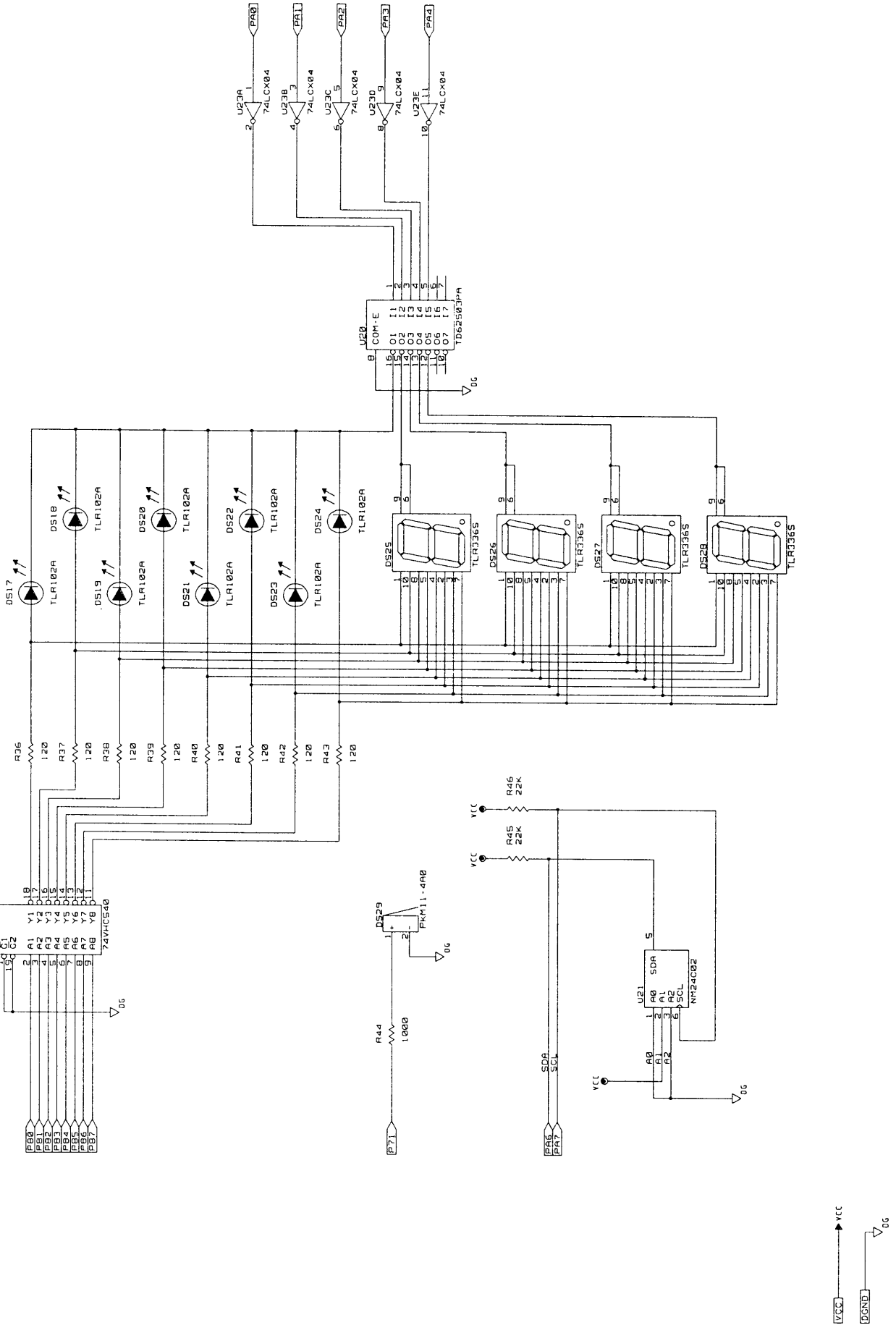

IORD-0
3,4,5
AD1[0..15]
2,3,4,5
A1[0..23]
2,3,4,5,6,7,8,9
ROM-0
CSIO-0
2,4,5,9
ROM0RDY-1
U15
74VHC138
U16
U17
74VHC541
R13
47K(SB)
R14
47K(SB)
S1
DS5104
VCC
DG
VCC
DGND

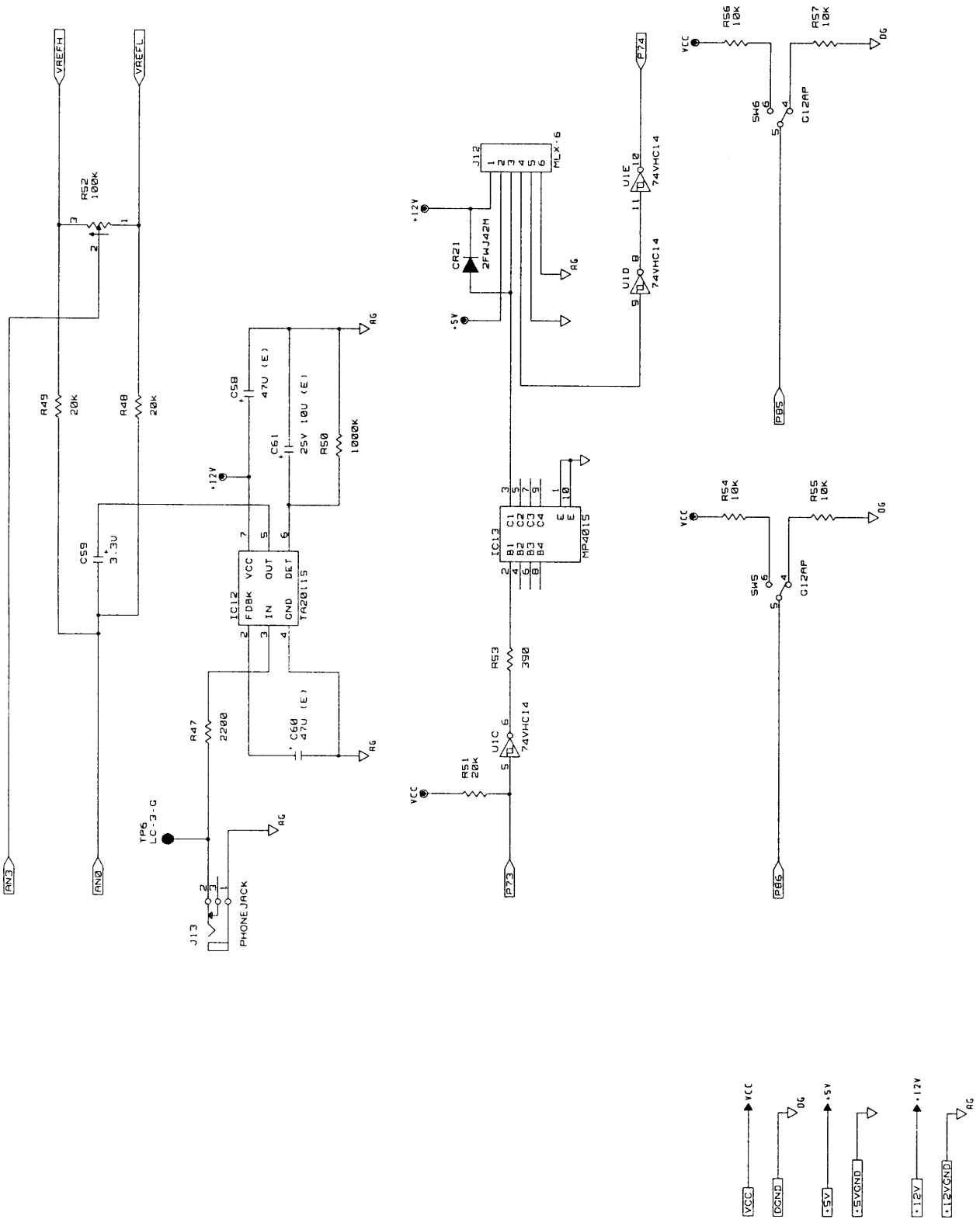

J10
DSUB-25S
J11
DSUB-9P
IC9
IC10
MAX3243CAI
C47 0.1U
C46 0.1U
C48 0.1U
C49 0.1U
C52 0.1U
C53 0.1U
C50 0.1U
C51 0.1U
R16 47K
R15 47K
VCC
DG
TX00-1
RX00-1
CTS0-0
TX01-1
RX01-1
CTS1-0
2,3,4,5
VCC
DGND


RESET-0
2,3,4,5,6
NMI-0
2,3,4,5
INT0-1
2,3,4,5
U18A
U18B
U18C
U18D
U18E
U18F
74VHC14
TP4
TP5
LC-3-G
R17
R18
R19
R20
R21
R22
10k
100
SW1
SW2
SW3
DPI-120
C54
C55
C56
16V 47U (E)
4.7U (E)
VCC
DG


VCC
DGND
VCC
DG


SPLIT CLEAR
SLEEP
IDLE
Write STOP
Read
74VHC540
TLP862
DSS104
ISS133
DP1-120
VCC
DGND


PA0
PA1
PA2
PA3
PA4
74LCX04
U20
TD62503P-A
DS17
DS18
DS19
DS20
DS21
DS22
DS23
DS24
TLR102A
DS25
DS26
DS27
DS28
TLR336S
R36
R37
R38
R39
R40
R41
R42
R43
120
U22
74VHC540
DS29
PKM11-4A0
R44
1000
R45
22K
R46
22K
VCC
U21
NM24C02
SDA
SCL
A0
A1
A2
P71
P86
P87
P80
P81
P82
P83
P84
P85
DG
VCC
DGND


VREFH
VREFL
R52 100K
R49 20K
R48 20K
C59 3.3U
C58 47U (E)
C61 25V 10U (E)
R50 1000K
+12V
IC12 FDBK VCC OUT IN GND DET TA20115
R47 220Ω
C60 47U (E)
AG
TP6 LC-3-G
J13 PHONEJACK
AN3
AN0
J12 MLX-6
CR21 2FW42M
+5V
IC13 MP4015
C1 C2 C3 C4 B1 B2 B3 B4 E E
R53 390
U1C 74VHC14
R51 20K
VCC
P73
U1D 74VHC14
U1E 74VHC14
P74
R56 10K
R57 10K
SW6 G12AP
P85
R54 10K
R55 10K
SW5 G12AP
P86
DG
VCC
DGND
+5V
+5VGND
+12V
+12VGND
AG

Chapter 3 Description of the Software

3.1 Introduction

Sections 3.1.1 and 3.1.2 of this chapter explain the common start-up routine and link command file which are used by all the sample programs. Interrupt handling is explained in Section 3.1.3.

The sample programs are compiled and assembled using Version 1.1 of the C compiler and assembler package for the TX19 Family of Toshiba microprocessors.

3.1.1 Overview of the start-up routine

The start-up routine is the program which is executed immediately after a reset; it performs the following processing operations:

- Definition of start-up routine variables and the like
- Initialization of the registers
- Initialization of system status (CP0 system control coprocessor settings)
- Initialization of external variable areas
- Call to *main* function

In addition, it may be necessary to include the following:

- Definitions of ANSI C library external variables and of certain functions from the ANSI C library
- Interrupt handler

This section explains the various definitions and processes which are coded in the start-up routine.

3.1.1.1 Start-up processing

For the purposes of explanation, the start-up routine is divided into four separate sections as follows:

1. Symbol definition section: Defines symbols and the like used in the start-up routine.
2. Start-up proper: Initializes the system, registers and memory, and calls the function *main*.
3. Exception handler section: Describes the processing which will be performed when an exception occurs.
4. Interrupt handler section: Describes the processing which will be performed when an interrupt occurs.

■ Symbol definition section

A typical start-up routine definition section is as follows:

```
;=== [User definition section]  This section must always be included.===  (a)
StackBase   equ             0x0xffffba00     ;Address of stack base
HeapTop     equ             0x0xffffba00     ;Address of base of heap area
HeapSize    equ             0x400            ;Size of heap area

;### Interrupt Vector Register                                            (b)
;IVR        equ              0xffffe040

;=== [External functions and variable definitions]
     This section must always be included.===                            (c)
```

```
t_area      section   data     small   ;Section where tiny-area variables whose initial
                                        values are undefined are located
t_data      section   data     small   ;Section where tiny-area variables whose initial
                                        values are defined are located
t_const     section   romdata  small   ;Section where tiny-area constants are located
n_area      section   data     medium  ;Section where near-area variables whose initial
                                        values are undefined are located
n_data      section   data     medium  ;Section where near-area variables whose initial
                                        values are defined are located
n_const     section   romdata  medium  ;Section where near-area constants are located
f_area      section   data     large   ;Section where far-area variables whose initial
                                        values are undefined are located
f_data      section   data     large   ;Section where far-area variables whose initial
                                        values are defined are located
f_const     section   romdata  large   ;Section where far-area constants are located

extern      large     TinyData,NearData,FarData
extern      large     TinyArea,NearArea,FarArea
extern      large     tinyAreaSize,nearAreaSize,farAreaSize
extern      large     tinyDataSize,nearDataSize,farDataSize
extern      large     tinyAreaOrg,nearAreaOrg,farAreaOrg
extern      large     tinyDataOrg,nearDataOrg,farDataOrg

extern      large     _main           ;External reference declarations for main() written
                                       using 32-bit ISA instruction set instructions

;When using interrupt functions, insert external declarations here.       (d)

;=== [Heap area definition]
     This section must always be included when malloc, calloc or realloc are used.===

n_data      ection    ata      medium
            public    _SBRK_break      ;Start pointer for heap area
            public    _SBRK_size       ;Size of heap area
            public    __allocb         ;Control pointer for heap area
_SBRK_break           dw       HeapTop
_SBRK_size            dw       HeapSize
_ _allocb             dw       0

HeapArea    section   data     abs=HeapTop           align=1,1
heap        dsb       HeapSize         ;Heap area

;=== [error definition]
     This section must always be included when arithmetic functions are used.===

n_area      section   data      medium
            public    _errno
_errno      dsw       1


;=== [Standard library functions (32-bit ISA): exit()]
     Rewrite this section before use. ===                                 (e)
f_code section code isa32
            public    _exit
_exit:      j         _exit
            nop

;=== [Standard library functions (16-bit ISA): exit()]
     Rewrite this section before use. ===                                 (f)
n_code section code isa16
            public     .exit
.exit:      b          .exit

;=== [Standard library functions (32-bit ISA): abort()]
     Rewrite this section before use.===                                  (g)
f_code section code isa32
            public     _abort
_abort:     j          _abort
            nop

;=== [Standard library functions (16-bit ISA): abort()]
     Rewrite this section before use.===                                  (h)
n_code section code isa16
            public     .abort
.abort:     b          .abort
```

The letters shown on the extreme right in the program listing correspond to the letters shown in the text in brackets before each section title.

(a) User definition section
This section defines the base addresses for the stack and heap areas, and the size of the heap area. In this case the stack and heap areas have the same base address, with the stack growing downwards in memory and the heap area growing upwards. The heap size is 400H; hence the heap area ranges from FFFFBA00H~FFFFBDFFH.

(b) Declaration of the Interrupt Vector Register
Declaring the Interrupt Vector Register IVR in this fashion facilitates the writing of code for the vector processing of hardware interrupts. For the TX1940 Series, IVR is at address FFFFE040H.

(c) Definition of external functions and variables
For some MCUs, part of the *tiny-area* (0H~00007FFFH) is missing. If this is the case, delete or comment out all declarations and routines associated with the *tiny-area*. In the above example this would entail deleting (or commenting out) *t_area*, *t_data*, *t_const*, *TinyArea*, *tinyAreaSize*, *tinyAreaOrg*, *TinyData*, *tinyDataSize* and *tinyDataOrg*. The same applies to the data initialization section explained below.

(d) External declaration of interrupt functions
If interrupt functions are to be used, they must be declared using *extern* declarations in the same way that the *main* function is declared. This enables the interrupt function names to be referenced in the interrupt vector definition section. Because interrupt functions are always written using 32-bit ISA instruction set instructions, they must be declared as shown below.

```
extern large    _intrptFunc
```

The above example illustrates the declaration of the 32-bit ISA function *intrptFunc*.

(e), (f), (g) and (h) Standard library functions
The standard library functions *exit*() and *abort*() are not included in the standard libraries (c9ib.lib and c9il.lib). To use these functions, it is necessary to write some additional program statements. As in the above example, this program code can be included in the file which contains the start-up routine.

■ Example of the start-up proper

```
ResetStart section code isa32
         public    _ _startup
_ _startup:                                                              a)
;------------------------------------------------------------------------
;[Register initialization] This section must always be included.         b)

        lui     sp,hi(StackBase)          ;Sets stack pointer.
        addiu   sp,sp,lo(StackBase)
        lui     gp,hi(__gp)               ;Sets global pointer.
        addiu   gp,gp,lo(__gp)

;Initializing the system control coprocessor (CP0)                       c)
        lui     r2,0x1040
        addiu   r2,r2,0x0000
        mtc0    r2,r12                    ;Sets Status Register.
```

```
;Clearing IVR register to 0                                              d)
        lui     r2,hi(IVR)
        addiu   r2,r2,lo(IVR)
        addu    r3,r0,r0
        sw      r3,0(r2)
;[Halting Watchdog timer]                                                e)
;
        lui     r4,0xffff
        ori     r4,r4,0xf090
        sb      r0,0(r4) ;WDMOD<WDTE>(0xfffff090) ← 0x00
        addiu   r2,r0,0xb1
        sb      r2,1(r4) ;WDCR(0xffff091) ← 0xb1
        nop

;-------------------------------------------------------------------------
;[Clearing tiny-area to 0] Unnecessary when no variables whose
 initial values are undefined are located in the tiny-area               f)

        addiu   r4,r0,tinyAreaSize
        beq     r4,r0,no_tiny_area
        nop
        addiu   r2,r0,TinyArea
clear_tiny_area:
        addiu   r4,r4,-1
        sb      r0,0(r2)
        bne     r4,r0,clear_tiny_area
        addiu   r2,r2,1
no_tiny_area:

;-------------------------------------------------------------------------
;[Clearing near-area to 0] Unnecessary when no variables whose
 initial values are undefined are located in the near-area               g)

        addiu   r4,r0,nearAreaSize
        beq     r4,r0,no_near_area
        nop
        lui     r2,hi(NearArea)
        addiu   r2,r2,lo(NearArea)
clear_near_area:
        addiu   r4,r4,-1
        sb      r0,0(r2)
        bne     r4,r0,clear_near_area
        addiu   r2,r2,1
no_near_area:

;-------------------------------------------------------------------------
;[Clearing far-area to 0] Unnecessary when no variables whose
 initial values are undefined are located in the far-area                h)

        lui     r4,hi(farAreaSize)
        addiu   r4,r4,lo(farAreaSize)
        beq     r4,r0,no_far_area
        nop
        lui     r2,hi(FarArea)
        addiu   r2,r2,lo(FarArea)
clear_far_area:
        addiu   r4,r4,-1
        sb      r0,0(r2)
        bne     r4,r0,clear_far_area
        addiu   r2,r2,1
no_far_area:

;-------------------------------------------------------------------------
;[Transferring variables whose initial values are defined to
 specified RAM area (tiny-area)]                                         i)
;Unnecessary when no variables whose initial values are defined are
 located in the tiny-area

        addiu   r4,r0,tinyDataSize         ;Number of bytes to transfer
        beq     r4,r0,no_tiny
        nop
        addiu   r2,r0,TinyData             ;Start address of transfer destination
```

```
        addiu    r3,r0,tinyDataOrg          ;Start address of transfer source
move_t_data:
        addiu    r4,r4,-1
        lb       r5,0(r3)
        addiu    r3,r3,1
        sb       r5,0(r2)
        bne      r4,r0,move_t_data
        addiu    r2,r2,1
no_tiny:

;---------------------------------------------------------------------
;[Transferring variables whose initial values are defined to
 specified RAM area (near-area)]                                      j)
;Unnecessary when no variables whose initial values are defined are
 located in the near-area

        addiu    r4,r0,nearDataSize         ;Number of bytes to transfer
        beq      r4,r0,no_near
        nop
        lui      r2,hi(NearData)            ;Start address of transfer destination
        addiu    r2,r2,lo(NearData)
        lui      r3,hi(nearDataOrg)         ;Start address of transfer source
        addiu    r3,r3,lo(nearDataOrg)
move_n_data:
        addiu    r4,r4,-1
        lb       r5,0(r3)
        addiu    r3,r3,1
        sb       r5,0(r2)
        bne      r4,r0,move_n_data
        addiu    r2,r2,1
no_near:

;---------------------------------------------------------------------
;[Transferring variables whose initial values are defined to
 specified RAM area (far-area)]                                       k)
;Unnecessary when no variables whose initial values are defined are
 located in the far-area

        lui      r4,hi(farDataSize)         ;Number of bytes to transfer
        addiu    r4,r4,lo(farDataSize)
        beq      r4,r0,no_far
        nop
        lui      r2,hi(FarData)             ;Start address of transfer destination
        addiu    r2,r2,lo(FarData)
        lui      r3,hi(farDataOrg)          ;Start address of transfer source
        addiu    r3,r3,lo(farDataOrg)
move_f_data:
        addiu    r4,r4,-1
        lb       r5,0(r3)
        addiu    r3,r3,1
        sb       r5,0(r2)
        bne      r4,r0,move_f_data
        addiu    r2,r2,1
no_far:

;---------------------------------------------------------------------
;[Calling main function]

        jalx.    main
        nop
self_jump:
        j        self_jump
        nop
```

The letters a) to k) below correspond to those shown on the extreme right in the preceding program listing.

a) *_ _startup*
The start-up routine included in the program sample is written to start from symbol *_ _startup* when reset. When you use Toshiba debuggers, add a description of "_ _startup" to Startup Label specification when creating a profile.

b) Register initialization

- Initializing the Stack Pointer (sp)
  In the C language local variables for functions are normally accessed using sp Register Relative Mode. Thus the register sp must be initialized before the function main is called. Initialize sp in the start-up routine. A typical line of code for initializing sp is shown below.

```
li      sp,StackBase
```

  StackBase is defined at the beginning of the start-up routine as explained in the definition section.

- Initializing the Global Pointer

```
li      gp, _ _gp
```

  _ _gp is the label symbol determined when linking, a special symbol which does not require an *extern* declaration. The value of *_ _gp* is the start address of the *medium-area* plus 8000H. The start address of the *medium-area* is either the start address of *data.m* or the start address of *romdata.m*, whichever is the lower.

  Alternatively, absolute addresses, which do not require the use of *_ _gp*, can be used. In this case when setting absolute addresses, make sure that the entire *medium-area*, as defined in the link command file, is within the address range accessible in gp Relative Mode.

c) Initializing the system control coprocessor (CP0)
The CP0 controls exception handling, system configuration and memory management. The CP0 Status Register (r12) is used for determining interrupt control and the operation mode; thus it must be properly initialized when interrupts are used.

The programmer can use the CP0 Status Register to define the exception vectors, interrupt mask levels, operation mode and interrupt enable state. The start-up routine shown is merely an example — it can be rewritten as necessary to suit the application at hand. Note, however, that for a TX1940 microprocessor, the BEV bit (bit 23 counting up from the LSB) must always be set to 1. For details, please refer to the databook entitled 32-Bit TX System RISC TX19 Family TX1940.

d) Clearing the register IVR to 0
The register IVR holds the vector for the interrupt source. The program branches to the interrupt routine pointed to by the vector held in the register IVR.

e) Halting the Watchdog Timer
The TX1940's Watchdog Timer is used for detecting CPU runaway. If the CPU starts operating erratically for some reason, e.g. because of noise, the Watchdog Timer detects the erroneous condition and sends a signal to the processor instructing it to return to its normal state.

Resetting the device enables the Watchdog Timer. In this example, however, it is disabled. To enable it, comment out this code segment. For details, please refer to the databook entitled 32-Bit TX System RISC TX19 Family TX1940.

f), g) and h) Clearing variables whose initial values are undefined to 0

The following code sample demonstrates a technique which can be used to clear the *t_area*, *n_area* and *f_area* sections to 0. (These are areas for variables whose initial values are undefined.) In the example below the *f_area* section is cleared to 0.

```
        lui     r4,hi(farAreaSize)     ;Size of f_area
        addiu   r4,r4,lo(farAreaSize)
        beq     r4,r0,no_far_area
        nop
        lui     r2,hi(FarArea)         ;Start address of f_area
        addiu   r2,r2,lo(FarArea)
clear_far_area:
        addiu   r4,r4,-1
        sb      r0,0(r2)
        bne     r4,r0,clear_far_area
        addiu   r2,r2,1
no_far_area:
```

The *farAreaSize* and *FarArea* symbols are defined in the link command file. They are replaced by 32-bit unsigned integers at link time. *farAreaSize* represents the size of the *f_area* area and *FarArea* represents the start address of the *f_area* area. This routine clears an area of size *farAreaSize* starting at the address *FarArea* to 0.

i), j) and k) Transferring variables whose initial values are defined from ROM to RAM

Two kinds of variable are stored in the *t_data*, *n_data* and *f_data* sections: variables whose initial values are defined and which are defined outside of any function, and variables defined within functions as *static* variables. The initial values for these kinds of variable are stored in ROM; however, since the values of these variables will change when instructions are executed, they must be copied to RAM.

In the example below the initial values are transferred from the *f_data* section of ROM to RAM.

```
        lui     r4,hi(farDataSize)     ;Number of bytes to transfer
        addiu   r4,r4,lo(farDataSize)
        beq     r4,r0,no_far
        nop
        lui     r2,hi(FarData)         ;Start address of transfer destination
        addiu   r2,r2,lo(FarData)
        lui     r3,hi(farDataOrg)      ;Start address of transfer source
        addiu   r3,r3,lo(farDataOrg)
move_f_data:
        addiu   r4,r4,-1
        lb      r5,0(r3)
        addiu   r3,r3,1
        sb      r5,0(r2)
        bne     r4,r0,move_f_data
        addiu   r2,r2,1
no_far:
```

They are replaced by 32-bit unsigned integers at link time. FarData represents the start address of the f_data area (the area in which the body of the variable with initial value is stored), and farDataOrg represents the start address of the ROM area in which the variable with initial value is stored. Namely, farDataOrg indicates the ROM address from which to transfer the data and FarData indicates the RAM address to which to transfer the data. They are replaced by 32-bit unsigned integers at link time. FarData represents the start address of the f_data area (the area in which the body of the variable with initial value is stored), and farDataOrg represents the start address of the ROM area in which the variable with initial value is stored. Namely, farDataOrg indicates the ROM address from which to transfer the data and FarData indicates the RAM address to which to transfer the data.

Figure 1.1 shows how the initial values are transferred from ROM to RAM by the sample program shown in Section 3.1.2.1. However, this differs from the sample application programs (see Section 3.1.2.2).

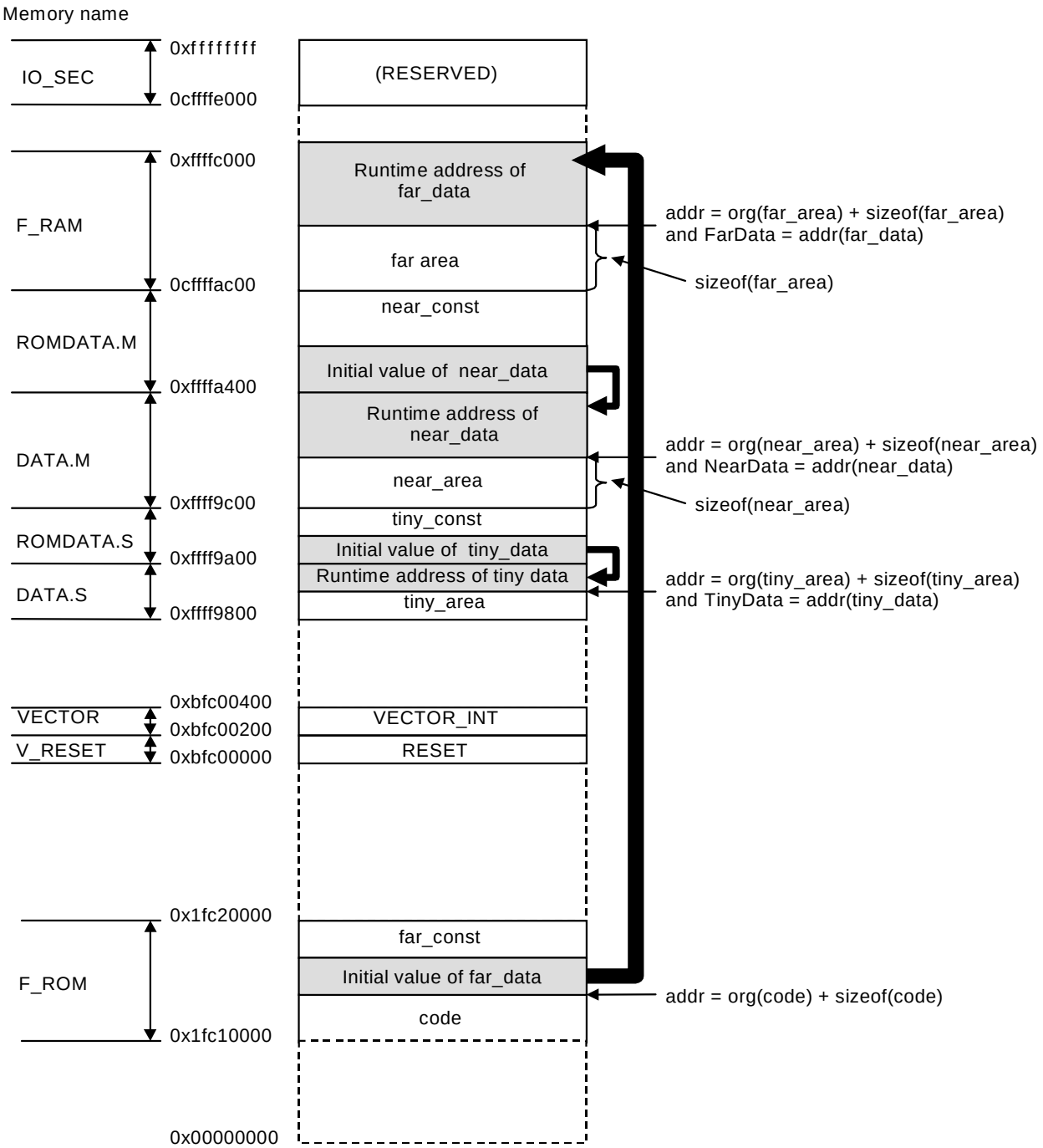


Figure 3.1.1 Memory map showing transfer of initial values from ROM to RAM

- Exception handler definition

When an exception occurs, the program will branch to the appropriate exception vector address once hardware exception processing (e.g. setting of the EPC) has been completed. The exception vector address depends on the type of exception which has occurred.

Type of Exception	Vector Address (virtual address)
Reset exception, non-maskable interrupt	0xbfc00000
Debug exception	0xbfc00200
Swi0 (maskable software interrupt)	0xbfc00210
Swi1 (maskable software interrupt)	0xbfc00220
Swi2 (maskable software interrupt)	0xbfc00230
Swi3 (maskable software interrupt)	0xbfc00240
Maskable hardware interrupt	0xbfc00260
Other exceptions	0xbfc00180

The exception handler definition section in the start-up routine is shown below.

```
;Exception handler definition
;Reset exception, non-maskable interrupt                                    a)
VECTOR_RESET section  code large abs=0xbfc00000
vReset:                              ;Reset: 0xbfc000000
       mfc0    r26,r12               ;Status(r12),NMI
       srl     r26,r26,20
       andi    r27,r26,1
       bne     r27,r0,J_Dummy        ;If NMI, go to dummy function.
       nop
       lui     r26,hi(__startup)     ;Otherwise, go to __startup.
       addiu   r26,r26,lo(__startup)
       jr      r26
       nop
J_Dummy:
       lui     r26,hi(_Int_dummy)
       addiu   r26,r26,lo(_Int_dummy)
       jr      r26
;Debug exception                                                            b)
VECTOR_DBG  section  code large abs=0xbfc00200
vDBG:                                ;debug:0xbfc0 0200
        mfc0    r26,r16              ;r16=Debug register
        andi    r26,r26,0x1          ;0bit=DSS(single step exception)
        bne     r26,r0,J_Dummy2      ;r26=1 ->single step exception
        nop
        lui     r26,hi(_Int_dummy)   ;debug break point exception
        addiu   r26,r26,lo(_Int_dummy)
        jr      r26
        nop
J_Dummy2:                            ;single step exception
        lui     r26,hi(_Int_dummy)
        addiu   r26,r26,lo(_Int_dummy)
        jr      r26
        nop
;Software interrupt                                                         c)
VECTOR_SWI0  section  code large abs=0xbfc00210
vSwi0:                               ;Swi0:0xbfc0 0210
        lui     r26,hi(_Int_dummy)
        addiu   r26,r26,lo(_Int_dummy)
        jr      r26
        nop
VECTOR_SWI1  section  code large abs=0xbfc00220
vSwi1:                               ;Swi1:0xbfc0 0220
        lui     r26,hi(_Int_dummy)
        addiu   r26,r26,lo(_Int_dummy)
        jr      r26
        nop
```

```
VECTOR_SWI2  section  code large abs=0xbfc00230
vSwi2:                                  ;Swi2:0xbfc0 0230
        lui     r26,hi(_Int_dummy)
        addiu   r26,r26,lo(_Int_dummy)
        jr      r26
        nop
VECTOR_SWI3  section  code large abs=0xbfc00240
vSwi3:                                  ;Swi3:0xbfc0 0240
        lui     r26,hi(_Int_dummy)
        addiu   r26,r26,lo(_Int_dummy)
        jr      r26
        nop

;Other exceptions
;Identify cause of exception from Cause Register.
;ExcCode bits (bits 2~6) of Cause Register (CP0:r13)
VECTOR_EXC section code large abs=0xbfc00180
    public __ExceptVector
__ExceptVector:
        mfc0    r26,r13
        andi    r26,r26,0x7c            ;Extract ExCode bits.
        addiu   r26,r26,-0x10
        lui     r27,hi(__ExceptTable)
        addu    r26,r26,r27
        addiu   r26,r26,lo(__ExceptTable)
        lw      r26,0(r26)
        jr      r26
        nop
;
;Exception table (sample)
;
;Write other exception processing in this table and correlate them with specific causes of
 interrupt.
EXCEPT_TAB section data large
    public __ExceptTable
__ExceptTable:
        dw    _Int_dummy                ;4 --- Address Error exception, load (AdEL)
        dw    _Int_dummy                ;5 --- Address Error exception, store (AdES)
        dw    _Int_dummy                ;6 --- Bus Error exception, instruction (IBE)
        dw    _Int_dummy                ;7 --- Bus Error exception, data (DBE)
        dw    _Int_dummy                ;8 --- System Call exception (Sys)
        dw    _Int_dummy                ;9 --- Breakpoint exception (Bp)
        dw    _Int_dummy                ;10--- Reserved Instruction exception (R1)
        dw    _Int_dummy                ;11--- Coprocessor Unusable exception (CpU)
        dw    _Int_dummy                ;12--- Overflow exception (Ov)

;Hardware interrupt                                                          e)
;Hardware interrupts are explained in Section 3.1.3, Interrupt Processing.
```

The letters below correspond to those shown on the extreme right in the preceding program listing.

a) Reset exception, non-maskable interrupt
On both a reset exception and a non-maskable interrupt, the program branches to BFC00000H. Hence, the cause of the exception must be identified from the setting of the NMI bit in the Status Register (r12). If the NMI bit = 1, it signifies that a non-maskable interrupt has occurred, in which case control is transferred to a dummy function. If the NMI bit = 0, it signifies that a reset exception has occurred, in which case the program branches to the *_startup* symbol (see the data initialization section).

b) Debug exception
There are two types of Debug exception: the Single-Step exception and the Debug Breakpoint exception. In the sample program the code for this section includes a branch which depends on the exception type, allowing the two types to be processed in different ways. If necessary, write separate handler routines for each type.

c) Software interrupt
In the sample program control is transferred to a dummy function for each of the interrupts Swi0 to Swi3. If necessary, write separate handler routines for each type.

d) Other exceptions
When an exception such as an Address Error exception or Bus Error exception occurs, the sample program transfers control to the appropriate routine as determined by the corresponding look-up table entry. Rewrite the table as necessary.

e) Hardware interrupt
Hardware interrupts are explained in Section 3.1.3, Interrupt Processing.
In the sample program shown later, control is transferred to a routine corresponding to the cause of the hardware interrupt. The cause is signified by the value of IVR.

3.1.1.2 Start-up routine (for sample applications)

This section includes a listing of the start-up routine which was used when compiling the sample applications. However, because the jump table for interrupt handling is different for each sample application, for details of a particular application's start-up routine, please refer to the specific individual section in this chapter, Description of the Software.

```
;************************************
;*          Application Note          *
;*                                    *
;*         (Start Up Routine)         *
;*                                    *
;*          MCU: TX1940FDBF           *
;*            1999/10/15              *
;*                                    *
;************************************

;=====================================================================
;[Address definition] Rewrite this section as necessary before use.
;=====================================================================
StackBase      equ    0xffffbc00       ;Base address of stack
HeapTop        equ    0xffffbc00       ;Base address of heap area
HeapSize       equ    0x400            ;Size of heap area
;### Interrupt Vector Register
IVR            equ    0xffffe040       ;Interrupt Vector Register

;=====================================================================
;Start-up program
;=====================================================================
;---------------------------------------------------------------------
;[Section] Section declarations are required even for sections for
           which no assembler variables are defined.
;---------------------------------------------------------------------
t_area  section  data    small   ;Section containing variables whose
                                  initial values are undefined
t_data  section  data    small   ;Section containing variables whose
                                  initial values are defined
t_const section  romdata small   ;Section containing constants
n_const section  romdata medium  ;Section containing constants
f_area  section  data    large   ;Section containing variables whose
                                  initial values are undefined
f_data  section  data    large   ;Section containing variables whose
                                  initial values are defined
f_const section  romdata large   ;Section containing constants
;---------------------------------------------------------------------
;[Global variable initialization area] As defined in the link command file
;---------------------------------------------------------------------
extern large    TinyData,FarData
extern large    TinyArea,FarArea
extern large    tinyAreaSize,farAreaSize
extern large    tinyDataSize,farDataSize
extern large    tinyAreaOrg,farAreaOrg
extern large    tinyDataOrg,farDataOrg
;### Interrupt functions used
extern large    _mnmi
extern large    _mint0
extern large    _minttb21
extern large    _mintta1
extern large    _mintta2
extern large    _mintad
extern large    _mintrx1
extern large    _minttx1
extern large    _mints2
extern large    _mintrtc
```

```
;---------------------------------------------------------------------
;[External reference declaration for main() function] (16bitISA)
;---------------------------------------------------------------------
extern medium   .main        ;If the main() function is a 32-bit ISA function,
                             ;replace ".main" with "_main"
;---------------------------------------------------------------------
;Start-up program proper
;---------------------------------------------------------------------
ResetStart section code isa32
        public     __startup
__startup:
;---------------------------------------------------------------------
;[Register initialization]
;This section must always be included.

        lui      sp,hi(StackBase)         ;Set Stack Pointer.
        addiu    sp,sp,lo(StackBase)
        lui      gp,hi(__gp)              ;Set Global Pointer.
        addiu    gp,gp,lo(__gp)

;Setting Status Register
;High-order 16 bits of Status Register (CP0:r12)
;           CU  0  RE 0  BEV TS NmI 0 PMask
;After reset  xxxx 00 x  00  1  0   0  0  xxx
;Low-order 16 bits of Status Register (CP0:r12)
;         CMask 0 SwiMask 0 KUo IEo KUp IEp KUc IEc
;After reset  xxx  0  xxxx   0  x   x   x   x   x   x
;0x1040_0000 means "enable CP0" & "BEV=1" & "disable interrupt"
        lui      r2,0x1040
        addiu    r2,r2,0x0000
        mtc0     r2,r12

        lui      r2,hi(IVR)
        addiu    r2,r2,lo(IVR)
        addu     r3,r0,r0
        sw       r3,0(r2)

;---------------------------------------------------------------------
;[Disabling Watchdog Timer]
;
        lui      r4,0xffff
        ori      r4,r4,0xf090
        sb       r0,0(r4)      ;WDMOD<WDTE>(0xfffff090) ← 0x00
        addiu    r2,r0,0xb1
        sb       r2,1(r4)      ;WDCR(0xffff091) ← 0xb1
        nop

;---------------------------------------------------------------------
;[Clearing Tiny Area to 0]
;Unnecessary if there is no tiny-area or if the tiny-area does not include
 any variables whose initial values are undefined

       addiu     r4,r0,tinyAreaSize
       beq       r4,r0,no_tiny_area
       addiu     r2,r0,TinyArea
clear_tiny_area:
       addiu     r4,r4,-1
       sb        r0,0(r2)
       bne       r4,r0,clear_tiny_area
       addiu     r2,r2,1
no_tiny_area:

;---------------------------------------------------------------------
;[Clearing Far Area to 0]
;Unnecessary if the far-area does not include any variables whose initial values are undefined
```

```
        lui     r4,hi(farAreaSize)
        addiu   r4,r4,lo(farAreaSize)
        beq     r4,r0,no_far_area
        lui     r2,hi(FarArea)
        addiu   r2,r2,lo(FarArea)
clear_far_area:
        addiu   r4,r4,-1
        sb      r0,0(r2)
        bne     r4,r0,clear_far_area
        addiu   r2,r2,1
no_far_area:

;---------------------------------------------------------------------
;[Transferring variables whose initial values are defined to specified
 RAM area (tiny-area)]
;Unnecessary if there is no tiny-area or if the tiny-area does not include
 any variables whose initial values are undefined

        addiu   r4,r0,tinyDataSize      ;Number of bytes to transfer
        beq     r4,r0,no_tiny
        addiu   r2,r0,TinyData          ;Start address of transfer destination
        addiu   r3,r0,tinyDataOrg       ;Start address of transfer source
move_t_data:
        addiu   r4,r4,-1
        lb      r5,0(r3)
        addiu   r3,r3,1
        sb      r5,0(r2)
        bne     r4,r0,move_t_data
        addiu   r2,r2,1
no_tiny:

;---------------------------------------------------------------------
;[Transferring variables whose initial values are defined to specified
 RAM area (far-area)]
;Unnecessary if the far-area does not include any variables whose initial
 values are undefined

        lui     r4,hi(farDataSize)      ;Number of bytes to transfer
        addiu   r4,r4,lo(farDataSize)
        beq     r4,r0,no_far
        lui     r2,hi(FarData)          ;Start address of transfer destination
        addiu   r2,r2,lo(FarData)
        lui     r3,hi(farDataOrg)       ;Start address of transfer source
        addiu   r3,r3,lo(farDataOrg)
move_f_data:
        addiu   r4,r4,-1
        lb      r5,0(r3)
        addiu   r3,r3,1
        sb      r5,0(r2)
        bne     r4,r0,move_f_data
        addiu   r2,r2,1
no_far:

;---------------------------------------------------------------------
;[Calling main function]
        jalx    .main
        nop
self_jump:
        j   self_jump
        nop

;=====================================================================
;[ANSI standard library functions] Change this section as necessary.
;=====================================================================
;---------------------------------------------------------------------
```

```
;[errno variable] This section must always be included when standard
 library functions are used.
;--------------------------------------------------------------------------
t_area        section     data    small
        public    _errno
_errno        dsw    1
;--------------------------------------------------------------------------
;[Heap area definition] This section must always be included
 when malloc, calloc or realloc are used.
;--------------------------------------------------------------------------
t_data        section     data    small
        public    _SBRK_break        ;Start pointer for heap area
        public    _SBRK_size         ;Size of heap area
        public    __allocb           ;Control pointer for heap area
_SBRK_break    dw    HeapTop
_SBRK_size     dw    HeapSize
__allocb       dw    0

HeapArea    section data    abs=HeapTop    align=1,1
heap        dsb    HeapSize           ;Heap area

;--------------------------------------------------------------------------
;[abort function] Include this section if abort is called from a
 32-bit ISA function.
;--------------------------------------------------------------------------
f_code        section    code    isa32
        public    _abort
_abort:        j    _abort
        nop
;--------------------------------------------------------------------------
;[abort function] Include this section if abort is called from a
 16-bit ISA function.
;--------------------------------------------------------------------------
n_code        section    code    isa16
        public    .abort
.abort:        b    .abort
;--------------------------------------------------------------------------
;[exit function] Include this section if exit is called from a
 32-bit ISA function.
;--------------------------------------------------------------------------
f_code        section    code    isa32
        public    _exit
_exit:        j    _exit
        nop
;--------------------------------------------------------------------------
;[exit function] Include this section if exit is called from a
 16-bit ISA function.
;--------------------------------------------------------------------------
n_code        section    code    isa16
        public    .exit
.exit:        b    .exit

;==========================================================================
;Exception vector address definition for TX1940
;==========================================================================
VECTOR_RESET section  code large abs=0xbfc00000
;Reset exception, non-maskable interrupt
vReset:                                ;Reset: 0xbfc000000
        mfc0       r26,r12             ;Status(r12),NMI
        srl        r26,r26,20
        andi       r27,r26,1
        bne        r27,r0,J_Dummy      ;If NMI, go to dummy function.
        nop
        lui        r26,hi(__startup)   ;Otherwise, go to startup.
        addiu      r26,r26,lo(__startup)
        jr         r26
```

```
        nop
J_Dummy:
        lui         r26,hi(_mnmi)
        addiu       r26,r26,lo(_mnmi)
        jr          r26
        nop
;Debug exception
VECTOR_DBG  section  code large abs=0xbfc00200
vDBG:                                   ;debug:0xbfc0 0200
        lui         r26,hi(_DBG_except)
        addiu       r26,r26,lo(_DBG_except)
        jr          r26
        nop
;Software interrupt 0
VECTOR_SWI0  section  code large abs=0xbfc00210
vSwi0:                                  ;Swi0:0xbfc0 0210
        lui         r26,hi(_Int_dummy)
        addiu       r26,r26,lo(_Int_dummy)
        jr          r26
        nop
;Software interrupt 1
VECTOR_SWI1  section  code large abs=0xbfc00220
vSwi1:                                  ;Swi1:0xbfc0 0220
        lui         r26,hi(_Int_dummy)
        addiu       r26,r26,lo(_Int_dummy)
        jr          r26
        nop
;Software interrupt 2
VECTOR_SWI2  section  code large abs=0xbfc00230
vSwi2:                                  ;Swi2:0xbfc0 0230
        lui         r26,hi(_Int_dummy)
        addiu       r26,r26,lo(_Int_dummy)
        jr          r26
        nop
;Software interrupt 3
VECTOR_SWI3  section  code large abs=0xbfc00240
vSwi3:                                 ;Swi3:0xbfc0 0240
        lui         r26,hi(_Int_dummy)
        addiu       r26,r26,lo(_Int_dummy)
        jr          r26
        nop
;---------------------------------------------------------------------
;Other exceptions
;Identify cause of interrupt from Cause Register.
;ExcCode bits (bits 2~6) of Cause Register (CP0:r13)
;---------------------------------------------------------------------
VECTOR_EXC section code large abs=0xbfc00180
        public __ExceptVector
__ExceptVector:
        mfc0        r26,r13
        andi        r26,r26,0x7c        ;Extract ExCode bits.
        addiu       r26,r26,-0x10
        lui         r27,hi(__ExceptTable)
        addu        r26,r26,r27
        addiu       r26,r26,lo(__ExceptTable)
        lw          r26,0(r26)
        jr          r26
        nop
;---------------------------------------------------------------------
;Exception table (example)
;---------------------------------------------------------------------
;Write other exception processing in this table and correlate them
 with specific causes of interrupt.
EXCEPT_TAB section data large
        public __ExceptTable
__ExceptTable:
```

```
        dw    _Int_dummy    ;4 --- Address Error exception, load (AdEL)
        dw    _Int_dummy    ;5 --- Address Error exception, store (AdES)
        dw    _Int_dummy    ;6 --- Bus Error exception, instruction (IBE)
        dw    _Int_dummy    ;7 --- Bus Error exception, data (DBE)
        dw    _Int_dummy    ;8 --- System Call exception (Sys)
        dw    _Int_dummy    ;9 --- Breakpoint exception (Bp)
        dw    _Int_dummy    ;10--- Reserved Instruction exception (R1)
        dw    _Int_dummy    ;11--- Coprocessor Unusable exception (CpU)
        dw    _Int_dummy    ;12--- Overflow exception (Ov)
;------------------------------------------------------------------------
;Hardware interrupt
;------------------------------------------------------------------------
VECTOR_INT section code large abs=0xbfc00260
    public __InterruptVector
__InterruptVector:
        lui       r26,hi(IVR)
        addiu     r26,r26,lo(IVR)
        lw        r26,0(r26)
        andi      r26,r26,0x03f0
        srl       r26,r26,2
        lui       r27,hi(__VectorTable)
        addu      r26,r26,r27
        addiu     r26,r26,lo(__VectorTable)
        lw        r26,0(r26)
        jr        r26
        nop
;------------------------------------------------------------------------
;Interrupt vector table (example)
;------------------------------------------------------------------------
;Write interrupt functions in this table and correlate them with specific
 causes of interrupt.
VECTOR_TAB section data large
    public __VectorTable
__VectorTable:
        dw    _Int_dummy    ;0 --- software set
        dw    _mint0        ;1 --- INT[0]
        dw    _Int_dummy    ;2 --- INT[1]
        dw    _Int_dummy    ;3 --- INT[2]
        dw    _Int_dummy    ;4 --- INT[3]
        dw    _Int_dummy    ;5 --- INT[4]
        dw    _Int_dummy    ;6 --- *
        dw    _Int_dummy    ;7 --- *
        dw    _Int_dummy    ;8 --- *
        dw    _Int_dummy    ;9 --- *
        dw    _Int_dummy    ;10--- INT[5]
        dw    _Int_dummy    ;11--- INT[6]
        dw    _Int_dummy    ;12--- INT[7]
        dw    _Int_dummy    ;13--- INT[8]
        dw    _Int_dummy    ;14--- INT[9]
        dw    _Int_dummy    ;15--- INT[A]
        dw    _Int_dummy    ;16--- *
        dw    _Int_dummy    ;17--- *
        dw    _Int_dummy    ;18--- *
        dw    _Int_dummy    ;19--- *
        dw    _Int_dummy    ;20--- INTTA0
        dw    _mintta1      ;21--- INTTA1
        dw    _mintta2      ;22--- INTTA2
        dw    _Int_dummy    ;23--- INTTA3
        dw    _Int_dummy    ;24--- *
        dw    _Int_dummy    ;25--- *
        dw    _Int_dummy    ;26--- *
        dw    _Int_dummy    ;27--- *
        dw    _Int_dummy    ;28--- INTTB00
        dw    _Int_dummy    ;29--- INTTB01
        dw    _Int_dummy    ;30--- INTTB10
        dw    _Int_dummy    ;31--- INTTB11
```

```
        dw    _Int_dummy    ;32--- INTTB20
        dw    _minttb21     ;33--- INTTB21
        dw    _Int_dummy    ;34--- INTTB30
        dw    _Int_dummy     ;35--- INTTB31
        dw    _Int_dummy    ;36--- *
        dw    _Int_dummy    ;37--- *
        dw    _Int_dummy    ;38--- *
        dw    _Int_dummy    ;39--- *
        dw    _Int_dummy    ;40--- INTTBOF0
        dw    _Int_dummy    ;41--- INTTBOF1
        dw    _Int_dummy    ;42--- INTTBOF2
        dw    _Int_dummy    ;43--- INTTBOF3
        dw    _Int_dummy    ;44--- *
        dw    _Int_dummy    ;45--- *
        dw    _Int_dummy    ;46--- *
        dw    _Int_dummy    ;47--- *
        dw    _Int_dummy    ;48--- INTRX0
        dw    _Int_dummy    ;49--- INTTX0
        dw    _mintrx1      ;50--- INTRX1
        dw    _minttx1      ;51--- INTTX1
        dw    _mints2       ;52--- INTS2
        dw    _Int_dummy    ;53--- *
        dw    _Int_dummy    ;54--- INTRX3
        dw    _Int_dummy    ;55--- INTTX3
        dw    _Int_dummy    ;56--- INTRX4
        dw    _Int_dummy    ;57--- INTTX4
        dw    _mintrtc      ;58--- INTRTC
        dw    _mintad       ;59--- INTAD
        dw    _Int_dummy    ;60--- INTDMA0
        dw    _Int_dummy    ;61--- INTDMA1
        dw    _Int_dummy    ;62--- INTDMA2
        dw    _Int_dummy    ;63--- INTDMA3
;---------------------------------------------------------------------
;Dummy function for unused interrupts
;---------------------------------------------------------------------
st_code  section code large
    public _Int_dummy
_Int_dummy:
        MFC0    r2,r14
        j       self_jump
        rfe
;---------------------------------------------------------------------
;Debug Exception interrupt
;---------------------------------------------------------------------
st_code  section code large
   public _DBG_except
_DBG_except:
        mfc0    r2,r16                ;r16=Debug register
        andi    r2,r2,0x1             ;0bit=DSS(single step exception)
        bne     r2,r0,J_Dummy2        ;r2=1 ->single step exception
        nop
        lui     r2,hi(_Int_dummy)     ;debug break point exception
        addiu   r2,r2,lo(_Int_dummy)
        jr      r2
        nop
J_Dummy2:                             ;single step exception
        lui     r2,hi(_Int_dummy)
        addiu   r2,r2,lo(_Int_dummy)
        jr      r2
        nop
        end
```

3.1.2 Compilation

The compiler used for the applications presented in these application notes is the C compiler for the TX19 Family of Toshiba microprocessors. The command used at compile time is as follows:

```
cc9i -Nr16 -Rle -ZA tiny -ZD tiny -ZC tiny -Zx tiny –O3
<start-up filename> <link command filename> <source filename>
```

The compile options used for the programs presented here are described below.

Compile Option	Description
-Nr16	Compiles in 16-Bit ISA Mode.
-Rle	Compiles using little-endian byte alignment. Selecting this option is obligatory with the TX1940 Family.
-ZA	Specifies area (*tiny*, *near* or *far*) for storing external variables whose initial values are undefined.
-ZD	Specifies area (*tiny*, *near* or *far*) for storing external variables whose initial values are defined.
-ZC	Specifies area (*tiny*, *near* or *far*) for storing *const* objects.
-Zx	Specifies area (*tiny*, *near* or *far*) for storing external variables declared as *extern*.

The table below shows other frequently used options.

Compile Option	Description
-Nr32	Compiles in 32-Bit ISA Mode.
-O[level]	Optimizes program. An optimization level of 1, 2 or 3 can be specified.
-g	Outputs information required by debugger. This makes source-level debugging possible.
-gn	Inserts nop instruction in delay slot for branch or jump instruction. This is required in order to be able to step through instructions in the debugger.
-f	Reads from specified file.
-I	Specifies search path for include file.
-o	Specifies output filename (abs file).
-zc	Suppresses output of Warning-501, Warning-505, Warning-517 and Warning-518 by assembler.
-zw	Suppresses output of Warning-520 by assembler.

3.1.2.1 Description of the link command file

This section explains the link command file in detail using an example. When a program is linked, principally the link command file carries out the following operations:

(1) Specifies the target device’s memory configuration.

(2) Specifies the sections to be linked and the order in which they are to be linked.

(3) Specifies the start addresses and sizes of the various sections.

(4) Defines global symbols.

An example of a link command file (named lnc9i.lcf) is shown below.

```
/*******************************************************/
/*Link Command File                                    */
/*Sample for TX19 (Direct Segment Mapping)             */
/*******************************************************/

/* MEMORY */
memory {
       DATA.S         :         org=0xffff9800,    len=0x00200
        ROMDATA.S     :         org=0xffff9a00,    len=0x00200
        DATA.M        :         org=0xffff9c00,    len=0x00800
        ROMDATA.M     :         org=0xffffa400,    len=0x00800
        F_RAM         :         org=0xffffac00,    len=0x01400
        F_ROM         :         org=0x1fc10000,    len=0x10000
        F_ROM2        :         org=0x1fc20000,    len=0x10000
        VECTOR        :         org=0xbfc00000,    len=0x00400
```

```
        IO_SEC        :         org=0xffffe000,    len=0x02000
}

/* SECTION */
sections
{
        tiny_area      org=0xffff9800 : {*(t_area)} > DATA.S
        tiny_data      org=0xffff9a00
                       addr=org(tiny_area)+sizeof(tiny_area)
                       : {*(t_data)}
        near_area      org=0xffff9c00 : {*(n_area)} > DATA.M
        near_data      org=0xffffa400
                       addr=org(near_area)+sizeof(near_area)
                       : {*(n_data)}
        far_area       org=0xffffac00 : {*(f_area)} > F_RAM
        far_code       org=0x1fc10000 : {*(ResetStart) *(f_code)} > F_ROM
        near_code      org=org(far_code)+sizeof(far_code): {*(n_code)}
        far_data       org=org(near_code)+sizeof(near_code)
                       addr=org(far_area)+sizeof(far_area)
                       : {*(f_data)}
        tiny_const     : {*(t_const)} > ROMDATA.S
        near_const     : {*(n_const)} > ROMDATA.M
        far_const      : {*(f_const)} > F_ROM

        TinyData = addr(tiny_data);
        NearData = addr(near_data);
        FarData  = addr(far_data);
        TinyArea = addr(tiny_area);
        NearArea = addr(near_area);
        FarArea  = addr(far_area);
        tinyAreaSize = sizeof(tiny_area);
        nearAreaSize = sizeof(near_area);
        farAreaSize  = sizeof(far_area);
        tinyDataSize = sizeof(tiny_data);
        nearDataSize = sizeof(near_data);
        farDataSize  = sizeof(far_data);
        tinyAreaOrg = org(tiny_area);
        nearAreaOrg = org(near_area);
        farAreaOrg  = org(far_area);
        tinyDataOrg = org(tiny_data);
        nearDataOrg = org(near_data);
        farDataOrg  = org(far_data);
}
```

- *memory* definition section

```
/* MEMORY */
memory {
        DATA.S         :         org=0xffff9800,    len=0x00200
        ROMDATA.S      :         org=0xffff9a00,    len=0x00200
        DATA.M         :         org=0xffff9c00,    len=0x00800
        ROMDATA.M      :         org=0xffffa400,    len=0x00800
        F_RAM          :         org=0xffffac00,    len=0x01400
        F_ROM          :         org=0x1fc10000,    len=0x10000
        F_ROM2         :         org=0x1fc20000,    len=0x10000
        VECTOR         :         org=0xbfc00000,    len=0x00400
        IO_SEC         :         org=0xffffe000,    len=0x02000
}
```

The memory definition section defines the memory area to be used. When coding this section, please refer to the memory map in the databook. DATA.S, F_RAM etc. are the names of the memory areas. The numeric value following the “org=” directive indicates the start address of each memory area; the numeric value following the “len=” directive indicates its size in bytes.

For example, the name F_RAM represents the memory area FFFFAC00H~FFFFBFFFH and the name F_ROM represents the memory area 1FC10000H~1FC1FFFH.

- DATA.S, ROMDATA.S

The DATA.S and ROMDATA.S sections are located in the *small-area*. The *small-area* corresponds to the compiler's *tiny-area* and can be specified to be in either the address range 0H~7FFFH or the address range FFFF8000H~FFFFFFFFH. When specifying the addresses, be careful not to exceed these ranges.

- DATA.M, ROMDATA.M

DATA.M and ROMDATA.M correspond to the assembler's *medium-area*, the accessible range for which is specified using the gp register. Hence, the combined DATA.M and ROMDATA.M area is from the start address of DATA.M or ROMDATA.M (whichever of the two is lower) to the end address of DATA.M or ROMDATA.M (whichever is the higher)and cannot exceed 64 Kbytes. The start address for this area can be any valid DATA.M or ROMDATA.M address.

- VECTOR

Exception vector addresses are specified in the VECTOR area. If a program is to incorporate exception handling, e.g. for Reset exceptions, this area must be included. For details on exception vector addresses, please refer to Section 3.1.1.1.

- IO_SEC

IO_SEC is an internal I/O area which is common to all the microprocessors in the TX19 Family. This area must be defined if the MCU header file or references to internal module registers such as the Interrupt Vector Register (IVR) are included in the start-up routine. Note, however, that the internal register configuration is different for each MCU.

■ *sections* definition section

```
sections
{
  tiny_area          org=0xffff9800 : {*(t_area)} > DATA.S
  tiny_data          org=0xffff9a00
                     addr=org(tiny_area)+sizeof(tiny_area)
                     : {*(t_data)}
  near_area          org=0xffff9c00 : {*(n_area)} > DATA.M
  near_data          org=0xffffa400
                     addr=org(near_area)+sizeof(near_area)
                     : {*(n_data)}
  far_area           org=0xffffac00 : {*(f_area)} > F_RAM
  far_code           org=0x1fc10000  : {*(ResetStart) *(f_code)} > F_ROM
  near_code          org=org(far_code)+sizeof(far_code): {*(n_code)}
  far_data           org=org(near_code)+sizeof(near_code)
                     addr=org(far_area)+sizeof(far_area)
                     : {*(f_data)}
  tiny_const         : {*(t_const)} > ROMDATA.S
  near_const         : {*(n_const)} > ROMDATA.M
  far_const          : {*(f_const)} > F_ROM

  TinyData = addr(tiny_data);
  NearData = addr(near_data);
  FarData  = addr(far_data);
  TinyArea = addr(tiny_area);
  NearArea = addr(near_area);
  FarArea  = addr(far_area);
  tinyAreaSize = sizeof(tiny_area);
  nearAreaSize = sizeof(near_area);
```

```
    farAreaSize  = sizeof(far_area);
    tinyDataSize = sizeof(tiny_data);
    nearDataSize = sizeof(near_data);
    farDataSize  = sizeof(far_data);
    tinyAreaOrg = org(tiny_area);
    nearAreaOrg = org(near_area);
    farAreaOrg  = org(far_area);
    tinyDataOrg = org(tiny_data);
    nearDataOrg = org(near_data);
    farDataOrg  = org(far_data);
  }
```

sections { } defines where the input sections are located. The format of this definition is as follows:

```
output section name [address definition]:
{filename (input section name)} [> output memory name]
```

- "*org=*" and "*addr=*"

  "*org=*" refers to the location address (address of initial value) and "*addr=*" refers to the start address (run-time address). Conversion of initial values to run-time addresses is performed in the start-up routine. In the above example "*addr=*" does not feature in the code and *tiny_area* definitions. If "*addr=*" is omitted in this fashion, "*org=*" and "*addr=*" take on the same value.

- "*org ( )*" and "*sizeof ( )*"

  "*org*" (output section name) is an operator for calculating the start address of the specified output section; "*sizeof*" (output section name) is an operator for calculating the size of the specified output section.

If "*org=*" and "*addr=*" point to different addresses, ">output memory name" cannot be specified. If "*addr=*" is omitted (i.e. if "*org=*" and "*addr=*" point to the same address), ">output memory name" can be specified. For example, an output memory name could not be defined in lines 4~6 of the *tiny_data* definition; however an output memory name could be defined in the 3rd line of the *tiny_area* definition since "*org=*" and "*addr=*" have taken on the same address. In the following case also, the output memory name can be defined:

```
tiny_const    : {*(t_const)} > ROMDATA.S
```

In the above example the output section *tiny_const* is defined to be in the free space of the ROMDATA.S memory area.

If an asterisk (*) is used in place of the input section filename, the effect is the same as specifying all input files. Hence, *{*(t_data)}* refers to all the input sections whose name includes the string "*t_data*" and which are specified as input files on the command line. If section names in the source program are changed or sections are created in the assembly language source file, do not forget to modify the section names here so that they correspond.

For example, if the code section name *newsec* is used in a C language source file and then a code section named *asmsec* is created in an assembly language source file *asmfile.mac*, the following lines must be added to the command language file.

```
code    org=0x1fc10000
         : {*(ResetStart) *(f_code))
           *(newsec) asmfile.rel(asmsec)} > F_ROM
```

3.1.2.2 Memory map (for sample applications)

A memory map for the sample applications is shown in Figure 3.1.2.

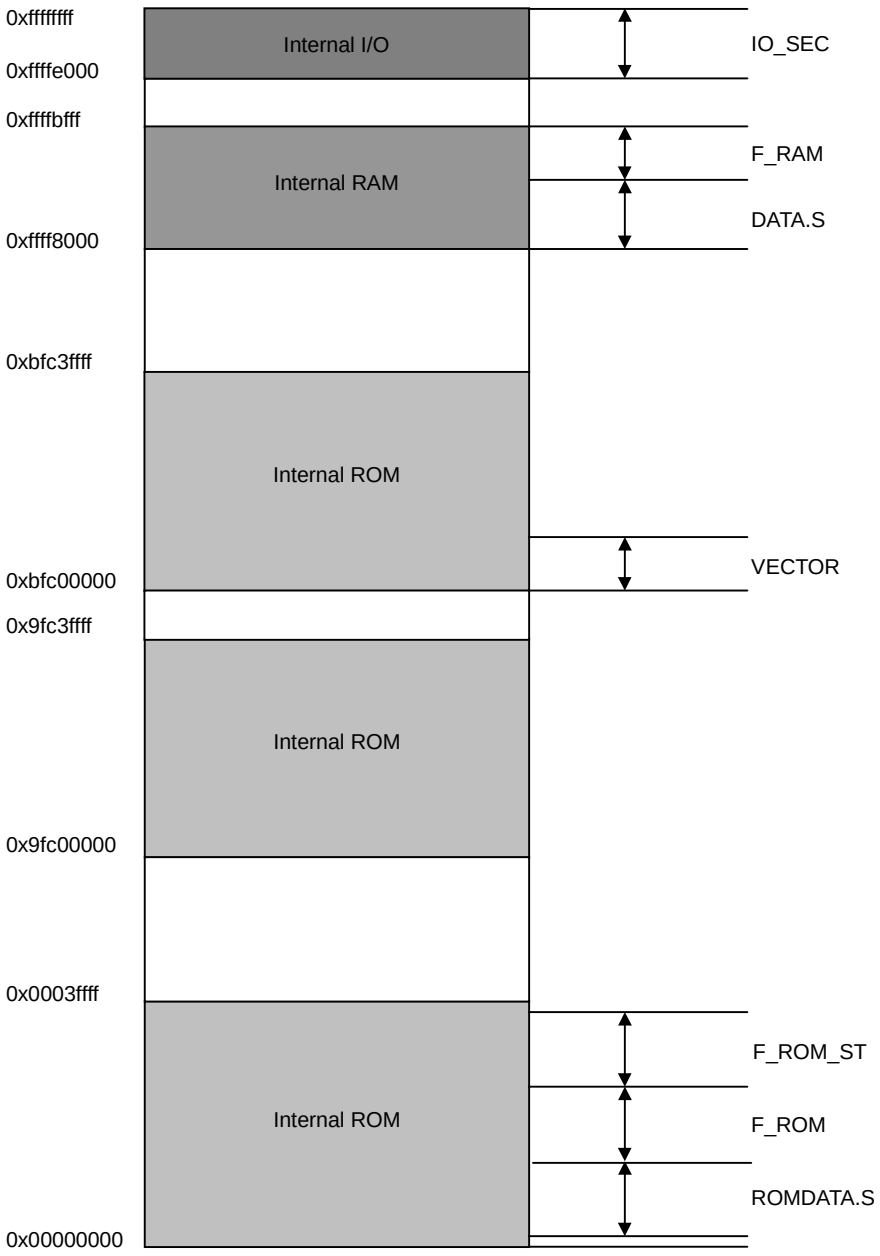


Figure 3.1.2 Memory map

3.1.2.3 Link command file (used in sample applications)

```
/*******************************************************/
/*                    Application Note                 */
/*                                                     */
/*                 <Link Command File>                 */
/*               (Direct Segment Mapping)              */
/*                                                     */
/*                    MCU: TX1940FDBF                  */
/*                      1999/10/15                     */
/*                                                     */
/*******************************************************/

/* MEMORY */
memory {
       /*-------------------------------------------------*/
       /* RAM area 0xffff8000 - 0xffffbfff (16KB)         */
       /*-------------------------------------------------*/
       DATA.S         :      org=0xffff8000,  len=0x03000      /* Locate Tiny variable */
       F_RAM          :      org=0xffffb000,  len=0x01000      /* Locate Far variable */

       /*-------------------------------------------------*/
       /* ROM area 0xbfc00000 - 0xbfc7ffff (512KB)  kseg1 */
       /*     or   0x00000000 - 0x0007ffff (512KB)  kuseg */
       /*-------------------------------------------------*/
       ROMDATA.S      :      org=0x00000400,  len=0x07c00      /* Locate Tiny constant */
       ROMDATA.M      :      org=0x00008000,  len=0x08000      /* Locate Near constant */
       F_ROM          :      org=0x00010000,  en=0x70000       /* Locate Code */
       VECTOR         :      org=0xbfc00000,  len=0x00400      /* Locate Vector */

       /*-------------------------------------------------*/
       /* SFR area 0xffffe000 - 0xffffffff (8KB)          */
       /*-------------------------------------------------*/
       IO_SEC         :      org=0xffffe000,  len=0x02000
}

/* SECTION */
sections
{
      tiny_area       org=0xffff8000
                      : {*(t_area)} > DATA.S
      tiny_data       org=0x00000400
                      addr=org(tiny_area)+sizeof(tiny_area)
                      : {*(t_data)}
      far_area        org=0xffffb000
                      : {*(f_area)} > F_RAM
      far_code        org=0x10000
                      : {*(ResetStart) *(st_code) *(f_code)} > F_ROM
      near_code       org=org(far_code)+sizeof(far_code)
                      : {*(n_code)}
      far_data        org=org(near_code)+sizeof(near_code)
                      addr=org(far_area)+sizeof(far_area)
                      : {*(f_data)}
      tiny_const      : {*(t_const)} > ROMDATA.S
      near_const      : {*(n_const)} > ROMDATA.M
      far_const       : {*(f_const)} > F_ROM

      TinyData = addr(tiny_data);
      FarData  = addr(far_data);
      TinyArea = addr(tiny_area);
      FarArea  = addr(far_area);
      tinyAreaSize = sizeof(tiny_area);
      farAreaSize  = sizeof(far_area);
      tinyDataSize = sizeof(tiny_data);
```

```
        farDataSize  = sizeof(far_data);
        tinyAreaOrg = org(tiny_area);
        farAreaOrg  = org(far_area);
        tinyDataOrg = org(tiny_data);
        farDataOrg  = org(far_data);
}
```

3.1.3 Interrupt handling

3.1.3.1 Hardware interrupts

Essentially the TX19 processor features two types of interrupt: non-maskable and maskable. Maskable interrupts are further sub-divided into software interrupts and hardware interrupts. The TX19 has a maximum of 64 interrupt sources for maskable hardware interrupts, each of which is assigned an individual interrupt vector. When a maskable hardware interrupt occurs, the interrupt controller's internal Interrupt Vector Register (IVR) is set to the corresponding interrupt vector value. The program can then read the vector value from this register and branch to the handler routine corresponding to the interrupt source.

3.1.3.2 Interrupt handler

The interrupt handler branches according to the interrupt number in the IVR register. A routine within the interrupt handler uses the interrupt vector table to determine the branch address which corresponds to the interrupt number.

- Interrupt handler

```
;Hardware interrupt
;Address of IVR must be predefined.
VECTOR_INT section code large abs=0xbfc00260
      public _ _InterruptVector
_ _InterruptVector:
      lui         r26,hi(IVR)
      addiu       r26,r26,lo(IVR)
      lw          r26,0(r26)                                ;(1)
      andi        r26,r26,0x03f0                            ;(2)
      srl         r26,r26,2
      lui         r27,hi(_ _VectorTable)
      addu        r26,r26,r27
      addiu       r26,r26,lo(_ _VectorTable)
      lw          r26,0(r26)                                ;(3)
      jr          r26                                       ;(4)
      nop
```

The numbers (1) to (4) below correspond to the numbers in the program listing above.

(1) (2) First, the contents of the IVR register are stored in r26. Since only the low-order bits 5~10 of the IVR register are used to hold the interrupt source, all other bits are masked to 0.

(3) (4) Next, the program looks up the address of the corresponding interrupt-processing function in the interrupt vector table and branches to it. Since this routine is located at address BFC00260H, the program will branch to this address when the maskable hardware interrupt occurs.

- Interrupt table

```
;Use this table to match interrupt functions to specific interrupt sources.
VECTOR_TAB section data large
      public _ _VectorTable
_ _VectorTable:
;Interrupt function name  Interrupt number  Interrupt source
      dw          _Int_dummy  ; 0 --- software set
      dw          _Int_dummy  ; 1 --- INT[0]
      dw          _Int_dummy  ; 2 --- INT[1]
      dw          _Int_sample ; 3 --- INT[2]
      dw          _Int_dummy  ; 4 --- INT[3]
```

```
        dw          _Int_dummy  ; 5 --- INT[4]
        dw          _Int_dummy  ; 6 --- *
        dw          _Int_dummy  ; 7 --- *
        dw          _Int_dummy  ; 8 --- *
        dw          _Int_dummy  ; 9 --- *
        dw          _Int_dummy  ; 10--- INT[5]
        dw          _Int_dummy  ; 11--- INT[6]
;Subsequent code omitted
```

The interrupt table look-up routine can thus determine the address of the interrupt-processing function corresponding to the interrupt which has occurred. Hence the handler can then jump to the appropriate interrupt-processing routine. In the above example, the routine will call the *Int_sample* function for interrupt number 3 (int[2]) and call a dummy function for all other interrupts. Create a dummy function and put its name into the table entries corresponding to the vectors for unused interrupts.

- Example of a dummy function

```
_Int_dummy:
        MFC0        r26,r14     ;Store interrupt-generated address in register r26.
        j           self_jump   ;Do not reverse the order of two lines shown on the left.
        rfe                     ;
```

3.1.3.3 Defining interrupt-processing functions

Interrupt-processing functions can be written in the C language.

When writing interrupt-processing functions in C, use the function qualifier _ _*interrupt*. When _ _*interrupt* is used, all registers used in the function are saved on entry to the function and restored on exit from the function. Note also that interrupt-processing functions must be of type *void* and cannot have any parameters.

- Sample description of an interrupt-processing function

```
void _ _interrupt Int_sample(){
        int i,j;

        for(i=0;i<10;i++)
          j*=i;
}
```

In order to enable the exception handler to branch to an interrupt-processing function written in C, the interrupt-processing function must be declared externally. For example, if the interrupt handler routine is included in the start-up program, declare the interrupt-processing function at the beginning of the start-up program as shown below.

```
extern large  _<interrupt-processing function name>
```

The interrupt-processing function is declared as *large* since interrupt functions are always 32-bit ISA functions.

Note: An interrupt-processing function written in C will be slower than one written in assembly language. If interrupt processing must be carried out at high speed, it is recommended that the function be written in assembly language.

3.1.4 I/O header

In the sample applications global variables are directly associated with addresses for use as I/O variables. I/O variables are unique to the Toshiba C compiler. Each of the sample applications includes the necessary I/O header.

3.1.4.1 I/O header file (used in sample applications)

```
———————— io1940.h ————————
/* io1940.h */
/* register map */

typedef unsigned int   _UINT_IO;
typedef unsigned char  _UCHR_IO;
typedef unsigned short _USHT_IO;
#ifndef EXTERN
#define EXTERN extern
#endif

/* PORT */
EXTERN _UCHR_IO __io(0xfffff000) IO_P0;
EXTERN _UCHR_IO __io(0xfffff002) IO_P0CR;
EXTERN _UCHR_IO __io(0xfffff001) IO_P1;
EXTERN _UCHR_IO __io(0xfffff004) IO_P1CR;
EXTERN _UCHR_IO __io(0xfffff005) IO_P1FC;
EXTERN _UCHR_IO __io(0xfffff012) IO_P2;
EXTERN _UCHR_IO __io(0xfffff014) IO_P2CR;
EXTERN _UCHR_IO __io(0xfffff015) IO_P2FC;
EXTERN _UCHR_IO __io(0xfffff018) IO_P3;
EXTERN _UCHR_IO __io(0xfffff01A) IO_P3CR;
EXTERN _UCHR_IO __io(0xfffff01B) IO_P3FC;
EXTERN _UCHR_IO __io(0xfffff01E) IO_P4;
EXTERN _UCHR_IO __io(0xfffff020) IO_P4CR;
EXTERN _UCHR_IO __io(0xfffff021) IO_P4FC;
EXTERN _UCHR_IO __io(0xfffff025) IO_P5;
EXTERN _UCHR_IO __io(0xfffff02B) IO_P7;
EXTERN _UCHR_IO __io(0xfffff02E) IO_P7CR;
EXTERN _UCHR_IO __io(0xfffff02F) IO_P7FC;
EXTERN _UCHR_IO __io(0xfffff030) IO_P8;
EXTERN _UCHR_IO __io(0xfffff032) IO_P8CR;
EXTERN _UCHR_IO __io(0xfffff033) IO_P8FC;
EXTERN _UCHR_IO __io(0xfffff031) IO_P9;
EXTERN _UCHR_IO __io(0xfffff034) IO_P9CR;
EXTERN _UCHR_IO __io(0xfffff035) IO_P9FC;
EXTERN _UCHR_IO __io(0xfffff036) IO_PA;
EXTERN _UCHR_IO __io(0xfffff038) IO_PACR;
EXTERN _UCHR_IO __io(0xfffff039) IO_PAFC;
EXTERN _UCHR_IO __io(0xfffff050) IO_ODE;

/* WDT */
EXTERN _UCHR_IO __io(0xfffff090) IO_WDMOD;
EXTERN _UCHR_IO __io(0xfffff091) IO_WDCR;

/* RT */
EXTERN _UCHR_IO __io(0xfffff0A0) IO_RTCCR;
EXTERN _UCHR_IO __io(0xfffff0A4) IO_RTCREG;

/* 8bit TIMER */
EXTERN _UCHR_IO __io(0xfffff100) IO_TA01RUN;
EXTERN _UCHR_IO __io(0xfffff102) IO_TA0REG;
EXTERN _UCHR_IO __io(0xfffff103) IO_TA1REG;
EXTERN _UCHR_IO __io(0xfffff104) IO_TA01MOD;
EXTERN _UCHR_IO __io(0xfffff105) IO_TA1FFCR;
```

```
EXTERN _UCHR_IO __io(0xfffff108) IO_TA23RUN;
EXTERN _UCHR_IO __io(0xfffff10A) IO_TA2REG;
EXTERN _UCHR_IO __io(0xfffff10B) IO_TA3REG;
EXTERN _UCHR_IO __io(0xfffff10C) IO_TA23MOD;
EXTERN _UCHR_IO __io(0xfffff10D) IO_TA3FFCR;

/* 16bit TIMER */
EXTERN _UCHR_IO __io(0xfffff180) IO_TB0RUN;
EXTERN _UCHR_IO __io(0xfffff182) IO_TB0MOD;
EXTERN _UCHR_IO __io(0xfffff183) IO_TB0FFCR;
EXTERN _UCHR_IO __io(0xfffff188) IO_TB0RG0L;
EXTERN _UCHR_IO __io(0xfffff189) IO_TB0RG0H;
EXTERN _UCHR_IO __io(0xfffff18A) IO_TB0RG1L;
EXTERN _UCHR_IO __io(0xfffff18B) IO_TB0RG1H;
EXTERN _UCHR_IO __io(0xfffff18C) IO_TB0CP0L;
EXTERN _UCHR_IO __io(0xfffff18D) IO_TB0CP0H;
EXTERN _UCHR_IO __io(0xfffff18E) IO_TB0CP1L;
EXTERN _UCHR_IO __io(0xfffff18F) IO_TB0CP1H;

EXTERN _UCHR_IO __io(0xfffff190) IO_TB1RUN;
EXTERN _UCHR_IO __io(0xfffff192) IO_TB1MOD;
EXTERN _UCHR_IO __io(0xfffff193) IO_TB1FFCR;
EXTERN _UCHR_IO __io(0xfffff198) IO_TB1RG0L;
EXTERN _UCHR_IO __io(0xfffff199) IO_TB1RG0H;
EXTERN _UCHR_IO __io(0xfffff19A) IO_TB1RG1L;
EXTERN _UCHR_IO __io(0xfffff19B) IO_TB1RG1H;
EXTERN _UCHR_IO __io(0xfffff19C) IO_TB1CP0L;
EXTERN _UCHR_IO __io(0xfffff19D) IO_TB1CP0H;
EXTERN _UCHR_IO __io(0xfffff19E) IO_TB1CP1L;
EXTERN _UCHR_IO __io(0xfffff19F) IO_TB1CP1H;

EXTERN _UCHR_IO __io(0xfffff1A0) IO_TB2RUN;
EXTERN _UCHR_IO __io(0xfffff1A2) IO_TB2MOD;
EXTERN _UCHR_IO __io(0xfffff1A3) IO_TB2FFCR;
EXTERN _UCHR_IO __io(0xfffff1A8) IO_TB2RG0L;
EXTERN _UCHR_IO __io(0xfffff1A9) IO_TB2RG0H;
EXTERN _UCHR_IO __io(0xfffff1AA) IO_TB2RG1L;
EXTERN _UCHR_IO __io(0xfffff1AB) IO_TB2RG1H;
EXTERN _UCHR_IO __io(0xfffff1AC) IO_TB2CP0L;
EXTERN _UCHR_IO __io(0xfffff1AD) IO_TB2CP0H;
EXTERN _UCHR_IO __io(0xfffff1AE) IO_TB2CP1L;
EXTERN _UCHR_IO __io(0xfffff1AF) IO_TB2CP1H;

EXTERN _UCHR_IO __io(0xfffff1B0) IO_TB3RUN;
EXTERN _UCHR_IO __io(0xfffff1B2) IO_TB3MOD;
EXTERN _UCHR_IO __io(0xfffff1B3) IO_TB3FFCR;
EXTERN _UCHR_IO __io(0xfffff1B8) IO_TB3RG0L;
EXTERN _UCHR_IO __io(0xfffff1B9) IO_TB3RG0H;
EXTERN _UCHR_IO __io(0xfffff1BA) IO_TB3RG1L;
EXTERN _UCHR_IO __io(0xfffff1BB) IO_TB3RG1H;
EXTERN _UCHR_IO __io(0xfffff1BC) IO_TB3CP0L;
EXTERN _UCHR_IO __io(0xfffff1BD) IO_TB3CP0H;
EXTERN _UCHR_IO __io(0xfffff1BE) IO_TB3CP1L;
EXTERN _UCHR_IO __io(0xfffff1BF) IO_TB3CP1H;

/* UART/SIO 0/1 */
EXTERN _UCHR_IO __io(0xfffff200) IO_SC0BUF;
EXTERN _UCHR_IO __io(0xfffff201) IO_SC0CR;
EXTERN _UCHR_IO __io(0xfffff202) IO_SC0MOD0;
EXTERN _UCHR_IO __io(0xfffff203) IO_BR0CR;
EXTERN _UCHR_IO __io(0xfffff204) IO_BR0ADD;
EXTERN _UCHR_IO __io(0xfffff205) IO_SC0MOD1;
EXTERN _UCHR_IO __io(0xfffff208) IO_SC1BUF;
EXTERN _UCHR_IO __io(0xfffff209) IO_SC1CR;
EXTERN _UCHR_IO __io(0xfffff20A) IO_SC1MOD0;
EXTERN _UCHR_IO __io(0xfffff20B) IO_BR1CR;
```

```
EXTERN _UCHR_IO __io(0xfffff20C) IO_BR1ADD;
EXTERN _UCHR_IO __io(0xfffff20D) IO_SC1MOD1;

/* ISCBUS/SIO */
EXTERN _UCHR_IO __io(0xfffff240) IO_SBI0CR1;
EXTERN _UCHR_IO __io(0xfffff241) IO_SBI0DBR;
EXTERN _UCHR_IO __io(0xfffff242) IO_I2C0AR;
EXTERN _UCHR_IO __io(0xfffff243) IO_SBI0CR2;
#define IO_SBI0SR IO_SBI0CR2
EXTERN _UCHR_IO __io(0xfffff244) IO_SBI0BR0;
EXTERN _UCHR_IO __io(0xfffff245) IO_SBI0BR1;

/* UART 3/4 */
EXTERN _UCHR_IO __io(0xfffff280) IO_SC3BUF;
EXTERN _UCHR_IO __io(0xfffff281) IO_SC3CR;
EXTERN _UCHR_IO __io(0xfffff282) IO_SC3MOD0;
EXTERN _UCHR_IO __io(0xfffff283) IO_BR3CR;
EXTERN _UCHR_IO __io(0xfffff284) IO_BR3ADD;
EXTERN _UCHR_IO __io(0xfffff285) IO_SC3MOD1;
EXTERN _UCHR_IO __io(0xfffff288) IO_SC4BUF;
EXTERN _UCHR_IO __io(0xfffff289) IO_SC4CR;
EXTERN _UCHR_IO __io(0xfffff28A) IO_SC4MOD0;
EXTERN _UCHR_IO __io(0xfffff28B) IO_BR4CR;
EXTERN _UCHR_IO __io(0xfffff28C) IO_BR4ADD;
EXTERN _UCHR_IO __io(0xfffff28D) IO_SC4MOD1;

/* 10bitADC */
EXTERN _UCHR_IO __io(0xfffff300) IO_ADREG04L;
EXTERN _UCHR_IO __io(0xfffff301) IO_ADREG04H;
EXTERN _UCHR_IO __io(0xfffff302) IO_ADREG15L;
EXTERN _UCHR_IO __io(0xfffff303) IO_ADREG15H;
EXTERN _UCHR_IO __io(0xfffff304) IO_ADREG26L;
EXTERN _UCHR_IO __io(0xfffff305) IO_ADREG26H;
EXTERN _UCHR_IO __io(0xfffff306) IO_ADREG37L;
EXTERN _UCHR_IO __io(0xfffff307) IO_ADREG37H;
EXTERN _UCHR_IO __io(0xfffff310) IO_ADMOD0;
EXTERN _UCHR_IO __io(0xfffff311) IO_ADMOD1;
EXTERN _UCHR_IO __io(0xfffff312) IO_ADMOD2;

/* INTC */
EXTERN _USHT_IO __io(0xffffe000) IO_IMC0L;
EXTERN _USHT_IO __io(0xffffe002) IO_IMC0H;
EXTERN _USHT_IO __io(0xffffe004) IO_IMC1L;
EXTERN _USHT_IO __io(0xffffe008) IO_IMC2L;
EXTERN _USHT_IO __io(0xffffe00a) IO_IMC2H;
EXTERN _USHT_IO __io(0xffffe00c) IO_IMC3L;
EXTERN _USHT_IO __io(0xffffe00e) IO_IMC3H;
EXTERN _USHT_IO __io(0xffffe014) IO_IMC5L;
EXTERN _USHT_IO __io(0xffffe016) IO_IMC5H;
EXTERN _USHT_IO __io(0xffffe01c) IO_IMC7L;
EXTERN _USHT_IO __io(0xffffe01e) IO_IMC7H;
EXTERN _USHT_IO __io(0xffffe020) IO_IMC8L;
EXTERN _USHT_IO __io(0xffffe022) IO_IMC8H;
EXTERN _USHT_IO __io(0xffffe028) IO_IMCAL;
EXTERN _USHT_IO __io(0xffffe02a) IO_IMCAH;
EXTERN _USHT_IO __io(0xffffe030) IO_IMCCL;
EXTERN _USHT_IO __io(0xffffe032) IO_IMCCH;
EXTERN _USHT_IO __io(0xffffe034) IO_IMCDL;
EXTERN _USHT_IO __io(0xffffe036) IO_IMCDH;
EXTERN _USHT_IO __io(0xffffe038) IO_IMCEL;
EXTERN _USHT_IO __io(0xffffe03a) IO_IMCEH;
EXTERN _USHT_IO __io(0xffffe03c) IO_IMCFL;
EXTERN _USHT_IO __io(0xffffe03e) IO_IMCFH;
EXTERN _USHT_IO __io(0xffffe040) IO_IVR;
EXTERN _USHT_IO __io(0xffffe060) IO_INTCLR;
```

```
/* DMAC ch.0 */
EXTERN _USHT_IO __io(0xffffe200) IO_CCR0;
EXTERN _USHT_IO __io(0xffffe204) IO_CSR0;
EXTERN _USHT_IO __io(0xffffe208) IO_SAR0;
EXTERN _USHT_IO __io(0xffffe20c) IO_DAR0;
EXTERN _USHT_IO __io(0xffffe210) IO_BCR0;
EXTERN _USHT_IO __io(0xffffe218) IO_DTCR0;

/* DMAC ch.1 */
EXTERN _USHT_IO __io(0xffffe220) IO_CCR1;
EXTERN _USHT_IO __io(0xffffe224) IO_CSR1;
EXTERN _USHT_IO __io(0xffffe228) IO_SAR1;
EXTERN _USHT_IO __io(0xffffe22c) IO_DAR1;
EXTERN _USHT_IO __io(0xffffe230) IO_BCR1;
EXTERN _USHT_IO __io(0xffffe238) IO_DTCR1;

/* DMAC ch.2 */
EXTERN _USHT_IO __io(0xffffe240) IO_CCR2;
EXTERN _USHT_IO __io(0xffffe244) IO_CSR2;
EXTERN _USHT_IO __io(0xffffe248) IO_SAR2;
EXTERN _USHT_IO __io(0xffffe24c) IO_DAR2;
EXTERN _USHT_IO __io(0xffffe250) IO_BCR2;
EXTERN _USHT_IO __io(0xffffe258) IO_DTCR2;

/* DMAC ch.3 */
EXTERN _USHT_IO __io(0xffffe260) IO_CCR3;
EXTERN _USHT_IO __io(0xffffe264) IO_CSR3;
EXTERN _USHT_IO __io(0xffffe268) IO_SAR3;
EXTERN _USHT_IO __io(0xffffe26c) IO_DAR3;
EXTERN _USHT_IO __io(0xffffe270) IO_BCR3;
EXTERN _USHT_IO __io(0xffffe278) IO_DTCR3;

EXTERN _USHT_IO __io(0xffffe280) IO_DCR;
EXTERN _USHT_IO __io(0xffffe28C) IO_DHR;

/* CS/WAIT */
EXTERN _UINT_IO __io(0xffffe400) IO_BMA0;
EXTERN _UINT_IO __io(0xffffe404) IO_BMA1;
EXTERN _UINT_IO __io(0xffffe408) IO_BMA2;
EXTERN _UINT_IO __io(0xffffe40C) IO_BMA3;
EXTERN _UINT_IO __io(0xffffe480) IO_B01CS;
EXTERN _UINT_IO __io(0xffffe484) IO_B23CS;
EXTERN _USHT_IO __io(0xffffe488) IO_BEXCS;

/* CG */
EXTERN _UCHR_IO __io(0xffffeE00) IO_SYSCR0;
EXTERN _UCHR_IO __io(0xffffeE01) IO_SYSCR1;
EXTERN _UCHR_IO __io(0xffffeE02) IO_SYSCR2;
EXTERN _UCHR_IO __io(0xffffeE03) IO_SYSCR3;
EXTERN _UCHR_IO __io(0xffffeE04) IO_ADCCLK;
EXTERN _UCHR_IO __io(0xffffeE10) IO_IMCGA0;
EXTERN _UCHR_IO __io(0xffffeE11) IO_IMCGA1;
EXTERN _UCHR_IO __io(0xffffeE12) IO_IMCGA2;
EXTERN _UCHR_IO __io(0xffffeE13) IO_IMCGA3;
EXTERN _UCHR_IO __io(0xffffeE14) IO_IMCGB0;
EXTERN _UCHR_IO __io(0xffffeE17) IO_IMCGB3;
EXTERN _UCHR_IO __io(0xffffeE20) IO_EICRCG;
```

——————— adc.h ———————

```
/* adc.h */
/* A/D Converter */
#ifndef _BIT32_DEF
#include "bit32def.h"
#endif

/* ADMOD0 */
```

```
#define EOCF   _BIT07
#define ADBF   _BIT06
#define ITM0   _BIT03
#define REPEAT _BIT02
#define SCAN   _BIT01
#define ADS    _BIT00

/* ADMOD1 */
#define VREFON _BIT07
#define I2AD   _BIT06
#define ADTRGE _BIT03
#define ADCH2  _BIT02
#define ADCH1  _BIT01
#define ADCH0  _BIT00

/* ADREG04L */
#define ADR01  _BIT07
#define ADR00  _BIT06
#define ADR0RF _BIT00

/* ADREG04H */
#define ADR09  _BIT07
#define ADR08  _BIT06
#define ADR07  _BIT05
#define ADR06  _BIT04
#define ADR05  _BIT03
#define ADR04  _BIT02
#define ADR03  _BIT01
#define ADR02  _BIT00

/* ADREG15L */
#define ADR11  _BIT07
#define ADR10  _BIT06
#define ADR1RF _BIT00

/* ADREG15H */
#define ADR19  _BIT07
#define ADR18  _BIT06
#define ADR17  _BIT05
#define ADR16  _BIT04
#define ADR15  _BIT03
#define ADR14  _BIT02
#define ADR13  _BIT01
#define ADR12  _BIT00

/* ADREG26L */
#define ADR21  _BIT07
#define ADR20  _BIT06
#define ADR2RF _BIT00

/* ADREG26H */
#define ADR29  _BIT07
#define ADR28  _BIT06
#define ADR27  _BIT05
#define ADR26  _BIT04
#define ADR25  _BIT03
#define ADR24  _BIT02
#define ADR23  _BIT01
#define ADR22  _BIT00

/* ADREG37L */
#define ADR31  _BIT07
#define ADR30  _BIT06
#define ADR3RF _BIT00

/* ADREG37H */
```

```
#define ADR39  _BIT07
#define ADR38  _BIT06
#define ADR37  _BIT05
#define ADR36  _BIT04
#define ADR35  _BIT03
#define ADR34  _BIT02
#define ADR33  _BIT01
#define ADR32  _BIT00
```

———— cg.h ————

```
/* cg.h */
/* CG */
#ifndef _BIT32_DEF
#include "bit32def.h"
#endif

/* SYSCR */
#define SCOSEL _BIT30
#define ALESEL _BIT28
#define LUPEG  _BIT25
#define LUPTM  _BIT24
#define WUPT1  _BIT21
#define WUPT0  _BIT20
#define STBY1  _BIT19
#define STBY0  _BIT18
#define DRVE   _BIT16
#define SYSCK  _BIT13
#define FPSEL  _BIT12
#define DFOSC  _BIT11
#define GEAR1  _BIT09
#define GEAR0  _BIT08
#define XEN    _BIT07
#define XTEN   _BIT06
#define RXEN   _BIT05
#define RXTEN  _BIT04
#define RSYSCK _BIT03
#define WUEF   _BIT02
#define PRCK1  _BIT01
#define PRCK0  _BIT00

/* ADCLK */
#define ADCCK1 _BIT01
#define ADCCK0 _BIT00

/* IMCGA */
#define EMCG31 _BIT29
#define EMCG30 _BIT28
#define INT3EN _BIT24
#define EMCG21 _BIT21
#define EMCG20 _BIT20
#define INT2EN _BIT16
#define EMCG11 _BIT13
#define EMCG10 _BIT12
#define INT1EN _BIT08
#define EMCG01 _BIT05
#define EMCG00 _BIT04
#define INT0EN _BIT00

/* IMCGB */
#define EMCG71   _BIT29
#define EMCG70   _BIT28
#define INTRTCEN _BIT24
#define EMCG41   _BIT05
#define EMCG40   _BIT04
#define INT4EN   _BIT00
```

```
/* EICRCG */
#define ICRCG2 _BIT02
#define ICRCG1 _BIT01
#define ICRCG0 _BIT00

———————— cs_wait.h ————————
/* cs_wait.h */
/* CS/WAIT Controller */
#ifndef _BIT32_DEF
#include "bit32def.h"
#endif

/* BMA0 */
#define MA0
(_BIT15|_BIT14|_BIT13|_BIT12|_BIT11|_BIT10|_BIT09|_BIT08|_BIT07|_BIT06|_BIT05|_BIT04|_BIT
03|_BIT02|_BIT01|_BIT00)
#define BA0
(_BIT31|_BIT30|_BIT29|_BIT28|_BIT27|_BIT26|_BIT25|_BIT24|_BIT23|_BIT22|_BIT21|_BIT20|_BIT
19|_BIT18|_BIT17|_BIT16)

/* BMA1 */
#define MA1
(_BIT15|_BIT14|_BIT13|_BIT12|_BIT11|_BIT10|_BIT09|_BIT08|_BIT07|_BIT06|_BIT05|_BIT04|_BIT
03|_BIT02|_BIT01|_BIT00)
#define BA1
(_BIT31|_BIT30|_BIT29|_BIT28|_BIT27|_BIT26|_BIT25|_BIT24|_BIT23|_BIT22|_BIT21|_BIT20|_BIT
19|_BIT18|_BIT17|_BIT16)

/* BMA2 */
#define MA2
(_BIT15|_BIT14|_BIT13|_BIT12|_BIT11|_BIT10|_BIT09|_BIT08|_BIT07|_BIT06|_BIT05|_BIT04|_BIT
03|_BIT02|_BIT01|_BIT00)
#define BA2
(_BIT31|_BIT30|_BIT29|_BIT28|_BIT27|_BIT26|_BIT25|_BIT24|_BIT23|_BIT22|_BIT21|_BIT20|_BIT
19|_BIT18|_BIT17|_BIT16)

/* BMA3 */
#define MA3
(_BIT15|_BIT14|_BIT13|_BIT12|_BIT11|_BIT10|_BIT09|_BIT08|_BIT07|_BIT06|_BIT05|_BIT04|_BIT
03|_BIT02|_BIT01|_BIT00)
#define BA3
(_BIT31|_BIT30|_BIT29|_BIT28|_BIT27|_BIT26|_BIT25|_BIT24|_BIT23|_BIT22|_BIT21|_BIT20|_BIT
19|_BIT18|_BIT17|_BIT16)

/* B01CS */
#define B0OM   (_BIT07|_BIT06)
#define B0BUS  _BIT04
#define B0W    (_BIT03|_BIT02|_BIT01|_BIT00)
#define B0E    _BIT11
#define B0RCV  (_BIT09|_BIT08)
#define B1OM   (_BIT23|_BIT22)
#define B1BUS  _BIT20
#define B1W    (_BIT19|_BIT18|_BIT17|_BIT16)
#define B1E    _BIT27
#define B1RCV  (_BIT25|_BIT24)

/* B23CS */
#define B2OM   (_BIT07|_BIT06)
#define B2BUS  _BIT04
#define B2W    (_BIT03|_BIT02|_BIT01|_BIT00)
#define B2E    _BIT11
#define B2M    _BIT10
#define B2RCV  (_BIT09|_BIT08)
#define B3OM   (_BIT23|_BIT22)
#define B3BUS  _BIT20
#define B3W    (_BIT19|_BIT18|_BIT17|_BIT16)
```

```
#define B3E    _BIT27
#define B3RCV  (_BIT25|_BIT24)

/* BEXCS */
#define BEXOM  (_BIT07|_BIT06)
#define BEXBUS _BIT04
#define BEXW   (_BIT03|_BIT02|_BIT01|_BIT00)
#define BEXRCV (_BIT09|_BIT08)
```

———————— dmac.h ————————

```
/* dmac.h */
/* DMAC */
#ifndef _BIT32_DEF
#include "bit32def.h"
#endif

/* CCR0,CCR1,CCR2,CCR3 */
#define Str    _BIT31
#define NIEn   _BIT23
#define AbIEn  _BIT22
#define Big    _BIT17
#define ExR    _BIT14
#define PosE   _BIT13
#define Lev    _BIT12
#define SReq   _BIT11
#define RelEn  _BIT10
#define SIO    _BIT09
#define SAC1   _BIT08
#define SAC0   _BIT07
#define DIO    _BIT06
#define DAC1   _BIT05
#define DAC0   _BIT04
#define TrSiz1 _BIT03
#define TrSiz0 _BIT02
#define DPS1   _BIT01
#define DPS0   _BIT00

/* CSR0,CSR1,CSR2,CSR3 */
#define Act    _BIT31
#define NC     _BIT23
#define AbC    _BIT22
#define BES    _BIT20
#define BED    _BIT19
#define Conf   _BIT18

/* SAR0,SAR1,SAR2,SAR3 */
#define SAddr0  _BIT00
#define SAddr1  _BIT01
#define SAddr2  _BIT02
#define SAddr3  _BIT03
#define SAddr4  _BIT04
#define SAddr5  _BIT05
#define SAddr6  _BIT06
#define SAddr7  _BIT07
#define SAddr8  _BIT08
#define SAddr9  _BIT09
#define SAddr10 _BIT10
#define SAddr11 _BIT11
#define SAddr12 _BIT12
#define SAddr13 _BIT13
#define SAddr14 _BIT14
#define SAddr15 _BIT15
#define SAddr16 _BIT16
#define SAddr17 _BIT17
#define SAddr18 _BIT18
#define SAddr19 _BIT19
```

```
#define SAddr20 _BIT20
#define SAddr21 _BIT21
#define SAddr22 _BIT22
#define SAddr23 _BIT23
#define SAddr24 _BIT24
#define SAddr25 _BIT25
#define SAddr26 _BIT26
#define SAddr27 _BIT27
#define SAddr28 _BIT28
#define SAddr29 _BIT29
#define SAddr30 _BIT30
#define SAddr31 _BIT31

/* DAR0,DAR1,DAR2,DAR3 */
#define DAddr0  _BIT00
#define DAddr1  _BIT01
#define DAddr2  _BIT02
#define DAddr3  _BIT03
#define DAddr4  _BIT04
#define DAddr5  _BIT05
#define DAddr6  _BIT06
#define DAddr7  _BIT07
#define DAddr8  _BIT08
#define DAddr9  _BIT09
#define DAddr10 _BIT10
#define DAddr11 _BIT11
#define DAddr12 _BIT12
#define DAddr13 _BIT13
#define DAddr14 _BIT14
#define DAddr15 _BIT15
#define DAddr16 _BIT16
#define DAddr17 _BIT17
#define DAddr18 _BIT18
#define DAddr19 _BIT19
#define DAddr20 _BIT20
#define DAddr21 _BIT21
#define DAddr22 _BIT22
#define DAddr23 _BIT23
#define DAddr24 _BIT24
#define DAddr25 _BIT25
#define DAddr26 _BIT26
#define DAddr27 _BIT27
#define DAddr28 _BIT28
#define DAddr29 _BIT29
#define DAddr30 _BIT30
#define DAddr31 _BIT31

/* BCR0,BCR1,BCR2,BCR3 */
#define DMABC0  _BIT00
#define DMABC1  _BIT01
#define DMABC2  _BIT02
#define DMABC3  _BIT03
#define DMABC4  _BIT04
#define DMABC5  _BIT05
#define DMABC6  _BIT06
#define DMABC7  _BIT07
#define DMABC8  _BIT08
#define DMABC9  _BIT09
#define DMABC10 _BIT10
#define DMABC11 _BIT11
#define DMABC12 _BIT12
#define DMABC13 _BIT13
#define DMABC14 _BIT14
#define DMABC15 _BIT15
#define DMABC16 _BIT16
#define DMABC17 _BIT17
```

```
#define DMABC18 _BIT18
#define DMABC19 _BIT19
#define DMABC20 _BIT20
#define DMABC21 _BIT21
#define DMABC22 _BIT22
#define DMABC23 _BIT23

/* DTCR0,DTCR1,DTCR2,DTCR3 */
#define DACM2   _BIT05
#define DACM1   _BIT04
#define DACM0   _BIT03
#define SACM2   _BIT02
#define SACM1   _BIT01
#define SACM0   _BIT00

/* DCR */
#define Rst     _BIT31

/* DHR0,DHR1,DHR2,DHR3 */
#define DOT0    _BIT00
#define DOT1    _BIT01
#define DOT2    _BIT02
#define DOT3    _BIT03
#define DOT4    _BIT04
#define DOT5    _BIT05
#define DOT6    _BIT06
#define DOT7    _BIT07
#define DOT8    _BIT08
#define DOT9    _BIT09
#define DOT10   _BIT10
#define DOT11   _BIT11
#define DOT12   _BIT12
#define DOT13   _BIT13
#define DOT14   _BIT14
#define DOT15   _BIT15
#define DOT16   _BIT16
#define DOT17   _BIT17
#define DOT18   _BIT18
#define DOT19   _BIT19
#define DOT20   _BIT20
#define DOT21   _BIT21
#define DOT22   _BIT22
#define DOT23   _BIT23
#define DOT24   _BIT24
#define DOT25   _BIT25
#define DOT26   _BIT26
#define DOT27   _BIT27
#define DOT28   _BIT28
#define DOT29   _BIT29
#define DOT30   _BIT30
#define DOT31   _BIT31

———————— intc.h ————————
/* intc.h */
/* INTC */
#ifndef _BIT32_DEF
#include "bit32def.h"
#endif

/* IMC0L */
#define EIM11 _BIT13
#define EIM10 _BIT12
#define DM1   _BIT11
#define IL12  _BIT10
#define IL11  _BIT09
#define IL10  _BIT08
```

```
#define EIM01 _BIT05
#define EIM00 _BIT04
#define DM0   _BIT03
#define IL02  _BIT02
#define IL01  _BIT01
#define IL00  _BIT00

/* IMC0H */
#define EIM31 _BIT29
#define EIM30 _BIT28
#define DM3   _BIT27
#define IL32  _BIT26
#define IL31  _BIT25
#define IL30  _BIT24
#define EIM21 _BIT21
#define EIM20 _BIT20
#define DM2   _BIT19
#define IL22  _BIT18
#define IL21  _BIT17
#define IL20  _BIT16

/* IMC1L */
#define EIM51 _BIT13
#define EIM50 _BIT12
#define DM5   _BIT11
#define IL52  _BIT10
#define IL51  _BIT09
#define IL50  _BIT08
#define EIM41 _BIT05
#define EIM40 _BIT04
#define DM4   _BIT03
#define IL42  _BIT02
#define IL41  _BIT01
#define IL40  _BIT00

/* IMC2H */
#define EIMB1 _BIT29
#define EIMB0 _BIT28
#define DMb   _BIT27
#define ILB2  _BIT26
#define ILB1  _BIT25
#define ILB0  _BIT24
#define EIA1  _BIT21
#define EIMA0 _BIT20
#define DMA   _BIT19
#define ILA2  _BIT18
#define ILA1  _BIT17
#define ILA0  _BIT16

/* IMC3L */
#define EIMD1 _BIT13
#define EIMD0 _BIT12
#define DMD   _BIT11
#define ILD2  _BIT10
#define ILD1  _BIT09
#define ILD0  _BIT08
#define EIMC1 _BIT05
#define EIMC0 _BIT04
#define DMC   _BIT03
#define ILC2  _BIT02
#define ILC1  _BIT01
#define ILC0  _BIT00

/* IMC3H */
#define EIMF1 _BIT29
#define EIMF0 _BIT28
```

```
#define DMF   _BIT27
#define ILF2  _BIT26
#define ILF1  _BIT25
#define ILF0  _BIT24
#define EIME1 _BIT21
#define EIME0 _BIT20
#define DME   _BIT19
#define ILE2  _BIT18
#define ILE1  _BIT17
#define ILE0  _BIT16

/* IMC5L */
#define EIM151 _BIT13
#define EIM150 _BIT12
#define DM15   _BIT11
#define IL152  _BIT10
#define IL151  _BIT09
#define IL150  _BIT08
#define EIM141 _BIT05
#define EIM140 _BIT04
#define DM14   _BIT03
#define IL142  _BIT02
#define IL141  _BIT01
#define IL140  _BIT00

/* IMC5H */
#define EIM171 _BIT29
#define EIM170 _BIT28
#define DM17   _BIT27
#define IL172  _BIT26
#define IL171  _BIT25
#define IL170  _BIT24
#define EIM161 _BIT21
#define EIM160 _BIT20
#define DM16   _BIT19
#define IL162  _BIT18
#define IL161  _BIT17
#define IL160  _BIT16

/* IMC7L */
#define EIMID1 _BIT13
#define EIMID0 _BIT12
#define DMID   _BIT11
#define ILID2  _BIT10
#define ILID1  _BIT09
#define ILID0  _BIT08
#define EIMIC1 _BIT05
#define EIMIC0 _BIT04
#define DMIC   _BIT03
#define ILIC2  _BIT02
#define ILIC1  _BIT01
#define ILIC0  _BIT00

/* IMC7H */
#define EIMIF1 _BIT29
#define EIMIF0 _BIT28
#define DMIF   _BIT27
#define ILIF2  _BIT26
#define ILIF1  _BIT25
#define ILIF0  _BIT24
#define EIMIE1 _BIT21
#define EIMIE0 _BIT20
#define DMIE   _BIT19
#define ILIE2  _BIT18
#define ILIE1  _BIT17
#define ILIE0  _BIT16
```

```
/* IMC8L */
#define EIM211 _BIT13
#define EIM210 _BIT12
#define DM21   _BIT11
#define IL212  _BIT10
#define IL211  _BIT09
#define IL210  _BIT08
#define EIM201 _BIT05
#define EIM200 _BIT04
#define DM20   _BIT03
#define IL202  _BIT02
#define IL201  _BIT01
#define IL200  _BIT00

/* IMC8H */
#define EIM231 _BIT29
#define EIM230 _BIT28
#define DM23   _BIT27
#define IL232  _BIT26
#define IL231  _BIT25
#define IL230  _BIT24
#define EIM221 _BIT21
#define EIM220 _BIT20
#define DM22   _BIT19
#define IL222  _BIT18
#define IL221  _BIT17
#define IL220  _BIT16

/* IMCAL */
#define EIM291 _BIT13
#define EIM290 _BIT12
#define DM29   _BIT11
#define IL292  _BIT10
#define IL291  _BIT09
#define IL290  _BIT08
#define EIM281 _BIT05
#define EIM280 _BIT04
#define DM28   _BIT03
#define IL282  _BIT02
#define IL281  _BIT01
#define IL280  _BIT00

/* IMCAH */
#define EIM2B1 _BIT29
#define EIM2B0 _BIT28
#define DM2B   _BIT27
#define IL2B2  _BIT26
#define IL2B1  _BIT25
#define IL2B0  _BIT24
#define EIM2A1 _BIT21
#define EIM2A0 _BIT20
#define DM2A   _BIT19
#define IL2A2  _BIT18
#define IL2A1  _BIT17
#define IL2A0  _BIT16

/* IMCCL */
#define EIM311 _BIT13
#define EIM310 _BIT12
#define DM31   _BIT11
#define IL312  _BIT10
#define IL311  _BIT09
#define IL310  _BIT08
#define EIM301 _BIT05
#define EIM300 _BIT04
```

```
#define DM30   _BIT03
#define IL302  _BIT02
#define IL301  _BIT01
#define IL300  _BIT00

/* IMCCH */
#define EIM331 _BIT29
#define EIM330 _BIT28
#define DM33   _BIT27
#define IL332  _BIT26
#define IL331  _BIT25
#define IL330  _BIT24
#define EIM321 _BIT21
#define EIM320 _BIT20
#define DM32   _BIT19
#define IL322  _BIT18
#define IL321  _BIT17
#define IL320  _BIT16

/* IMCDL */
#define EIM341 _BIT05
#define EIM340 _BIT04
#define DM34   _BIT03
#define IL342  _BIT02
#define IL341  _BIT01
#define IL340  _BIT00

/* IMCDH */
#define EIM371 _BIT29
#define EIM370 _BIT28
#define DM37   _BIT27
#define IL372  _BIT26
#define IL371  _BIT25
#define IL370  _BIT24
#define EIM361 _BIT21
#define EIM360 _BIT20
#define DM36   _BIT19
#define IL362  _BIT18
#define IL361  _BIT17
#define IL360  _BIT16

/* IMCEL */
#define EIM391 _BIT13
#define EIM390 _BIT12
#define DM39   _BIT11
#define IL392  _BIT10
#define IL391  _BIT09
#define IL390  _BIT08
#define EIM381 _BIT05
#define EIM380 _BIT04
#define DM38   _BIT03
#define IL382  _BIT02
#define IL381  _BIT01
#define IL380  _BIT00

/* IMCEH */
#define EIM3B1 _BIT29
#define EIM3B0 _BIT28
#define DM3B   _BIT27
#define IL3B2  _BIT26
#define IL3B1  _BIT25
#define IL3B0  _BIT24
#define EIM3A1 _BIT21
#define EIM3A0 _BIT20
#define DM3A   _BIT19
#define IL3A2  _BIT18
```

```
#define IL3A1  _BIT17
#define IL3A0  _BIT16

/* IMCFL */
#define EIM3D1 _BIT13
#define EIM3D0 _BIT12
#define DM3D   _BIT11
#define IL3D2  _BIT10
#define IL3D1  _BIT09
#define IL3D0  _BIT08
#define EIM3C1 _BIT05
#define EIM3C0 _BIT04
#define DM3C   _BIT03
#define IL3C2  _BIT02
#define IL3C1  _BIT01
#define IL3C0  _BIT00

/* IMCFH */
#define EIM3F1 _BIT29
#define EIM3F0 _BIT28
#define DM3F   _BIT27
#define IL3F2  _BIT26
#define IL3F1  _BIT25
#define IL3F0  _BIT24
#define EIM3E1 _BIT21
#define EIM3E0 _BIT20
#define DM3E   _BIT19
#define IL3E2  _BIT18
#define IL3E1  _BIT17
#define IL3E0  _BIT16

/* IVR */
#define IVR9   _BIT09
#define IVR8   _BIT08
#define IVR7   _BIT07
#define IVR6   _BIT06
#define IVR5   _BIT05
#define IVR4   _BIT04

/* INTCLR */
#define EICLR6 _BIT06
#define EICLR5 _BIT05
#define EICLR4 _BIT04
#define EICLR3 _BIT03
#define EICLR2 _BIT02
#define EICLR1 _BIT01
#define EICLR0 _BIT00
```

——————— pio.h ———————

```
/* pio.h */
/* PIO */
#ifndef _BIT32_DEF
#include "bit32def.h"
#endif

/* PORT0 */
/* P0 */
#define P07   _BIT07
#define P06   _BIT06
#define P05   _BIT05
#define P04   _BIT04
#define P03   _BIT03
#define P02   _BIT02
#define P01   _BIT01
#define P00   _BIT00
/* P0CR */
```

```
#define P07C  _BIT07
#define P06C  _BIT06
#define P05C  _BIT05
#define P04C  _BIT04
#define P03C  _BIT03
#define P02C  _BIT02
#define P01C  _BIT01
#define P00C  _BIT00

/* PORT1 */
/* P1 */
#define P17   _BIT07
#define P16   _BIT06
#define P15   _BIT05
#define P14   _BIT04
#define P13   _BIT03
#define P12   _BIT02
#define P11   _BIT01
#define P10   _BIT00
/* P1CR */
#define P17C  _BIT07
#define P16C  _BIT06
#define P15C  _BIT05
#define P14C  _BIT04
#define P13C  _BIT03
#define P12C  _BIT02
#define P11C  _BIT01
#define P10C  _BIT00
/* P1FC */
#define P17F  _BIT07
#define P16F  _BIT06
#define P15F  _BIT05
#define P14F  _BIT04
#define P13F  _BIT03
#define P12F  _BIT02
#define P11F  _BIT01
#define P10F  _BIT00

/* PORT2 */
/* P2 */
#define P27   _BIT07
#define P26   _BIT06
#define P25   _BIT05
#define P24   _BIT04
#define P23   _BIT03
#define P22   _BIT02
#define P21   _BIT01
#define P20   _BIT00
/* P2CR */
#define P27C  _BIT07
#define P26C  _BIT06
#define P25C  _BIT05
#define P24C  _BIT04
#define P23C  _BIT03
#define P22C  _BIT02
#define P21C  _BIT01
#define P20C  _BIT00
/* P2FC */
#define P27F  _BIT07
#define P26F  _BIT06
#define P25F  _BIT05
#define P24F  _BIT04
#define P23F  _BIT03
#define P22F  _BIT02
#define P21F  _BIT01
#define P20F  _BIT00
```

```
/* PORT3 */
/* P3 */
#define P37   _BIT07
#define P36   _BIT06
#define P35   _BIT05
#define P34   _BIT04
#define P33   _BIT03
#define P32   _BIT02
#define P31   _BIT01
#define P30   _BIT00
/* P3CR */
#define P37C  _BIT07
#define P36C  _BIT06
#define P35C  _BIT05
#define P34C  _BIT04
#define P33C  _BIT03
#define P32C  _BIT02
/* P3FC */
#define P36F  _BIT06
#define P35F  _BIT05
#define P34F  _BIT04
#define P32F  _BIT02
#define P31F  _BIT01
#define P30F  _BIT00

/* PORT4 */
/* P4 */
#define P44   _BIT04
#define P43   _BIT03
#define P42   _BIT02
#define P41   _BIT01
#define P40   _BIT00
/* P4CR */
#define P44C  _BIT04
#define P43C  _BIT03
#define P42C  _BIT02
#define P41C  _BIT01
#define P40C  _BIT00
/* P4FC */
#define P44F  _BIT04
#define P43F  _BIT03
#define P42F  _BIT02
#define P41F  _BIT01
#define P40F  _BIT00

/* PORT5 */
/* P5 */
#define P57   _BIT07
#define P56   _BIT06
#define P55   _BIT05
#define P54   _BIT04
#define P53   _BIT03
#define P52   _BIT02
#define P51   _BIT01
#define P50   _BIT00

/* PORT7 */
/* P7 */
#define P77   _BIT07
#define P76   _BIT06
#define P75   _BIT05
#define P74   _BIT04
#define P73   _BIT03
#define P72   _BIT02
#define P71   _BIT01
```

```
#define P70   _BIT00
/* P7CR */
#define P77C  _BIT07
#define P76C  _BIT06
#define P75C  _BIT05
#define P74C  _BIT04
#define P73C  _BIT03
#define P72C  _BIT02
#define P71C  _BIT01
#define P70C  _BIT00
/* P7FC */
#define P76F  _BIT06
#define P75F  _BIT05
#define P74F  _BIT04
#define P73F  _BIT03
#define P72F  _BIT02
#define P71F  _BIT01
#define P70F  _BIT00

/* PORT8 */
/* P8 */
#define P87   _BIT07
#define P86   _BIT06
#define P85   _BIT05
#define P84   _BIT04
#define P83   _BIT03
#define P82   _BIT02
#define P81   _BIT01
#define P80   _BIT00
/* P8CR */
#define P87C  _BIT07
#define P86C  _BIT06
#define P85C  _BIT05
#define P84C  _BIT04
#define P83C  _BIT03
#define P82C  _BIT02
#define P81C  _BIT01
#define P80C  _BIT00
/* P8FC */
#define P87F  _BIT07
#define P86F  _BIT06
#define P85F  _BIT05
#define P84F  _BIT04
#define P83F  _BIT03
#define P82F  _BIT02
#define P81F  _BIT01
#define P80F  _BIT00

/* PORT9 */
/* P9 */
#define P97   _BIT07
#define P96   _BIT06
#define P95   _BIT05
#define P94   _BIT04
#define P93   _BIT03
#define P92   _BIT02
#define P91   _BIT01
#define P90   _BIT00
/* P9CR */
#define P97C  _BIT07
#define P96C  _BIT06
#define P95C  _BIT05
#define P94C  _BIT04
#define P93C  _BIT03
#define P92C  _BIT02
#define P91C  _BIT01
```

```
#define P90C  _BIT00
/* P9FC */
#define P95F  _BIT05
#define P93F  _BIT03
#define P92F  _BIT02
#define P90F  _BIT00

/* PORTA */
/* PA */
#define PA7   _BIT07
#define PA6   _BIT06
#define PA5   _BIT05
#define PA4   _BIT04
#define PA3   _BIT03
#define PA2   _BIT02
#define PA1   _BIT01
#define PA0   _BIT00
/* PACR */
#define PA7C  _BIT07
#define PA6C  _BIT06
#define PA5C  _BIT05
#define PA4C  _BIT04
#define PA3C  _BIT03
#define PA2C  _BIT02
#define PA1C  _BIT01
#define PA0C  _BIT00
/* PAFC */
#define PA7F  _BIT07
#define PA6F  _BIT06
#define PA5F  _BIT05
#define PA4F  _BIT04

/* ODE */
#define ODE72 _BIT05
#define ODE70 _BIT04
#define ODEA7 _BIT03
#define ODEA6 _BIT02
#define ODE93 _BIT01
#define ODE90 _BIT00

———————— sio.h ————————
/* sio.h */
/* SIO */
#ifndef _BIT32_DEF
#include "bit32def.h"
#endif

/* SC0CR,SC1CR,SC3CR,SC4CR */
#define RB8     _BIT07
#define EVEN    _BIT06
#define PE      _BIT05
#define OERR    _BIT04
#define PERR    _BIT03
#define FERR    _BIT02
#define SCLKS   _BIT01
#define IOC     _BIT00

/* SC0MOD0,SC1MOD0,SC3MOD0,SC4MOD0 */
#define TB8     _BIT07
#define CTSE    _BIT06
#define RXE     _BIT05
#define WU      _BIT04
#define SM1     _BIT03
#define SM0     _BIT02
#define SC1     _BIT01
#define SC0     _BIT00
```

```
/* BR0CR */
#define BR0ADDE _BIT06
#define BR0CK1  _BIT05
#define BR0CK0  _BIT04
#define BR0S3   _BIT03
#define BR0S2   _BIT02
#define BR0S1   _BIT01
#define BR0S0   _BIT00

/* BR1CR */
#define BR1ADDE _BIT06
#define BR1CK1  _BIT05
#define BR1CK0  _BIT04
#define BR1S3   _BIT03
#define BR1S2   _BIT02
#define BR1S1   _BIT01
#define BR1S0   _BIT00

/* BR3CR */
#define BR3ADDE _BIT06
#define BR3CK1  _BIT05
#define BR3CK0  _BIT04
#define BR3S3   _BIT03
#define BR3S2   _BIT02
#define BR3S1   _BIT01
#define BR3S0   _BIT00

/* BR4CR */
#define BR4ADDE _BIT06
#define BR4CK1  _BIT05
#define BR4CK0  _BIT04
#define BR4S3   _BIT03
#define BR4S2   _BIT02
#define BR4S1   _BIT01
#define BR4S0   _BIT00

/* SC0MOD1,SC1MOD1,SC3MOD1,SC4MOD1 */
#define I2S0    _BIT07
#define FDPX0   _BIT06

/* BR0ADD */
#define BR0K3   _BIT03
#define BR0K2   _BIT02
#define BR0K1   _BIT01
#define BR0K0   _BIT00

/* BR1ADD */
#define BR1K3   _BIT03
#define BR1K2   _BIT02
#define BR1K1   _BIT01
#define BR1K0   _BIT00

/* BR3ADD */
#define BR3K3   _BIT03
#define BR3K2   _BIT02
#define BR3K1   _BIT01
#define BR3K0   _BIT00

/* BR4ADD */
#define BR4K3   _BIT03
#define BR4K2   _BIT02
#define BR4K1   _BIT01
#define BR4K0   _BIT00

/* SC0BUF,SC1BUF,SC3BUF,SC4BUF */
```

```
#define SC7     _BIT07
#define SC6     _BIT06
#define SC5     _BIT05
#define SC4     _BIT04
#define SC3     _BIT03
#define SC2     _BIT02
#define SC1     _BIT01
#define SC0     _BIT00
#define RB7     _BIT07
#define RB6     _BIT06
#define RB5     _BIT05
#define RB4     _BIT04
#define RB3     _BIT03
#define RB2     _BIT02
#define RB1     _BIT01
#define RB0     _BIT00

/* I2CBUS/SIO */
/* SBI0CR1 */
#define BC2     _BIT07
#define SIOS    _BIT07
#define BC1     _BIT06
#define SIOINH  _BIT06
#define BC0     _BIT05
#define SIOM1   _BIT05
#define ACK     _BIT04
#define SIOM0   _BIT04
#define SCK2    _BIT02
#define SCK1    _BIT01
#define SCK0    _BIT00

/* SBI0DBR */
#define DB7     _BIT07
#define DB6     _BIT06
#define DB5     _BIT05
#define DB4     _BIT04
#define DB3     _BIT03
#define DB2     _BIT02
#define DB1     _BIT01
#define DB0     _BIT00

/* I2C0AR */
#define SA6     _BIT07
#define SA5     _BIT06
#define SA4     _BIT05
#define SA3     _BIT04
#define SA2     _BIT03
#define SA1     _BIT02
#define SA0     _BIT01
#define ALS     _BIT00

/* SBI0CR2,SBI0SR */
#define MST     _BIT07
#define TRX     _BIT06
#define BB      _BIT05
#define PIN     _BIT04
#define SBIM1   _BIT03
#define AL      _BIT03
#define SIOF    _BIT03
#define SBIM0   _BIT02
#define AAS     _BIT02
#define SEF     _BIT02
#define SWRST1  _BIT01
#define AD0     _BIT01
#define SWRST0  _BIT00
#define LRB     _BIT00
```

```
/* SBI0BR0 */
#define SBI0BR0 _BIT06

/* SBI0BR1 */
#define P4EN    _BIT07
```

———— tmr.h ————

```
/* tmr.h */
/* TMR */
#ifndef _BIT32_DEF
#include "bit32def.h"
#endif

/* 8bit timer */
/* TA01RUN */
#define TA0RDE   _BIT07
#define I2TA01   _BIT03
#define TA01PRUN _BIT02
#define TA1RUN   _BIT01
#define TA0RUN   _BIT00
/* TA23RUN */
#define TA2RDE   _BIT07
#define I2TA23   _BIT03
#define TA23PRUN _BIT02
#define TA3RUN   _BIT01
#define TA2RUN   _BIT00
/* TA01MOD */
#define TA01M1   _BIT07
#define TA01M0   _BIT06
#define PWM01    _BIT05
#define PWM00    _BIT04
#define TA1CLK1  _BIT03
#define TA1CLK0  _BIT02
#define TA0CLK1  _BIT01
#define TA0CLK0  _BIT00
/* TA23MOD */
#define TA23M1   _BIT07
#define TA23M0   _BIT06
#define PWM21    _BIT05
#define PWM20    _BIT04
#define TA3CLK1  _BIT03
#define TA3CLK0  _BIT02
#define TA2CLK1  _BIT01
#define TA2CLK0  _BIT00
/* TA1FFCR */
#define TAFF1C1  _BIT03
#define TAFF1C0  _BIT02
#define TAFF1IE  _BIT01
#define TAFF1IS  _BIT00
/* TA3FFCR */
#define TAFF3C1  _BIT03
#define TAFF3C0  _BIT02
#define TAFF3IE  _BIT01
#define TAFF3IS  _BIT00

/* 16bit timer */
/* TB0RUN */
#define TB0RDE   _BIT07
#define I2TB0    _BIT03
#define TB0PRUN  _BIT02
#define TB0RUN   _BIT00
/* TB1RUN */
#define TB1RDE   _BIT07
#define I2TB1    _BIT03
#define TB1PRUN  _BIT02
```

```
#define TB1RUN   _BIT00
/* TB2RUN */
#define TB2RDE   _BIT07
#define I2TB2    _BIT03
#define TB2PRUN  _BIT02
#define TB2RUN   _BIT00
/* TB3RUN */
#define TB3RDE   _BIT07
#define I2TB3    _BIT03
#define TB3PRUN  _BIT02
#define TB3RUN   _BIT00
/* TB0MOD */
#define TB0CP0I  _BIT05
#define TB0CPM1  _BIT04
#define TB0CPM0  _BIT03
#define TB0CLE   _BIT02
#define TB0CLK1  _BIT01
#define TB0CLK0  _BIT00
/* TB1MOD */
#define TB1CP0I  _BIT05
#define TB1CPM1  _BIT04
#define TB1CPM0  _BIT03
#define TB1CLE   _BIT02
#define TB1CLK1  _BIT01
#define TB1CLK0  _BIT00
/* TB2MOD */
#define TB2CP0I  _BIT05
#define TB2CPM1  _BIT04
#define TB2CPM0  _BIT03
#define TB2CLE   _BIT02
#define TB2CLK1  _BIT01
#define TB2CLK0  _BIT00
/* TB3MOD */
#define TB3CP0I  _BIT05
#define TB3CPM1  _BIT04
#define TB3CPM0  _BIT03
#define TB3CLE   _BIT02
#define TB3CLK1  _BIT01
#define TB3CLK0  _BIT00
/* TB0FFCR */
#define TB0C1T1  _BIT05
#define TB0C0T1  _BIT04
#define TB0E1T1  _BIT03
#define TB0E0T1  _BIT02
#define TB0FF0C1 _BIT01
#define TB0FF0C0 _BIT00
/* TB1FFCR */
#define TB1C1T1  _BIT05
#define TB1C0T1  _BIT04
#define TB1E1T1  _BIT03
#define TB1E0T1  _BIT02
#define TB1FF0C1 _BIT01
#define TB1FF0C0 _BIT00
/* TB2FFCR */
#define TB2C1T1  _BIT05
#define TB2C0T1  _BIT04
#define TB2E1T1  _BIT03
#define TB2E0T1  _BIT02
#define TB2FF0C1 _BIT01
#define TB2FF0C0 _BIT00
/* TB0FFCR */
#define TB3C1T1  _BIT05
#define TB3C0T1  _BIT04
#define TB3E1T1  _BIT03
#define TB3E0T1  _BIT02
#define TB3FF0C1 _BIT01
```

```
#define TB3FF0C0 _BIT00

/* WDT */
#define WDTE     _BIT07
#define WDTP1    _BIT06
#define WDTP0    _BIT05
#define I2WDT    _BIT02
#define RESCR    _BIT01

/* RT */
#define RTCRCLR  _BIT03
#define RTCSEL1  _BIT02
#define RTCSEL0  _BIT01
#define RTCRUN   _BIT00
```

———— bit32def.h ————

```
/* _BIT32_DEF */

#ifndef _BIT32_DEF
#define _BIT32_DEF
#endif

#define _BIT00     (0x00000001)
#define _BIT01     (0x00000002)
#define _BIT02     (0x00000004)
#define _BIT03     (0x00000008)
#define _BIT04     (0x00000010)
#define _BIT05     (0x00000020)
#define _BIT06     (0x00000040)
#define _BIT07     (0x00000080)
#define _BIT08     (0x00000100)
#define _BIT09     (0x00000200)
#define _BIT10     (0x00000400)
#define _BIT11     (0x00000800)
#define _BIT12     (0x00001000)
#define _BIT13     (0x00002000)
#define _BIT14     (0x00004000)
#define _BIT15     (0x00008000)
#define _BIT16     (0x00010000)
#define _BIT17     (0x00020000)
#define _BIT18     (0x00040000)
#define _BIT19     (0x00080000)
#define _BIT20     (0x00100000)
#define _BIT21     (0x00200000)
#define _BIT22     (0x00400000)
#define _BIT23     (0x00800000)
#define _BIT24     (0x01000000)
#define _BIT25     (0x02000000)
#define _BIT26     (0x04000000)
#define _BIT27     (0x08000000)
#define _BIT28     (0x10000000)
#define _BIT29     (0x20000000)
#define _BIT30     (0x40000000)
#define _BIT31     (0x80000000)
```

———— io1940.c ————

```
#ifndef EXTERN
#define EXTERN
#endif
#include "io1940.h"
```

When the io1940.c file is created in this way, it should be compiled together with the io1940.h source file.

3.2 DC Motor Drive

To change the speed of the built-in DC motor, turn the control knob on the circuit board.

3.2.1 Functional specifications

3.2.1.1 Motor operation method

To start the built-in DC motor, press the INT0 switch on the TB1940 board. (*Take care to avoid personal injury or damage to other equipment from the motor's rotating shaft.) Turn the control knob on the circuit board clockwise to increase the motor speed and counterclockwise to reduce the motor speed. The motor speed is indicated by the decimal number on the 7-segment LEDs (in units of rpm). The motor speed shown is obtained by reading the pulse output from the motor's encoder using the TX1940 and converting it into a number of revolutions. (The motor speed is controlled using open-loop control, and without using feedback control) The INT0 switch can also be used to stop the motor. The 7-segment LEDs display the number of revolutions while the motor is moving and "STOP" while it is not.

3.2.1.2 Basic specifications

- The motor speed is controlled using PWM. The duty cycle of the PWM waveform is determined by the voltage level for the control knob.
- The voltage from the control knob is converted into a digital value by an AD converter.
- The motor speed can be increased by turning the control knob clockwise.
- The 7-segment LEDs show the rate of revolution based on the output from the motor's encoder.
- The motor encoder's period is calculated using the TX1940's capture function.
- The built-in motor generates 200 pulses per revolution.
- The INT0 switch can be used to start and stop the motor.

3.2.2 Functional block diagram

Figure 3.2.1 shows the functional blocks of the TX1940 which are used by the sample program.

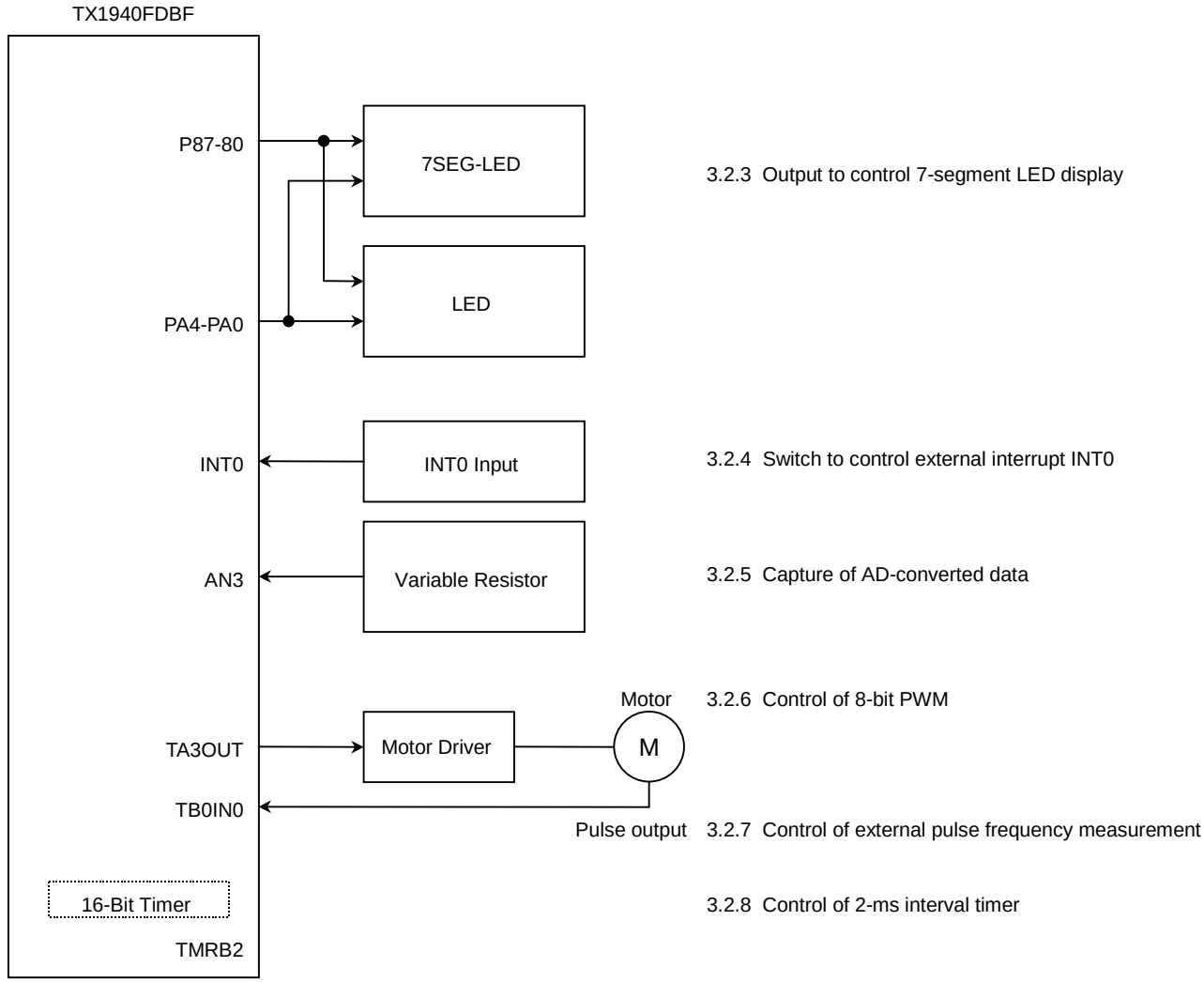


Figure 3.2.1 Functional block diagram

3.2.3 Control of 7-segment LED display output

3.2.3.1 Overview of 7-segment LED display

- The display data is dynamically displayed on the 7-segment LEDs and updated every 2 ms.

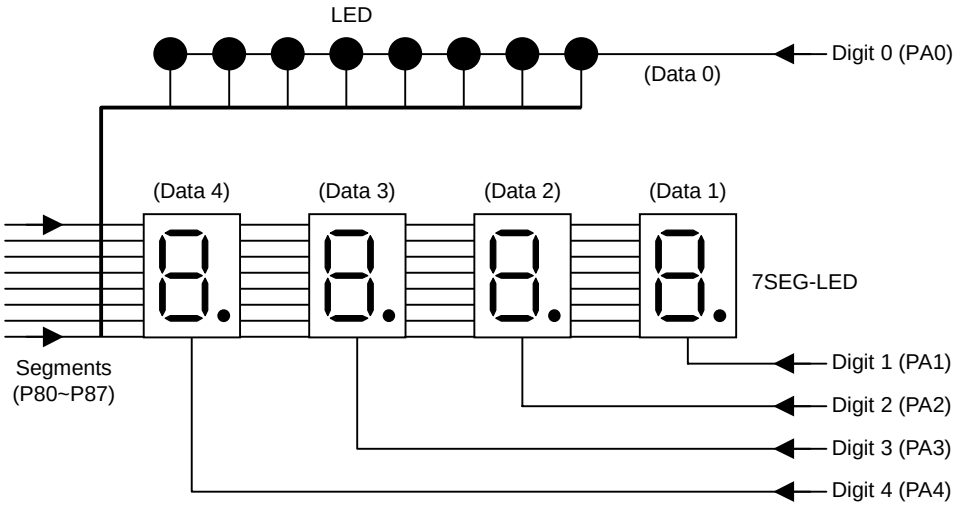
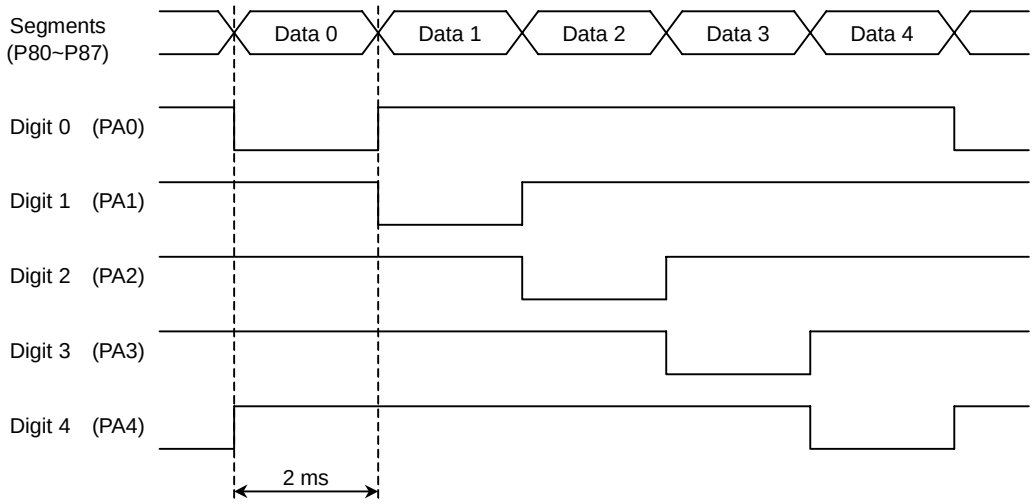


Figure 3.2.2 Correspondence between digits and segments

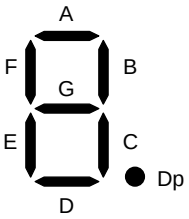
- Figure 3.2.3 shows the timing for the digit and segment outputs.



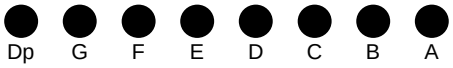
Note: The segment and digit outputs are all active-Low.

Figure 3.2.3 Timing of display digit update

- To light any given display segment, output a Low signal on the corresponding ports. The correspondence between display segments and port outputs is shown in Table 3.2.1.



7SEG-LED



LED

Table 3.2.1 Correspondence between display segments and port outputs

	P87(Dp)	P86(G)	P85(F)	P84(E)	P83(D)	P82(C)	P81(B)	P80(A)
0	*	1	0	0	0	0	0	0
1	*	1	1	1	1	0	0	1
2	*	0	1	0	0	1	0	0
3	*	0	1	1	0	0	0	0
4	*	0	0	1	1	0	0	1
5	*	0	0	1	0	0	1	0
6	*	0	0	0	0	0	1	0
7	*	1	0	1	1	0	0	0
8	*	0	0	0	0	0	0	0
9	*	0	0	1	1	0	0	0
A	*	0	0	0	1	0	0	0
B	*	0	0	0	0	0	1	1
C	*	1	0	0	0	1	1	0
D	*	0	1	0	0	0	0	1
E	*	0	0	0	0	1	1	0
F	*	0	0	0	1	1	1	0
Blank	1	1	1	1	1	1	1	1

Note 1: All segment outputs are active-Low.

Note 2: Set port output * to 0 to display Dp and to 1 to turn Dp off.

3.2.3.2 Control method for 7-segment LED display

■ Initial settings

		7	6	5	4	3	2	1	0	
P8	←	1	1	1	1	1	1	1	1	Set P80~P87 output latches to 1.
P8CR	←	1	1	1	1	1	1	1	1	Set P80~P87 for output.
P8FC	←	0	0	0	0	0	0	0	0	Set P80~P87 to be port.
PA	←	-	-	-	1	1	1	1	1	Set PA0~PA4 output latches to 1.
PACR	←		-	-	1	1	1	1	1	Set PA0~PA4 for output.
PAFC	←	-	-	-	0	0	0	0	0	Set PA0~PA4 to be port.

Note: X denotes Don't care; "–" denotes No change.

■ Processing of the display output

Create the display data in the main routine and output display data for each digit to the ports within a 2-ms Interval Timer interrupt. For details of the 2-ms interval timer, please refer to Section 3.3.8.

The brightness of the 7-segment and indicator LEDs can be adjusted by changing the time at which display data is output to the ports. The display output control flow is shown in Figure 3.2.4.

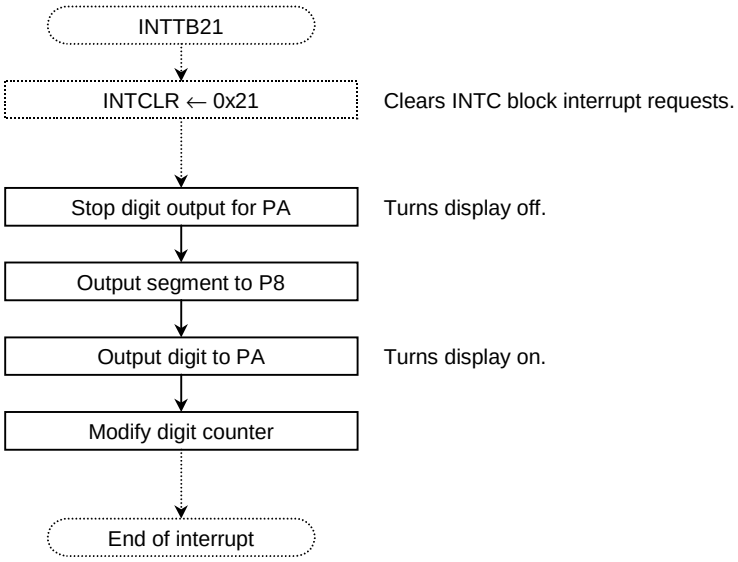


Figure 3.2.4 Display output control flow

3.2.4 Controlling of the INT0 external interrupt switch

3.2.4.1 Overview of the external interrupt switch

The INT0 interrupt is an edge-triggered interrupt; the interrupt is recognized on the falling edge of INT0. The active state of this interrupt is set by the CG block of the TX1940. In the sample program here, the INT0 interrupt is used as an external interrupt switch.

3.2.4.2 Method for controlling the external interrupt switch

- Initial settings

		7	6	5	4	3	2	1	0		
P7CR	←	0	-	-	-	-	-	-	-		Set P77 to be INT0 input pin.
P7FC	←	0	-	-	-	-	-	-	-		
IMCGA0	←	X	X	1	0	X	X	X	1		Set the interrupt's active state to falling edge using the CG block.
EICRCG=0x00											
IMC0L	←	X	X	0	1	0	1	1	0	High-order	Make sure interrupt sources are recognized as active when High.
		X	X	X	X	X	X	X	X	Low-order	Set the interrupt level to 6. Any desired value can be set in the Interrupt Register.
INTCLR=0x01											Clear the interrupt request during initialization.

- Interrupt handling

Figure 3.2.5 shows the control flow for handling of the INT0 interrupt. When INT0 and other external interrupts are handled, the CG block interrupt requests must be cleared in software. Also, INTC block interrupt requests must be processed in software, irrespective of how interrupt handling is performed.

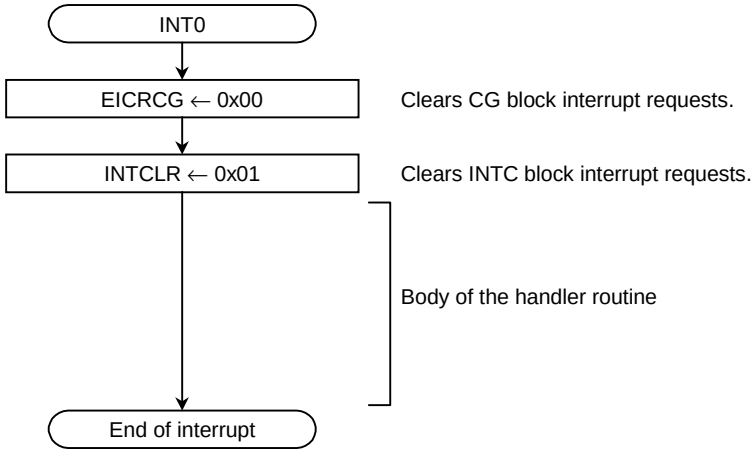


Figure 3.2.5 INT0 interrupt control flow

3.2.5 Capturing AD-converted data

3.2.5.1 Overview of AD conversion

- When the control knob on the board is turned, a voltage proportional to the knob position is applied to AN3. In the sample program the analog data on AN3 is sampled every 2.0 ms. The 2.0-ms interval time is effected using an interval timer interrupt. (For details of how to set the interval timer interrupt, please refer to Section 3.2.8.)
- AD conversion is performed in Fixed-Channel Single-Conversion Mode.
- The voltage on AN3 is used to calculate the duty cycle for the PWM waveform. (For details of how to calculate the duty cycle, please refer to Section 3.2.6.)

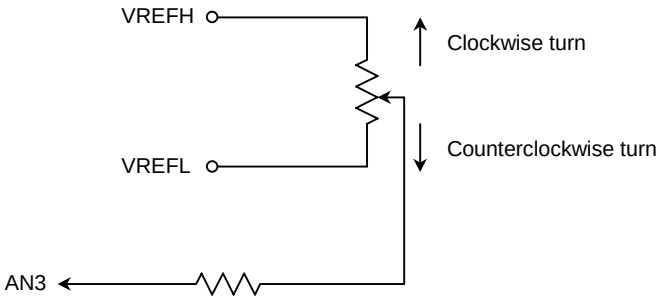


Figure 3.2.6 Control knob switch circuit

3.2.5.2 AD conversion control method

■ Initial settings

Initial settings in the main routine

		7	6	5	4	3	2	1	0		
ADCCLK	←	X	X	X	X	X	X	0	1		Set AD conversion time to 10.75 µs (when fsys = 32 MHz).
INTCLR=0x3b											
IMCEH	←	X	X	1	1	0	0	1	1	High-order	Set active state of INTAD to be rising edge and set level to 3.
		X	X	X	X	X	X	X	X	Low-order	The Interrupt Register can be set to any desired value.
ADMOD1	←	1	X	X	X	0	0	1	1		Set analog input channel to AN3 and select Vref as voltage to apply.

Note: X denotes Don't care.

Settings for 2-ms interval timer

		7	6	5	4	3	2	1	0	
ADMOD0	←	X	X	0	0	0	0	0	1	Start conversion in Fixed-Channel Single-Conversion Mode.

■ Capturing AD-converted data

- When an interrupt is generated to signal completion of AD conversion, read out the data from the AD Conversion Result Register.
- The control flow for capturing AD converted data is shown in Figure 3.2.7.

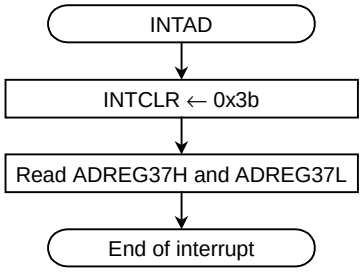


Figure 3.2.7 INTAD interrupt control flow

3.2.6 Controlling 8-bit PWM

3.2.6.1 Overview of 8-bit PWM

- Using the 8-Bit Timer (TMRA2), generate a PWM signal of 7-bit resolution.
- Using TMRA2 and 2^n-1 overflow control, output a PWM signal with any desired duty cycle in 126 steps from the timer flip-flop output pin TA3OUT. Figure 3.2.8 shows the PWM output waveform.
- The PWM period is set to 31.75 µs. The duty cycle is rewritten every 2 ms based on the value of the AD conversion result (AN3). (The 2-ms interval time is effected using a TMRB Interval Timer interrupt.)
- The motor drive circuit on the TB1940 is active-Low (see Figure 3.2.9). Therefore, the revolution speed of the DC motor in this program is at its lowest when TAREG2 = 1 and at its highest when TAREG2 = 126. (When motor revolution is enabled, TA3FF is initially set to 1.)
- Control of the PWM duty cycle
  TA2REG = 1 Low-level duration = 0.25 µs Duty cycle = 99.21%
  TA2REG = 126 Low-level duration = 31.5 µs Duty cycle = 0.79%
  Adjust the waveform so that the Low-level duration lies between these two extremes. The Low-level duration can be set in increments of 0.25 µs.
- In the sample program the seven high-order bits of the 10-bit AN3 conversion result are stored in TAREG2. However, because TAREG2 < (2^7 – 1) and TAREG2 ≠ 0, TAREG2 can take any value from 1 to 126.

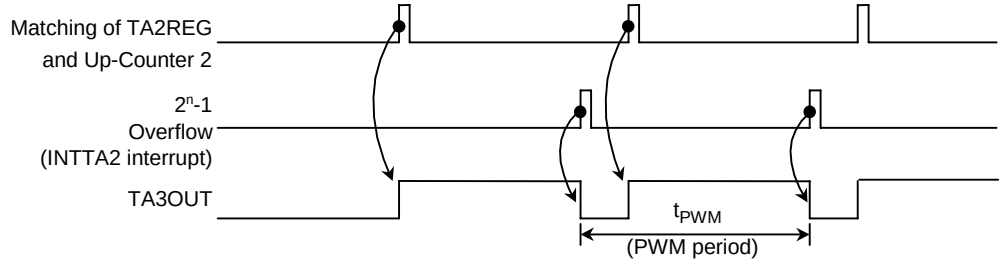


Figure 3.2.8 PWM timing waveform

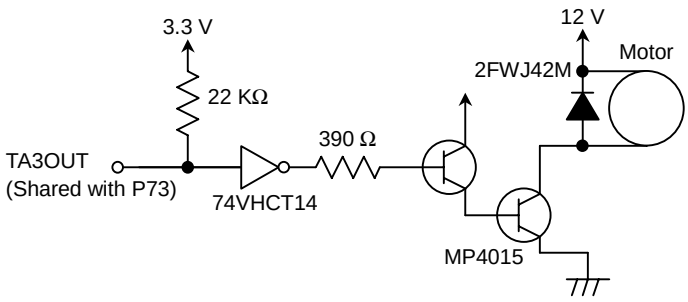


Figure 3.2.9 Motor drive circuit

Table 3.2.2 PWM period

@fc = 32 MHz

Peripheral Clock Selection <FPSEL>	Clock Gear Value <GEAR1:0>	Selection of Prescaler Clock <PRCK1:0>	PWM Period								
			$2^6 - 1$			$2^7 - 1$			$2^8 - 1$		
			φT1	φT4	φT16	φT1	φT4	φT16	φT1	φT4	φT16
0 (fgear)	00 (fc)	00 (fperiph/4)	15.8 μs	63 μs	252 μs	31.8 μs	127 μs	508 μs	63.8 μs	255 μs	1020 μs
		01 (fperiph/2)	7.9 μs	31.5 μs	126 μs	15.9 μs	63.5 μs	254 μs	31.9 μs	127.5 μs	510 μs
		10 (fperiph)	—	15.8 μs	63 μs	—	31.8 μs	127 μs	—	63.8 μs	255 μs
	01 (fc/2)	00 (fperiph/4)	31.5 μs	126 μs	504 μs	63.5 μs	254 μs	1016 μs	127.5 μs	510 μs	2040 μs
		01 (fperiph/2)	15.8 μs	63 μs	252 μs	31.8 μs	127 μs	508 μs	63.8 μs	255 μs	1020 μs
		10 (fperiph)	—	31.5 μs	126 μs	—	63.5 μs	254 μs	—	127.5 μs	510 μs
	10 (fc/4)	00 (fperiph/4)	63 μs	252 μs	1008 μs	127 μs	508 μs	2032 μs	255 μs	1020 μs	4080 μs
		01 (fperiph/2)	31.5 μs	126 μs	504 μs	63.5 μs	254 μs	1016 μs	127.5 μs	510 μs	2040 μs
		10 (fperiph)	—	63 μs	252 μs	—	127 μs	508 μs	—	255 μs	1020 μs
	11 (fc/8)	00 (fperiph/4)	126 μs	504 μs	2016 μs	254 μs	1016 μs	4064 μs	510 μs	2040 μs	8160 μs
		01 (fperiph/2)	63 μs	252 μs	1008 μs	127 μs	508 μs	2032 μs	255 μs	1020 μs	4080 μs
		10 (fperiph)	—	126 μs	504 μs	—	254 μs	1016 μs	—	510 μs	2040 μs
1 (fc)	00 (fc)	00 (fperiph/4)	15.8 μs	63 μs	252 μs	31.8 μs	127 μs	508 μs	63.8 μs	255 μs	1020 μs
		01 (fperiph/2)	7.9 μs	31.5 μs	126 μs	15.9 μs	63.5 μs	254 μs	31.9 μs	127.5 μs	510 μs
		10 (fperiph)	—	15.8 μs	63 μs	—	31.8 μs	127 μs	—	63.8 μs	255 μs
	01 (fc/2)	00 (fperiph/4)	15.8 μs	63 μs	252 μs	31.8 μs	127 μs	508 μs	63.8 μs	255 μs	1020 μs
		01 (fperiph/2)	—	31.5 μs	126 μs	—	63.5 μs	254 μs	—	127.5 μs	510 μs
		10 (fperiph)	—	15.8 μs	63 μs	—	31.8 μs	127 μs	—	63.8 μs	255 μs
	10 (fc/4)	00 (fperiph/4)	—	63 μs	252 μs	—	127 μs	508 μs	—	255 μs	1020 μs
		01 (fperiph/2)	—	31.5 μs	126 μs	—	63.5 μs	254 μs	—	127.5 μs	510 μs
		10 (fperiph)	—	—	63 μs	—	—	127 μs	—	—	255 μs
	11 (fc/8)	00 (fperiph/4)	—	63 μs	252 μs	—	127 μs	508 μs	—	255 μs	1020 μs
		01 (fperiph/2)	—	—	126 μs	—	—	254 μs	—	—	510 μs
		10 (fperiph)	—	—	63 μs	—	—	127 μs	—	—	255 μs

Note 1: The prescaler's output clock φTn must be selected such that the relationship φTn < fsys/2 is satisfied (i.e. φTn must be slower than fsys/2).

Note 2: The dash character, —, in the table indicates a prohibited setting.

3.2.6.2 PWM control method

■ Initial settings

Clock conditions

System clock :High-speed (fc = 32 MHz)
High-speed clock gear :x1 (fc)
Prescaler clock :fperiph/4 (fperiph = fsys)

		MSB 7	6	5	4	3	2	1	LSB 0	
P7CR	←	–	–	–	–	1	–	–	–	Set P73 to be TA3OUT pin.
P7FC	←	–	–	–	–	1	–	–	–	
TA23RUN	←	–	X	X	X	–	–	–	0	Stop TMRA2 and clear it to 0.
TA23MOD	←	1	1	1	0	–	–	0	1	Select PWM Mode (with period = 2^7– 1) and select φT1 as input clock.
TA2REG	←	0	0	0	0	0	0	0	1	Initial value for PWM duty cycle Low-level duration
TA3FFCR	←	X	X	X	X	0	1	0	0	Set TA3FF to 1 and set it to be inverted by a match detection signal from TMRA2. Set INT0 interrupt to enable inversion. TMRA2 count is started by INT0.

(Note: X denotes Don't care; "–" denotes No change.

■ PWM duty cycle set-up

		MSB							LSB	
		7	6	5	4	3	2	1	0	
TA2REG	←	0	-	-	-	-	-	-	-	Set Low-level duration <according to AD3 conversion result>

TAREG2 can be set to any value from 1 to 126. Figure 3.2.10 shows a diagram of the PWM waveform when the motor's rate of revolution is set to its minimum and its maximum values by the program.

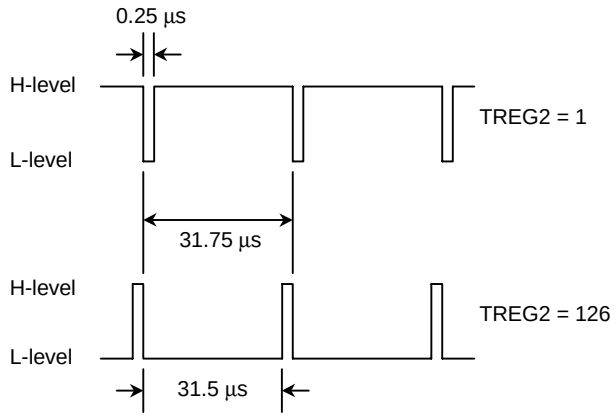


Figure 3.2.10 Sample PWM waveform

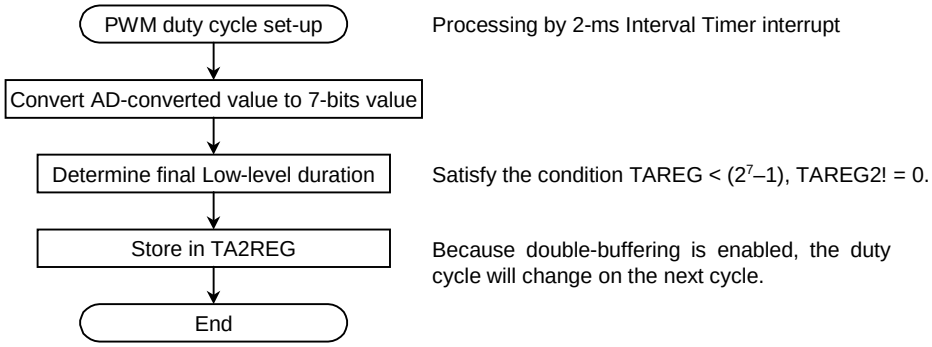


Figure 3.2.11 Flowchart showing how PWM duty cycle is determined

■ Processing using the INT0_interrupt

- Stop and start the motor using the INT0 interrupt switch.
  (*For details of how to set the INT0 interrupt, please refer to Section 3.2.4.)
- Settings to start the motor

		MSB							LSB	
		7	6	5	4	3	2	1	0	
TA23RUN	←	–	X	X	X	–	–	–	0	Stop TA2RUN and clear to 0.
TA3FFCR	←	X	X	X	X	1	0	1	0	Clear TA3FF and enable TMRA2 for inversion.
TA23RUN	←	–	X	X	X	–	–	–	1	TMRA2 starts counting.

Note: X denotes Don't care; "–" denotes No change.

- Settings to stop the motor

		MSB							LSB	
		7	6	5	4	3	2	1	0	
TA23RUN	←	–	X	X	X	–	–	–	0	Stop TA2RUN and clear to 0.
TA3FFCR	←	X	X	X	X	0	1	0	0	Set TA3FF and disable inversion.

Figure 3.2.12 shows the flowchart for starting and stopping the motor.

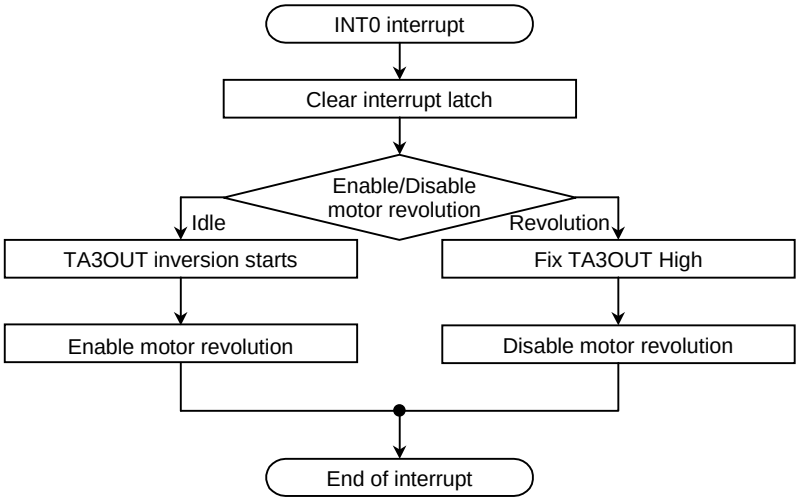


Figure 3.2.12 Flowchart for starting and stopping the Motor

3.2.7 Controlling external pulse frequency measurement

3.2.7.1 Overview of frequency measurement feature

- Using the 16-Bit Timer capture feature, measure the encoder pulse output frequency for the built-in DC motor.
- The built-in DC motor generates 200 encoder pulses per revolution.
- The motor's encoder output is connected to the TB0IN0 input to enable the frequency of the external pulses to be ascertained.
  - TA1OUT is selected for capture timing.
  - TA1FF is inverted using TMRA1 as the source clock.
  - The Low-level duration of TA1FF is set to 8192 μs (to generate an INTTA1 interrupt every 8192 μs). The number of pulses on TB0IN0 during this interval is measured. The INTTA1 generation interval can be set to any desired time by changing the value of TA1REG.
  - The DC motor's encoder period (TB0IN0) is ascertained by the INTTA1 interrupt routine.

3.2.7.2 Method for controlling frequency measurement

■ Initial settings

		7	6	5	4	3	2	1	0		
TMRA1 setting											
TA01RUN	←	-	X	X	X	-	-	1	-		Stop TMRA1 and clear it to 0.
TA01MOD	←	0	0	X	X	1	1	X	X		Select ϕT256 as the input clock.
TA1FFCR	←	X	X	X	X	1	0	1	1		Clear TA1FF and select TMRA1 with inversion enabled.
TA1REG	←	128									64 μs × 128 = 8192 μs
IMC5L	←	X	X	1	1	0	1	0	1	High-order	Enable INTTA1, set interrupt level to 5 and active state to be rising edge. (Note 1)
		X	X	X	X	X	X	X	X	Low-order	
TMRB setting											
TB0RUN	←	-	0	X	X	-	-	-	0		Stop TMRB0 and clear it to 0.
TB0MOD	←	0	0	1	1	1	0	0	0		Select TA1OUT for capture timing, disable up-counter from being cleared and select TB0IN0 as source clock.
TB0RUN	←	0	0	X	X	-	1	X	1		Start TMRB0.
TA01RUN	←	0	X	X	X	-	1	1	X		Start TMRA1.

Note 1: The interrupt level can be set to any desired value.

■ INTTA1 interrupt processing

- TMRB0 latches the value of UC0 into TB0CP0 on the rise of TA1OUT and the value of UC0 into TB0CP1 on the fall of TA1OUT.
- The difference between TB0CP0 and TB0CP1 is ascertained when the INTTA1 interrupt is generated on the rise of TA1OUT, enabling the number of pulses to be calculated.
- A flowchart showing INTTA1 interrupt control is shown in Figure 3.2.13.
- In the sample program the number of pulses is sampled 32 times. Each time the average of the sampled values is calculated by the INTTA1 interrupt routine. Figure 3.2.14 illustrates how the average of the sampled data is calculated.

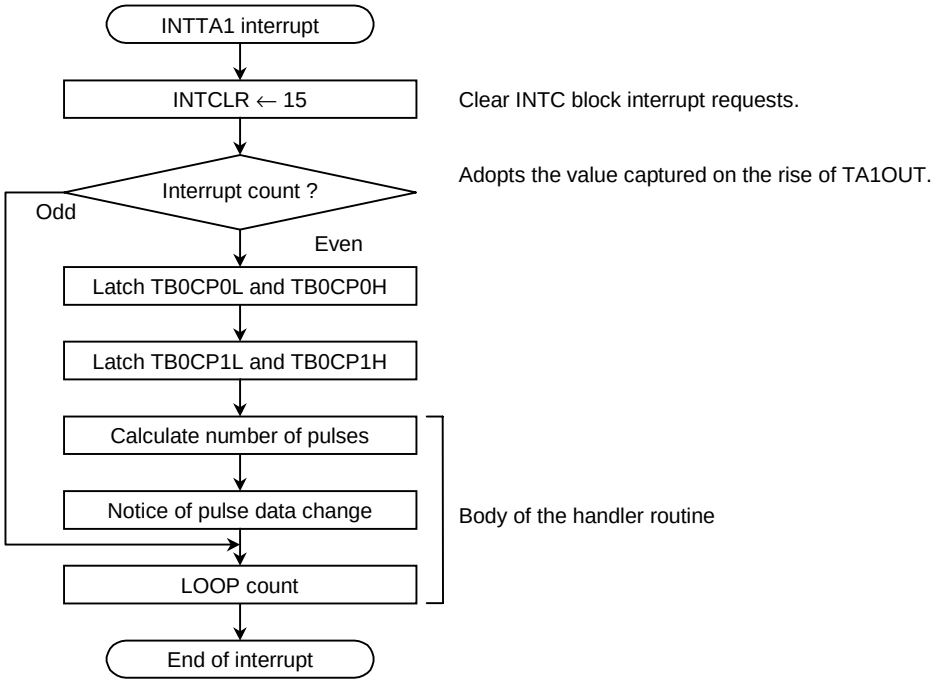


Figure 3.2.13 INTTA1 control flowchart

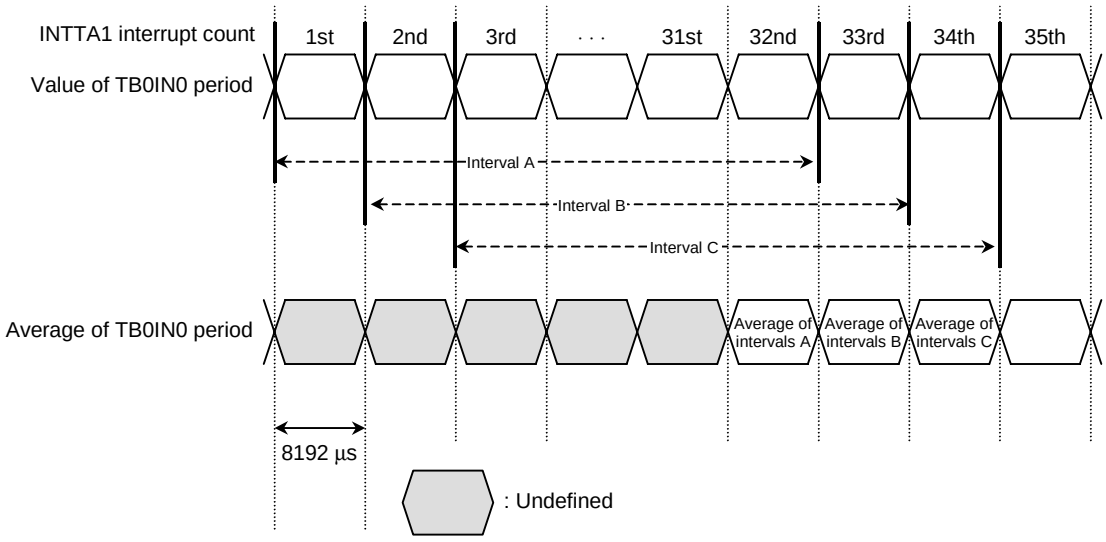


Figure 3.2.14 Method for calculating average value

3.2.8 Controlling the 2-ms Interval Timer

3.2.8.1 Overview of the 2-ms Interval Timer

- Generates a repeated interrupt with a period of 2 ms.
- Uses the 16-Bit Timer (TMRB2).
- Sets the interval time at which to generate INTTB21 in the Timer Register TB2RG1.

3.2.8.2 Control method for 2-ms Interval Timer

		7	6	5	4	3	2	1	0	
TB2RUN	←	0	0	0	0	0	0	0	0	Stop TMRB2.
TB2FFCR	←	1	1	0	0	0	0	1	1	Disable trigger.
TB2MOD	←	0	0	1	0	0	1	1	1	Select prescaler output clock for input clock and disable capture feature.
TB2RG1L	←	0	0	0	0	0	0	0	1	Set interval time.
TB2RG1H	←	1	1	1	1	0	1	0	0	Set TB2RG1 to 2000 μs ÷ φT16 = 500.
IMC8L	←	0x33--								
TB2RUN	←	0	0	X	X	–	1	X	1	Start TMRB0.

3.2.9 Sample programs

3.2.9.1 Generic flowchart

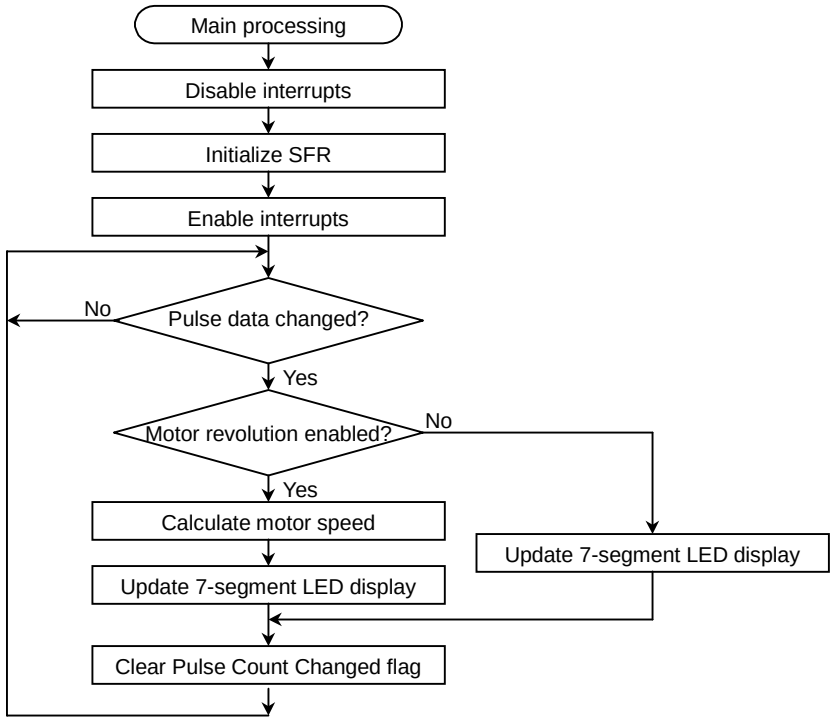


Figure 3.2.15 Main processing

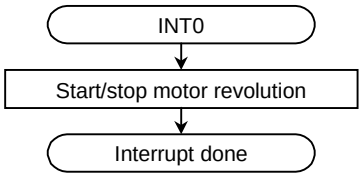


Figure 3.2.16 INT0 interrupt processing

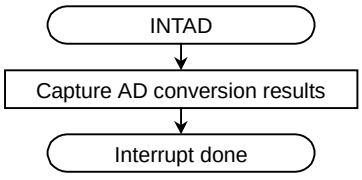


Figure 3.2.17 INTAD interrupt processing

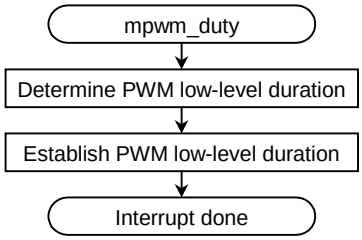


Figure 3.2.18 Determining the PWM duty cycle

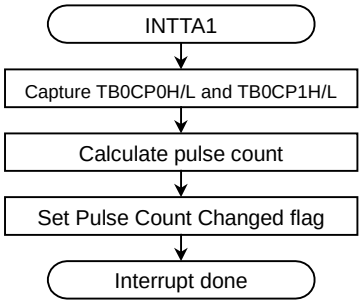


Figure 3.2.19 INTTA1 interrupt processing

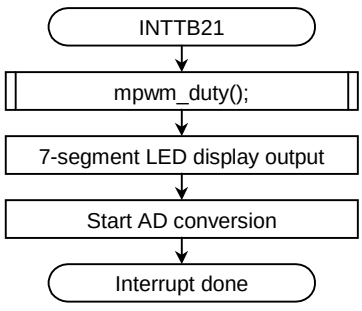


Figure 3.2.20 INTTB21 interrupt processing

3.2.9.2 File configuration

The configuration of files used in the sample program is shown in Table 3.2.3.

Table 3.2.3 File configuration

Filename	Contents	Page Reference
Stc9i16_dcm.asm	Start-up routine	3-12
io1940.c	Defines special function registers	3-51
dcmotor_drv.c	DC motor drive main processing	3-70
intrpt_dcmotor.c	Interrupt processing	3-73
func_dcmotor.c	Processing for various functions	3-75
io1940.h	Declares special function registers	3-28
ram_dcmotor.h	Declares external variables	3-77
led_dcmotor.h	Defines constants	3-78

3.2.9.3 Vector table

The vector table used in the sample programs is shown below. Replace the vector table section in the start-up routine with the vector table given below.

```
_VecTable:
      dw      _Int_dummy      ;0 --- software set
      dw      _mint0          ;1 --- INT[0]
      dw      _Int_dummy      ;2 --- INT[1]
      dw      _Int_dummy      ;3 --- INT[2]
      dw      _Int_dummy      ;4 --- INT[3]
      dw      _Int_dummy      ;5 --- INT[4]
      dw      _Int_dummy      ;6 --- *
      dw      _Int_dummy      ;7 --- *
      dw      _Int_dummy      ;8 --- *
      dw      _Int_dummy      ;9 --- *
      dw      _Int_dummy      ;10--- INT[5]
      dw      _Int_dummy      ;11--- INT[6]
      dw      _Int_dummy      ;12--- INT[7]
      dw      _Int_dummy      ;13--- INT[8]
      dw      _Int_dummy      ;14--- INT[9]
      dw      _Int_dummy      ;15--- INT[A]
      dw      _Int_dummy      ;16--- *
      dw      _Int_dummy      ;17--- *
      dw      _Int_dummy      ;18--- *
      dw      _Int_dummy      ;19--- *
      dw      _Int_dummy      ;20--- INTTA0 : 8bit Timer 0
      dw      _mintta1        ;21--- INTTA1 : 8bit Timer 1
      dw      _Int_dummy      ;22--- INTTA2 : 8bit Timer 2
      dw      _Int_dummy      ;23--- INTTA3 : 8bit Timer 3
      dw      _Int_dummy      ;24--- *
      dw      _Int_dummy      ;25--- *
      dw      _Int_dummy      ;26--- *
      dw      _Int_dummy      ;27--- *
      dw      _Int_dummy      ;28--- INTTB00 :16bit Timer 0 (TB0RG0)
      dw      _Int_dummy      ;29--- INTTB01 :16bit Timer 0 (TB0RG1)
      dw      _Int_dummy      ;30--- INTTB10 :16bit Timer 1 (TB1RG0)
      dw      _Int_dummy      ;31--- INTTB11 :16bit Timer 1 (TB1RG1)
      dw      _Int_dummy      ;32--- INTTB20 :16bit Timer 2 (TB2RG0)
      dw      _minttb21       ;33--- INTTB21 :16bit Timer 2 (TB2RG1)
      dw      _Int_dummy      ;34--- INTTN30 :16bit Timer 3 (TB3RG0)
      dw      _Int_dummy      ;35--- INTTB31 :16bit Timer 3 (TB3RG1)
      dw      _Int_dummy      ;36--- *
      dw      _Int_dummy      ;37--- *
      dw      _Int_dummy      ;38--- *
      dw      _Int_dummy      ;39--- *
      dw      _Int_dummy      ;40--- INTTBOF0:16bit Timer 0 (OverFlow)
```

```
dw      _Int_dummy      ;41--- INTTBOF1:16bit Timer 1 (OverFlow)
dw      _Int_dummy      ;42--- INTTBOF2:16bit Timer 2 (OverFlow)
dw      _Int_dummy      ;43--- INTTBOF3:16bit Timer 3 (OverFlow)
dw      _Int_dummy      ;44--- *
dw      _Int_dummy      ;45--- *
dw      _Int_dummy      ;46--- *
dw      _Int_dummy      ;47--- *
dw      _Int_dummy      ;48--- INTRX0 :Serial receive (channel.0)
dw      _Int_dummy      ;49--- INTTX0 :Serial transmit (channel.0)
dw      _Int_dummy      ;50--- INTRX1 :Serial receive (channel.1)
dw      _Int_dummy      ;51--- INTTX1 :Serial transmit (channel.1)
dw      _Int_dummy      ;52--- INTS2  :Serial Channel 2 interrupt
dw      _Int_dummy      ;53--- *
dw      _Int_dummy      ;54--- INTRX3 :Serial receive (channel.3)
dw      _Int_dummy      ;55--- INTTX3 :Serial transmit (channel.3)
dw      _Int_dummy      ;56--- INTRX4 :Serial receive (channel.4)
dw      _Int_dummy      ;57--- INTTX4 :Serial transmit (channel.4)
dw      _Int_dummy      ;58--- INTRTC :Timer for RTC interrupt
dw      _mintad         ;59--- INTAD  :AD conversion finished
dw      _Int_dummy      ;60--- INTDMA0:DMA transfer finished (channel.0)
dw      _Int_dummy      ;61--- INTDMA1:DMA transfer finished (channel.1)
dw      _Int_dummy      ;62--- INTDMA2:DMA transfer finished (channel.2)
dw      _Int_dummy      ;63--- INTDMA3:DMA transfer finished (channel.3)
```

3.2.9.4 Source code

- Filename: dcmotor_drv.c

```
/*
        ************************************************
        **              Application Note              **
        **                                            **
        **              ( MAIN ROUTINE )              **
        **                                            **
        **                   MCU                      **
        **                  TX1940                    **
        **                                            **
        **              fc      = 32MHz               **
        **              fsys    = 32MHz               **
        **              fperiph = 32MHz               **
        **                                            **
        **                                            **
        ************************************************
        ************************************************
        *      COPYRIGHT(C) 1999 TOSHIBA CORPORATION     *
        *              ALL RIGHTS RESERVED               *
        ************************************************/

/***********************************************/
/*              Loading header file            */
/***********************************************/
#include "io1940.h"
#include "ram_dcmotor.h"
#include "led_dcmotor.h"
#include <stdlib.h>


/*****************************************************************************/
/* Module name  :                                                            */
/*****************************************************************************/
/* Function     :Initializes DC motor drive mode.                            */
/* Input        :None                                                        */
/* Output       :SFR                                                         */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/*****************************************************************************/
void mdcmoter_init(void){

/*--- Setting CG ---*/
      IO_SYSCR0  = 0xf0;   /* <XEN>=1 <XTEN>=1 <RXEN>=1 <RXTEN>=1 */
                           /* <RSYSCR>=0 <WUEF>=0 <PRCK>=00 */
      IO_SYSCR1  = 0x10;   /* <SYSCR>=0 <FPSEL>=1 <DF0SC>=0 <GEAR>=0 */
      IO_ADCCLK  = 0x01;   /* <ADCCLK>=01 */

/*--- Setting I/O ports ---*/
      IO_P0      = 0xff;   /* Output High on P00-P07 */
      IO_P0CR    = 0xff;   /* Set P00-P07 for output */

      IO_P1      = 0xff;   /* Output High on P10-P17 */
      IO_P1CR    = 0xff;   /* Set P10-P17 for output */
      IO_P1FC    = 0x00;   /* Set P10-P17 to be a port */

      IO_P2      = 0xff;   /* Output High on P20-P27 */
      IO_P2CR    = 0xff;   /* Set P20-P27 for output */
      IO_P2FC    = 0x00;   /* Set P20-P27 to be a port */

      IO_P3      = 0xff;   /* Output High on P30-P31 and set P32-P37 for pull-up */
      IO_P3CR    = 0x00;   /* Set P30-P31 for output and P32-P37 for input */
      IO_P3FC    = 0x00;   /* Set P30-P37 to be a port */
```

```
        IO_P4      = 0x1f;   /* Set P40-P43 for pull-up and output High on P44 */
        IO_P4CR    = 0x10;   /* Set P40-P43 for input and P44 for output */
        IO_P4FC    = 0x00;   /* Set P40-P44 to be a port */


        IO_P7      = 0x65;   /* Output High on P70,P72,P75,P76 */
        IO_P7CR    = 0x67;   /* Set P70,P71,P72,P75,P76 for output and P73,P74,P77 for
                                input */
        IO_P7FC    = 0x00;   /* Set P70-P77 to be a port */

        IO_P7      = 0x6d;   /* Output High on P76,P75,P73,P72,P70 */
        IO_P7CR    = 0x6f;   /* Set P76,P75,P74,P73,P72,P71,P70 for output and P77,P74 for
                                input */
        IO_P7FC    = 0x98;   /* Use P74 as TB0IN0 and P73 as TA3OUT; set others to be ports */

        IO_P8      = 0xff;   /* Output High on P87-P80 */
        IO_P8CR    = 0xff;   /* Set P87-P80 for output */
        IO_P8FC    = 0x00;   /* Set P87-P80 to be a port */

        IO_P9      = 0x3f;   /* Output Low on P96-P97 */
        IO_P9CR    = 0xc0;   /* Set P90-P95 for input and P96-P97 for output */
        IO_P9FC    = 0x00;   /* Set P90-P97 to be a port */

        IO_PA      = 0x1f;   /* Output High on PA4-PA0 */
        IO_PACR    = 0x3f;   /* Set PA5-PA0 for output and PA7-PA6 for input */
        IO_PAFC    = 0x00;   /* Set PA7-PA0 to be a port*/

    /*--- INT0 switch ---*/
        IO_IMCGA0  = 0x21;   /* <EMCG>=10 <INT0EN>=1 */ /* falling edge */
        IO_IMC0L   = 0x1600; /* High level, interrupt level 6 */
        IO_INTCLR  = 0x01;   /* Interrupt request clear */

    /*--- Settings for AD conversion ---*/
        IO_ADMOD0  = 0x00;   /* <ITM0>=0 <REPEAT>=0 <SCAN>=0 <ADS>=0 */
        IO_ADMOD1  = 0x83;   /* <VREFN>=1 <12AD>=0 <ADTREG>=0 <ADCH2:0>=011 */
        IO_IMCEH   = 0x3330; /* Rising edge, interrupt level 3 */

    /*--- 2-ms timer settings ---*/
    /* TMRB2 */
        IO_TB2RUN  = 0x00;   /* <TB2RDE>=0 <I2TB2>=0 <TB2PRUN>=0 <TB2RUN>=0 */
        IO_TB2FFCR = 0xc3;   /* <TB2C1T1>=0 <TB2C0T1>=0 <TB2E1T1>=0 */
                             /* <TB2E0T1>=0 <TB2FF0C>=11 */
        IO_TB2MOD  = 0x27;   /* <TB2CP0I>=1 <TB2CPM>=00 <TB2CLE>=1 <TB2CLK>=11 */
                             /* φT16=4.0usec */
        IO_TB2RG1L = 0xf4;   /* TB2RG1 = 2000/4 */
        IO_TB2RG1H = 0x01;   /*                 */
        IO_IMC8L   = 0x3530; /* Rising edge, interrupt level 5 */

    /*--- PWM settings ---*/
    /* TMRA2 */
        IO_TA23RUN = 0x00;   /* <TA2RDE>=0 <I2TA23>=0 <TA23PRUN>=0 <TA2RUN>=0 */
        IO_TA23MOD = 0xe1;   /* PWM frequency 31.5 kHz, CLK:φT1(0.25usec) */
        IO_TA2REG  = 1;
        IO_TA3FFCR = 0x04;   /* <TAFF3C>=01 <TAFF3IE>=0 <TAFF3IS>=0 */
                             /* Set enable by INT0 */

    /*--- Settings for capture feature ---*/
    /* TMRA1 */
        IO_TA01RUN = 0x00;   /* <TA0RDE>=0 <I2TA01>=0 <TA01PRUN>=0 */
                             /* <TA1RUN>=0 <TA0RUN>=0 */
        IO_TA01MOD = 0x0c;   /* <TA01M>=00 <PWM>=00 <TA1CLK>=11 <TA1CLK>=00 */
        IO_TA1FFCR = 0x0b;   /* <TAFF1C>=10 <TAFF1IE>=1 <TA1FF1IS>=1 */
        IO_TA1REG  = 128;    /* Generate interrupt approximately every about 8192 us */
        IO_IMC5L   = 0x3430; /* Rising edge, interrupt level 4 */
    /* TMRB0 */
```

```
	IO_TB0RUN  = 0x00;   /* <TB0RDE>=0 <I2TB0>=0 <TB0PRUN>=0 <TB0RUN>=0 */
	IO_TB0MOD  = 0x38;   /* <TB0CP0I>=1 <TB0CPM>=11 <TB0CLE>=0 <TB0CLK>=00 */
	                     /* Capture timing=TA1OU,T CLK=TB0IN0 */

/*--- Timer start ---*/
	IO_TA01RUN = 0x06;   /* <TA0RDE>=0 <I2TA01>=0 <TA01PRUN>=1 <TA1RUN>=1 <TA0RUN>=0 */
	IO_TB0RUN  = 0x05;   /* <TB0RDE>=0 <I2TB0>=0 <TB0PRUN>=1 <TB0RUN>=1 */
	IO_TB2RUN  = 0x05;   /* <TB2RDE>=0 <I2TB2>=0 <TB2PRUN>=1 <TB2RUN>=1 */

}


/*************************************************************************/
/* Module name   :main                                                   */
/*************************************************************************/
/* Function      :Main processing                                        */
/* Input         :None                                                   */
/* Output        :None                                                   */
/* Parameters    :None                                                   */
/* Return value  :None                                                   */
/*************************************************************************/
void main(void){
	__DI();
/*--- Initialize SFR ---*/
	mdcmoter_init();
	__EI();

	for(;;){
		switch(frpm_disp){
		case  0:
			break;
		case  1:
			/*--- Change 7 segment display data(revolution rate) ---*/
			switch(fmotor_rot_able){
			/*--- 7 segment LED display data during revolution of motor ---*/
			case   DRV_ENABLE:

				/*--- Calculate revolution rate of motor ---*/
				mrpm_calc();

				g7seg_data[3] = t7seg_ch[grpm3];
				g7seg_data[2] = t7seg_ch[grpm2];
				g7seg_data[1] = t7seg_ch[grpm1];
				g7seg_data[0] = t7seg_ch[grpm0];
				break;
			/*--- 7 segment LED display data during stop motor ---*/
			case   DRV_DISABLE:
				g7seg_data[3] = c7seg_S;
				g7seg_data[2] = c7seg_t;
				g7seg_data[1] = c7seg_o;
				g7seg_data[0] = c7seg_p;
				break;
			}
			frpm_disp = 0;
			break;
		}
	}
}
```

- Filename: intrpt_dcmotor.c

```
/*
        ************************************************
        **             Application Note               **
        **                                            **
        **           ( INTERRUPT ROUTINE )            **
        **                                            **
        **                  MCU                       **
        **                TX1940                      **
        **                                            **
        **             fc      = 32MHz                **
        **             fsys    = 32MHz                **
        **             fperiph = 32MHz                **
        **                                            **
        **                                            **
        ************************************************
        ************************************************
        *      COPYRIGHT(C) 1999 TOSHIBA CORPORATION     *
        *               ALL RIGHTS RESERVED              *
        ************************************************/

/***********************************************/
/*             Loading header file             */
/***********************************************/
#include "io1940.h"
#include "ram_dcmotor.h"


/***********************************************/
/*             Constant definitions            */
/***********************************************/
unsigned char    fmotor_rot_able = DRV_DISABLE;  /* Motor Revolution Enable flag */
unsigned short   gad3_data;          /* Store AN3 conversion result */
unsigned short   gcp0, gcp1;         /* Store captured value */
unsigned char    frpm_disp = 0;      /* Flag of 7SEG-LED display Enable */


/*****************************************************************************/
/* Module name  :mintad                                                      */
/*****************************************************************************/
/* Function     :Generate interrupt on completion of AD conversion. (interrupt)*/
/* Input        :ADREG37H,ADREG37L                                           */
/* Output       :INTCLR,gad3_data                                            */
/*****************************************************************************/
void __interrupt mintad(void){

      IO_INTCLR = 0x3b;
      gad3_data = (IO_ADREG37H<<2)+(IO_ADREG37L>>6);    /* Capture value of AN3 */

}


/*****************************************************************************/
/* Module name   :minttb21                                                   */
/*****************************************************************************/
/* Function      :2-ms Interval Timer (interrupt)                            */
/* Input         :None                                                       */
/* Output        :INTCLR,ADMOD0                                              */
/*****************************************************************************/
void __interrupt minttb21(void){

      IO_INTCLR = 0x21;
      mpwm_duty();
      p7seg_disp();
```

```
      IO_ADMOD0 = 0x01;             /* <ITM0>=0 <REPEAT>=0 <SCAN>=0 <ADS>=1 */
}


/*************************************************************************/
/* Module name   :mintta1                                                */
/*************************************************************************/
/* Function      :INTTA1 interrupt (interrupt) capture                   */
/* Input         :TB0CP0L,TB0CP0H,TB0CP1L,TB0CP1H                        */
/* Output        :INTCLR,gcp0,gcp1,frpm_disp                             */
/*************************************************************************/
void __interrupt mintta1(void){

      unsigned char cp0l,cp0h,cp1l,cp1h;
      static unsigned int  loop_count = 0;

      IO_INTCLR = 0x15;
      if( loop_count%2 != 0 ){
           cp0l      = IO_TB0CP0L;
           cp0h      = IO_TB0CP0H;
           cp1l      = IO_TB0CP1L;
           cp1h      = IO_TB0CP1H;
           gcp0      = (cp0h<<8)+cp0l;          /* Store Capture Register value */
           gcp1      = (cp1h<<8)+cp1l;          /* Store Capture Register value */
           frpm_disp = 1 ;
      }
      loop_count++;
}


/*************************************************************************/
/* Module name   :mint0                                                  */
/*************************************************************************/
/* Function      :INT0 interrupt (external interrupt)                    */
/* Input         :None                                                   */
/* Output        :EICRCG,INTCLR,TA23RUN,TA3FFCR, fmotor_rot_able         */
/*************************************************************************/
void __interrupt mint0(void){

      IO_EICRCG  = 0x00;
      IO_INTCLR  = 0x01;
      IO_TA23RUN = 0x80;            /* <TA2RDE>=1 <I2TA23>=0 <TA23PRUN>=0 */
                                    /* <TA3RUN>=0 <TA2RUN>=0 */
      switch(fmotor_rot_able){
/*--- Start motor revolution---*/
      case    DRV_DISABLE:
              IO_TA3FFCR = 0x0a;    /* <TAFF3C>=10 <TAFF3IE>=1 <TAFF3IS>=0 */
              IO_TA23RUN = 0x85;    /* <TA2RDE>=1 <I2TA23>=0 <TA23PRUN>=1  */
                                    /* <TA3RUN>=0 <TA2RUN>=1 */
              fmotor_rot_able = DRV_ENABLE;     /* Set Enable flag */
              break;
/*--- Stop motor revolution ---*/
      case    DRV_ENABLE:
              IO_TA3FFCR = 0x04;    /* <TAFF3C>=01 <TAFF3IE>=0 <TAFF3IS>=0 */
              fmotor_rot_able = DRV_DISABLE;    /* Prohibit Enable flag */
              break;
      }

}
```

- Filename: func_dcmotor.c

```
/*
        ************************************************
        **              Application Note              **
        **                                            **
        **             ( FUNCTION ROUTINE )           **
        **                                            **
        **                    MCU                     **
        **                  TX1940                    **
        **                                            **
        **              fc      = 32MHz               **
        **              fsys    = 32MHz               **
        **              fperiph = 32MHz               **
        **                                            **
        **                                            **
        ************************************************
        ************************************************
        *      COPYRIGHT(C) 1999 TOSHIBA CORPORATION     *
        *             ALL RIGHTS RESERVED                *
        ************************************************/

/***********************************************/
/*             Loading header file             */
/***********************************************/
#include "io1940.h"
#include "ram_dcmotor.h"
#include "led_dcmotor.h"

/***********************************************/
/*            Constant definitions             */
/***********************************************/
/*--- Definitions of port output (7SEG-LED digit) ---*/
#define  c7seg_digit_out0    0x1e    /* PA (00011110) */
#define  c7seg_digit_out1    0x1d    /* PA (00011101) */
#define  c7seg_digit_out2    0x1b    /* PA (00011011) */
#define  c7seg_digit_out3    0x17    /* PA (00010111) */
#define  c7seg_digit_out4    0x0f    /* PA (00001111) */
/*--- Number of buffer array ---*/
#define  LIMIT          32

/*--- Digit count ---*/
#define  digit(pos)          ((pos+1)%5)
/*--- Count of buffer number ---*/
#define  next(pos)           ((pos+1)%LIMIT)


/***********************************************/
/*         External variable definitions       */
/***********************************************/
unsigned char   g7seg_data[4];                  /* 7SEG-LED display data */
unsigned char   gled_data = cled_all_off;       /* LED display data */
unsigned char   grpm0,grpm1,grpm2,grpm3;        /* store data of motor revolution number */


/*************************************************************************/
/* Module name  :p7seg_disp                                              */
/*************************************************************************/
/* Function     :7-segment LED display output (using 2-ms interrupt)     */
/* Input        :gled_data,g7seg_data                                    */
/* Output       :P8,PA                                                   */
/* Parameters   :None                                                    */
/* Return value :None                                                    */
/*************************************************************************/
void p7seg_disp(void){
```

```
	static unsigned char ldisp_digit = 0;

/*--- Turning display off ---*/
	IO_P8 = 0xff;

/*--- Display output ---*/
	switch (ldisp_digit) {
	case 0:
		IO_PA = c7seg_digit_out0;		/* Change digits */
		IO_P8 = gled_data;			/* Output LED */
		break;
	case 1:
		IO_PA = c7seg_digit_out1;		/* Change digits */
		IO_P8 = g7seg_data[0];			/* Output 7 segment LED */
		break;
	case 2:
		IO_PA = c7seg_digit_out2;		/* Change digits */
		IO_P8 = g7seg_data[1];			/* Output 7 segment LED */
		break;
	case 3:
		IO_PA = c7seg_digit_out3;		/* Change digits */
		IO_P8 = g7seg_data[2];			/* Output 7 segment LED */
		break;
	case 4:
		IO_PA = c7seg_digit_out4;		/* Change digits */
		IO_P8 = g7seg_data[3];			/* Output 7 segment LED */
		break;
	}

/*--- Digit count processing ---*/
	ldisp_digit = digit(ldisp_digit);

}

/*********************************************/
/*             Display data table            */
/*********************************************/
/*--- 7SEG-LED ---*/
const unsigned char t7seg_ch[16] = {
	0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xd8,
	0x80,0x98,0x88,0x83,0xc6,0xa1,0x86,0x8e,
};


/*****************************************************************************/
/* Module name  :mpwm_duty                                                   */
/*****************************************************************************/
/* Function     :Determine PWM Low-level duration (using 2-ms interrupt)     */
/* Input        :gad3_data                                                   */
/* Output       :SFR,g7seg_data[]                                            */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/*****************************************************************************/
void mpwm_duty(void){				/* PWM frequency:31kHz */

	volatile unsigned char   pls_w;


	pls_w = (unsigned char)((gad3_data >> 3)); /* Setting with TR2REG<=126 required */
	if(pls_w == 0)		pls_w++;
	else if(pls_w == 127)	pls_w--;

	IO_TA2REG =		pls_w;
}
```

```
/*****************************************************************************/
/* Module name  :mrpm_disp                                                   */
/*****************************************************************************/
/* Function     :Outputs motor revolutions to 7-segment LEDs                 */
                 (using 2-ms interrupt)                                      */
/* Input        :gcp0,gcp1                                                   */
/* Output       :SFR,g7seg_data[]                                            */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/*****************************************************************************/
void mrpm_calc(void){

      unsigned int                  rpm;
      static unsigned int           sum = 0;
      static unsigned char          index = 0;
      static unsigned short         buffer[LIMIT];

      if(gcp1 >= gcp0){
      /*--- Storing number of pulses ---*/
             sum        -= buffer[index];
             buffer[index] = (gcp1-gcp0);
             sum        += buffer[index];

      /*--- Incrementing array index ---*/
             index = next(index);

      /*--- Determining motor revolution rate [rpm] ---*/
             /* Encoder output = 200(pluse/rev) */
             /* Number of buffers 32, 8192 us -> 18 buffers shifted  */
             rpm = (300000*sum) >> 18;

      /*--- Converting revolution rate [rpm] into decimal number ---*/
             grpm3 = (rpm/1000);               /* Calculate 4th digit */
             rpm  = rpm-(grpm3*1000);
             grpm2 = (rpm/100);                /* Calculate 3rd digit */
             rpm  = rpm-(grpm2*100);
             grpm1 = (rpm/10);                 /* Calculate second digit */
             grpm0 = (rpm%10);                 /* Calculate first digit  */
      }
}
```

- Filename: ram_dcmotor.h

```
extern      unsigned char       g7seg_data[4];
extern      unsigned char       gled_data;

extern      unsigned short      gcp0;
extern      unsigned short      gcp1;

extern      unsigned short      gad3;
extern      unsigned char       fenable;
```

- Filename: led_dcmotor.h

```
/*
        ************************************************
        **              TOSHIBA CORPORATION           **
        **                                            **
        **                ( LED HEADER )              **
        **                                            **
        **                     MCU                    **
        **                    TX1940                  **
        **                                            **
        **                                            **
        **                                            **
        ************************************************
        ************************************************
        *      COPYRIGHT(C) 1999 TOSHIBA CORPORATION     *
        *              ALL RIGHTS RESERVED               *
        ************************************************/

/*** Definition of 7-segment LEDs (Display characters) ***/
     #define   c7seg_S         0x92             /* S */
     #define   c7seg_t         0x87             /* t */
     #define   c7seg_o         0xa3             /* o */
     #define   c7seg_p         0x8c             /* P */

/*** Definition of LEDs ***/
     #define   cled_all_on     0x00             /* Turn all LEDs on */
     #define   cled_all_off    0xff             /* Turn all LEDs off */
```

3.3 E²PROM

One byte of data can be read from or written to the E2PROM using the I2C Bus Mode of the serial bus interface (SBI).

3.3.1 Specifications

3.3.1.1 Basic specifications

- Uses the I²C Bus Mode of the serial bus interface (SBI) for data transfer.
- Uses the matrix keys (0~F) and the AD conversion keys (Read and Write) for key input.
- Sounds the buzzer on each key input.
- Uses 7-segment LEDs to display addresses and data.

Table 3.3.1 Key list

Key Name	Type of Key	Key Function
0 ~ F	Matrix	Used for entry of addresses and data
Read	AD conversion	Executes read
Write	AD conversion	Executes write

3.3.1.2 Method for reading data from E²PROM

The following text explains how to read from the TB1940 board.

Step 1: Enter a two-digit address using the keys 0~F.
(The data is shifted left with each key press.)

Step 2: Press the Read key.

Step 3: The data at the specified address is displayed.

Step 4: Press the Read key or any of the keys 0~F to return to step 1.
(If the keys 0~F are used, the data entered becomes the new address.)

Example: Figure 3.3.1 shows how to read the data from address 04H.

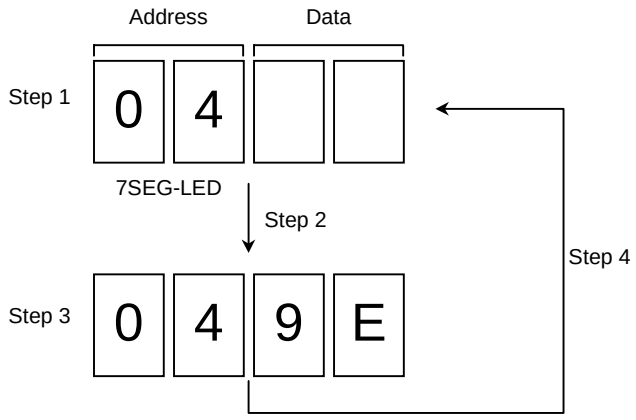


Figure 3.3.1 Procedure for reading data from the E²PROM

3.3.1.3 Method for writing data to the E^2PROM

The following text explains how to write to the TB1940 board.

Step 1: Enter a two-digit address using the keys 0~F.
(The data is shifted left with each key press.)

Step 2: Press the Write key. (The saved data is immediately displayed.)

Step 3: Enter two digits data using the keys 0~F.

Step 4: Press the Write key to return to step 1. (This saves the data to the E^2PROM.)

Example: Figure 3.3.2 shows how to write the data A5H to address 04H.

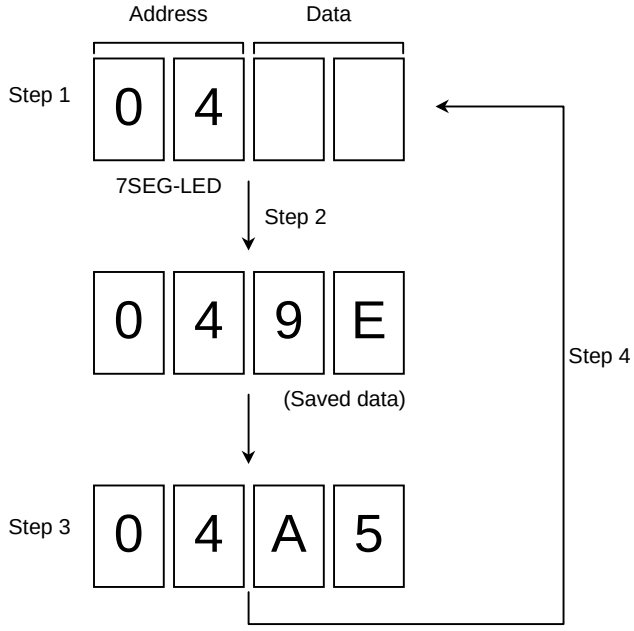


Figure 3.3.2 Procedure for writing data to the E^2PROM

3.3.2 Functional block diagram

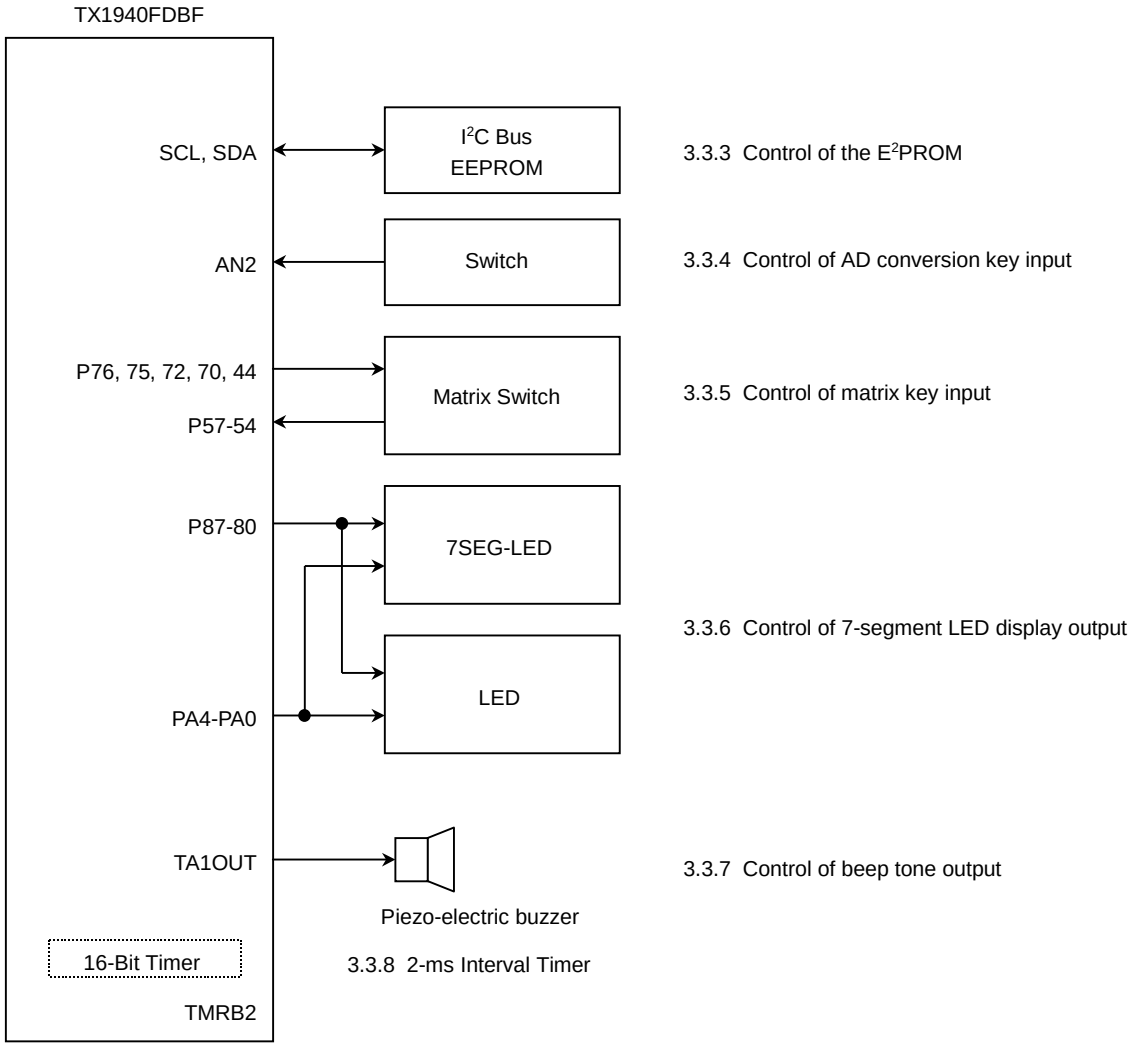


Figure 3.3.3 Functional block diagram

3.3.3 Control of the E²PROM

3.3.3.1 Overview of the E²PROM

- AC characteristics of the E²PROM

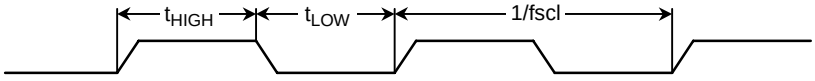
  Table 3.3.2 below shows the AC characteristics of the E²PROM.

Table 3.3.2 AC characteristics of E²PROM

Parameter	Symbol	Min	Max	Unit
SCL Clock Frequency	f_{SCL}		100	kHz
SDA and SCL Noise Cancel Time	T_I		100	ns
Bus Release Time Before Start of Transfer	t_{BUF}	4.7		μs
Start Condition Hold Time	$t_{HD:STA}$	4.0		μs
SCL Low-Level Time	t_{LOW}	4.7		μs
SCL High-Level Time	t_{HIGH}	4.0		μs
Start Condition Set-up Time	$t_{SU:STA}$	4.7		μs
Data Hold Time	$t_{HD:DAT}$	0		ns
Data Set-up Time	$t_{SU:DAT}$	250		ns
SDA and SCL Signal Rise Time	t_R		1	μs
SDA and SCL Signal Fall Time	t_F		300	ns
Stop Condition Set-up Time	$t_{SU:STO}$	4.7		μs
Output Data Hold Time	t_{DH}	300		ns

- Calculating the internal SCL clock frequency

  System clock: fc (= 32 MHz)
  Clock gear: fc/1
  ϕT0 = fperiph/4 (= 8 MHz)



$t_{LOW} = 2^n/\phi T0$

$t_{HIGH} = 2^n/\phi T0 + 4/\phi T0$

$fscl = 1/(t_{Low} + t_{HIGH})$

$$= \frac{\phi T0}{2^n + 4}$$

SBI0CR1<SCK2:0>	n	SCL Frequency
000	4	400 kHz
001	5	222 kHz
010	6	117 kHz
011	7	60.6 kHz
100	8	30.7 kHz
101	9	15.5 kHz
110	10	7.78 kHz
111		Reserved

Figure 3.3.4 Clock source

Since the maximum SCL clock frequency for the E²PROM is 100 kHz, the setting n = 7 (a clock frequency of 60.6 kHz) is used in these sample programs.

■ Setting slave addresses

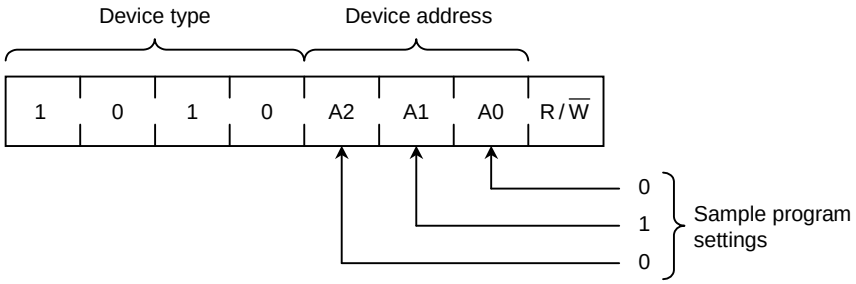
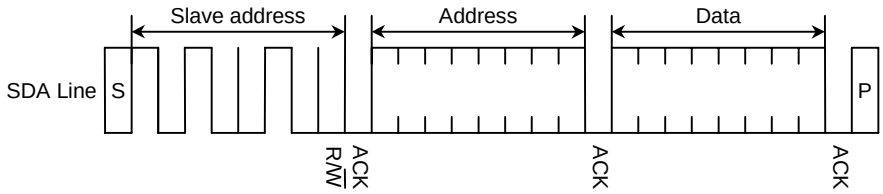


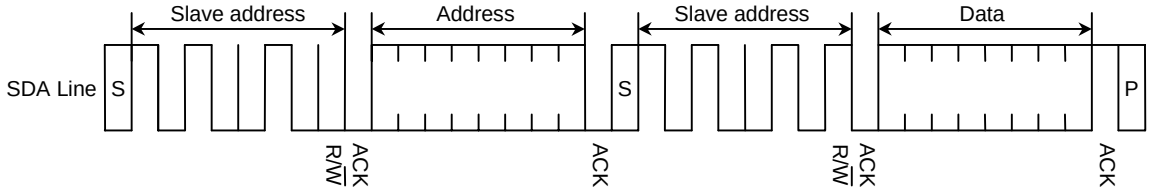
Figure 3.3.5 Setting slave addresses

■ I^2C bus data format

(a) 1-byte data write



(b) 1-byte data read



Note: S = start condition

R / $\overline{W}$ = Direction bit

ACK = Acknowledge bit

P = stop condition

Figure 3.3.6 I^2C bus data format used with the E^2PROM

3.3.3.2 E²PROM control method

■ Initial settings

		7	6	5	4	3	2	1	0	
PACR	←	1	1	-	-	-	-	-	-	Set PA6 to be SDA.
PAFC	←	1	1	-	-	-	-	-	-	Set PA7 to be SCL.
ODE	←	X	X	-	-	1	1	-	-	Set PA7 and PA6 to be open-drain.
IMCDL	←	X	X	1	1	0	0	0	1	Set INTS2 for rising edge and to level 1.

Note 1: Note: X denotes Don't care; "–" denotes No change.

Note 2: The interrupt level can be set to any desired value.

- Device initialization

		7	6	5	4	3	2	1	0	
SBI0BR0	←	X	0	X	X	X	X	X	X	Stop when idle.
SBI0BR1	←	1	X	X	X	X	X	X	X	Activate internal baud rate circuit.
SBI0CR1	←	0	0	0	1	X	0	1	1	Set SCL clock to 60.6 kHz and put interface in Acknowledgement Mode.
I2C0AR	←	0	0	0	0	0	0	0	0	Set to recognize slave address.
SBI0CR2	←	0	0	0	1	1	0	0	0	Set initial state to Slave Receiver Mode.

Note: X denotes Don't care.

■ Controlling writing to E²PROM

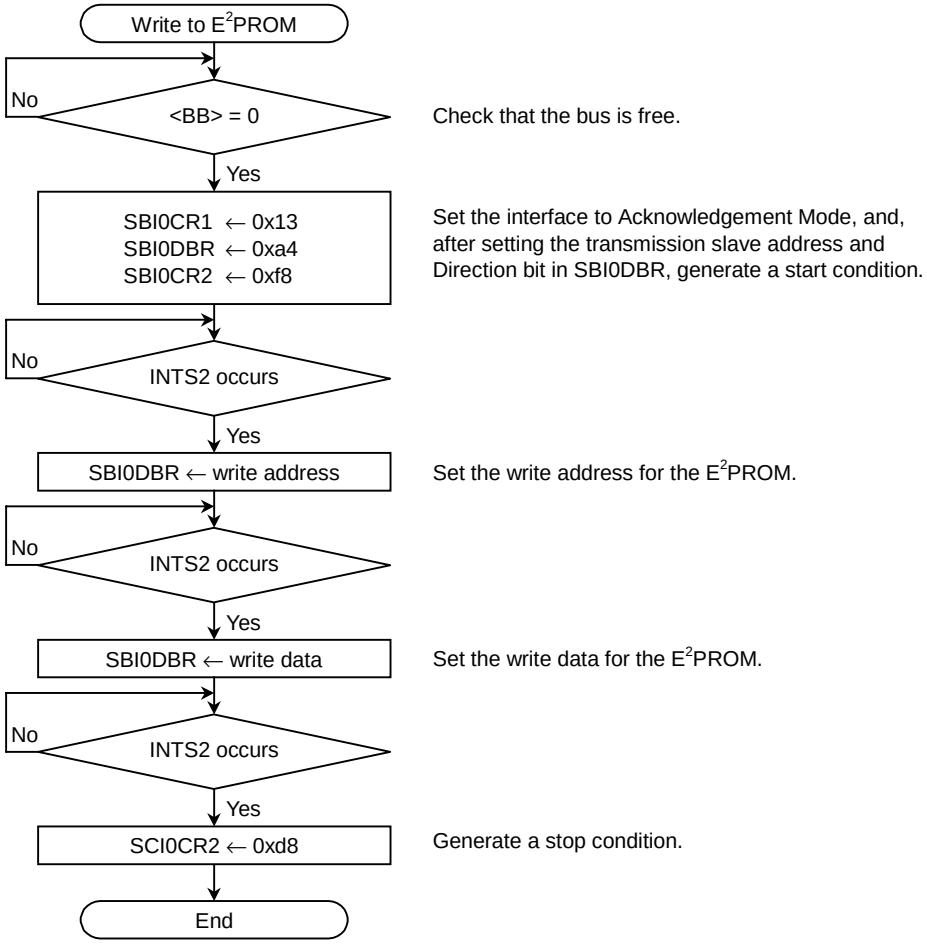
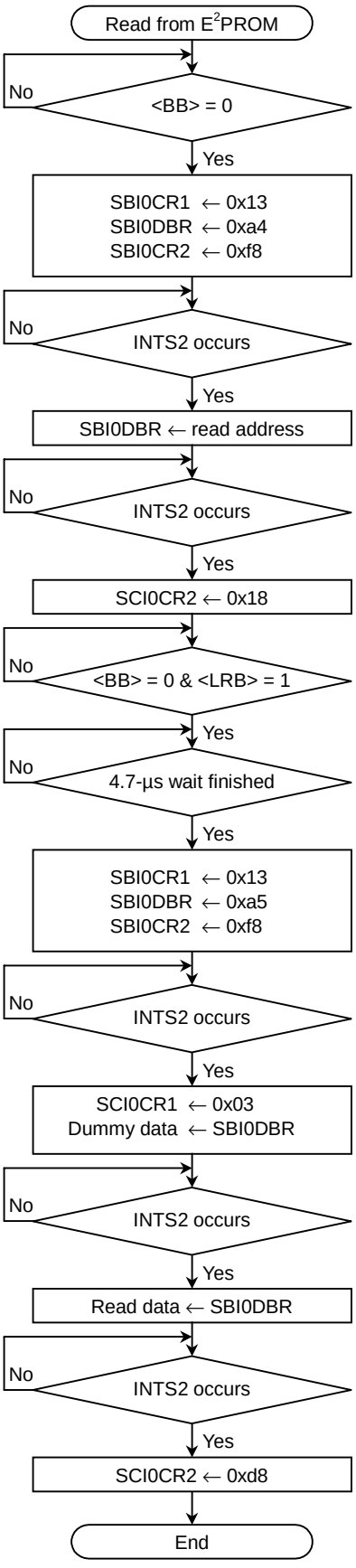


Figure 3.3.7 Operation flow for control of writing to the E²PROM

■ Controlling reading from the E^2PROM



Check that the bus is free.

Set the interface to Acknowledgement Mode, and after setting the transmission slave address and Direction bit in SBI0DBR, generate a start condition.

Set the read address for the E^2PROM.

Release the bus.

Check that the bus is free and that no other devices are pulling the SCL line Low.

Set the interface to Acknowledgement Mode, and after setting the transmission slave address and Direction bit in SBI0DBR, generate a start condition.

Exit Acknowledgement Mode and read the dummy data.

Store the read data.

Generate a stop condition.

Figure 3.3.8 Operation flow for control of reading from the E^2PROM

3.3.4 Control of AD conversion key input

3.3.4.1 Overview of AD conversion keys

- Determine whether the AD conversion keys are on or off by reading the voltage from each key.
- Eliminate ON-chattering for a 30-ms period and eliminate OFF-chattering for a 30-ms period.
- Set the resistance value for each resistor so that the voltages across each resistor are the same.
- If two or more keys are pressed simultaneously, the voltage of the key which is located closer to Vdd is read. (In the sample program, key number 5 is read.)

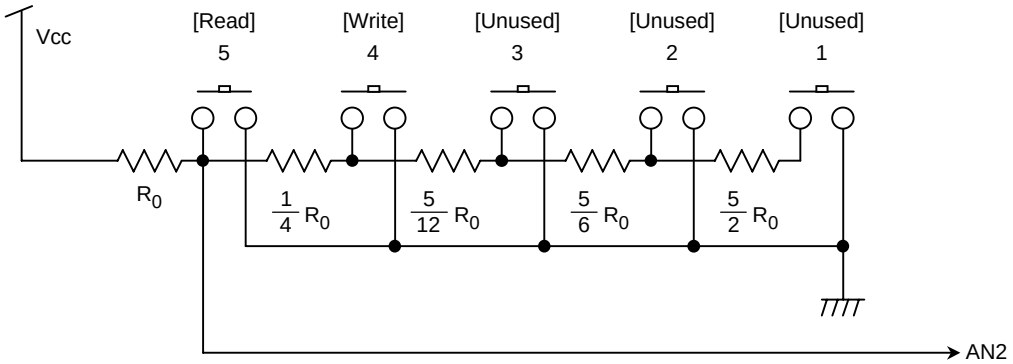


Figure 3.3.9 Key numbers

Table 3.3.3 Correspondence between AD-converted values and voltages

Key Number	Voltage when Key ON (logical value)	AD-Converted Value when Key ON (logical value)	AD-Converted Value of 8 High-Order Bits (logical value)
OFF	3.30 V	1023	255
1	2.65 V	821	205
2	1.99 V	616	154
3	1.34 V	415	103
4	0.67 V	207	51
5	0.00 V	0	0

Note 1: This applies when Vcc = 3.3 V.

Note 2: In the sample program only the eight high-order bits of the AD-converted value are used (the two low-order bits are discarded).

3.3.4.2 Method for controlling AD conversion key input

■ Initial settings

		7	6	5	4	3	2	1	0		
ADCCLK	←	X	X	X	X	X	X	0	1		Set AD conversion time to 8 µs (when fsys = 32 MHz).
ADMOD0	←	0	0	0	0	0	0	0	0		Select Fixed-Channel Single-Conversion Mode.
ADMOD1	←	1	0	X	X	0	0	1	0		Select Channel 2 and VREF as the voltage to apply.
IMCEH	←	X	X	-	-	-	-	-	-	L	Set INTAD for rising edge and to level 2.
		X	X	1	1	0	0	1	0	H	

Note 1: X denotes Don't care; "–" denotes No change.

Note 2: The interrupt level can be set to any desired value.

■ AD conversion key input processing

When an interrupt is generated on completion of AD conversion, the AD Conversion Result Register is read and the contents is converted into key data. Based on this converted key data, chattering-elimination processing is performed using a 2-ms Interval Timer interrupt.

The processing for each key is performed in the main routine only after it has been confirmed that chattering elimination has been completed. For details of the 2-ms Interval Timer, please refer to Section 3.3.8.

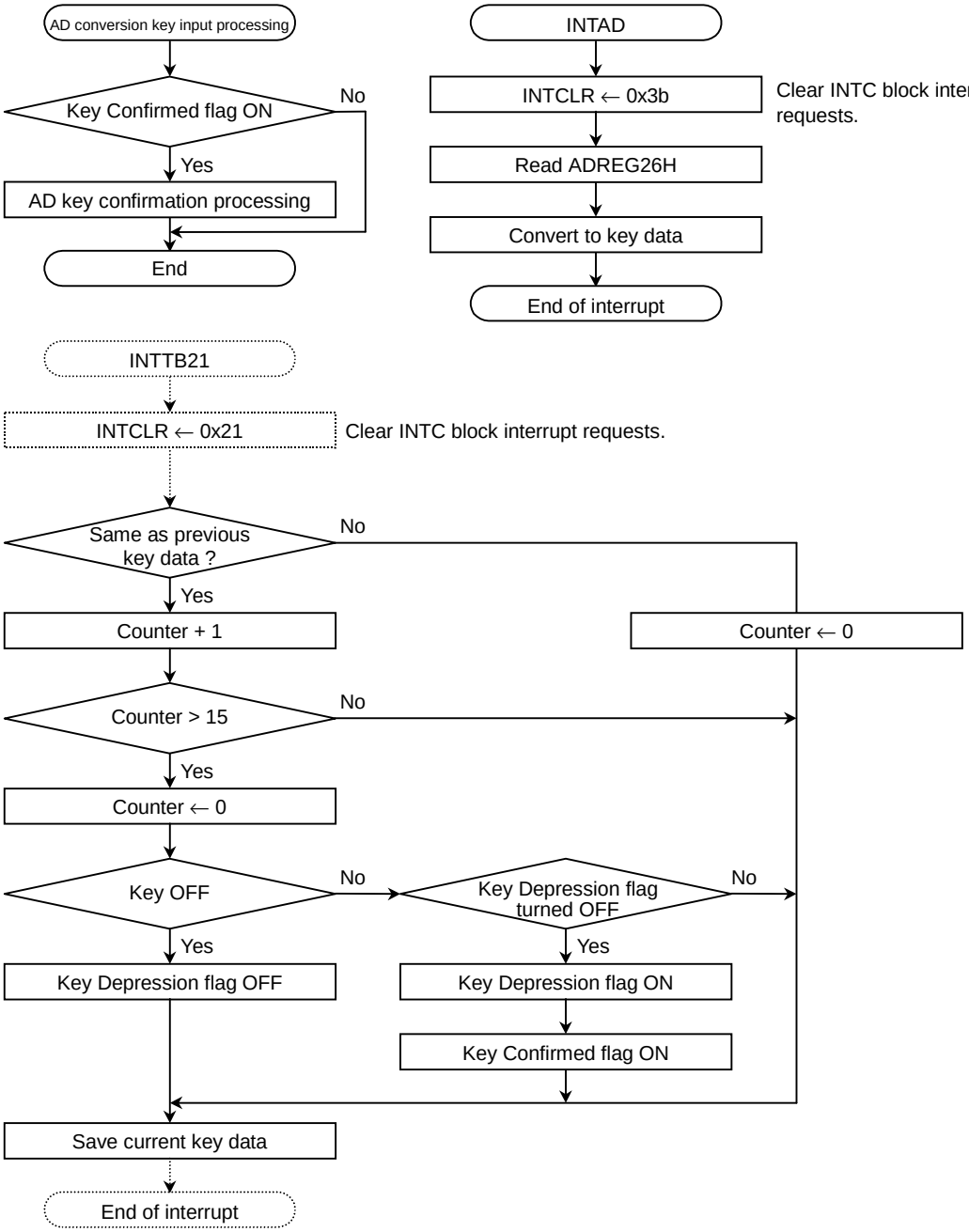


Figure 3.3.10 Operation flow for control of AD conversion key input

3.3.5 Control of matrix key input

3.3.5.1 Overview of matrix key input

- Key input data is updated every 2 ms.
- Does not perform simultaneous key depression processing.
- Does not perform DIP switch read processing.
- Allows up to 16 keys to be connected.

■ Digit output and key data input

The row which is outputting a Low signal to the digit ports (P70, P72, P75 and P76) is selected. The key input ports (P54~P57) change state according to the key status of the selected row. A key input port corresponding to a key which is depressed goes High. Key input ports corresponding to keys which are not depressed go Low.

Note: Ensure that the key input ports are read 10 µs or more after the digit output has changed. Avoid reading the key input ports immediately after changing the digit output, since this may not yield the correct value.

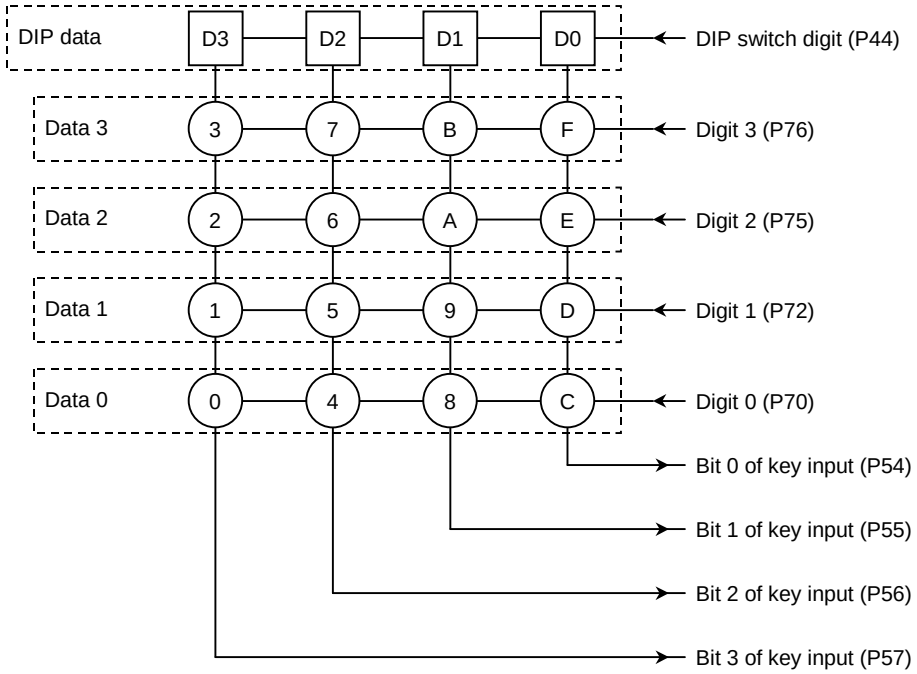
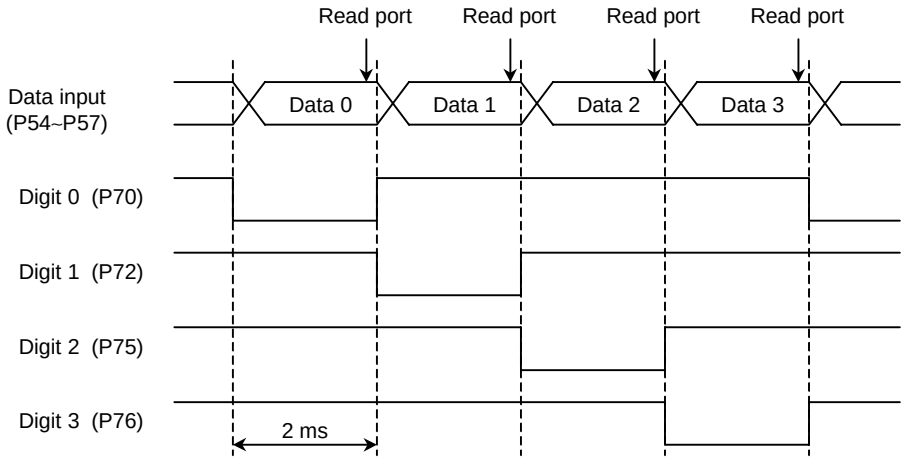


Figure 3.3.11 Relationship between digits and input data

In the sample program the key input ports are read at the beginning of a 2-ms interrupt; on completion of the reading the next digit row to be scanned outputs a Low signal. Processing key input in this way generates a wait time of at least 2 ms from the time at which the digits are changed to the time at which the key input ports are read. The data is always read correctly.

Figure 3.3.12 shows the timing chart for reading in data and changing the digits.



Note: All digit outputs are active-Low.

Figure 3.3.12 Timing chart for reading in data and changing the digits

Two bytes of memory space are allocated for key data. Each bit holds the key state (whether depressed or released). For example, if only key 1 is depressed, the 8th bit is set to 1 and all other bits are 0. When multiple keys are depressed simultaneously, all the corresponding bits are set to 1.

Table 3.3.4 shows the data structure of key data.

Table 3.3.4 Data structure of key data

		Key Data (16 Bits)																
		Data 3				Data 2				Data 1				Data 0				
Matrix Key Name	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0x0008
	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0x0080
	2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0x0800
	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x8000
	4	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0x0004
	5	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0x0040
	6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0x0400
	7	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x4000
	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0x0002
	9	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0x0020
	A	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0x0200
	B	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0x2000
	C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0x0001
	D	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0x0010
	E	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0x0100
	F	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0x1000
OFF		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x0000
Multiple depressions		*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	Other than the above

3.3.5.2 Method for controlling matrix key input

- Initial settings

In the initial setting none of the digit rows is selected.

		7	6	5	4	3	2	1	0	
P4	←	0	0	0	1	-	-	-	-	Set P44 output latch to 1.
P4CR	←	0	0	0	1	-	-	-	-	Set P44 for output.
P4FC	←	0	0	0	0	-	-	-	-	Set P44 for port.
P7	←	-	1	1	-	-	1	-	1	Set P70, P72, P75 and P76 output latches to 1.
P7CR	←	-	1	1	-	-	1	-	1	Set P70, P72, P75 and P76 for output.
P7FC	←	-	0	0	-	-	0	-	0	Set P70, P72, P75 and P76 to be ports.

Note: X denotes Don't care; "–" denotes No change.

- Matrix key input processing

After port input and digit update, chattering-elimination processing is performed using a 2-ms Interval Timer interrupt. The processing for each key is performed in the main routine only after it has been confirmed that chattering elimination has been completed. For details of the 2-ms Interval Timer, please refer to Section 3.3.8.

Figure 3.3.13 shows the flow for control of matrix key input.

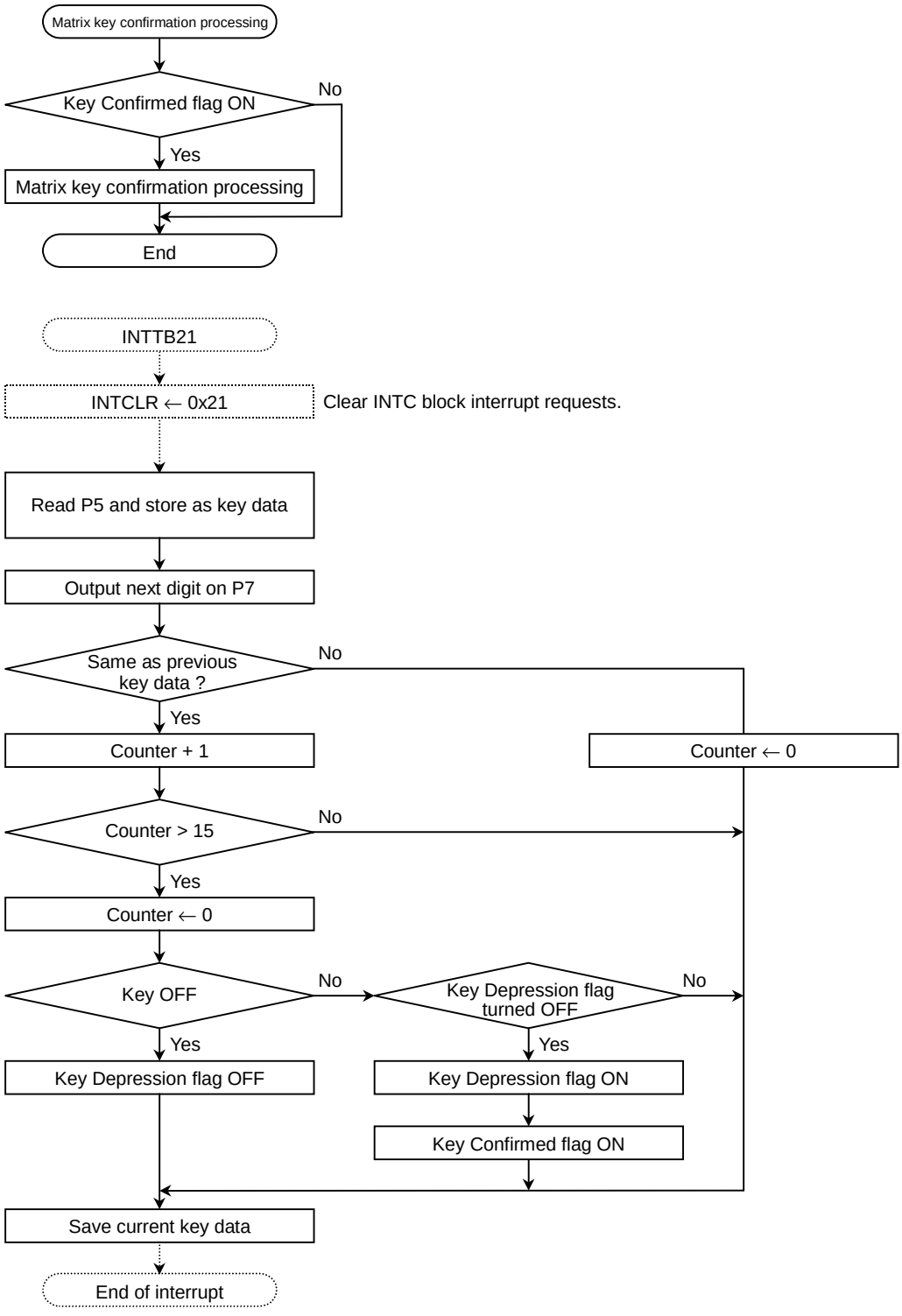


Figure 3.3.13 Operation flow for control of matrix key input

3.3.6 Control of 7-segment LED display output

3.3.6.1 Overview of 7-segment LED display

- The display data is dynamically displayed on the 7-segment LEDs and updated every 2 ms.

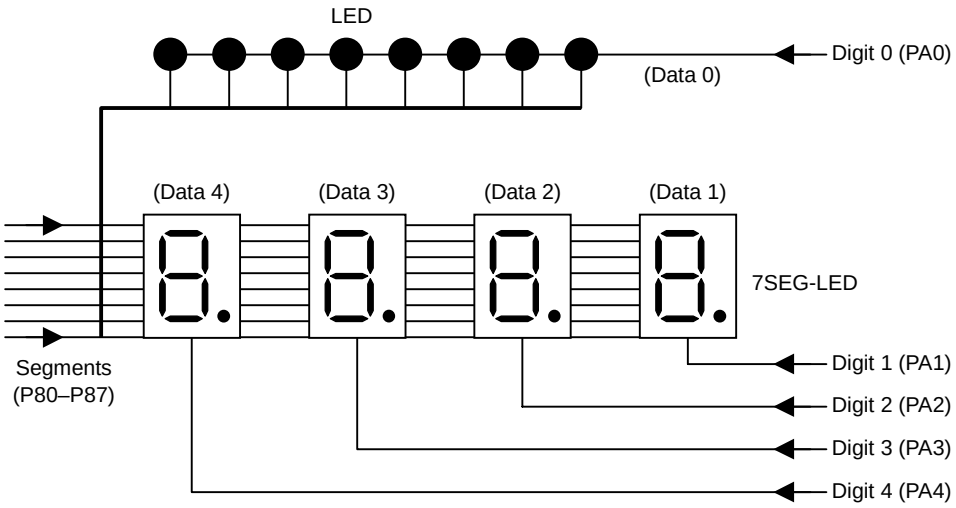
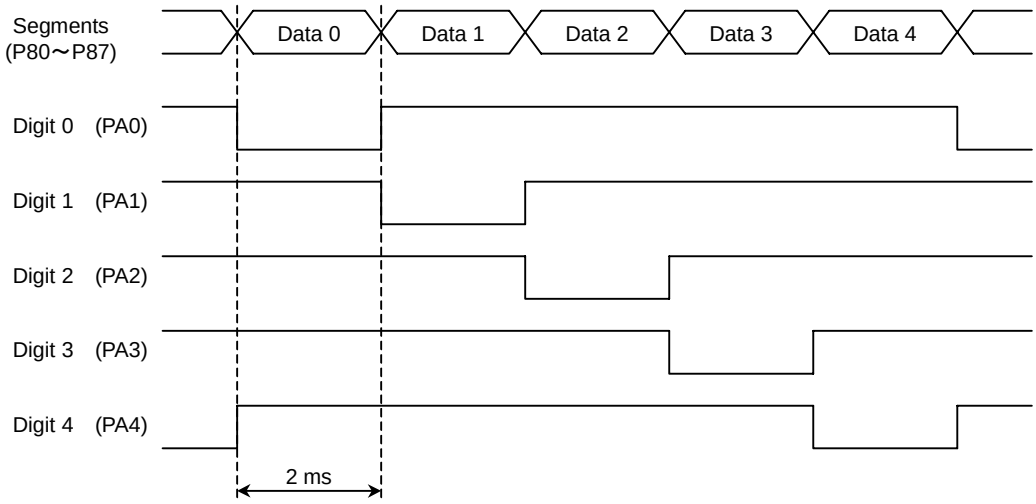


Figure 3.3.14 Correspondence between digits and segments

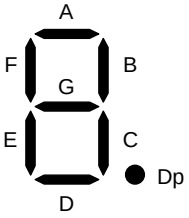
- Figure 3.3.15 shows the timing for the digit and segment outputs.



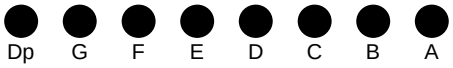
Note: The segment and digit outputs are all active-Low.

Figure 3.3.15 Timing of display digit update

- To light any given display segment, output a Low signal on the corresponding ports. The correspondence between display segments and port outputs is shown in Table 3.3.5.



7SEG-LED



LED

Table 3.3.5 Correspondence between display segments and port outputs

	P87(Dp)	P86(G)	P85(F)	P84(E)	P83(D)	P82(C)	P81(B)	P80(A)
0	*	1	0	0	0	0	0	0
1	*	1	1	1	1	0	0	1
2	*	0	1	0	0	1	0	0
3	*	0	1	1	0	0	0	0
4	*	0	0	1	1	0	0	1
5	*	0	0	1	0	0	1	0
6	*	0	0	0	0	0	1	0
7	*	1	0	1	1	0	0	0
8	*	0	0	0	0	0	0	0
9	*	0	0	1	1	0	0	0
A	*	0	0	0	1	0	0	0
B	*	0	0	0	0	0	1	1
C	*	1	0	0	0	1	1	0
D	*	0	1	0	0	0	0	1
E	*	0	0	0	0	1	1	0
F	*	0	0	0	1	1	1	0
Blank	1	1	1	1	1	1	1	1

Note 1: All segment outputs are active-Low.

Note 2: Set port output * to 0 to display Dp and to 1 to turn Dp off.

3.3.6.2 Control method for 7-segment LED display

■ Initial settings

		7	6	5	4	3	2	1	0	
P8	←	1	1	1	1	1	1	1	1	Set P80~P87 output latches to 1.
P8CR	←	1	1	1	1	1	1	1	1	Set P80~P87 for output.
P8FC	←	0	0	0	0	0	0	0	0	Set P80~P87 to be port.
PA	←	-	-	-	1	1	1	1	1	Set PA0~PA4 output latches to 1.
PACR	←	-	-	-	1	1	1	1	1	Set PA0~PA4 for output.
PAFC	←	-	-	-	0	0	0	0	0	Set PA0~PA4 to be port.

Note: X denotes Don't care; "–" denotes No change.

■ Processing of the display output

Create the display data in the main routine and output display data for each digit to the ports within a 2-ms Interval Timer interrupt. For details of the 2-ms interval timer, please refer to Section 3.3.8.

The brightness of the 7-segment and indicator LEDs can be adjusted by changing the time at which display data is output to the ports. The display output control flow is shown in Figure 3.3.16.

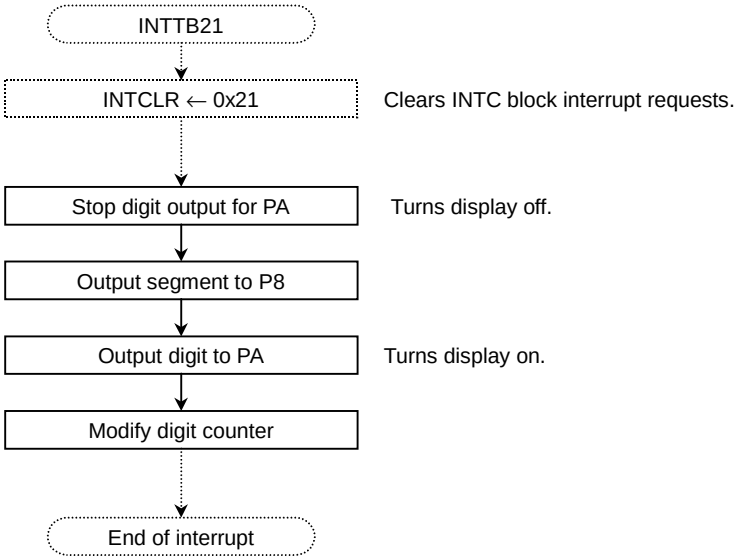


Figure 3.3.16 Display output control flow

3.3.7 Control of beep tone output

3.3.7.1 Overview of beep tone output

The beep tone is sounded for 100 ms at a frequency of 1 kHz (with a 50% duty cycle).

(1) Using TMRA1, invert the value of the timer flip-flop TA1FF every 0.5 ms and output the inverted value on the Timer Flip-Flop Output pin TA1OUT.

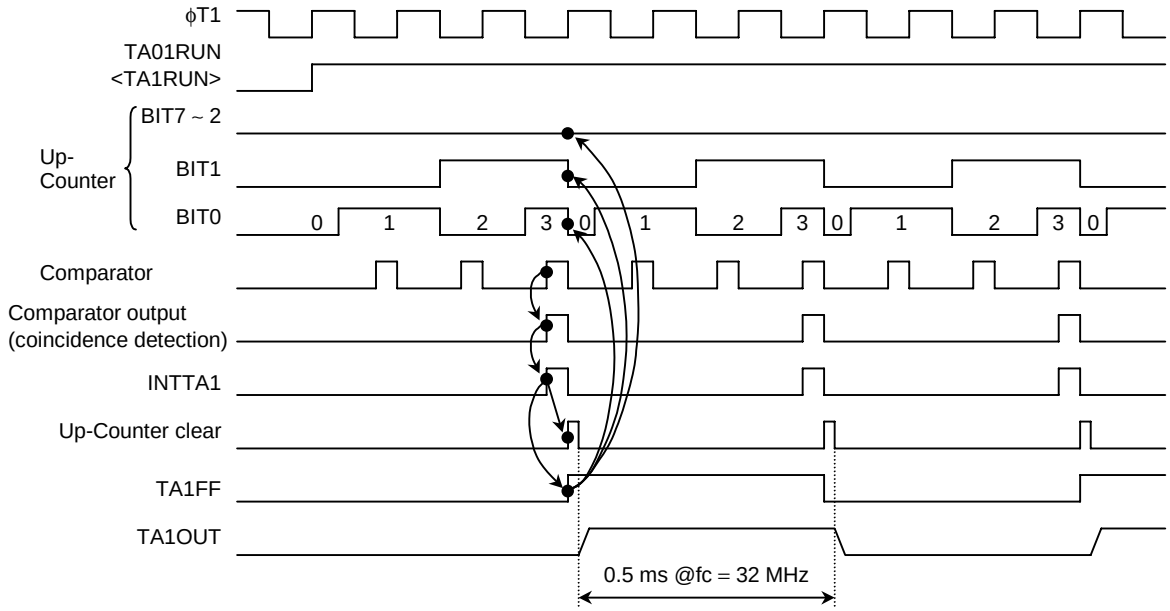


Figure 3.3.17 Square wave output timing chart (with 50% duty cycle)

(2) Timer Register calculation

Table 3.3.6 Prescaler output clock resolution

@fc = 32 MHz

Peripheral Clock Selection <FPSEL>	Clock Gear Value <GEAR1:0>	Selection of Prescaler Clock <PRCK1:0>	Minimum of Prescaler Output Clock Internal			
			ϕT1	ϕT4	ϕT16	ϕT256
0 (fgear)	00 (fc)	00 (fperiph/4)	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)	$fc/2^{11}$ (64 μs)
		01 (fperiph/2)	$fc/2^2$ (0.125 μs)	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)	$fc/2^{10}$ (32 μs)
		10 (fperiph)	—	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)	$fc/2^9$ (16 μs)
	01 (fc/2)	00 (fperiph/4)	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)	$fc/2^8$ (8.0 μs)	$fc/2^{12}$ (128 μs)
		01 (fperiph/2)	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)	$fc/2^{11}$ (64 μs)
		10 (fperiph)	—	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)	$fc/2^{10}$ (32 μs)
	10 (fc/4)	00 (fperiph/4)	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)	$fc/2^9$ (16 μs)	$fc/2^{13}$ (256 μs)
		01 (fperiph/2)	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)	$fc/2^8$ (8.0 μs)	$fc/2^{12}$ (128 μs)
		10 (fperiph)	—	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)	$fc/2^{11}$ (64 μs)
	11 (fc/8)	00 (fperiph/4)	$fc/2^6$ (2.0 μs)	$fc/2^8$ (8.0 μs)	$fc/2^{10}$ (32 μs)	$fc/2^{14}$ (512 μs)
		01 (fperiph/2)	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)	$fc/2^9$ (16 μs)	$fc/2^{13}$ (256 μs)
		10 (fperiph)	—	$fc/2^6$ (2.0 μs)	$fc/2^8$ (8.0 μs)	$fc/2^{12}$ (128 μs)
1 (fc)	00 (fc)	00 (fperiph/4)	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)	$fc/2^{11}$ (64 μs)
		01 (fperiph/2)	$fc/2^2$ (0.125 μs)	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)	$fc/2^{10}$ (32 μs)
		10 (fperiph)	—	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)	$fc/2^9$ (16 μs)
	01 (fc/2)	00 (fperiph/4)	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)	$fc/2^{11}$ (64 μs)
		01 (fperiph/2)	—	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)	$fc/2^{10}$ (32 μs)
		10 (fperiph)	—	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)	$fc/2^9$ (16 μs)
	10 (fc/4)	00 (fperiph/4)	—	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)	$fc/2^{11}$ (64 μs)
		01 (fperiph/2)	—	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)	$fc/2^{10}$ (32 μs)
		10 (fperiph)	—	—	$fc/2^5$ (1.0 μs)	$fc/2^9$ (16 μs)
	11 (fc/8)	00 (fperiph/4)	—	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)	$fc/2^{11}$ (64 μs)
		01 (fperiph/2)	—	—	$fc/2^6$ (2.0 μs)	$fc/2^{10}$ (32 μs)
		10 (fperiph)	—	—	$fc/2^5$ (1.0 μs)	$fc/2^9$ (16 μs)

Note 1: The prescaler's output clock ϕTn must be selected such that the relationship ϕTn < fsys/2 is satisfied (i.e. ϕTn must be slower than fsys/2).

Note 2: Do not change the clock gear value while the timer is operating.

Note 3: The dash character, —, in the table indicates a prohibited setting.

For ϕT1 ($fc/2^3$)

1000 μs ÷ 0.25 μs ÷ 2 = 2000 (7D0H)

Cannot be set with 8-Bit Timer.

For ϕT4 ($fc/2^5$)

1000 μs ÷ 1 μs ÷ 2 = 500 (1F4H)

Cannot be set with 8-Bit Timer.

For ϕT16 ($fc/2^7$)

1000 μs ÷ 4.0 μs ÷ 2 = 125 (7DH)

Set TA1REG to 7DH.

For ϕT256 ($fc/2^{11}$)

1000 μs ÷ 64 μs ÷ 2 = 7.8125 (08H)

Set TA1REG to 08H. In this case the period is 1024 μs, with a 2.4% margin of error.

3.3.7.2 Method for control of beep tone output

■ Initial settings

		7	6	5	4	3	2	1	0	
P7CR	←	-	-	-	-	-	-	1	-	Set P71 to be TA1OUT output pin.
P7FC	←	-	-	-	-	-	-	1	-	
TA01RUN	←	0	X	X	X	0	-	0	-	Stop TMRA1 and clear it to 0.
TA01MOD	←	0	0	0	0	1	0	0	0	Select 8-Bit Timer Mode and set input clock to φT16 (4 μs @fc = 32 MHz).
TA1REG	←	0	1	1	1	1	1	0	1	Set TA1REG to 1000 μs ÷ φT16 ÷ 2 = 125.
TA1FFCR	←	X	X	X	X	1	0	1	1	Clear TA1FF to 0 and set it to be inverted by a match detection signal from TMRA1.
IMC5L	←	X	X	-	-	-	-	-	- L	Set INTTA1 for rising edge and to level 4.
		X	X	1	1	0	1	0	0 H	

Note 1: X denotes Don't care; "–" denotes No change.
Note 2: The interrupt level can be set to any desired value.

■ Beep tone processing

Using an INTTA1 interrupt count the time for which the beep sounds. When 100 ms has elapsed, stop the timer and turn the beep tone off.

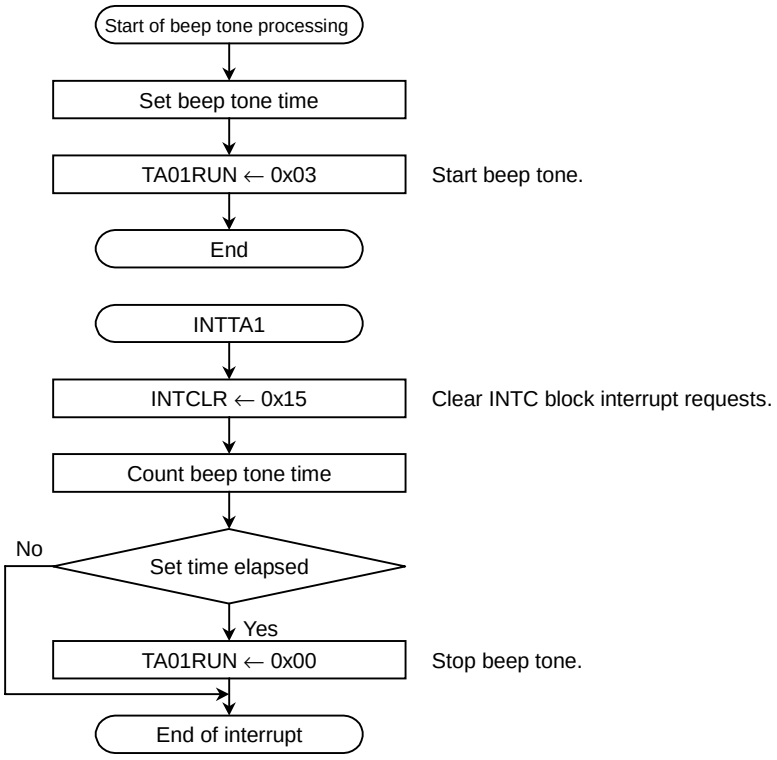


Figure 3.3.18 Beep tone processing flow

3.3.8 2-ms Interval Timer

3.3.8.1 Overview of the 2-ms Interval Timer

- Using the 16-Bit Timer (TMRB), generate an interrupt every 2 ms.
- Set the interval time in the Timer Register TB2RG1 and generate INTTB21.

■ Timer Register calculation

The prescaler output resolutions needed for calculation are shown in Table 3.3.7.

Table 3.3.7 Prescaler output clock resolution

@fc = 32 MHz

Peripheral Clock Selection <FPSEL>	Clock Gear Value <GEAR1:0>	Selection of Prescaler Clock <PRCK1:0>	Resolution of Prescaler Output Clock		
			ϕT1	ϕT4	ϕT16
0 (gear)	00 (fc)	00 (fperiph/4)	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)
		01 (fperiph/2)	$fc/2^2$ (0.125 μs)	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)
		10 (fperiph)	—	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)
	01 (fc/2)	00 (fperiph/4)	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)	$fc/2^8$ (8.0 μs)
		01 (fperiph/2)	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)
		10 (fperiph)	—	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)
	10 (fc/4)	00 (fperiph/4)	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)	$fc/2^9$ (16 μs)
		01 (fperiph/2)	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)	$fc/2^8$ (8.0 μs)
		10 (fperiph)	—	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)
	11 (fc/8)	00 (fperiph/4)	$fc/2^6$ (2.0 μs)	$fc/2^8$ (8.0 μs)	$fc/2^{10}$ (32 μs)
		01 (fperiph/2)	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)	$fc/2^9$ (16 μs)
		10 (fperiph)	—	$fc/2^6$ (2.0 μs)	$fc/2^8$ (8.0 μs)
1 (fc)	00 (fc)	00 (fperiph/4)	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)
		01 (fperiph/2)	$fc/2^2$ (0.125 μs)	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)
		10 (fperiph)	—	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)
	01 (fc/2)	00 (fperiph/4)	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)
		01 (fperiph/2)	—	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)
		10 (fperiph)	—	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)
	10 (fc/4)	00 (fperiph/4)	—	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)
		01 (fperiph/2)	—	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)
		10 (fperiph)	—	—	$fc/2^5$ (1.0 μs)
	11 (fc/8)	00 (fperiph/4)	—	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)
		01 (fperiph/2)	—	—	$fc/2^6$ (2.0 μs)
		10 (fperiph)	—	—	$fc/2^5$ (1.0 μs)

Note 1: The prescaler's output clock ϕTn must be selected such that the relationship ϕTn < fsys/2 is satisfied (i.e. ϕTn must be slower than fsys/2).
Note 2: Do not change the clock gear value while the timer is operating.
Note 3: The dash character, —, in the table indicates a prohibited setting.

For ϕT1 ($fc/2^3$)

2000 μs ÷ 0.25 μs = 8000 (1F40H)

Set TB2RG1L to 00H and TB2RG1H to 1FH.

For ϕT4 ($fc/2^5$)

2000 μs ÷ 1.0 μs = 2000 (07D0H)

Set TB2RG1L to D0H and TB2RG1H to 07H.

For ϕT16 ($fc/2^7$)

2000 μs ÷ 4.0 μs = 500 (01F4H)

Set TB2RG1L to F4H and TB2RG1H to 01H.

3.3.8.2 Method for controlling the 2-ms Interval Timer

■ Initial settings

		7	6	5	4	3	2	1	0		
											Stop TMRB2.
TB2RUN	←	0	0	X	X	0	0	0	0		Disable trigger.
TB2FFCR	←	1	1	0	0	0	0	1	1		Select prescaler output clock as input clock and disable
TB2MOD	←	0	0	1	0	0	1	1	1		capture function.
											Set interval time.
TB2RG1L	←	1	1	1	1	0	1	0	0		Set TB2RG1 to 2000 μs ÷ ϕT16 = 500.
TB2RG1H	←	0	0	0	0	0	0	0	1		
IMC8L	←	X	X	-	-	-	-	-	-	L	Set INTTB21 for rising edge and to level 4.
		X	X	1	1	0	0	1	1	H	
TB2RUN	←	0	0	X	X	0	1	0	1		Start TMRB2.

Note 1: X denotes Don't care; "–" denotes No change.

Note 2: The interrupt level can be set to any desired value.

- Always set the eight low-order data bits in the Timer Register first, followed by the eight high-order bits.
- In the sample program chattering-elimination processing for the matrix keys and AD conversion keys and 7-segment LED display output processing is performed using a INTTB21 interrupt.

3.3.9 Sample programs

3.3.9.1 Generic flowchart

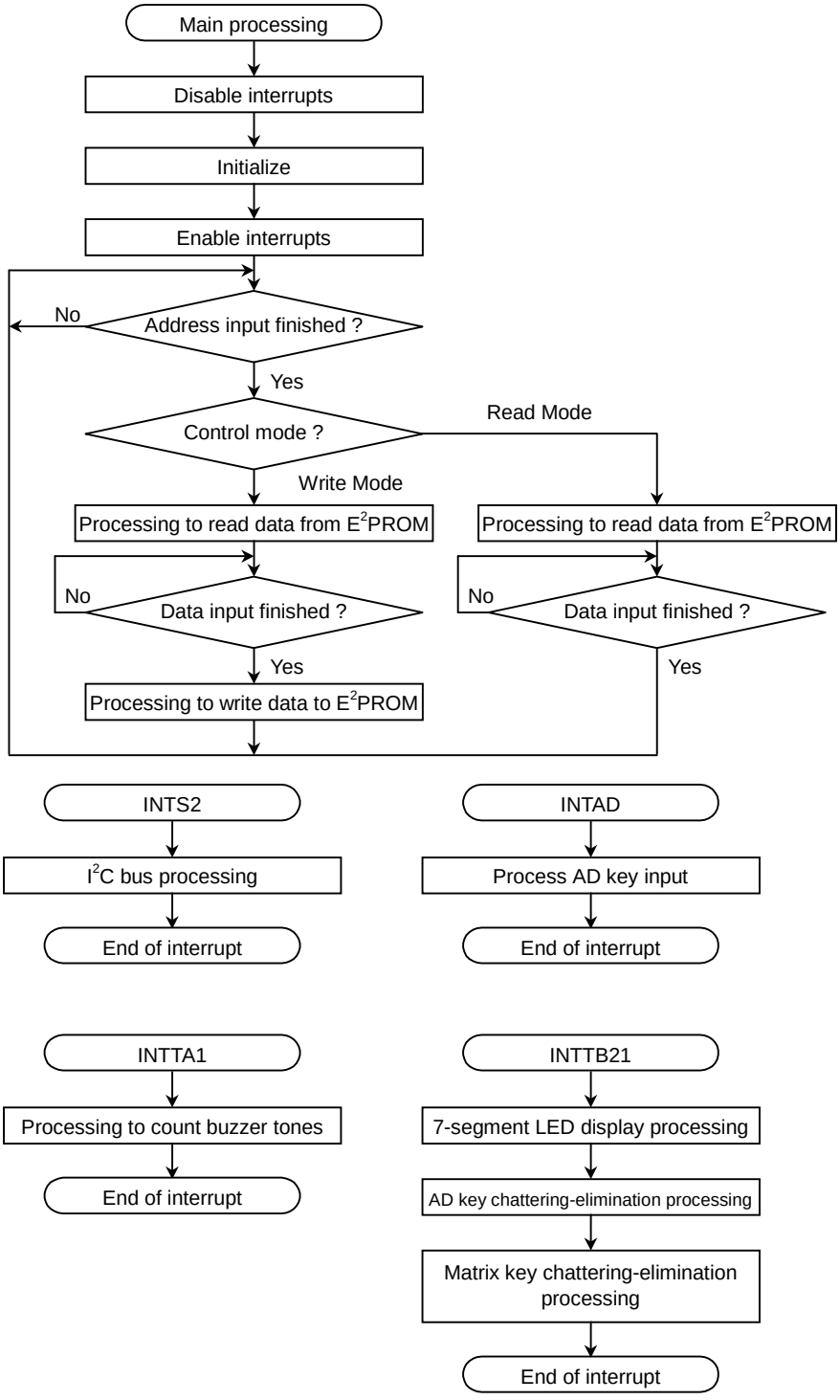


Figure 3.3.19 Generic flow

3.3.9.2 File configuration

The configuration of files used in the sample programs is listed in Table 3.3.8.

Table 3.3.8 File configuration

Filename	Contents	Page References
eeprom.c	E^2PROM Mode main section	3-103
func_eeprom.c	E^2PROM Mode functional section	3-110
led.h	7-segment LED display definitions	3-117
key.h	Key definitions	3-118
Stc9i16.asm	Start-up routine	3-12
io1940.c	Special function registers	3-51
io1940.h	Special function register definitions	3-28
Bit32def.h	32-bit macro definitions	3-42
adc.h	AD conversion register definitions	3-31
sio.h	Serial interface register definitions	3-46

3.3.9.3 Vector table

The vector table used in the sample programs is shown below. Replace the vector table section in the start-up routine with the vector table given below.

- Interrupt vector table section

```
_VecTable:
      dw      _Int_dummy      ;0 --- software set
      dw      _Int_dummy      ;1 --- INT[0]
      dw      _Int_dummy      ;2 --- INT[1]
      dw      _Int_dummy      ;3 --- INT[2]
      dw      _Int_dummy      ;4 --- INT[3]
      dw      _Int_dummy      ;5 --- INT[4]
      dw      _Int_dummy      ;6 --- *
      dw      _Int_dummy      ;7 --- *
      dw      _Int_dummy      ;8 --- *
      dw      _Int_dummy      ;9 --- *
      dw      _Int_dummy      ;10--- INT[5]
      dw      _Int_dummy      ;11--- INT[6]
      dw      _Int_dummy      ;12--- INT[7]
      dw      _Int_dummy      ;13--- INT[8]
      dw      _Int_dummy      ;14--- INT[9]
      dw      _Int_dummy      ;15--- INT[A]
      dw      _Int_dummy      ;16--- *
      dw      _Int_dummy      ;17--- *
      dw      _Int_dummy      ;18--- *
      dw      _Int_dummy      ;19--- *
      dw      _Int_dummy      ;20--- INTTA0 : 8bit Timer 0
      dw      _mintta1        ;21--- INTTA1 : 8bit Timer 1
      dw      _Int_dummy      ;22--- INTTA2 : 8bit Timer 2
      dw      _Int_dummy      ;23--- INTTA3 : 8bit Timer 3
      dw      _Int_dummy      ;24--- *
      dw      _Int_dummy      ;25--- *
      dw      _Int_dummy      ;26--- *
      dw      _Int_dummy      ;27--- *
      dw      _Int_dummy      ;28--- INTTB00 :16bit Timer 0 (TB0RG0)
      dw      _Int_dummy      ;29--- INTTB01 :16bit Timer 0 (TB0RG1)
      dw      _Int_dummy      ;30--- INTTB10 :16bit Timer 1 (TB1RG0)
      dw      _Int_dummy      ;31--- INTTB11 :16bit Timer 1 (TB1RG1)
      dw      _Int_dummy      ;32--- INTTB20 :16bit Timer 2 (TB2RG0)
      dw      _minttb21       ;33--- INTTB21 :16bit Timer 2 (TB2RG1)
      dw      _Int_dummy      ;34--- INTTN30 :16bit Timer 3 (TB3RG0)
```

```
dw      _Int_dummy      ;35--- INTTB31 :16bit Timer 3 (TB3RG1)
dw      _Int_dummy      ;36--- *
dw      _Int_dummy      ;37--- *
dw      _Int_dummy      ;38--- *
dw      _Int_dummy      ;39--- *
dw      _Int_dummy      ;40--- INTTBOF0:16bit Timer 0 (OverFlow)
dw      _Int_dummy      ;41--- INTTBOF1:16bit Timer 1 (OverFlow)
dw      _Int_dummy      ;42--- INTTBOF2:16bit Timer 2 (OverFlow)
dw      _Int_dummy      ;43--- INTTBOF3:16bit Timer 3 (OverFlow)
dw      _Int_dummy      ;44--- *
dw      _Int_dummy      ;45--- *
dw      _Int_dummy      ;46--- *
dw      _Int_dummy      ;47--- *
dw      _Int_dummy      ;48--- INTRX0 :Serial receive (Channel 0)
dw      _Int_dummy      ;49--- INTTX0 :Serial transmit (Channel 0)
dw      _Int_dummy      ;50--- INTRX1 :Serial receive (Channel 1)
dw      _Int_dummy      ;51--- INTTX1 :Serial transmit (Channel 1)
dw      _mints2         ;52--- INTS2  :Serial Channel 2 interrupt
dw      _Int_dummy      ;53--- *
dw      _Int_dummy      ;54--- INTRX3 :Serial receive (Channel 3)
dw      _Int_dummy      ;55--- INTTX3 :Serial transmit (Channel 3)
dw      _Int_dummy      ;56--- INTRX4 :Serial receive (Channel 4)
dw      _Int_dummy      ;57--- INTTX4 :Serial transmit (Channel 4)
dw      _Int_dummy      ;58--- INTRTC :Timer for RTC interrupt
dw      _mintad         ;59--- INTAD  :AD conversion finished
dw      _Int_dummy      ;60--- INTDMA0:DMA transfer finished (Channel 0)
dw      _Int_dummy      ;61--- INTDMA1:DMA transfer finished (Channel 1)
dw      _Int_dummy      ;62--- INTDMA2:DMA transfer finished (Channel 2)
dw      _Int_dummy      ;63--- INTDMA3:DMA transfer finished (Channel 3)
```

3.3.9.4 Source code

- Filename: eeprom.c

```
/*
        ************************************************
        **              Application Note              **
        **                                            **
        **             ( EEPROM ROUTINE )             **
        **                                            **
        **                   MCU                      **
        **                  TX1940                    **
        **                                            **
        **              fc      = 32MHz               **
        **              fsys    = 32MHz               **
        **              fperiph = 32MHz               **
        **              φT0     = 8MHz                **
        **                                            **
        **                  2000/2/3                  **
        **                                            **
        ************************************************
        ************************************************
        *      COPYRIGHT(C) 1999 TOSHIBA CORPORATION     *
        *              ALL RIGHTS RESERVED               *
        ************************************************/

/************************************************/
/*              Loading header file             */
/************************************************/
#include "io1940.h"      /* Special function registers */
#include "sio.h"         /* Serial communications */
#include "key.h"         /* Key definitions */
#include "led.h"         /* Display definitions */
#include <stdlib.h>

/************************************************/
/*             Constant definitions             */
/************************************************/
/*--- Mode(gmd_eeprom) ---*/
#define cmd_address             0           /* Address input mode */
#define cmd_read_step1          1           /* Read Mode 1  */
#define cmd_write_step1         2           /* Write Mode 1 */
#define cmd_read_step2          3           /* Read Mode 2  */
#define cmd_write_step2         4           /* Write Mode 2 */
#define cmd_write_step3         5           /* Write Mode 3 */
/*--- Key ---*/
#define cadkey_write            cadkey_4    /* Write key */
#define cadkey_read             cadkey_5    /* Read key */
/*--- I2C ---*/
#define  csbi0cr1               0x03        /* Serial Control Register 1 (SCL=60.6kHz) */
#define  ceeprom_slave_add      0xa4        /* EEPROM slave address */
#define  ci2c_write             0x00        /* Read Bit */
#define  ci2c_read              0x01        /* Write Bit */

/************************************************/
/*             extern declaration               */
/************************************************/
extern   void padkey_in(void);
extern   void padkey_chat(void);
extern   void pmatkey_chat(void);
extern   void padkey_decode(void);
extern   void pmatkey_decode(void);
extern   unsigned char pnumkey_ch(unsigned short);
extern   void p7seg_disp(void);
```

```
extern   void pbeep_start(unsigned short);
extern   void pbeep_timer(void);

extern unsigned char     g7seg_data[4];
extern unsigned char     gled_data;
extern unsigned char     gadkey_code;
extern unsigned char     fadkey_push;
extern unsigned short    gmatkey_code;
extern unsigned char     fmatkey_push;
extern unsigned char     t7seg_ch[];
extern unsigned char     tled_ch1[];

/***********************************************/
/*                    RAM                      */
/***********************************************/
unsigned char            gmd_eeprom;            /* Mode data */
unsigned char            gmd_eeprom_read;       /* Read Mode data */
unsigned char            gmd_eeprom_write;      /* Write Mode data */
unsigned char            geeprom_add;           /* EEPROM address */
unsigned char            geeprom_data;          /* EEPROM data */

/***********************************************/
/*                    CODE                     */
/***********************************************/

/*****************************************************************************/
/* Module name  :meeprom_init                                                */
/*****************************************************************************/
/* Function     :Initialize E2PROM mode                                      */
/* Input        :None                                                        */
/* Output       :SFR                                                         */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/*****************************************************************************/
void meeprom_init(void){

/*--- CG settings ---*/
      IO_SYSCR1     = 0x10;      /* <SYSCK>=0 <FPSEL>=1 <DFOSC>=0 <GEAR>=00 */
      IO_ADCCLK     = 0x01;      /* <ADCCK>=01 */

/*--- I/O port settings ---*/
      IO_P0         = 0xff;      /* Output High on P00-P07 */
      IO_P0CR       = 0xff;      /* Set P00-P07 for output */

      IO_P1         = 0xff;      /* Output High on P10-P17 */
      IO_P1CR       = 0xff;      /* Set P10-P17 for output */
      IO_P1FC       = 0x00;      /* Set P10-P17 to be a port */

      IO_P2         = 0xff;      /* Output High on P20-P27 */
      IO_P2CR       = 0xff;      /* Set P20-P27 for output */
      IO_P2FC       = 0x00;      /* Set P20-P27 to be a port */

      IO_P3         = 0xff;      /* Output High on P30-P31 and set P32-P37 for pull-up */
      IO_P3CR       = 0x00;      /* Set P30-P31 for output and P32-P37 for input */
      IO_P3FC       = 0x00;      /* Set P30-P37 to be a port */

      IO_P4         = 0x1f;      /* Set P40-P43 for pull-up and output High on P44 */
      IO_P4CR       = 0x10;      /* Set P40-P43 for input and P44 for output */
      IO_P4FC       = 0x00;      /* Set P40-P44 to be a port */

      IO_P7         = 0x65;      /* Output High on P70,P72,P75,P76 */
      IO_P7CR       = 0x67;      /* Set P70,P71,P72,P75,P76 for output and P73,P74,P77
                                    for input */
      IO_P7FC       = 0x02;      /* Use P71 to be TA1OUT and set others to be a ports */

      IO_P8         = 0xff;      /* Output High on P80-P87 */
```

```
        IO_P8CR       = 0xff;       /* Set P80-P87 for output */
        IO_P8FC       = 0x00;       /* Set P80-P87 to be a port */

        IO_P9         = 0x3f;       /* Output Low on P96-P97 */
        IO_P9CR       = 0xc0;       /* Set P90-P95 for input and P96-P97 for output */
        IO_P9FC       = 0x00;       /* Set P90-P97 to be a port */

        IO_PA         = 0x1f;       /* Output High on PA0-PA4 */
        IO_PACR       = 0xdf;       /* Set PA5 for input and set others for output */
        IO_PAFC       = 0xc0;       /* Use PA7 as SCL and PA6 as SDA, and set others to be
                                                ports */
        IO_ODE        = 0x0c;       /* <ODEA7,6>=11 */

    /*--- Timer settings ---*/
        IO_TB2RUN     = 0x00;       /* <TB2RDE>=0 <I2TB2>=0 <TB2PRUN>=0 <TB2RUN>=0 */
        IO_TB2FFCR    = 0xc3;       /* <TB2C1T1>=0 <TB2C0T1>=0 <TB2E1T1>=0 */
                                    /* <TB2E0T1>=0 <TB2FF0C>=11 */
        IO_TB2MOD     = 0x27;       /* <TB2CP0I>=1 <TB2CPM>=00 <TB2CLE>=1 <TB2CLK>=11 */
        IO_TB2RG1L    = 0xf4;       /* TB2RG1 = 2000 / 4 */
        IO_TB2RG1H    = 0x01;       /*  */
        IO_IMC8L      = 0x3330;     /* ↑ LEVEL3 */

    /*--- Buzzer settings ---*/
        IO_TA01RUN    = 0x00;       /* <TA0RDE>=0 <I2TA01>=0 <TA01PRUN>=0 */
                                    /* <TA1RUN>=0 <TA0RUN>=0 */
        IO_TA01MOD    = 0x08;       /* <TA01M>=00 <PWM0>=00 <TA1CLK>=10 <TA0CLK>=00 */
        IO_TA1REG     = 0x7d;       /* TA1REG = 500 / 4 */
        IO_TA1FFCR    = 0x0b;       /* <TAFF1C>=10 <TAFF1IE>=1 <TAFF1IS>=1 */
        IO_IMC5L      = 0x3430;     /* ↑ LEVEL4 */

    /*--- AD conversion settings ---*/
        IO_ADMOD0     = 0x00;       /* <ITM0L>=0 <REPERT>=0 <SCAN>=0 <ADS>=0 */
        IO_ADMOD1     = 0x82;       /* <VREFON>=1 <I2AD>=0 <ADTRGE>=0 <ADCH>=010 */
        IO_IMCEH      = 0x3210;     /* ↑ LEVEL2 */

    /*--- I2C settings ---*/
        IO_SBI0BR0    = 0x00;       /* <I2SBI0>=0 */
        IO_SBI0BR1    = 0x80;       /* <P4EN>=1 */
        IO_SBI0CR1    = csbi0cr1 | ACK;            /* <ACK>=1 */
        IO_I2C0AR     = 0x00;       /* <ALS>=0 */
        IO_SBI0CR2    = 0x18;       /* <PIN>=1 <SBIM>=10 */
        IO_IMCDL      = 0x31;       /* ↑ LEVEL1 */

    /*--- Data initialization ---*/
        g7seg_data[0] = c7seg_spc;
        g7seg_data[1] = c7seg_spc;
        g7seg_data[2] = c7seg_0;
        g7seg_data[3] = c7seg_0;
        gled_data     = cled_all_off;

    /*--- Timer start ---*/
        IO_TB2RUN     = 0x05;       /* <TB2RDE>=0 <I2TB2>=0 <TB2PRUN>=1 <TB2RUN>=1 */
    }

    /*************************************************************************/
    /* Module name  :peeprom_read                                            */
    /*************************************************************************/
    /* Function     :Start reading from EEPROM                               */
    /* Input        :SBI0SR                                                  */
    /* Output       :SBI0CR1,SBI0DBR,SBI0CR2,gmd_eeprom_read                 */
    /* Parameters   :None                                                    */
    /* Return value :None                                                    */
    /*************************************************************************/
    void peeprom_read(void){

        while ((IO_SBI0SR & BB) != 0x00);         /* Check to see if bus is free */
```

```
        IO_SBI0CR1    = csbi0cr1 | ACK;           /* Set ACK */
        IO_SBI0DBR    = ceeprom_slave_add | ci2c_write;   /* Set slave address */
        IO_SBI0CR2    = 0xf8;                     /* Generate start condition */
        gmd_eeprom_read = 1;
}

/*****************************************************************************/
/* Module name  :peeprom_write                                               */
/*****************************************************************************/
/* Function     :Start writing to EEPROM                                     */
/* Input        :SBI0SR                                                      */
/* Output       :SBI0CR1,SBI0DBR,SBI0CR2,gmd_eeprom_read                     */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/*****************************************************************************/
void peeprom_write(void){

        while ((IO_SBI0SR & BB) != 0x00);         /* Check whether bus is free */
        IO_SBI0CR1    = csbi0cr1 | ACK;           /* ACK Set */
        IO_SBI0DBR    = ceeprom_slave_add | ci2c_write; /* Set slave address */
        IO_SBI0CR2    = 0xf8;                     /* Generate start condition */
        gmd_eeprom_write = 1;
}

/*****************************************************************************/
/* Module name  :main                                                        */
/*****************************************************************************/
/* Function     :Main processing                                             */
/* Input        :fadkey_push,fmatkey_push,gmatkey_code                       */
/* Output       :gmatkey_code,gadkey_code,gmd_eeprom,geeprom_add,g7seg_data   */
/*              :gadkey_code,gled_data                                       */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/*****************************************************************************/
void main(void){

        __DI();

        meeprom_init();                           /* Initialize */

        __EI();

/*************************************/
/*                                   */
/*              Main loop            */
/*                                   */
/*************************************/

for(;;){
        padkey_decode();
        pmatkey_decode();

        /*--- Disabling simultaneous key depressions ---*/
        if (fadkey_push == 1 && fmatkey_push == 1) {
             gmatkey_code = cmatkey_no;
             gadkey_code = cadkey_no;
        }

        switch (gmd_eeprom) {
        /*--- Address input ---*/
        case cmd_address:
             switch (gmatkey_code) {        /* Numeric key processing */
             case cmatkey_0:
             case cmatkey_1:
             case cmatkey_2:
             case cmatkey_3:
```

```
        case cmatkey_4:
        case cmatkey_5:
        case cmatkey_6:
        case cmatkey_7:
        case cmatkey_8:
        case cmatkey_9:
        case cmatkey_a:
        case cmatkey_b:
        case cmatkey_c:
        case cmatkey_d:
        case cmatkey_e:
        case cmatkey_f:
              pbeep_start(100);
              geeprom_add = (geeprom_add<<4) + pnumkey_ch(gmatkey_code);
              g7seg_data[2] = t7seg_ch[geeprom_add & 0x0f];
              g7seg_data[3] = t7seg_ch[geeprom_add >> 4];
              break;
        }

        switch (gadkey_code) {
        case cadkey_read:              /* Read key processing */
              pbeep_start(100);
              peeprom_read();
              gmd_eeprom = cmd_read_step1;
              break;
        case cadkey_write:             /* Write key processing */
              pbeep_start(100);
              gled_data = cled_all_on;
              peeprom_read();
              gmd_eeprom = cmd_write_step1;
              break;
        }
        break;
    /*--- Reading data from EEPROM ---*/
    case cmd_read_step1:
    case cmd_write_step1:
        if (gmd_eeprom_read == 6) {
              gmd_eeprom_read = 0;
              g7seg_data[0] = t7seg_ch[geeprom_data & 0x0f];
              g7seg_data[1] = t7seg_ch[geeprom_data >> 4];
              if (gmd_eeprom == cmd_read_step1) gmd_eeprom = cmd_read_step2;
              else gmd_eeprom = cmd_write_step2;
        }
        break;
    /*--- Check for end during Read Mode ---*/
    case cmd_read_step2:
        switch (gmatkey_code) {        /* Numeric key processing */
        case cmatkey_0:
        case cmatkey_1:
        case cmatkey_2:
        case cmatkey_3:
        case cmatkey_4:
        case cmatkey_5:
        case cmatkey_6:
        case cmatkey_7:
        case cmatkey_8:
        case cmatkey_9:
        case cmatkey_a:
        case cmatkey_b:
        case cmatkey_c:
        case cmatkey_d:
        case cmatkey_e:
        case cmatkey_f:
              pbeep_start(100);
              geeprom_add = (geeprom_add<<4) + pnumkey_ch(gmatkey_code);
              g7seg_data[0] = c7seg_spc;
```

```
                g7seg_data[1] = c7seg_spc;
                g7seg_data[2] = t7seg_ch[geeprom_add & 0x0f];
                g7seg_data[3] = t7seg_ch[geeprom_add >> 4];
                gmd_eeprom = cmd_address;
                break;
          }
          if(gadkey_code == cadkey_read) {      /* Read key processing */
                pbeep_start(100);
                g7seg_data[0] = c7seg_spc;
                g7seg_data[1] = c7seg_spc;
                gmd_eeprom = cmd_address;
          }
          break;
     /*--- Data input during Write Mode ---*/
     case cmd_write_step2:
          switch (gmatkey_code) {        /* Numeric key processing */
          case cmatkey_0:
          case cmatkey_1:
          case cmatkey_2:
          case cmatkey_3:
          case cmatkey_4:
          case cmatkey_5:
          case cmatkey_6:
          case cmatkey_7:
          case cmatkey_8:
          case cmatkey_9:
          case cmatkey_a:
          case cmatkey_b:
          case cmatkey_c:
          case cmatkey_d:
          case cmatkey_e:
          case cmatkey_f:
                pbeep_start(100);
                geeprom_data = (geeprom_data<<4) + pnumkey_ch(gmatkey_code);
                g7seg_data[0] = t7seg_ch[geeprom_data & 0x0f];
                g7seg_data[1] = t7seg_ch[geeprom_data >> 4];
                break;
          }
          if(gadkey_code == cadkey_write) {     /* Write key processing */
                pbeep_start(100);
                peeprom_write();
                gmd_eeprom = cmd_write_step3;
          }
          break;
     /*--- Writing data to EEPROM ---*/
     case cmd_write_step3:
          if (gmd_eeprom_write == 4) {
                gmd_eeprom_write = 0;
                g7seg_data[0] = c7seg_spc;
                g7seg_data[1] = c7seg_spc;
                gled_data = cled_all_off;
                gmd_eeprom = cmd_address;
          }
          break;
     }
}
}

/*****************************************************************************/
/* Module name  :mints2                                                      */
/*****************************************************************************/
/* Function     :I2C interrupt                                               */
/* Input        :gmd_eeprom_read,SBI0SR,SBI0DBR                              */
/* Output       :gmd_eeprom_read,SBI0CR1,SBI0DBR,SBI0CR2                     */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
```

```
/*****************************************************************************/
void __interrupt mints2(void){

      volatile unsigned int i;

      IO_INTCLR = 0x34;                         /* Clear interrupt latch */
/*--- Reading ---*/
      switch (gmd_eeprom_read) {
      /*--- Address output ---*/
      case 1:
         IO_SBI0DBR = geeprom_add;
         gmd_eeprom_read = 2;
         break;
      /*--- Restarting ---*/
      case 2:
         IO_SBI0CR2 = 0x18;                     /* Release bus */
         while ((IO_SBI0SR & BB) != 0x00);      /* Check whether bus is free */
         while ((IO_SBI0SR & LRB) != 0x01);     /* Check LRB */
         for (i=0 ;i<80 ;i++);                  /* Wait for 4.7 us */
         IO_SBI0CR1 = csbi0cr1 | ACK;           /* Restart */
         IO_SBI0DBR = ceeprom_slave_add | ci2c_read;
         IO_SBI0CR2 = 0xf8;
         gmd_eeprom_read = 3;
         break;
      /*-- Dummy read  ---*/
      case 3:
         IO_SBI0CR1 = csbi0cr1;
         geeprom_data = IO_SBI0DBR;
         gmd_eeprom_read = 4;
         break;
      /*--- Reading data ---*/
      case 4:
         IO_SBI0CR1 = csbi0cr1 | 0x20;
         geeprom_data = IO_SBI0DBR;
         gmd_eeprom_read = 5;
         break;
      /*--- Stop ---*/
      case 5:
         IO_SBI0CR2 = 0xd8;
         gmd_eeprom_read = 6;
         break;
      }

/*--- Writing ---*/
      switch (gmd_eeprom_write) {
      /*--- Address output ---*/
      case 1:
         IO_SBI0DBR = geeprom_add;
         gmd_eeprom_write = 2;
         break;
      /*--- Writing data ---*/
      case 2:
         IO_SBI0DBR = geeprom_data;
         gmd_eeprom_write = 3;
         break;
      /*--- Stop ---*/
      case 3:
         IO_SBI0CR2 = 0xd8;
         gmd_eeprom_write = 4;
         break;
      }
}

/*****************************************************************************/
/* Module name :minttb21                                                     */
/*****************************************************************************/
```

```
/* Function    :2-ms Interval Timer interrupt                                  */
/* Input       :None                                                           */
/* Output      :None                                                           */
/*******************************************************************************/
void __interrupt minttb21(void){

      IO_INTCLR = 0x21;             /* Clear interrupt latch */
      p7seg_disp();
      padkey_chat();
      pmatkey_chat();
}

/*******************************************************************************/
/* Module name :mintta1                                                        */
/*******************************************************************************/
/* Function    :Beep tone timer interrupt                                      */
/* Input       :None                                                           */
/* Output      :None                                                           */
/*******************************************************************************/
void __interrupt mintta1(void){

      IO_INTCLR = 0x15;             /* Clear interrupt latch */
      pbeep_timer();
}

/*******************************************************************************/
/* Module name :mintad                                                         */
/*******************************************************************************/
/* Function    :AD Conversion Finished interrupt                               */
/* Input       :None                                                           */
/* Output      :None                                                           */
/*******************************************************************************/
void __interrupt mintad(void){

      IO_INTCLR = 0x3b;             /* Clear interrupt latch */
      padkey_in();
}
```

- Filename: func_eeprom.c

```
/*
        ************************************************
        **              Application Note              **
        **                                            **
        **          (EEPROM FUNCTION ROUTINE )        **
        **                                            **
        **                   MCU                      **
        **                  TX1940                    **
        **                                            **
        **              fc      = 32MHz               **
        **              fsys    = 32MHz               **
        **              fperiph = 32MHz               **
        **              ϕT0     = 8MHz                **
        **                                            **
        **                 1999/9/20                  **
        **                                            **
        ************************************************
        ************************************************
        *      COPYRIGHT(C) 1999 TOSHIBA CORPORATION     *
        *              ALL RIGHTS RESERVED               *
        ************************************************/

/***********************************************/
/*              Loading header file            */
/***********************************************/
#include "io1940.h"
#include "adc.h"
#include "key.h"

/***********************************************/
/*              Constant definitions           */
/***********************************************/
/*--- Port output definitions (key digits) ---*/
#define  ckey_digit_out0       0x64             /* P7 (01100100) */
#define  ckey_digit_out1       0x61             /* P7 (01100001) */
#define  ckey_digit_out2       0x45             /* P7 (01000101) */
#define  ckey_digit_out3       0x25             /* P7 (00100101) */
/*--- Port output definitions (7-segment LED digits) ---*/
#define  c7seg_digit_out0      0x1e             /* PA (00011110) */
#define  c7seg_digit_out1      0x1d             /* PA (00011101) */
#define  c7seg_digit_out2      0x1b             /* PA (00011011) */
#define  c7seg_digit_out3      0x17             /* PA (00010111) */
#define  c7seg_digit_out4      0x0f             /* PA (00001111) */

/***********************************************/
/*                    RAM                      */
/***********************************************/
unsigned char            g7seg_data[4];         /* 7-segment LED display data */
unsigned char            gled_data;             /* LED display data */
unsigned char            gadkey_data;           /* AD key data */
unsigned char            gadkey_code;           /* AD key confirmed data */
unsigned char            fadkey_push;           /* AD Key Depression Status flag */
unsigned char            fadkey_ok;             /* AD Key Confirmed flag */
unsigned short           gmatkey_data;          /* Matrix key data */
unsigned short           gmatkey_code;          /* Matrix key confirmed data */
unsigned char            fmatkey_push;          /* Matrix Key Depression Status flag */
unsigned char            fmatkey_ok;            /* Matrix Key Confirmed flag */
unsigned short           gbeep_cnt;             /* Beep tone count */

/***********************************************/
/*                    CODE                     */
/***********************************************/
```

```
/*****************************************************************************/
/* Module name  :padkey_in                                                   */
/*****************************************************************************/
/* Function     :Entry of AD key data (using AD finished interrupt)          */
/* Input        :ADREG26H                                                    */
/* Output       :gadkey_data                                                 */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/*****************************************************************************/
void padkey_in(void) {

      unsigned char lad_data;

/*--- Entering AD-converted data ---*/
      lad_data = IO_ADREG26H;

/*--- Converting AD data into key data ---*/
      if (lad_data > 225) gadkey_data = 0;
      else if (lad_data > 175) gadkey_data = 1;
      else if (lad_data > 125) gadkey_data = 2;
      else if (lad_data >  75) gadkey_data = 3;
      else if (lad_data >  25) gadkey_data = 4;
      else gadkey_data = 5;
}

/*****************************************************************************/
/* Module name  :padkey_chat                                                 */
/*****************************************************************************/
/* Function     :Eliminates AD key chattering (using 2-ms interrupt)         */
/* Input        :gadkey_data                                                 */
/* Output       :ADMOD0 ,fadkey_ok ,fadkey_push                              */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/*****************************************************************************/
void padkey_chat(void) {

      static unsigned char ladkey_chat = 0 ,ladkey_buff = 0x00;

/*--- Eliminating chattering ---*/
      if (gadkey_data == ladkey_buff) {
         ladkey_chat++;
         if (ladkey_chat > 15) {
                 ladkey_chat = 0;
                 if (gadkey_data == 0x00) fadkey_push = 0;
                 else {
                          if (fadkey_push == 0) {
                                   fadkey_ok = 1;
                                   fadkey_push = 1;
                          }
                 }
         }
      }
      else ladkey_chat = 0;
/*--- Holding key data ---*/
      ladkey_buff = gadkey_data;
/*--- Starting AD conversion ---*/
      IO_ADMOD0 |= ADS;
}

/*****************************************************************************/
/* Module name  :pmatkey_chat                                                */
/*****************************************************************************/
/* Function     :Eliminates matrix key chattering (using 2-ms interrupt)     */
/* Input        :P5                                                          */
/* Output       :P7 ,gmatkey_data ,fmatkey_ok ,fmatkey_push                  */
/* Parameters   :None                                                        */
```

```
/* Return value :None                                                          */
/*****************************************************************************/
void pmatkey_chat(void) {

      static unsigned char  lmatkey_chat = 0 ,lmatkey_digit = 0;
      static unsigned short lmatkey_buff = 0x0000;
      unsigned short lmat_data;

/*--- Reading data ---*/
      lmat_data = IO_P5 & 0xf0;

/*--- Storing key data ---*/
      switch (lmatkey_digit) {
      case 0:
         gmatkey_data = (gmatkey_data & 0xfff0) + (lmat_data >>4);
         /* Storing data */
         lmatkey_digit = 1;                      /* Change digit count */
         IO_P7 = ckey_digit_out1;/* Output next key digit */
         break;
      case 1:
         gmatkey_data = (gmatkey_data & 0xff0f) + lmat_data;
         /* Storing data */
         lmatkey_digit = 2;                      /* Change digit count */
         IO_P7 = ckey_digit_out2;/* Output next key digit */
         break;
      case 2:
         gmatkey_data = (gmatkey_data & 0xf0ff) + (lmat_data <<4);
         /* Storing data */
         lmatkey_digit = 3;                      /* Change digit count */
         IO_P7 = ckey_digit_out3;/* Output next key digit */
         break;
      case 3:
         gmatkey_data = (gmatkey_data & 0x0fff) + (lmat_data <<8);
         /* Storing data */
         lmatkey_digit = 0;                      /* Change digit count */
         IO_P7 = ckey_digit_out0;/* Output next key digit */
         break;
      }

/*--- Eliminating chattering ---*/
      if (gmatkey_data == lmatkey_buff) {
         lmatkey_chat++;
         if (lmatkey_chat > 15) {
                 lmatkey_chat = 0;
                 if (gmatkey_data == 0x0000) fmatkey_push = 0;
                 else {
                          if (fmatkey_push == 0) {
                                   fmatkey_ok = 1;
                                   fmatkey_push = 1;
                          }
                 }
         }
      }
      else lmatkey_chat = 0;

/*--- Holding key data ---*/
      lmatkey_buff = gmatkey_data;
}

/*****************************************************************************/
/* Module name  :padkey_decode                                               */
/*****************************************************************************/
/* Function     :Decodes AD key                                              */
/* Input        :gadkey_data ,fadkey_ok                                      */
/* Output       :gadkey_code ,fadkey_ok                                      */
/* Parameters   :None                                                        */
```

```
/* Return value :None                                                         */
/****************************************************************************/
void padkey_decode(void) {

      gadkey_code = cadkey_no;
      if (fadkey_ok == 1) {
         fadkey_ok = 0;
         gadkey_code = gadkey_data;
      }
}

/****************************************************************************/
/* Module name  :pmatkey_decode                                               */
/****************************************************************************/
/* Function     :Decodes matrix key                                           */
/* Input        :gmatkey_data ,fmatkey_ok                                     */
/* Output       :gmatkey_code ,fmatkey_ok                                     */
/* Parameters   :None                                                         */
/* Return value :None                                                         */
/****************************************************************************/
void pmatkey_decode(void) {

      gmatkey_code = cmatkey_no;
      if (fmatkey_ok == 1) {
         fmatkey_ok = 0;
         gmatkey_code = gmatkey_data;
      }
}

/****************************************************************************/
/* Module name  :pnumkey_ch                                                   */
/****************************************************************************/
/* Function     :Converts numeric key into numeric data                       */
/* Input        :None                                                         */
/* Output       :None                                                         */
/* Parameters   :Key code                                                     */
/* Return value :Key number                                                   */
/****************************************************************************/
unsigned char pnumkey_ch(unsigned short lkey_code) {
      switch (lkey_code) {
      case cmatkey_0:    return      0x00;
      case cmatkey_1:    return      0x01;
      case cmatkey_2:    return      0x02;
      case cmatkey_3:    return      0x03;
      case cmatkey_4:    return      0x04;
      case cmatkey_5:    return      0x05;
      case cmatkey_6:    return      0x06;
      case cmatkey_7:    return      0x07;
      case cmatkey_8:    return      0x08;
      case cmatkey_9:    return      0x09;
      case cmatkey_a:    return      0x0a;
      case cmatkey_b:    return      0x0b;
      case cmatkey_c:    return      0x0c;
      case cmatkey_d:    return      0x0d;
      case cmatkey_e:    return      0x0e;
      case cmatkey_f:    return      0x0f;
      default:           return      0xff;
      }
}

/****************************************************************************/
/* Module name  :p7seg_disp                                                   */
/****************************************************************************/
/* Function     :Output for 7-segment LED display (using 2-ms interrupt)      */
/* Input        :gled_data,g7seg_data                                         */
/* Output       :P8,PA                                                        */
```

```
/* Parameters   :None                                                          */
/* Return value :None                                                          */
/*****************************************************************************/
void p7seg_disp(void){

      static unsigned char ldisp_digit = 0;

/*--- Turning display off ---*/
      IO_P8 = 0xff;

/*--- Output for display ---*/
      switch (ldisp_digit) {
      case 0:
         IO_PA = c7seg_digit_out0;              /* Output digits */
         IO_P8 = gled_data;                     /* Output for LED */
         break;
      case 1:
         IO_PA = c7seg_digit_out1;              /* Output digits */
         IO_P8 = g7seg_data[0];                 /* Output for 7-segment LED */
         break;
      case 2:
         IO_PA = c7seg_digit_out2;              /* Output digits */
         IO_P8 = g7seg_data[1];                 /* Output for 7-segment LED */
         break;
      case 3:
         IO_PA = c7seg_digit_out3;   /* Output digits */
         IO_P8 = g7seg_data[2];                 /* Output for 7-segment LED */
         break;
      case 4:
         IO_PA = c7seg_digit_out4;   /* Output digits */
         IO_P8 = g7seg_data[3];                 /* Output for 7-segment LED */
         break;
      }

/*--- Digit count ---*/
      ldisp_digit++;
      if (ldisp_digit > 4) ldisp_digit = 0;
}

/*********************************************/
/*              Display data table           */
/*********************************************/
/*--- 7SEG-LED ---*/
const unsigned char t7seg_ch[] = {
      0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xd8,
      0x80,0x98,0x88,0x83,0xc6,0xa1,0x86,0x8e,
};
/*--- LED ---*/
const unsigned char tled_ch1[] = {
      0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f
};

/*****************************************************************************/
/* Module name  :pbeep_start                                                 */
/*****************************************************************************/
/* Function     :Starts beep tone                                            */
/* Input        :None                                                        */
/* Output       :gbeep_cnt,TA01RUN                                           */
/* Parameters   :Beep tone time                                              */
/* Return value :None                                                        */
/*****************************************************************************/
void pbeep_start(unsigned short lbeep_on_time){

      gbeep_cnt = lbeep_on_time * 2;
      IO_TA01RUN = 0x06;
}
```

```
/*****************************************************************************/
/* Module name  :pbeep_timer                                                 */
/*****************************************************************************/
/* Function     :Beep tone                                                   */
/* Input        :gbeep_cnt                                                   */
/* Output       :gbeep_cnt                                                   */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/*****************************************************************************/
void pbeep_timer(void){

      gbeep_cnt--;
      if(gbeep_cnt == 0) IO_TA01RUN = 0x00;
}
```

- Filename: led.h

```
/*
        ************************************************
        **             TOSHIBA CORPORATION            **
        **                                            **
        **               ( LED HEADER )               **
        **                                            **
        **                   MCU                      **
        **                  TX1940                    **
        **                                            **
        **                1999/9/20                   **
        **                                            **
        ************************************************
        ************************************************
        *      COPYRIGHT(C) 1999 TOSHIBA CORPORATION     *
        *              ALL RIGHTS RESERVED               *
        ************************************************/

/*** Definition of 7-segment LEDs ***/
        #define         c7seg_0                 0xc0    /* 0 */
        #define         c7seg_1                 0xf9    /* 1 */
        #define         c7seg_2                 0xa4    /* 2 */
        #define         c7seg_3                 0xb0    /* 3 */
        #define         c7seg_4                 0x99    /* 4 */
        #define         c7seg_5                 0x92    /* 5 */
        #define         c7seg_6                 0x82    /* 6 */
        #define         c7seg_7                 0xd8    /* 7 */
        #define         c7seg_8                 0x80    /* 8 */
        #define         c7seg_9                 0x98    /* 9 */
        #define         c7seg_a                 0x88    /* A */
        #define         c7seg_b                 0x83    /* B */
        #define         c7seg_c                 0xc6    /* C */
        #define         c7seg_d                 0xa1    /* D */
        #define         c7seg_e                 0x86    /* E */
        #define         c7seg_f                 0x8e    /* F */
        #define         c7seg_p                 0x8c    /* P */
        #define         c7seg_spc               0xff    /* Blank */

/*** Definition of LEDs ***/
        #define         cled_all_on             0x00    /* Turn all LEDs on */
        #define         cled_all_off            0xff    /* Turn all LEDs off */
```

- Filename: key.h

```
/*
        ************************************************
        **             TOSHIBA CORPORATION            **
        **                                            **
        **               ( KEY HEADER )               **
        **                                            **
        **                    MCU                     **
        **                   TX1940                   **
        **                                            **
        **                 1999/9/20                  **
        **                                            **
        ************************************************
        ************************************************
        *      COPYRIGHT(C) 1999 TOSHIBA CORPORATION     *
        *              ALL RIGHTS RESERVED               *
        ************************************************/

/*--- Definition of matrix keys ---*/
#define  cmatkey_no      0x0000
#define  cmatkey_0       0x0008
#define  cmatkey_1       0x0080
#define  cmatkey_2       0x0800
#define  cmatkey_3       0x8000
#define  cmatkey_4       0x0004
#define  cmatkey_5       0x0040
#define  cmatkey_6       0x0400
#define  cmatkey_7       0x4000
#define  cmatkey_8       0x0002
#define  cmatkey_9       0x0020
#define  cmatkey_a       0x0200
#define  cmatkey_b       0x2000
#define  cmatkey_c       0x0001
#define  cmatkey_d       0x0010
#define  cmatkey_e       0x0100
#define  cmatkey_f       0x1000

/*--- Definition of AD keys ---*/
#define  cadkey_no       0x00
#define  cadkey_1        0x01
#define  cadkey_2        0x02
#define  cadkey_3        0x03
#define  cadkey_4        0x04
#define  cadkey_5        0x05
```

3.4 Stopwatch

This is a minute/seconds stopwatch with a split time facility. It also incorporates a feature for reducing the stopwatch's current consumption in Standby Mode.

3.4.1 Specifications

3.4.1.1 Basic specifications

- Allows key input using the INT0 key (START/STOP), AD conversion keys (SPLIT/CLEAR, STOP, IDLE and SLEEP) and the NMI key.
- Sounds a buzzer each time a key is pressed (except for the NMI, STOP, IDLE and SLEEP keys).
- Displays the time on the 7-segment LEDs.

Table 3.4.1 Key list

Key Name	Type of Key	Key Function
START/STOP	INT0	Starts or stops count
STOP	AD conversion	Puts device into STOP Mode
IDLE	AD conversion	Puts device into IDLE Mode
SLEEP	AD conversion	Puts device into SLEEP Mode
SPLIT/CLEAR	AD conversion	Displays split time or clears count

3.4.1.2 How to use the stopwatch

Figure 3.4.1 below shows how to use the stopwatch on the TB1940 board.

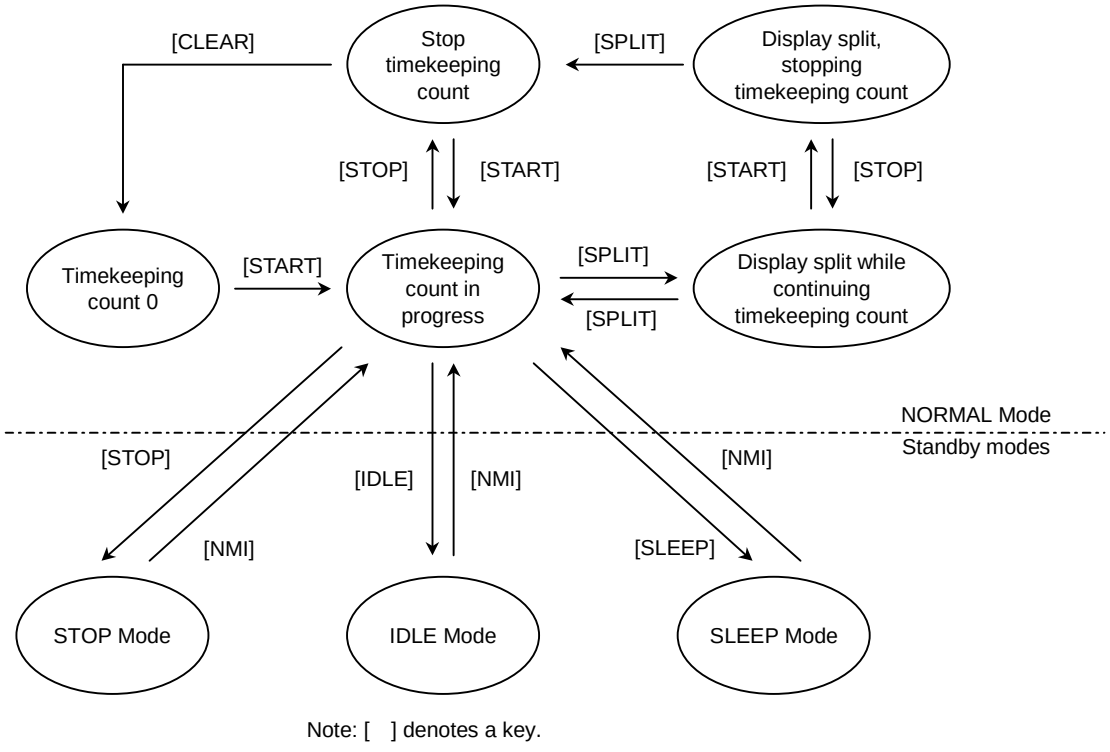


Figure 3.4.1 How to use the stopwatch

■ In NORMAL Mode

Press [START] and the stopwatch starts counting; press [STOP] and the stopwatch stops counting. Pressing [SPLIT] while the stopwatch is counting displays the split time. Pressing [CLEAR] while the stopwatch is idle clears the count. Pressing [STOP], [IDLE] or [SLEEP] while the stopwatch is counting puts the board into the corresponding standby mode.

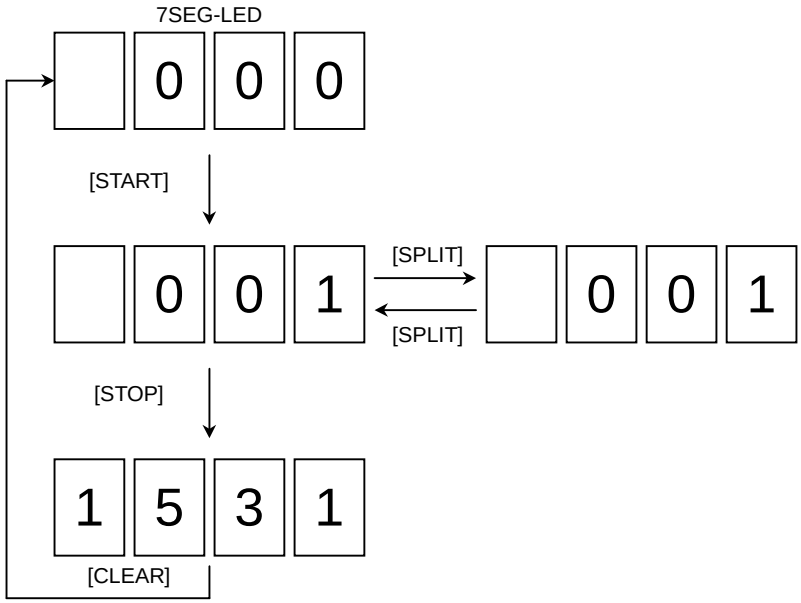


Figure 3.4.2 Typical stopwatch display in NORMAL Mode

■ In SLEEP Mode

The stopwatch display is cleared. The Timer for RTC continues to count. Count can be continued for up to two minutes (approximately). Pressing [NMI] returns the board from SLEEP Mode to NORMAL Mode, causing the count to continue.

■ In IDLE Mode

The stopwatch display is cleared. The count stops and the value reached in the previous mode retained. Pressing [NMI] returns the board from IDLE Mode to NORMAL Mode, causing the count to resume.

■ In STOP Mode

The stopwatch display is cleared. The count stops and the value reached in the previous mode retained. Pressing [NMI] returns the board from STOP Mode to NORMAL Mode, causing the count to resume.

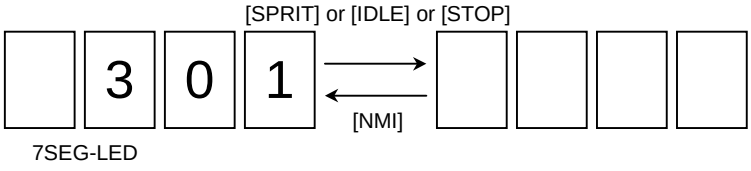


Figure 3.4.3 Typical stopwatch display during SLEEP, IDLE or STOP Mode

3.4.2 Functional block diagram

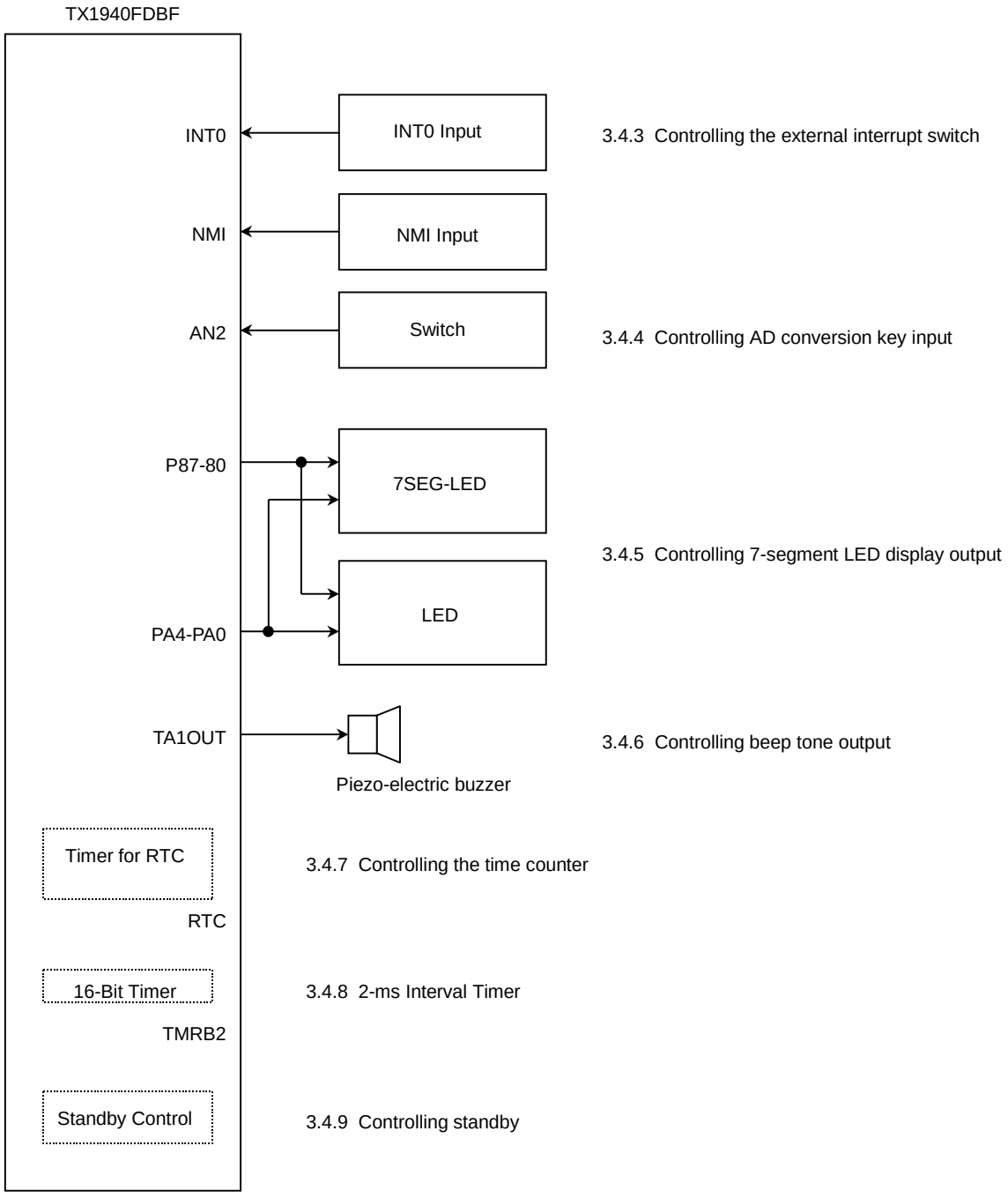


Figure 3.4.4 Functional block diagram

3.4.3 Controlling the external interrupt switch

3.4.3.1 Overview of the external interrupt switch

If the key connected to the P77 (INT0) pin is pressed, an INT0 interrupt will be generated on the next falling edge of the signal on this pin.

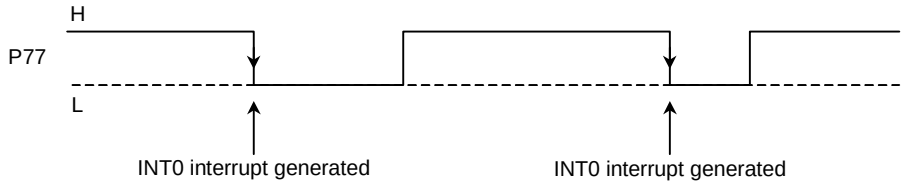


Figure 3.4.5 INT0 interrupt generation timing

3.4.3.2 Controlling the external interrupt switch

■ Initial settings

		7	6	5	4	3	2	1	0		
P7CR	←	0	-	-	-	-	-	-	-	}	Set P77 to be INT0 input pin.
P7FC	←	0	-	-	-	-	-	-	-	}	
IMCGA0	←	X	X	1	0	X	X	X	1		Set the interrupt's active state to falling edge using the CG block.
EICRCG=0x00											
IMC0L	←	X	X	0	1	0	1	1	0	High-order	Make sure interrupt sources are recognized as active when High.
		X	X	X	X	X	X	X	X	Low-order	Set the interrupt level to 6. Any desired value can be set in the Interrupt Register.
INTCLR=0x01											Clear the interrupt request during initialization.

Note 1: X denotes Don't care; "–" denotes No change.

Note 2: The interrupt level can be set to any desired value.

The sample program starts or stops the time count using an INT0 interrupt.

3.4.4 Control of AD conversion key input

3.4.4.1 Overview of AD conversion keys

- Determine whether the AD conversion keys are on or off by reading the voltage from each key.
- Eliminate ON-chattering for a 30-ms period and eliminate OFF-chattering for a 30-ms period.
- Set the resistance value for each resistor so that the voltages across each resistor are the same.
- If two or more keys are pressed simultaneously, the voltage of the key which is located closer to Vdd is read. (In the sample program, key number 5 is read.)

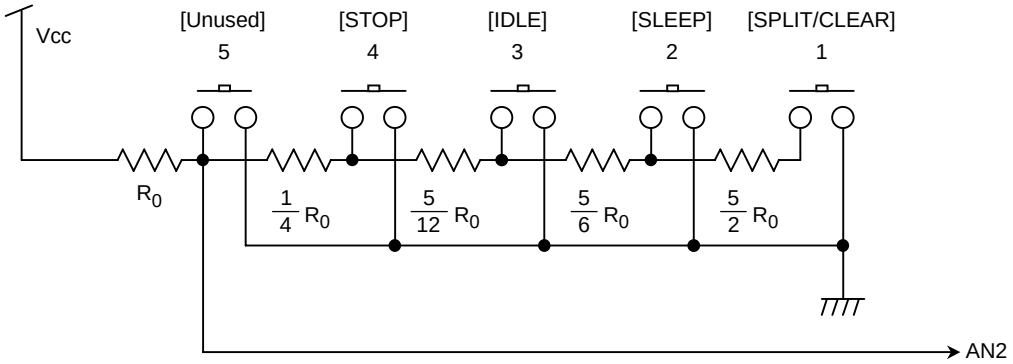


Figure 3.4.6 Key numbers

Table 3.4.2 Correspondence between AD-converted values and voltages

Key Number	Voltage when Key ON (logical value)	AD-Converted Value when Key ON (logical value)	AD-Converted Value of 8 High-Order Bits (logical value)
OFF	3.30 V	1023	255
1	2.65 V	821	205
2	1.99 V	616	154
3	1.34 V	415	103
4	0.67 V	207	51
5	0.00 V	0	0

Note 1: This applies when Vcc = 3.3 V.

Note 2: In the sample program only the eight high-order bits of the AD-converted value are used (the two low-order bits are discarded).

3.4.4.2 Method for controlling AD conversion key input

■ Initial settings

		7	6	5	4	3	2	1	0		
ADCCLK	←	X	X	X	X	X	X	0	1		Set AD conversion time to 10.75 µs (when fsys = 32 MHz).
ADMOD0	←	0	0	0	0	0	0	0	0		Select Fixed-Channel Single Conversion Mode.
ADMOD1	←	1	0	X	X	0	0	1	0		Select Channel 2 and VREF as the voltage to apply.
IMCEH	←	X	X	-	-	-	-	-	-	L	Set INTAD for rising edge and to level 2.
		X	X	1	1	0	0	1	0	H	

Note 1: X denotes Don't care; "–" denotes No change.

Note 2: The interrupt level can be set to any desired value.

■ AD conversion key input processing

When an interrupt is generated on completion of AD conversion, the AD Conversion Result Register is read and the contents is converted into key data. Based on this converted key data, chattering-elimination processing is performed using a 2-ms Interval Timer interrupt.

The processing for each key is performed in the main routine only after it has been confirmed that chattering elimination has been completed. For details of the 2-ms Interval Timer, please refer to Section 3.4.8.

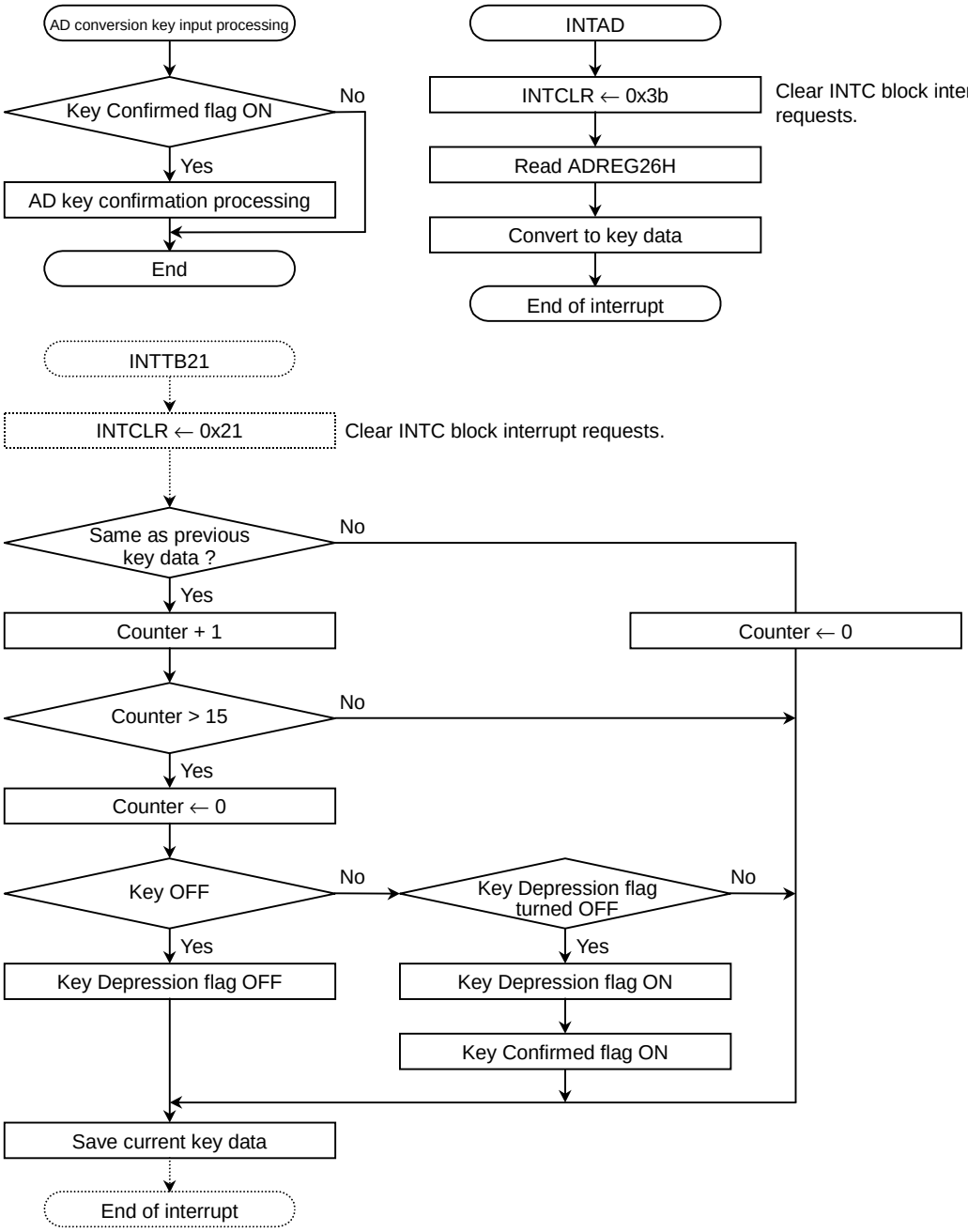


Figure 3.4.7 Operation flow for control of AD conversion key input

3.4.5 Control of 7-segment LED display output

3.4.5.1 Overview of 7-segment LED display

- The display data is dynamically displayed on the 7-segment LEDs and updated every 2 ms.

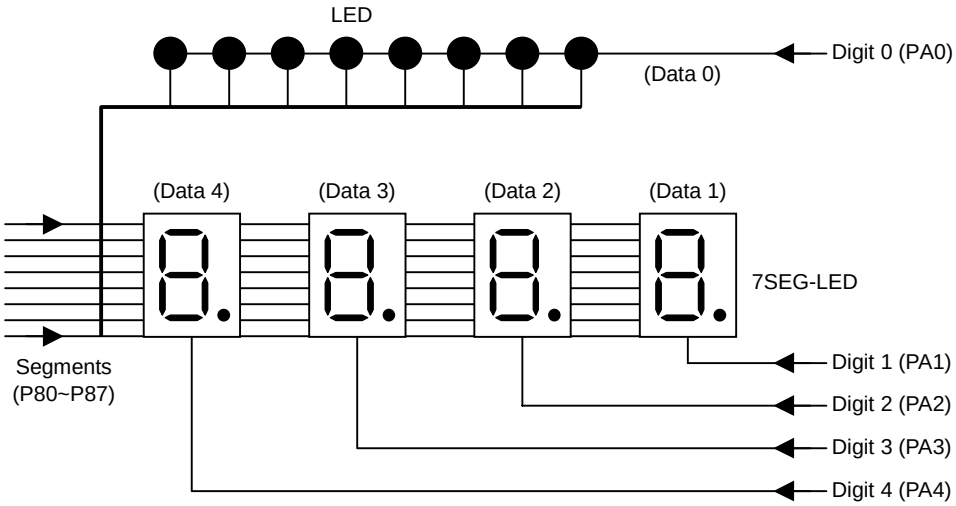
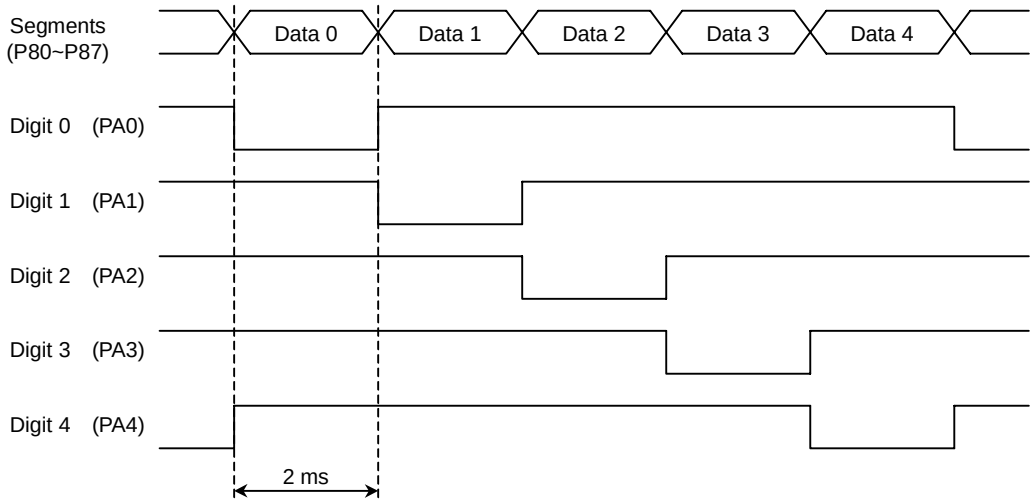


Figure 3.4.8 Correspondence between digits and segments

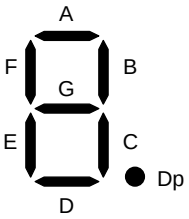
- Figure 3.4.9 shows the timing for the digit and segment outputs.



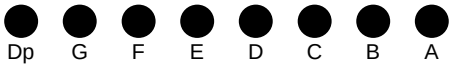
Note: The segment and digit outputs are all active-Low.

Figure 3.4.9 Timing of display digit update

- To light any given display segment, output a Low signal on the corresponding ports. The correspondence between display segments and port outputs is shown in Table 3.4.3.



7SEG-LED



LED

Table 3.4.3 Correspondence between display segments and port outputs

	P87(Dp)	P86(G)	P85(F)	P84(E)	P83(D)	P82(C)	P81(B)	P80(A)
0	*	1	0	0	0	0	0	0
1	*	1	1	1	1	0	0	1
2	*	0	1	0	0	1	0	0
3	*	0	1	1	0	0	0	0
4	*	0	0	1	1	0	0	1
5	*	0	0	1	0	0	1	0
6	*	0	0	0	0	0	1	0
7	*	1	0	1	1	0	0	0
8	*	0	0	0	0	0	0	0
9	*	0	0	1	1	0	0	0
A	*	0	0	0	1	0	0	0
B	*	0	0	0	0	0	1	1
C	*	1	0	0	0	1	1	0
D	*	0	1	0	0	0	0	1
E	*	0	0	0	0	1	1	0
F	*	0	0	0	1	1	1	0
Blank	1	1	1	1	1	1	1	1

Note 1: All segment outputs are active-Low.

Note 2: Set port output * to 0 to display Dp and to 1 to turn Dp off.

3.4.5.2 Control method for 7-segment LED display

■ Initial settings

		7	6	5	4	3	2	1	0	
P8	←	1	1	1	1	1	1	1	1	Set P80~P87 output latches to 1.
P8CR	←	1	1	1	1	1	1	1	1	Set P80~P87 for output.
P8FC	←	0	0	0	0	0	0	0	0	Set P80~P87 to be port.
PA	←	-	-	-	1	1	1	1	1	Set PA0~PA4 output latches to 1.
PACR	←	-	-	-	1	1	1	1	1	Set PA0~PA4 for output.
PAFC	←	-	-	-	0	0	0	0	0	Set PA0~PA4 to be port.

Note: X denotes Don't care; "–" denotes No change.

■ Processing of the display output

Create the display data in the main routine and output display data for each digit to the ports within a 2-ms Interval Timer interrupt. For details of the 2-ms interval timer, please refer to Section 3.3.8.

The brightness of the 7-segment and indicator LEDs can be adjusted by changing the time at which display data is output to the ports. The display output control flow is shown in Figure 3.4.10.

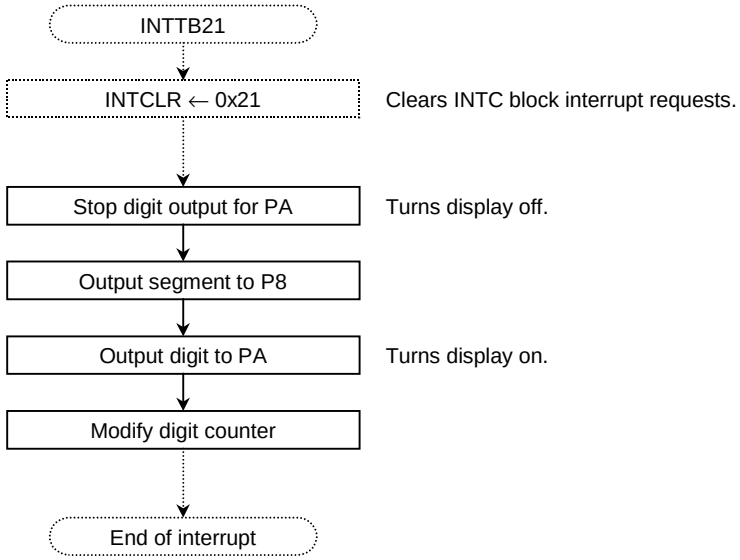


Figure 3.4.10 Display output control flow

3.4.6 Control of beep tone output

3.4.6.1 Overview of beep tone output

The beep tone is sounded for 100 ms at a frequency of 1 kHz (with a 50% duty cycle).

(1) Using TMRA1, invert the value of the timer flip-flop TA1FF every 0.5 ms and output the inverted value on the Timer Flip-Flop Output pin TA1OUT.

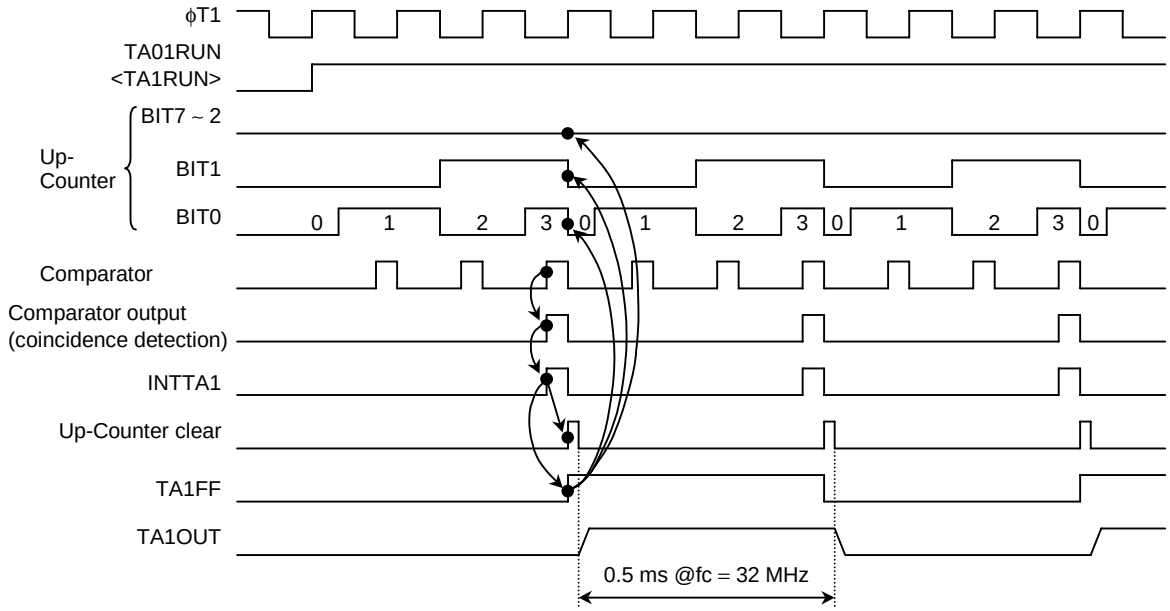


Figure 3.4.11 Square wave output timing chart (with 50% duty cycle)

(2) Timer Register calculation

Table 3.4.4 Prescaler output clock resolution

@fc = 32 MHz

Peripheral Clock Selection <FPSEL>	Clock Gear Value <GEAR1:0>	Selection of Prescaler Clock <PRCK1:0>	Minimum of Prescaler Output Clock Internal			
			φT1	φT4	φT16	φT256
0 (fgear)	00 (fc)	00 (fperiph/4)	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)	$fc/2^{11}$ (64 μs)
		01 (fperiph/2)	$fc/2^2$ (0.125 μs)	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)	$fc/2^{10}$ (32 μs)
		10 (fperiph)	—	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)	$fc/2^9$ (16 μs)
	01 (fc/2)	00 (fperiph/4)	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)	$fc/2^8$ (8.0 μs)	$fc/2^{12}$ (128 μs)
		01 (fperiph/2)	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)	$fc/2^{11}$ (64 μs)
		10 (fperiph)	—	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)	$fc/2^{10}$ (32 μs)
	10 (fc/4)	00 (fperiph/4)	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)	$fc/2^9$ (16 μs)	$fc/2^{13}$ (256 μs)
		01 (fperiph/2)	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)	$fc/2^8$ (8.0 μs)	$fc/2^{12}$ (128 μs)
		10 (fperiph)	—	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)	$fc/2^{11}$ (64 μs)
	11 (fc/8)	00 (fperiph/4)	$fc/2^6$ (2.0 μs)	$fc/2^8$ (8.0 μs)	$fc/2^{10}$ (32 μs)	$fc/2^{14}$ (512 μs)
		01 (fperiph/2)	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)	$fc/2^9$ (16 μs)	$fc/2^{13}$ (256 μs)
		10 (fperiph)	—	$fc/2^6$ (2.0 μs)	$fc/2^8$ (8.0 μs)	$fc/2^{12}$ (128 μs)
1 (fc)	00 (fc)	00 (fperiph/4)	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)	$fc/2^{11}$ (64 μs)
		01 (fperiph/2)	$fc/2^2$ (0.125 μs)	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)	$fc/2^{10}$ (32 μs)
		10 (fperiph)	—	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)	$fc/2^9$ (16 μs)
	01 (fc/2)	00 (fperiph/4)	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)	$fc/2^{11}$ (64 μs)
		01 (fperiph/2)	—	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)	$fc/2^{10}$ (32 μs)
		10 (fperiph)	—	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)	$fc/2^9$ (16 μs)
	10 (fc/4)	00 (fperiph/4)	—	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)	$fc/2^{11}$ (64 μs)
		01 (fperiph/2)	—	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)	$fc/2^{10}$ (32 μs)
		10 (fperiph)	—	—	$fc/2^5$ (1.0 μs)	$fc/2^9$ (16 μs)
	11 (fc/8)	00 (fperiph/4)	—	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)	$fc/2^{11}$ (64 μs)
		01 (fperiph/2)	—	—	$fc/2^6$ (2.0 μs)	$fc/2^{10}$ (32 μs)
		10 (fperiph)	—	—	$fc/2^5$ (1.0 μs)	$fc/2^9$ (16 μs)

Note 1: The prescaler's output clock φTn must be selected such that the relationship φTn < fsys/2 is satisfied (i.e. φTn must be slower than fsys/2).

Note 2: Do not change the clock gear value while the timer is operating.

Note 3: The dash character, —, in the table indicates a prohibited setting.

For φT1 ($fc/2^3$) ÷

1000 μs ÷ 0.25 μs ÷ 2 = 2000 (7D0H)

Cannot be set with 8-Bit Timer.

For φT4 ($fc/2^5$)

1000 μs ÷ 1 μs ÷ 2 = 500 (1F4H)

Cannot be set with 8-Bit Timer.

For φT16 ($fc/2^7$)

1000 μs ÷ 4.0 μs ÷ 2 = 125 (7DH)

Set TA1REG to 7DH.

For φT256 ($fc/2^{11}$)

1000 μs ÷ 64 μs ÷ 2 = 7.8125 (08H)

Set TA1REG to 08H. In this case the period is 1024 μs, with a 2.4% margin of error.

3.4.6.2 Method for control of beep tone output

■ Initial settings

		7	6	5	4	3	2	1	0	
P7CR	←	-	-	-	-	-	-	1	-	Set P71 to be TA1OUT output pin.
P7FC	←	-	-	-	-	-	-	1	-	
TA01RUN	←	0	X	X	X	0	-	0	-	Stop TMRA1 and clear it to 0.
TA01MOD	←	0	0	0	0	1	0	0	0	Select 8-Bit Timer Mode and set input clock to φT16 (4 μs @fc = 32 MHz).
TA1REG	←	0	1	1	1	1	1	0	1	Set TA1REG to 1000 μs ÷ φT16 ÷ 2 = 125.
TA1FFCR	←	X	X	X	X	1	0	1	1	Clear TA1FF to 0 and set it to be inverted by the coincidence detection signal from TMRA1.
IMC5L	←	X	X	-	-	-	-	-	- L	Set INTTA1 for rising edge and to level 4.
		X	X	1	1	0	1	0	0 H	

Note 1: X denotes Don't care; "–" denotes No change.
Note 2: The interrupt level can be set to any desired value.

■ Beep tone processing

Using an INTTA1 interrupt count the time for which the beep sounds. When 100 ms has elapsed, stop the timer and turn the beep tone off.

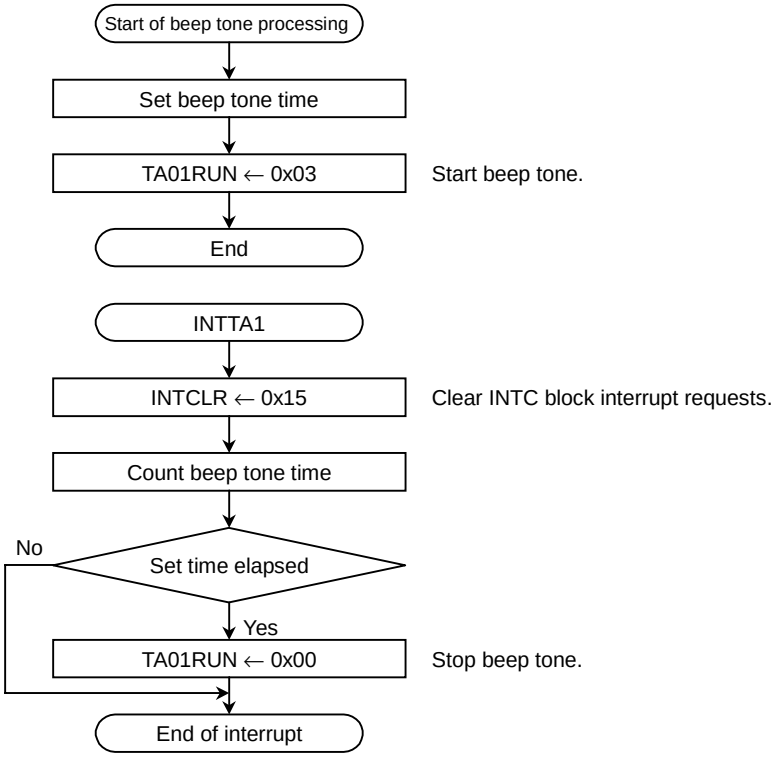


Figure 3.4.12 Beep tone processing flow

3.4.7 Controlling the time counter

3.4.7.1 Overview of the time counter

- An interrupt for counting the time is generated every 0.5 seconds.
- The count can go up to 59 minutes 59 seconds before wrapping around to 0 minutes 0 seconds.

3.4.7.2 Controlling the time counter

■ Initial settings

		7	6	5	4	3	2	1	0		
RTCCR	←	0	X	X	X	0	0	0	0		Stops and clears the Timer for RTC.
IMCGB3	←	X	X	1	1	X	X	X	1		Sets INTRTC for rising edge and to level 5.
IMCEH	←	X	X	0	1	0	1	0	1	L	
		X	X	-	-	-	-	-	-	H	
EICRCG	←	X	X	X	X	X	1	1	1		Clears the INTRTC interrupt latch.
INTCLR	←	X	X	1	1	1	0	1	0		

Note 1: The interrupt level can be set to any desired value.
Note 2: X denotes Don't care; "–" denotes No change.
Note 3: When setting the Timer for RTC interrupt, be sure to set the active state to rising edge.

To use the INTRTC, the CG block must also be set up.

■ Time counter

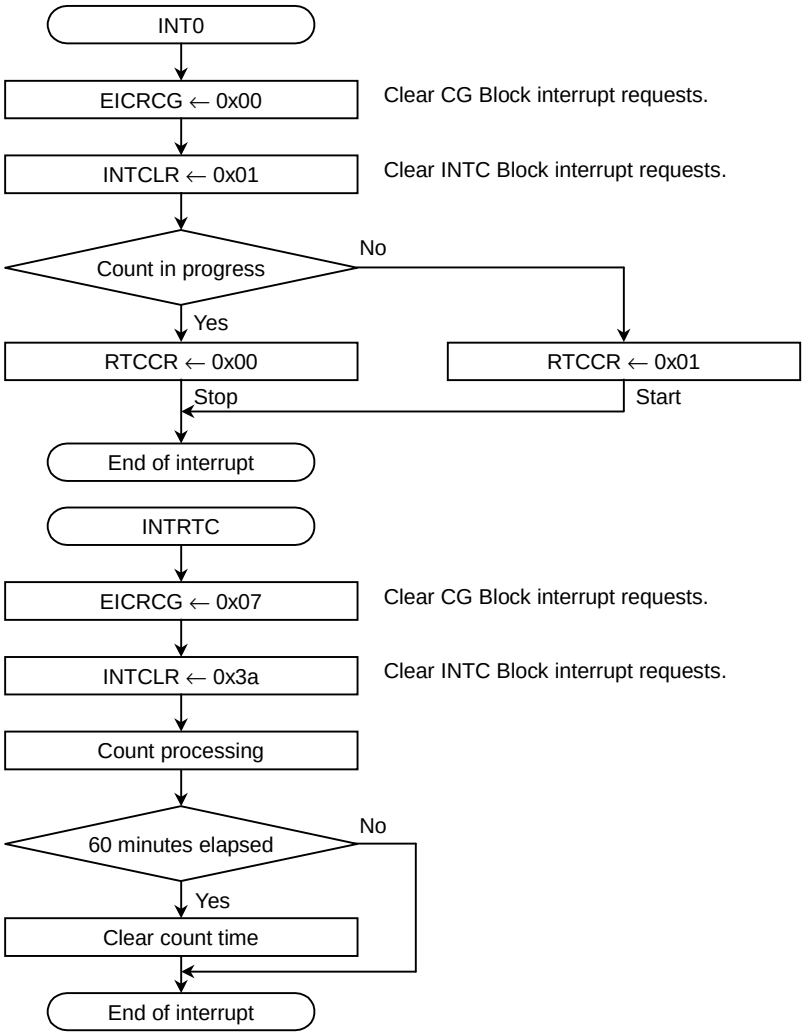


Figure 3.4.13 Flowchart for time control

3.4.8 2-ms Interval Timer

3.4.8.1 Overview of the 2-ms Interval Timer

- Using the 16-Bit Timer (TMRB), generate an interrupt every 2 ms.
- Set the interval time in the Timer Register TB2RG1 and generate INTTB21.

■ Timer Register calculation

The prescaler output resolutions needed for calculation are shown in Table 3.4.5.

Table 3.4.5 Prescaler output clock resolution

@fc = 32 MHz

Peripheral Clock Selection <FPSEL>	Clock Gear Value <GEAR1:0>	Selection of Prescaler Clock <PRCK1:0>	Resolution of Prescaler Output Clock		
			φT1	φT4	φT16
0 (gear)	00 (fc)	00 (fperiph/4)	fc/2^3 (0.25 μs)	fc/2^5 (1.0 μs)	fc/2^7 (4.0 μs)
		01 (fperiph/2)	fc/2^2 (0.125 μs)	fc/2^4 (0.5 μs)	fc/2^6 (2.0 μs)
		10 (fperiph)	—	fc/2^3 (0.25 μs)	fc/2^5 (1.0 μs)
	01 (fc/2)	00 (fperiph/4)	fc/2^4 (0.5 μs)	fc/2^6 (2.0 μs)	fc/2^8 (8.0 μs)
		01 (fperiph/2)	fc/2^3 (0.25 μs)	fc/2^5 (1.0 μs)	fc/2^7 (4.0 μs)
		10 (fperiph)	—	fc/2^4 (0.5 μs)	fc/2^6 (2.0 μs)
	10 (fc/4)	00 (fperiph/4)	fc/2^5 (1.0 μs)	fc/2^7 (4.0 μs)	fc/2^9 (16 μs)
		01 (fperiph/2)	fc/2^4 (0.5 μs)	fc/2^6 (2.0 μs)	fc/2^8 (8.0 μs)
		10 (fperiph)	—	fc/2^5 (1.0 μs)	fc/2^7 (4.0 μs)
	11 (fc/8)	00 (fperiph/4)	fc/2^6 (2.0 μs)	fc/2^8 (8.0 μs)	fc/2^{10} (32 μs)
		01 (fperiph/2)	fc/2^5 (1.0 μs)	fc/2^7 (4.0 μs)	fc/2^9 (16 μs)
		10 (fperiph)	—	fc/2^6 (2.0 μs)	fc/2^8 (8.0 μs)
1 (fc)	00 (fc)	00 (fperiph/4)	fc/2^3 (0.25 μs)	fc/2^5 (1.0 μs)	fc/2^7 (4.0 μs)
		01 (fperiph/2)	fc/2^2 (0.125 μs)	fc/2^4 (0.5 μs)	fc/2^6 (2.0 μs)
		10 (fperiph)	—	fc/2^3 (0.25 μs)	fc/2^5 (1.0 μs)
	01 (fc/2)	00 (fperiph/4)	fc/2^3 (0.25 μs)	fc/2^5 (1.0 μs)	fc/2^7 (4.0 μs)
		01 (fperiph/2)	—	fc/2^4 (0.5 μs)	fc/2^6 (2.0 μs)
		10 (fperiph)	—	fc/2^3 (0.25 μs)	fc/2^5 (1.0 μs)
	10 (fc/4)	00 (fperiph/4)	—	fc/2^5 (1.0 μs)	fc/2^7 (4.0 μs)
		01 (fperiph/2)	—	fc/2^4 (0.5 μs)	fc/2^6 (2.0 μs)
		10 (fperiph)	—	—	fc/2^5 (1.0 μs)
	11 (fc/8)	00 (fperiph/4)	—	fc/2^5 (1.0 μs)	fc/2^7 (4.0 μs)
		01 (fperiph/2)	—	—	fc/2^6 (2.0 μs)
		10 (fperiph)	—	—	fc/2^5 (1.0 μs)

Note 1: The prescaler's output clock φTn must be selected such that the relationship φTn < fsys/2 is satisfied (i.e. φTn must be slower than fsys/2).
Note 2: Do not change the clock gear value while the timer is operating.
Note 3: The dash character, —, in the table indicates a prohibited setting.

For φT1 (fc/2^3)

2000 μs ÷ 0.25 μs = 8000 (1F40H)

Set TB2RG1L to 00H and TB2RG1H to 1FH.

For φT4 (fc/2^5)

2000 μs ÷ 1.0 μs = 2000 (07D0H)

Set TB2RG1L to D0H and TB2RG1H to 07H.

For ϕT16 ($fc/2^7$)

2000 μs ÷ 4.0 μs = 500 (01F4H)

Set TB2RG1L to F4H and TB2RG1H to 01H.

3.4.8.2 Method for controlling the 2-ms Interval Timer

■ Initial settings

		7	6	5	4	3	2	1	0	
										Stop TMRB2.
TB2RUN	←	0	0	X	X	0	0	0	0	Disable trigger.
TB2FFCR	←	1	1	0	0	0	0	1	1	Select prescaler output clock as input clock and disable
TB2MOD	←	0	0	1	0	0	1	1	1	capture function.
										Set interval time.
TB2RG1L	←	1	1	1	1	0	1	0	0	Set TB2RG1 to 2000 μs ÷ ϕT16 = 500.
TB2RG1H	←	0	0	0	0	0	0	0	1	
IMC8L	←	X	X	-	-	-	-	-	-	Set INTTB21 for rising edge and to level 4.
		X	X	1	1	0	0	1	1	
TB2RUN	←	0	0	X	X	0	1	0	1	Start TMRB2.

Note 1: X denotes Don't care; "–" denotes No change.

Note 2: The interrupt level can be set to any desired value.

- Always set the eight low-order data bits in the Timer Register first, followed by the eight high-order bits.
- In the sample program AD conversion key chattering-elimination processing and 7-segment LED display output processing is performed using an INTTB21 interrupt.

3.4.9 Controlling standby

3.4.9.1 SLEEP Mode

The processor stops running and only the internal low-speed oscillator and the Timer for RTC continue to operate. To exit this mode use a reset, an NMI interrupt or a CG Block interrupt. In the sample program only a reset or an NMI interrupt may be used to exit SLEEP Mode so that only the Timer for RTC operates.

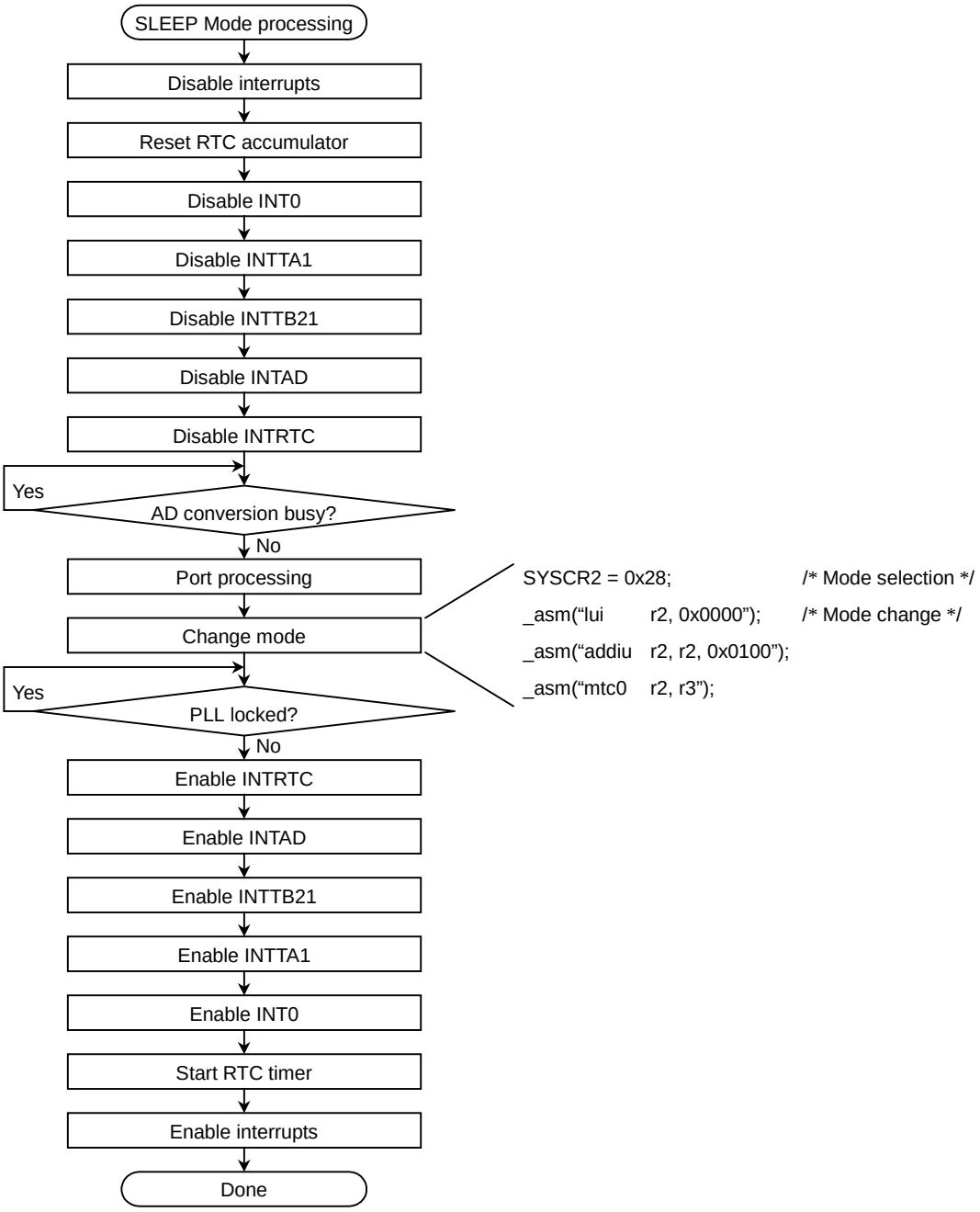


Figure 3.4.14 Flow of control in SLEEP Mode

3.4.9.2 IDLE Mode

The processor stops running and the internal I/O devices operate or halt as determined by the corresponding register bits. Each module has in its internal registers one bit which determines whether the module will continue to operate or stop operating in IDLE Mode.

Caution: The RTC cannot be set to continue to operate or stop operating in IDLE Mode.

To exit this mode use a reset, an NMI interrupt, a CG Block interrupt or an INTC Block interrupt. In the sample program only a RESET or an NMI interrupt may be used to exit IDLE Mode.

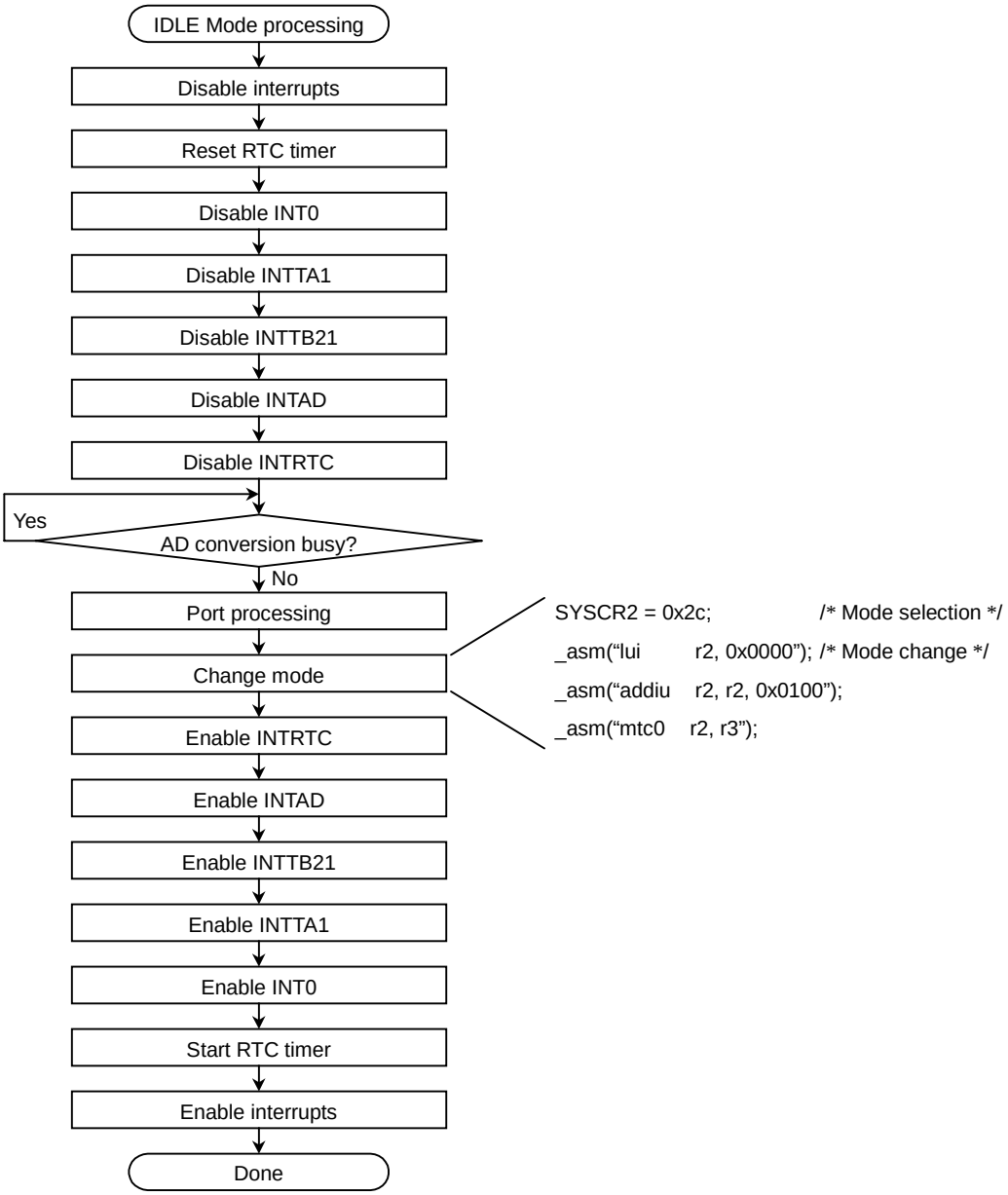


Figure 3.4.15 Flow of control in IDLE Mode

3.4.9.3 STOP Mode

The processor stops running and all internal I/O devices also stop operating. To exit this mode use a reset, an NMI interrupt or a CG Block interrupt. In the sample program only a RESET or an NMI interrupt may be used to exit STOP Mode.

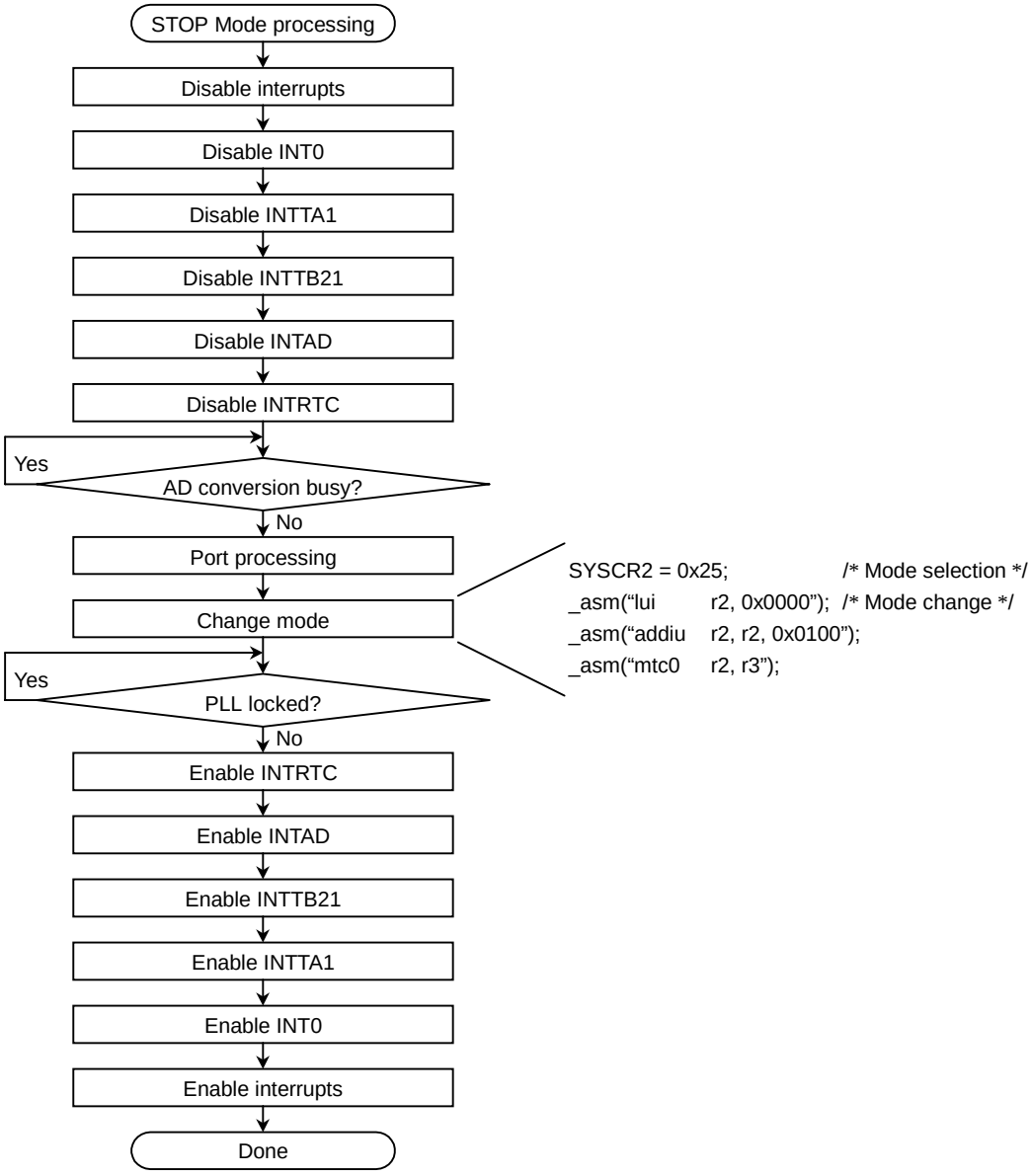


Figure 3.4.16 Flow of control in STOP Mode

3.4.10 Sample Program

3.4.10.1 Generic flowchart

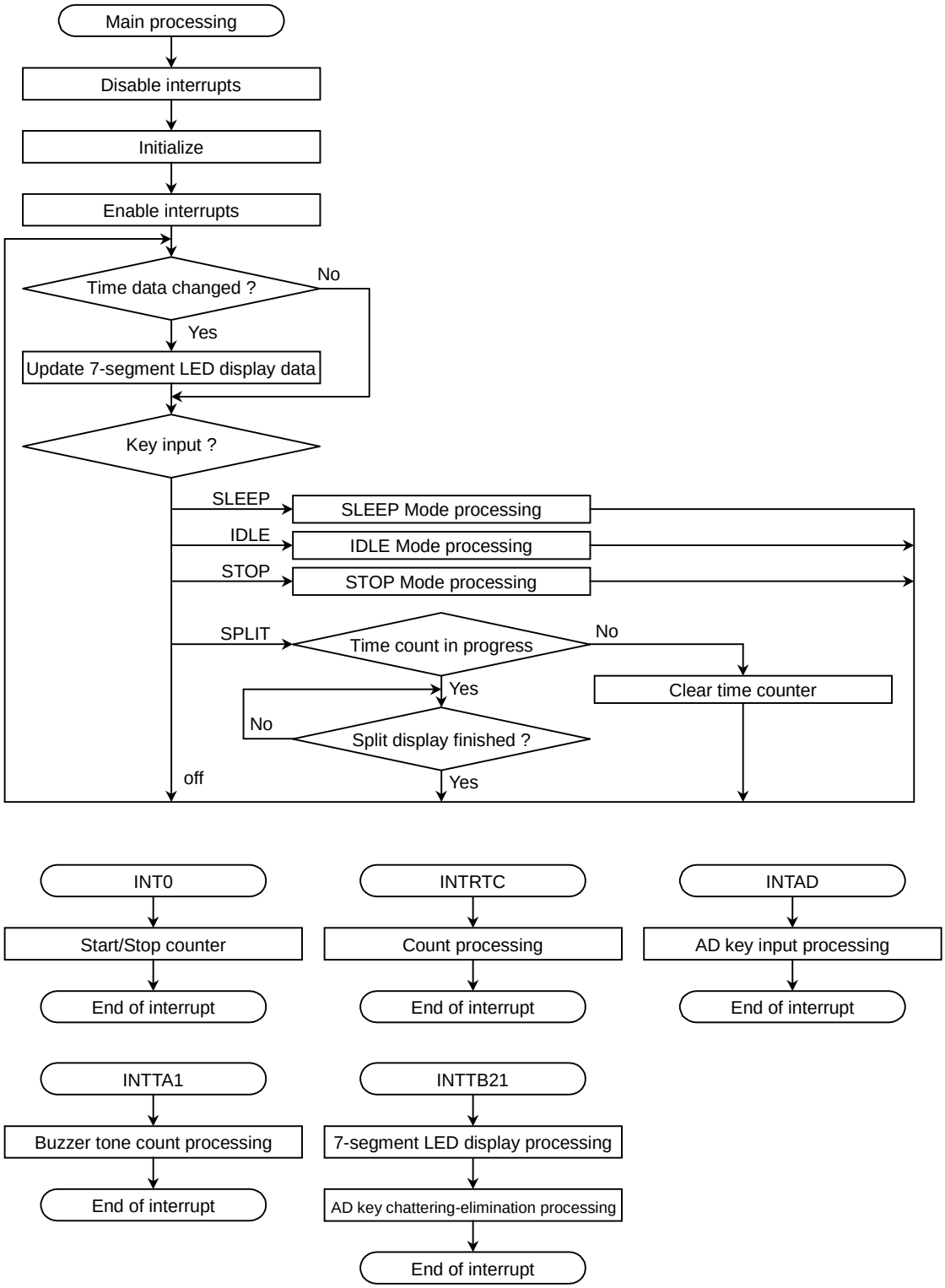


Figure 3.4.17 Generalized flow of control in Stopwatch Mode

3.4.10.2 File configuration

Table 3.4.6 below lists the files used in the sample programs.

Table 3.4.6 File configuration

Filename	Contents	Pages References
stopwatch.c	Stopwatch Mode main section	3-140
func_stopwatch.c	Stopwatch Mode main functional section	3-147
led.h	7-segment LED display definitions	3-151
key.h	Key definitions	3-152
Stc9i16.asm	Start-up routine	3-12
io1940.c	Special function registers	3-51
io1940.h	Special function register definitions	3-28
Bit32def.h	32-bit macro definitions	3-42
adc.h	AD conversion register definitions	3-31
tmr.h	Timer register definitions	3-49

3.4.10.3 Vector table

The vector table used in the sample programs is shown below. Replace the vector table section in the start-up routine with the vector table given below.

- Interrupt vector table section

```
_VecTable:
      dw      _Int_dummy      ;0 --- software set
      dw      _mint0          ;1 --- INT[0]
      dw      _Int_dummy      ;2 --- INT[1]
      dw      _Int_dummy      ;3 --- INT[2]
      dw      _Int_dummy      ;4 --- INT[3]
      dw      _Int_dummy      ;5 --- INT[4]
      dw      _Int_dummy      ;6 --- *
      dw      _Int_dummy      ;7 --- *
      dw      _Int_dummy      ;8 --- *
      dw      _Int_dummy      ;9 --- *
      dw      _Int_dummy      ;10--- INT[5]
      dw      _Int_dummy      ;11--- INT[6]
      dw      _Int_dummy      ;12--- INT[7]
      dw      _Int_dummy      ;13--- INT[8]
      dw      _Int_dummy      ;14--- INT[9]
      dw      _Int_dummy      ;15--- INT[A]
      dw      _Int_dummy      ;16--- *
      dw      _Int_dummy      ;17--- *
      dw      _Int_dummy      ;18--- *
      dw      _Int_dummy      ;19--- *
      dw      _Int_dummy      ;20--- INTTA0 : 8bit Timer 0
      dw      _mintta1        ;21--- INTTA1 : 8bit Timer 1
      dw      _Int_dummy      ;22--- INTTA2 : 8bit Timer 2
      dw      _Int_dummy      ;23--- INTTA3 : 8bit Timer 3
      dw      _Int_dummy      ;24--- *
      dw      _Int_dummy      ;25--- *
      dw      _Int_dummy      ;26--- *
      dw      _Int_dummy      ;27--- *
      dw      _Int_dummy      ;28--- INTTB00 :16bit Timer 0 (TB0RG0)
      dw      _Int_dummy      ;29--- INTTB01 :16bit Timer 0 (TB0RG1)
      dw      _Int_dummy      ;30--- INTTB10 :16bit Timer 1 (TB1RG0)
      dw      _Int_dummy      ;31--- INTTB11 :16bit Timer 1 (TB1RG1)
      dw      _Int_dummy      ;32--- INTTB20 :16bit Timer 2 (TB2RG0)
      dw      _minttb21       ;33--- INTTB21 :16bit Timer 2 (TB2RG1)
      dw      _Int_dummy      ;34--- INTTN30 :16bit Timer 3 (TB3RG0)
```

```
dw      _Int_dummy      ;35--- INTTB31 :16bit Timer 3 (TB3RG1)
dw      _Int_dummy      ;36--- *
dw      _Int_dummy      ;37--- *
dw      _Int_dummy      ;38--- *
dw      _Int_dummy      ;39--- *
dw      _Int_dummy      ;40--- INTTBOF0:16bit Timer 0 (OverFlow)
dw      _Int_dummy      ;41--- INTTBOF1:16bit Timer 1 (OverFlow)
dw      _Int_dummy      ;42--- INTTBOF2:16bit Timer 2 (OverFlow)
dw      _Int_dummy      ;43--- INTTBOF3:16bit Timer 3 (OverFlow)
dw      _Int_dummy      ;44--- *
dw      _Int_dummy      ;45--- *
dw      _Int_dummy      ;46--- *
dw      _Int_dummy      ;47--- *
dw      _Int_dummy      ;48--- INTRX0 :Serial receive (Channel 0)
dw      _Int_dummy      ;49--- INTTX0 :Serial transmit (Channel 0)
dw      _Int_dummy      ;50--- INTRX1 :Serial receive (Channel 1)
dw      _Int_dummy      ;51--- INTTX1 :Serial transmit (Channel 1)
dw      _Int_dummy      ;52--- INTS2  :Serial Channel 2 interrupt
dw      _Int_dummy      ;53--- *
dw      _Int_dummy      ;54--- INTRX3 :Serial receive (Channel 3)
dw      _Int_dummy      ;55--- INTTX3 :Serial transmit (Channel 3)
dw      _Int_dummy      ;56--- INTRX4 :Serial receive (Channel 4)
dw      _Int_dummy      ;57--- INTTX4 :Serial transmit (Channel 4)
dw      _mintrtc        ;58--- INTRTC :Timer for RTC interrupt
dw      _mintad         ;59--- INTAD  :AD conversion finished
dw      _Int_dummy      ;60--- INTDMA0:DMA transfer finished (Channel 0)
dw      _Int_dummy      ;61--- INTDMA1:DMA transfer finished (Channel 1)
dw      _Int_dummy      ;62--- INTDMA2:DMA transfer finished (Channel 2)
dw      _Int_dummy      ;63--- INTDMA3:DMA transfer finished (Channel 3)
```

3.4.10.4 Source code

- Filename: stopwatch.c

```
/*
        ************************************************
        **              Application Note              **
        **                                            **
        **           ( STOP WATCH ROUTINE )           **
        **                                            **
        **                   MCU                      **
        **                  TX1940                    **
        **                                            **
        **              fc       = 32MHz              **
        **              fsys     = 32MHz              **
        **              fperiph  = 32MHz              **
        **              ϕT0      = 8MHz               **
        **                                            **
        **                  2000/2/3                  **
        **                                            **
        ************************************************
        ************************************************
        *      COPYRIGHT(C) 1999 TOSHIBA CORPORATION     *
        *              ALL RIGHTS RESERVED               *
        ************************************************/

/***********************************************/
/*              Loading header files           */
/***********************************************/
#include "io1940.h"                             /* Special registers */
#include "cg.h"                                 /* CG */
#include "adc.h"                                /* AD conversion */
#include "tmr.h"                                /* Timers */
#include "led.h"                                /* Display definitions */
#include "key.h"                                /* Key definitions */
#include <stdlib.h>

/***********************************************/
/*              Constant definitions           */
/***********************************************/
/*--- Mode definitions ---*/
#define cmd_watch_run           0               /* Normal Mode */
#define cmd_watch_split         1               /* Split Mode */
/*--- Key definitions ---*/
#define  csplit_key             cadkey_1        /* Split key */
#define  csleep_key             cadkey_2        /* Sleep operation key */
#define  cidle_key              cadkey_3        /* Idle operation key  */
#define  cstop_key              cadkey_4        /* Stop operation key  */

/***********************************************/
/*               extern declaration            */
/***********************************************/
extern   void padkey_in(void);
extern   void padkey_chat(void);
extern   void padkey_decode(void);
extern   void p7seg_disp(void);
extern   void pbeep_start(unsigned short);
extern   void pbeep_timer(void);

extern unsigned char     g7seg_data[4];
extern unsigned char     gled_data;
extern unsigned char     gadkey_code;
extern unsigned char     fadkey_push;
extern unsigned char     t7seg_ch[];
```

```
extern unsigned char     tled_ch1[];

/*********************************************/
/*                    RAM                    */
/*********************************************/
unsigned char            gmd_watch;            /* Mode data */
unsigned short           gwatch_time;          /* Time data */

/*********************************************/
/*                    CODE                   */
/*********************************************/

/*****************************************************************************/
/* Module name  :mwatch_init                                                 */
/*****************************************************************************/
/* Function     :Initializes stopwatch                                       */
/* Inputs       :None                                                        */
/* Outputs      :SFR                                                         */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/*****************************************************************************/
void mwatch_init(void){

/*--- CG settings ---*/
         IO_SYSCR0     = 0xf0;    /* <XEN>=1 <XTEN>=1 <RXEN>=1 <RXTEN>=1 */
                                  /* <RSYSCK>=0 <WUEF>=0 <PRCK>=00 */
         IO_SYSCR1     = 0x10;    /* <SYSCK>=0 <FPSEL>=1 <DFOSC>=0 <GEAR>=00 */
         IO_ADCCLK     = 0x01;    /* <ADCCK>=01 */

/*--- I/O port settings ---*/
         IO_P0         = 0xff;    /* Output High on P00-P07 */
         IO_P0CR       = 0xff;    /* Set P00-P07 for output */

         IO_P1         = 0xff;    /* Output High on P10-P17 */
         IO_P1CR       = 0xff;    /* Set P10-P17 for output */
         IO_P1FC       = 0x00;    /* Set P10-P17 to be a port */

         IO_P2         = 0xff;    /* Output High on P20-P27 */
         IO_P2CR       = 0xff;    /* Set P20-P27 for output */
         IO_P2FC       = 0x00;    /* Set P20-P27 to be a port */

         IO_P3         = 0xff;    /* Output High on P30-P31 and set P32-P37 for pull-up */
         IO_P3CR       = 0x00;    /* Set P30-P31 for output and P32-P37 for input */
         IO_P3FC       = 0x00;    /* Set P30-P37 to be a port */

         IO_P4         = 0x1f;    /* Set P40-P43 for pull-up and output High on P44 */
         IO_P4CR       = 0x10;    /* Set P40-P43 for input and P44 for output */
         IO_P4FC       = 0x00;    /* Set P40-P44 to be a port */

         IO_P7         = 0x65;    /* Output High on P70,P72,P75,P76 */
         IO_P7CR       = 0x67;    /* Set P70,P71,P72,P75,P76 for output and P73,P74,P77 for
                                     input */
         IO_P7FC       = 0x02;    /* Set P71 to be TA1OUT output pin and set others to be
                                     ports */

         IO_P8         = 0xff;    /* Output High on P80-P87 */
         IO_P8CR       = 0xff;    /* Set P80-P87 for output */
         IO_P8FC       = 0x00;    /* Set P80-P87 to be a port */

         IO_P9         = 0x3f;    /* Output Low on P96-P97 */
         IO_P9CR       = 0xc0;    /* Set P90-P95 for input and P96-P97 for output */
         IO_P9FC       = 0x00;    /* Set P90-P97 to be a port */

         IO_PA         = 0x1f;    /* Output High on PA0-PA4 */
         IO_PACR       = 0x1f;    /* Set PA0-PA4 for output and PA5-PA7 for input */
         IO_PAFC       = 0x00;    /* Set PA0-PA7 to be a port */
```

```
/*--- INT0 switch settings ---*/
        IO_IMCGA0     = 0x21;    /* ↓ LEVEL6 */
        IO_IMC0L      = 0x1630;  /*  */
        IO_EICRCG     = 0x00;    /* Clear interrupt latch */
        IO_INTCLR     = 0x01;

/*--- AD conversion settings ---*/
        IO_ADMOD0     = 0x00;    /* <ITM0L>=0 <REPERT>=0 <SCAN>=0 <ADS>=0 */
        IO_ADMOD1     = 0x82;    /* <VREFON>=1 <I2AD>=0 <ADTRGE>=0 <ADCH>=010 */
        IO_IMCEH      = 0x3210;  /* ↑ LEVEL2 */

/*--- Timer for RTC settings ---*/
        IO_RTCCR      = 0x00;    /* Stop and clear Timer for RTC */
        IO_IMCGB3     = 0x31;    /* LEVEL5 */
        IO_IMCEH     |= 0x0015;  /*  */
        IO_EICRCG     = 0x07;    /* Clear interrupt latch */
        IO_INTCLR     = 0x3a;

/*--- Timer settings ---*/
        IO_TB2RUN     = 0x00;    /* <TB2RDE>=0 <I2TB2>=0 <TB2PRUN>=0 <TB2RUN>=0 */
        IO_TB2FFCR    = 0xc3;    /* <TB2C1T1>=0 <TB2C0T1>=0 <TB2E1T1>=0 */
                                 /* <TB2E0T1>=0 <TB2FF0C>=11 */
        IO_TB2MOD     = 0x27;    /* <TB2CP0I>=1 <TB2CPM>=00 <TB2CLE>=1 <TB2CLK>=11 */
        IO_TB2RG1L    = 0xf4;    /* TB2RG1 = 2000 / 4 */
        IO_TB2RG1H    = 0x01;    /*  */
        IO_IMC8L      = 0x3330;  /* ↑ LEVEL3 */

/*--- Buzzer settings ---*/
        IO_TA01RUN = 0x00;       /* <TA0RDE>=0 <I2TA01>=0 <TA01PRUN>=0 */
                                 /* <TA1RUN>=0 <TA0RUN>=0 */
        IO_TA01MOD    = 0x08;    /* <TA01M>=00 <PWM0>=00 <TA1CLK>=10 <TA0CLK>=00 */
        IO_TA1REG     = 0x7d;    /* TA1REG = 500 / 4 */
        IO_TA1FFCR    = 0x0b;    /* <TAFF1C>=10 <TAFF1IE>=1 <TAFF1IS>=1 */
        IO_IMC5L      = 0x3430;  /* ↑ LEVEL4 */

/*--- Data initialization ---*/
        g7seg_data[0] = c7seg_0;
        g7seg_data[1] = c7seg_0;
        g7seg_data[2] = c7seg_0;
        g7seg_data[3] = c7seg_spc;
        gled_data     = 0xfe;
        gwatch_time   = 0;
        gmd_watch     = cmd_watch_run;

/*--- Timer start ---*/
        IO_TB2RUN     = 0x05;    /* <TB2RDE>=0 <I2TB2>=0 <TB2PRUN>=1 <TB2RUN>=1 */
}

/*****************************************************************************/
/* Module name  :pwatch_sleep                                                */
/*****************************************************************************/
/* Function     :SLEEP Mode processing                                       */
/* Inputs       :ADMOD0                                                      */
/* Outputs      :RTCCR,IMCGA0,IMC0L,IMCGB3,IMCEH,P8,PA,SYSCR2,EICRCG,INTCLR  */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/* ************************************************************************/
void __isa32 pwatch_sleep(void){
        __DI();
        IO_RTCCR      = 0x01;                /* Reset RTC accumulator */
        IO_IMC0L     &= 0xf0ff;              /* Disable INT0 */
        IO_IMCGA0     = 0x20;
        IO_IMC5L     &= 0xf0ff;              /* Disable INTTA1 */
        IO_IMC8L     &= 0xf0ff;              /* Disable INTTB21 */
        IO_IMCEH     &= 0xf0ff;              /* Disable INTAD */
```

```
        IO_IMCEH     &= 0xfff0;                /* Disable INTRTC */
        IO_IMCGB3     = 0x30;
        while ((IO_ADMOD0 & ADBF) != 0x00);     /* Check AD conversion */
        IO_P8         = 0xff;                   /* Port processing */
        IO_PA         = 0x1f;
        IO_SYSCR2     = 0x28;                   /* Select mode */
        __SYNC();
        __asm("lui    r2,0x0000");              /* Change mode */
        __asm("addiu  r2,r2,0x0100");
        __asm("mtc0   r2,r3");
        __asm("nop");                           /* Dummy instructions */
        while ((IO_SYSCR3 & LUPFG) != 0x00);
        IO_IMCGB3     = 0x31;                   /* Enable INTRTC */
        IO_IMCEH      = (IO_IMCEH & 0xff00) | 0x0015;
        IO_EICRCG     = 0x07;
        IO_INTCLR     = 0x3a;
        IO_IMCEH      = (IO_IMCEH & 0x00ff) | 0x3200;  /* Enable INTAD */
        IO_IMC8L      = (IO_IMC8L & 0x00ff) | 0x3300;  /* Enable INTTB21 */
        IO_IMC5L      = (IO_IMC5L & 0x00ff) | 0x3400;  /* Enable INTTA1 */
        IO_IMCGA0     = 0x21;                   /* Enable INT0 */
        IO_IMC0L      = (IO_IMC0L & 0x00ff) | 0x1600;
        IO_EICRCG     = 0x00;
        IO_INTCLR     = 0x01;
        gwatch_time  += IO_RTCREG;              /* Increment RTC accumulator */
        __EI();
}

/*****************************************************************************/
/* Module name  :pwatch_idle                                                 */
/*****************************************************************************/
/* Function     :IDLE Mode processing                                        */
/* Inputs       :ADMOD0                                                      */
/* Outputs      :IMCGA0,IMC0L,RTCCR,EICRCG,INTCLR,P8,PA,SYSCR2               */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/*****************************************************************************/
void __isa32 pwatch_idle(void){
        __DI();
        IO_RTCCR      = 0x08;                   /* Stop Timer for RTC */
        IO_IMC0L     &= 0xf0ff;                 /* Disable INT0 */
        IO_IMCGA0     = 0x20;
        IO_IMC5L     &= 0xf0ff;                 /* Disable INTTA1 */
        IO_IMC8L     &= 0xf0ff;                 /* Disable INTTB21 */
        IO_IMCEH     &= 0xf0ff;                 /* Disable INTAD */
        IO_IMCEH     &= 0xfff0;                 /* Disable INTRTC */
        IO_IMCGB3     = 0x30;
        while ((IO_ADMOD0 & ADBF) != 0x00);     /* Check AD conversion */
        IO_P8         = 0xff;                   /* Port processing */
        IO_PA         = 0x1f;
        IO_SYSCR2     = 0x2c;                   /* Select mode */
        __SYNC();
        __asm("lui    r2,0x0000");/* Change mode */
        __asm("addiu  r2,r2,0x0100");
        __asm("mtc0   r2,r3");
        __asm("nop");                           /* Dummy instructions */
        IO_IMCGB3     = 0x31;                   /* Enable INTRTC */
        IO_IMCEH      = (IO_IMCEH & 0xff00) | 0x0015;
        IO_EICRCG     = 0x07;
        IO_INTCLR     = 0x3a;
        IO_IMCEH      = (IO_IMCEH & 0x00ff) | 0x3200;  /* Enable INTAD */
        IO_IMC8L      = (IO_IMC8L & 0x00ff) | 0x3300;  /* Enable INTTB21 */
        IO_IMC5L      = (IO_IMC5L & 0x00ff) | 0x3400;  /* Enable INTTA1 */
        IO_IMCGA0     = 0x21;                   /* Enable INT0 */
        IO_IMC0L      = (IO_IMC0L & 0x00ff) | 0x1600;
        IO_EICRCG     = 0x00;
        IO_INTCLR     = 0x01;
```

```
        IO_RTCCR      = 0x09;                   /* Start Timer for RTC */
        __EI();
}

/*****************************************************************************/
/* Module name  :pwatch_stop                                                 */
/*****************************************************************************/
/* Function     :STOP Mode processing                                        */
/* Inputs       :ADMOD0                                                      */
/* Outputs      :IMCGA0,IMC0L,P8,PA,SYSCR2,EICRCG,INTCLR                     */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/*****************************************************************************/
void __isa32 pwatch_stop(void){
        __DI();
        IO_IMC0L     &= 0xf0ff;                 /* Disable INT0 */
        IO_IMCGA0     = 0x20;
        IO_IMC5L     &= 0xf0ff;                 /* Disable INTTA1 */
        IO_IMC8L     &= 0xf0ff;                 /* Disable INTTB21 */
        IO_IMCEH     &= 0xf0ff;                 /* Disable INTAD */
        IO_IMCEH     &= 0xfff0;                 /* Disable INTRTC */
        IO_IMCGB3     = 0x30;
        while ((IO_ADMOD0 & ADBF) != 0x00);     /* Check AD conversion */
        IO_P8         = 0xff;                   /* Port processing */
        IO_PA         = 0x1f;
        IO_SYSCR2     = 0x25;                   /* Select mode */
        __SYNC();
        __asm("lui    r2,0x0000");              /* Change mode */
        __asm("addiu  r2,r2,0x0100");
        __asm("mtc0   r2,r3");
        __asm("nop");                           /* Dummy instructions */
        while ((IO_SYSCR3 & LUPFG) != 0x00);
        IO_IMCGB3     = 0x31;                   /* Enable INTRTC */
        IO_IMCEH      = (IO_IMCEH & 0xff00) | 0x0015;
        IO_EICRCG     = 0x07;
        IO_INTCLR     = 0x3a;
        IO_IMCEH      = (IO_IMCEH & 0x00ff) | 0x3200;  /* Enable INTAD */
        IO_IMC8L      = (IO_IMC8L & 0x00ff) | 0x3300;  /* Enable INTTB21 */
        IO_IMC5L      = (IO_IMC5L & 0x00ff) | 0x3400;  /* Enable INTTA1 */
        IO_IMCGA0     = 0x21;                   /* Enable INT0 */
        IO_IMC0L      = (IO_IMC0L & 0x00ff) | 0x1600;
        IO_EICRCG     = 0x00;
        IO_INTCLR     = 0x01;
        __EI();
}

/*****************************************************************************/
/* Module name  :main                                                        */
/*****************************************************************************/
/* Function     :Stopwatch Mode main processing                              */
/* Inputs       :gmd_watch,gwatch_time,RTCCR,ADMOD0,RTCREG,gadkey_code       */
/* Outputs      :g7seg_data,gled_data,gmd_watch,IMCGA0,IMCGB3,RTCCR,P8,PA,SYSCR2*/
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/*****************************************************************************/
void main(void){

        static unsigned short  lwatch_buff = 0;
        unsigned int lminute ,lsecond;

        __DI();

        mwatch_init();                          /* Initialize */

        __EI();
```

```
/*************************************/
/*                                   */
/*              Main loop            */
/*                                   */
/*************************************/

for(;;){
        padkey_decode();

        switch (gmd_watch) {
        case cmd_watch_run:
              /*--- Display update processing ---*/
              if (lwatch_buff != gwatch_time) {
                    lminute = gwatch_time / 120;      /* Calculate time in minutes */
                    lsecond = (gwatch_time / 2) % 60; /* Calculate time in seconds */
                    g7seg_data[0] = t7seg_ch[lsecond % 10];
                    g7seg_data[1] = t7seg_ch[lsecond / 10];
                    g7seg_data[2] = t7seg_ch[lminute % 10];
                    lminute /= 10;
                    if (lminute == 0) g7seg_data[3] = c7seg_spc;
                    else g7seg_data[3] = t7seg_ch[lminute];
                    gled_data = tled_ch1[gwatch_time & 0x0007];
                    lwatch_buff = gwatch_time;        /* Store time data */
              }
              switch (gadkey_code) {
              /*--- SPLIT & CLEAR processing ---*/
              case csplit_key:
                    pbeep_start(100);
                    if ((IO_RTCCR & RTCRUN) != 0x00) {
                          gled_data = 0x00;
                          gmd_watch = cmd_watch_split;
                    }
                    else gwatch_time = 0;
                    break;
              /*--- SLEEP Mode processing ---*/
              case csleep_key:
                    if ((IO_RTCCR & RTCRUN) != 0x00) {
                          pwatch_sleep();
                    }
                    break;
              /*--- IDLE Mode processing ---*/
              case cidle_key:
                    if ((IO_RTCCR & RTCRUN) != 0x00) {
                          pwatch_idle();
                    }
                    break;
              /*--- STOP Mode processing ---*/
              case cstop_key:
                    if ((IO_RTCCR & RTCRUN) != 0x00) {
                          pwatch_stop();
                    }
                    break;
              }
              break;
        case cmd_watch_split: /* SPLIT display wait processing */
              if (gadkey_code == csplit_key) {
                    pbeep_start(100);
                    gled_data = tled_ch1[gwatch_time & 0x0007];
                    gmd_watch = cmd_watch_run;
              }
              break;
        }
}
}

/*****************************************************************************/
```

```
/* Module name  :mintrtc                                                      */
/****************************************************************************/
/* Function     :Time counter processing (using Timer for RTC interrupt)     */
/* Inputs       :gwatch_time                                                  */
/* Outputs      :gwatch_time                                                  */
/* Parameters   :None                                                         */
/* Return value :None                                                         */
/****************************************************************************/
void __interrupt mintrtc(void){

        IO_EICRCG = 0x07;                           /* Clear interrupt latch */
        IO_INTCLR = 0x3a;

        gwatch_time++;                              /* Time count */
        if (gwatch_time == 7200) gwatch_time = 0;   /* Check for time overflow */
}

/****************************************************************************/
/* Module name  :mint0                                                        */
/****************************************************************************/
/* Function     :Time count start (using INT0 interrupt)                      */
/* Inputs       :None                                                         */
/* Outputs      :RTCCR                                                        */
/* Parameters   :None                                                         */
/* Return value :None                                                         */
/****************************************************************************/
void __interrupt mint0(void){

        IO_EICRCG = 0x00;                           /* Clear interrupt latch */
        IO_INTCLR = 0x01;

        if ((IO_RTCCR & RTCRUN) != 0x00) {
               IO_RTCCR = 0x00;                     /* Stop count */
               pbeep_start(100);
        }
        else {
               IO_RTCCR = 0x01;                     /* Start count */
               pbeep_start(100);
        }
}

/****************************************************************************/
/* Module name  :minttb21                                                     */
/****************************************************************************/
/* Function    :2-ms Interval Timer interrupt                                 */
/* Inputs      :None                                                          */
/* Outputs     :None                                                          */
/****************************************************************************/
void __interrupt minttb21(void){

        IO_INTCLR = 0x21;                           /* Clear interrupt latch */
        p7seg_disp();
        padkey_chat();
}

/****************************************************************************/
/* Module name :mintta1                                                       */
/****************************************************************************/
/* Function    :Beep Tone Timer interrupt                                     */
/* Inputs      :None                                                          */
/* Outputs     :None                                                          */
/****************************************************************************/
void __interrupt mintta1(void){

        IO_INTCLR = 0x15;                           /* Clear interrupt latch */
        pbeep_timer();
```

```
}

/************************************************************************/
/* Module name  :mintad                                                 */
/************************************************************************/
/* Function     :AD Conversion-Finished interrupt                       */
/* Inputs       :None                                                   */
/* Outputs      :None                                                   */
/************************************************************************/
void __interrupt mintad(void){

        IO_INTCLR = 0x3b;                           /* Clear interrupt latch */
        padkey_in();
}

/************************************************************************/
/* Module name  :mnmi                                                   */
/************************************************************************/
/* Function     :NMI interrupt                                          */
/* Inputs       :None                                                   */
/* Outputs      :None                                                   */
/************************************************************************/
void __interrupt mnmi(void){

        __asm("NOP");

}
```

- Filename: func_stopwatch.c

```
/*
        ************************************************
        **              Application Note              **
        **                                            **
        **        ( STOP WATCH FUNCTION ROUTINE )     **
        **                                            **
        **                   MCU                      **
        **                  TX1940                    **
        **                                            **
        **              fc       = 32MHz              **
        **              fsys     = 32MHz              **
        **              fperiph  = 32MHz              **
        **              ϕT0      = 8MHz               **
        **                                            **
        **                  1999/9/20                 **
        **                                            **
        ************************************************
        ************************************************
        *      COPYRIGHT(C) 1999 TOSHIBA CORPORATION    *
        *              ALL RIGHTS RESERVED              *
        ************************************************/

/************************************************/
/*              Loading header files            */
/************************************************/
#include "io1940.h"
#include "adc.h"
#include "key.h"

/************************************************/
/*              Constant definitions            */
/************************************************/
/*--- Port output definitions(7SEG-LED digits) ---*/
```

```
#define  c7seg_digit_out0          0x1e          /* PA (00011110) */
#define  c7seg_digit_out1          0x1d          /* PA (00011101) */
#define  c7seg_digit_out2          0x1b          /* PA (00011011) */
#define  c7seg_digit_out3          0x17          /* PA (00010111) */
#define  c7seg_digit_out4          0x0f          /* PA (00001111) */

/*********************************************/
/*                    RAM                    */
/*********************************************/
unsigned char            g7seg_data[4];          /* 7-segment LED display data */
unsigned char            gled_data;              /* LED display data */
unsigned char            gadkey_data;            /* AD key data */
unsigned char            gadkey_code;            /* AD key confirmed data */
unsigned char            fadkey_push;            /* AD Key Depression Status flag */
unsigned char            fadkey_ok;              /* AD Key Confirmed flag */
unsigned short           gbeep_cnt;              /* Beep tone count */

/*********************************************/
/*                    CODE                   */
/*********************************************/

/*****************************************************************************/
/* Module name  :padkey_in                                                   */
/*****************************************************************************/
/* Function     :AD key data input (using AD Conversion-Finished interrupt)  */
/* Inputs       :ADREG26H                                                    */
/* Outputs      :gadkey_data                                                 */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/*****************************************************************************/
void padkey_in(void) {

        unsigned char lad_data;

/*--- Entering AD conversion data ---*/
        lad_data = IO_ADREG26H;

/*--- Converting AD data into key data ---*/
        if (lad_data > 225) gadkey_data = 0;
        else if (lad_data > 175)    gadkey_data = 1;
        else if (lad_data > 125)    gadkey_data = 2;
        else if (lad_data >  75)    gadkey_data = 3;
        else if (lad_data >  25)    gadkey_data = 4;
        else                        gadkey_data = 5;
}

/*****************************************************************************/
/* Module name  :padkey_chat                                                 */
/*****************************************************************************/
/* Function     :AD key chattering elimination (using 2-ms interrupt)        */
/* Inputs       :gadkey_data                                                 */
/* Outputs      :ADMOD0 ,fadkey_ok ,fadkey_push                              */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/*****************************************************************************/
void padkey_chat(void) {

        static unsigned char ladkey_chat = 0 ,ladkey_buff = 0x00;

/*--- Eliminating chattering ---*/
        if (gadkey_data == ladkey_buff) {
                ladkey_chat++;
                if (ladkey_chat > 15) {
                        ladkey_chat = 0;
                        if (gadkey_data == 0x00) fadkey_push = 0;
                        else {
```

```
                                if (fadkey_push == 0) {
                                        fadkey_ok = 1;
                                        fadkey_push = 1;
                                }
                        }
                }
        }
        else ladkey_chat = 0;
/*--- Holding key data ---*/
        ladkey_buff = gadkey_data;
/*--- Starting AD decode ---*/
        IO_ADMOD0 |= ADS;
}

/*****************************************************************************/
/* Module name  :padkey_decode                                               */
/*****************************************************************************/
/* Function     :AD key decoding                                             */
/* Inputs       :gadkey_data ,fadkey_ok                                      */
/* Outputs      :gadkey_code ,fadkey_ok                                      */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/*****************************************************************************/
void padkey_decode(void) {

        gadkey_code = cadkey_no;
        if (fadkey_ok == 1) {
                fadkey_ok = 0;
                gadkey_code = gadkey_data;
        }
}

/*****************************************************************************/
/* Module name  :p7seg_disp                                                  */
/*****************************************************************************/
/* Function     :7-segment LED display output (using 2-ms interrupt)         */
/* Inputs       :gled_data,g7seg_data                                        */
/* Outputs      :P8,PA                                                       */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/*****************************************************************************/
void p7seg_disp(void){

        static unsigned char ldisp_digit = 0;

/*--- Turning display off ---*/
        IO_P8 = 0xff;

/*--- Display output ---*/
        switch (ldisp_digit) {
        case 0:
                IO_PA = c7seg_digit_out0;       /* Digit output */
                IO_P8 = gled_data;              /* LED output */
                        break;
        case 1:
                IO_PA = c7seg_digit_out1;       /* Digit output */
                IO_P8 = g7seg_data[0];          /* 7-segment LED output */
                break;
        case 2:
                IO_PA = c7seg_digit_out2;       /* Digit output */
                IO_P8 = g7seg_data[1];          /* 7-segment LED output */
                break;
        case 3:
                IO_PA = c7seg_digit_out3;       /* Digit output */
                IO_P8 = g7seg_data[2];          /* 7-segment LED output */
                break;
```

```
        case 4:
                IO_PA = c7seg_digit_out4;       /* Digit output */
                IO_P8 = g7seg_data[3];          /* 7-segment LED output */
                break;
        }

/*--- Digit count ---*/
        ldisp_digit++;
        if (ldisp_digit > 4) ldisp_digit = 0;
}

/*********************************************/
/*              Display data table           */
/*********************************************/
/*--- 7SEG-LED ---*/
const unsigned char t7seg_ch[] = {
        0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xd8,
        0x80,0x98,0x88,0x83,0xc6,0xa1,0x86,0x8e,
};
/*--- LED ---*/
const unsigned char tled_ch1[] = {
        0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f
};

/*****************************************************************************/
/* Module name  :pbeep_start                                                 */
/*****************************************************************************/
/* Function     :Beep tone start                                             */
/* Inputs       :None                                                        */
/* Outputs      :gbeep_cnt,TA01RUN                                           */
/* Parameters   :Beep tone time                                              */
/* Return value :None                                                        */
/*****************************************************************************/
void pbeep_start(unsigned short lbeep_on_time){

        gbeep_cnt = lbeep_on_time * 2;
        IO_TA01RUN = 0x06;
}

/*****************************************************************************/
/* Module name  :pbeep_timer                                                 */
/*****************************************************************************/
/* Function     :Beep tone                                                   */
/* Inputs       :gbeep_cnt                                                   */
/* Outputs      :gbeep_cnt                                                   */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/*****************************************************************************/
void pbeep_timer(void){

        gbeep_cnt--;
        if(gbeep_cnt == 0) IO_TA01RUN = 0x00;
}
```

■ Filename: led.h

```
/*
        ************************************************
        **              TOSHIBA CORPORATION           **
        **                                            **
        **                ( LED HEADER )              **
        **                                            **
        **                     MCU                    **
        **                   TX1940                   **
        **                                            **
        **                  1999/9/20                 **
        **                                            **
        ************************************************
        ************************************************
        *      COPYRIGHT(C) 1999 TOSHIBA CORPORATION     *
        *              ALL RIGHTS RESERVED               *
        ************************************************/

/*** 7-segment LED definition ***/
        #define         c7seg_0                 0xc0    /* 0 */
        #define         c7seg_1                 0xf9    /* 1 */
        #define         c7seg_2                 0xa4    /* 2 */
        #define         c7seg_3                 0xb0    /* 3 */
        #define         c7seg_4                 0x99    /* 4 */
        #define         c7seg_5                 0x92    /* 5 */
        #define         c7seg_6                 0x82    /* 6 */
        #define         c7seg_7                 0xd8    /* 7 */
        #define         c7seg_8                 0x80    /* 8 */
        #define         c7seg_9                 0x98    /* 9 */
        #define         c7seg_a                 0x88    /* A */
        #define         c7seg_b                 0x83    /* B */
        #define         c7seg_c                 0xc6    /* C */
        #define         c7seg_d                 0xa1    /* D */
        #define         c7seg_e                 0x86    /* E */
        #define         c7seg_f                 0x8e    /* F */
        #define         c7seg_p                 0x8c    /* P */
        #define         c7seg_spc               0xff    /* Blank */

/*** LED difinition ***/
        #define         cled_all_on             0x00    /* Turn all LEDs on */
        #define         cled_all_off            0xff    /* Turn all LEDs off */
```

- Filename: key.h

```
/*
        ************************************************
        **              TOSHIBA CORPORATION           **
        **                                            **
        **                ( KEY HEADER )              **
        **                                            **
        **                     MCU                    **
        **                    TX1940                  **
        **                                            **
        **                  1999/9/20                 **
        **                                            **
        ************************************************
        ************************************************
        *      COPYRIGHT(C) 1999 TOSHIBA CORPORATION     *
        *              ALL RIGHTS RESERVED               *
        ************************************************/

/*--- Matrix key definition ---*/
#define  cmatkey_no      0x0000
#define  cmatkey_0       0x0008
#define  cmatkey_1       0x0080
#define  cmatkey_2       0x0800
#define  cmatkey_3       0x8000
#define  cmatkey_4       0x0004
#define  cmatkey_5       0x0040
#define  cmatkey_6       0x0400
#define  cmatkey_7       0x4000
#define  cmatkey_8       0x0002
#define  cmatkey_9       0x0020
#define  cmatkey_a       0x0200
#define  cmatkey_b       0x2000
#define  cmatkey_c       0x0001
#define  cmatkey_d       0x0010
#define  cmatkey_e       0x0100
#define  cmatkey_f       0x1000

/*--- AD key definition ---*/
#define  cadkey_no       0x00
#define  cadkey_1        0x01
#define  cadkey_2        0x02
#define  cadkey_3        0x03
#define  cadkey_4        0x04
#define  cadkey_5        0x05
```

3.5 PC Communications (simple desktop calculator)

This facility enables calculation formulae to be received from a personal computer and then returns the results of the calculation to the personal computer.

3.5.1 Functional block diagram

3.5.1.1 Basic specifications

- The serial channels' (SIO) UART Mode is used for communicating with the personal computer.
- Calculations can only use the operators +, –, x and ÷ and can only be performed on pairs of positive numbers.
- Calculation operands and results are always 32 bits in length;
  (operands: 0~2147483647), (results: –2147483647~2147483647).

  Note: If an operation results in an overflow, the result might not be produced correctly.
- All data is received and transmitted as ASCII data.
- The 7-segment LEDs are used for display in PC Communications Mode.

3.5.1.2 Operation method

Enter a calculation formula into the personal computer and press [Enter]. The calculation result will be output.

Example 1: Calculation formula: 354363+546454[Enter]
Calculation result: 900817

Example 2: Calculation formula: 67/25[Enter]
Calculation result: 2

Example 3: Calculation formula: 65757+746464-6476473[Enter] ←An error is generated because there are more than two terms.
Calculation result: Error

Example 4: Calculation formula: 6566+746464647456473[Enter] ←An error is generated because the data is too long.
Calculation result: Error

Example 5: Calculation formula: 100/0[Enter] ←An error is generated because an attempt has been made to divide by 0.
Calculation result: Error

```
TB1940
Calculator Ver.1
>354363+546454
900817

>67/25
2

>65757+746464-6476473
Error

>6566+746464647456473
Error

>100/0
Error
```

Figure 3.5.1 PC screen display example

Table 3.5.1 lists the valid ASCII codes which can be used in a sample program. Any other ASCII codes which are present in the data sent to the personal computer will be ignored.

Table 3.5.1 List of ASCII codes

Character Code	ASCII Code	Character Code	ASCII Code
0	30	A	3A
1	31	B	3B
2	32	C	3C
3	33	D	3D
4	34	E	3E
5	35	F	3F
6	36	+	2B
7	37	–	2D
8	38	×	2A
9	39	÷	2F

Control codes	ASCII Code
CR	0D

3.5.2 Functional block diagram

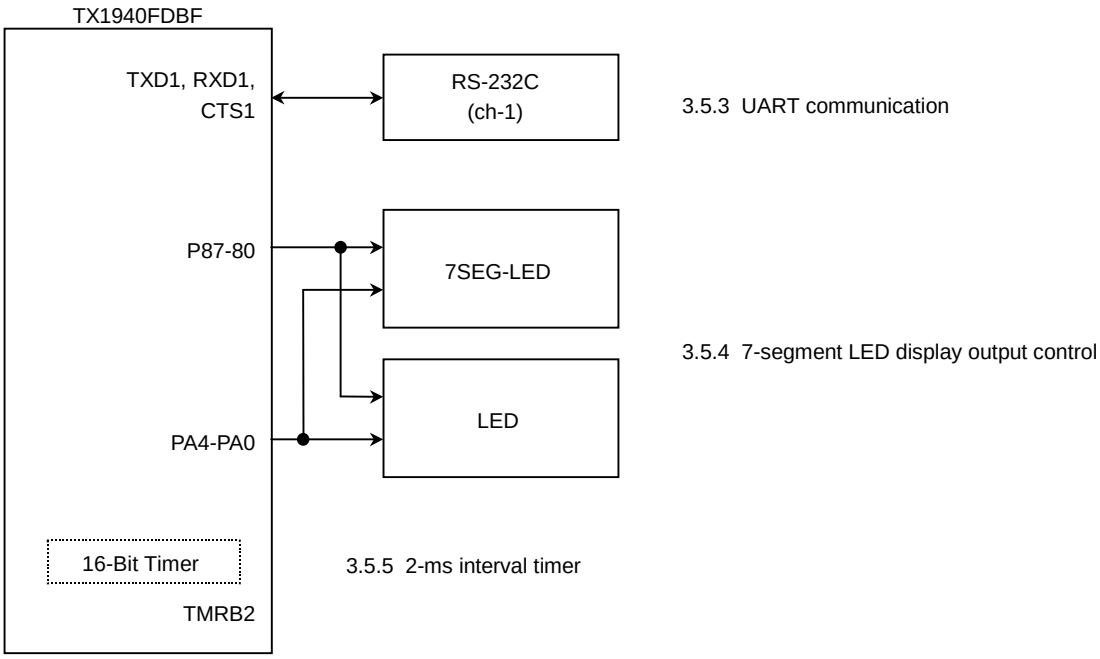


Figure 3.5.2 Block diagram

3.5.3 UART communication

3.5.3.1 Overview of UART communication

■ Transfer data settings are shown below.

Baud rate	9600bps
Data bits	8 bits
Parity	None
Stop bits	1 bit

Note: The sample program is not equipped to handle communications errors.

■ Method for calculating the baud rate

The following equation demonstrates how the baud rate is calculated when the baud rate generator is used.

$$\text{Baud rate} = \frac{\text{Baud rate generator divisor}}{\text{Baud rate generator input clock}} \div 16$$

- Divide by integer

If fc = 32 MHz is selected for fperiph and ϕT0 = fperiph/4, to set the baud rate to 9600 bps, the input clock is calculated and the divisor selected as follows.

For ϕT0 ($fc/2^2$)

$$9600 = \frac{fc/4}{\text{Divisor}} \div 16$$

$$\text{Divisor} = \frac{32{,}000{,}000}{9600 \times 4 \times 16} \qquad \text{Divisor} = 52.0833 \ \langle 52 \rangle$$

For ϕT2 ($fc/2^4$)

$$9600 = \frac{fc/16}{\text{Divisor}} \div 16$$

$$\text{Divisor} = \frac{32{,}000{,}000}{9600 \times 16 \times 16} \qquad \text{Divisor} = 13.0208 \ \langle 13 \rangle$$

For ϕT8 ($fc/2^6$)

$$9600 = \frac{fc/64}{\text{Divisor}} \div 16$$

$$\text{Divisor} = \frac{32{,}000{,}000}{9600 \times 64 \times 16} \qquad \text{Divisor} = 3.2552 \ \langle 3 \rangle$$

For ϕT32 ($fc/2^8$)

$$9600 = \frac{fc/256}{\text{Divisor}} \div 16$$

$$\text{Divisor} = \frac{32{,}000{,}000}{9600 \times 256 \times 16} \qquad \text{Divisor} = 0.8138 \ \langle 1 \rangle$$

Note: Values in < > are divisor values which have been rounded to the nearest integer.

Hence:

For ϕT0 the baud rate cannot be set since the maximum divisor value is 15.

For ϕT2 the baud rate is 9615.38 bps with a margin for error of 0.16%.

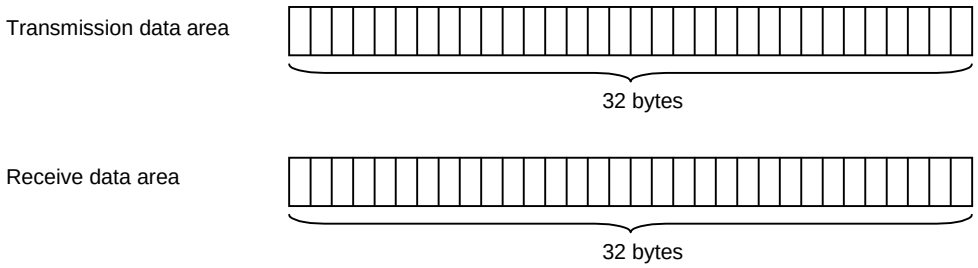
For ϕT8 the baud rate is 10416.67 bps with a margin for error of 8.5%.

For ϕT32 the baud rate is 7812.50 bps with a margin for error of 18.6%.

Therefore ϕT2 is used in the sample program.

- Data storage area

  In the sample program of PC communications the transmission data and receive data each have 32 bytes of storage area set aside for them.



3.5.3.2 Method for controlling UART communications

■ Initial settings

		7	6	5	4	3	2	1	0		
P9CR	←	-	-	0	0	1	-	-	-	}	Set P93 to be TxD1.
P9FC	←	X	X	0	X	1	-	X	-	}	Set P94 to be RxD1.
SC1MOD0	←	0	0	0	0	1	0	0	1		Select 8-Bit UART Mode and baud rate generator.
SC1MOD1	←	0	0	X	X	X	X	X	X		Select to Half-Duplex Synchronous Mode and set to stop when IDLE.
SC1CR	←	0	0	0	0	0	0	0	0		Select No parity.
BR1CR	←	0	0	0	1	1	1	0	1		Set baud rate to 9600 bps (when fc = 32 MHz).
IMCCH	←	X	X	0	1	0	1	0	1	L	Set INTRX1 for rising edge and to level 5.
		X	X	1	1	0	1	0	0	H	Set INTTX1 for rising edge and to level 4.

Note 1: X denotes Don't care; "–" denotes No change.
Note 2: The interrupt level can be set to any desired value.

■ Receive control

Until the reception-finished code (0D) is received, the received data is returned directly to the PC as transmission data.

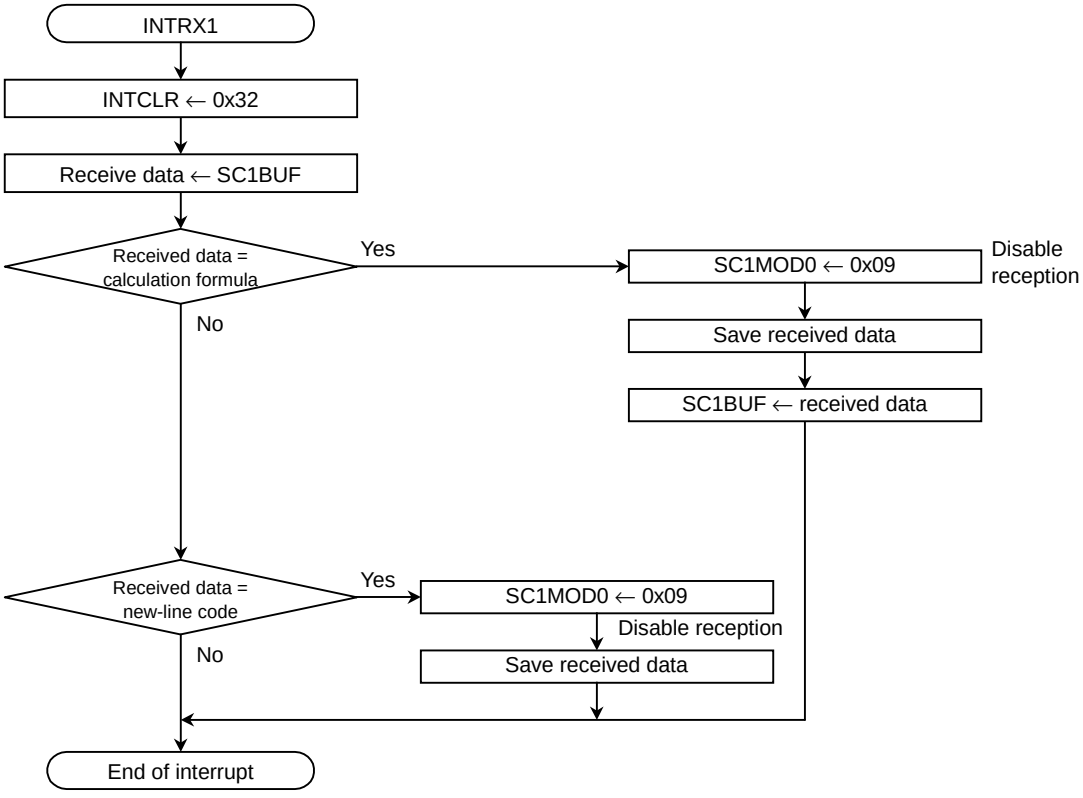


Figure 3.5.3 Flow of control for transmission/receive

■ Transmission control

To transmit/receive data, enable receive operation. When transmitting multiple data sets, store the transmission data in a buffer.

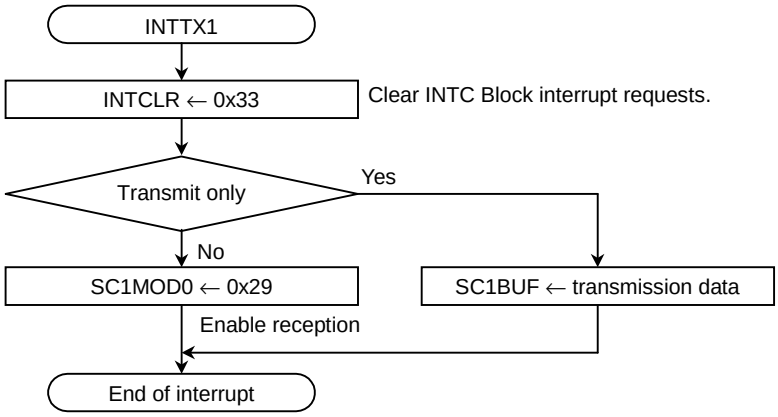


Figure 3.5.4 Flow of control for transmission

3.5.4 Control of 7-segment LED display output

3.5.4.1 Overview of 7-segment LED display

- The display data is dynamically displayed on the 7-segment LEDs and updated every 2 ms.

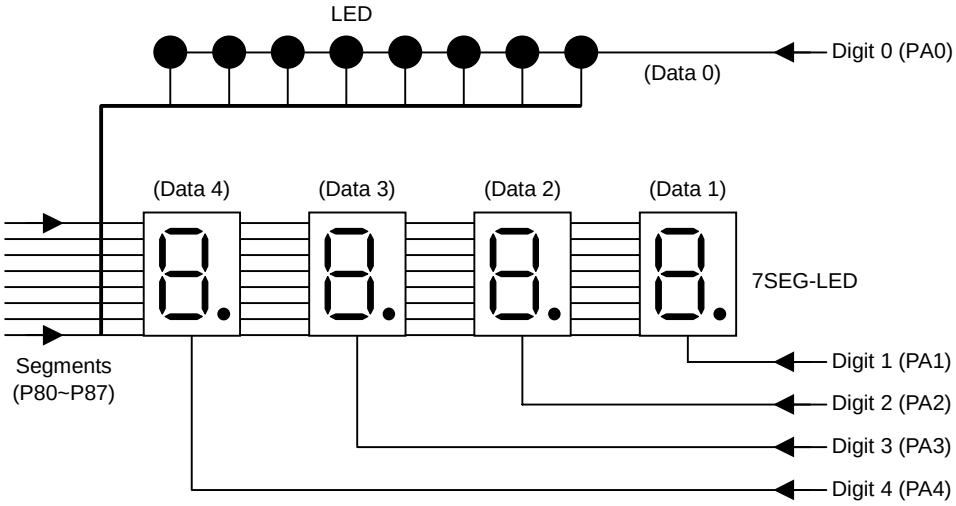
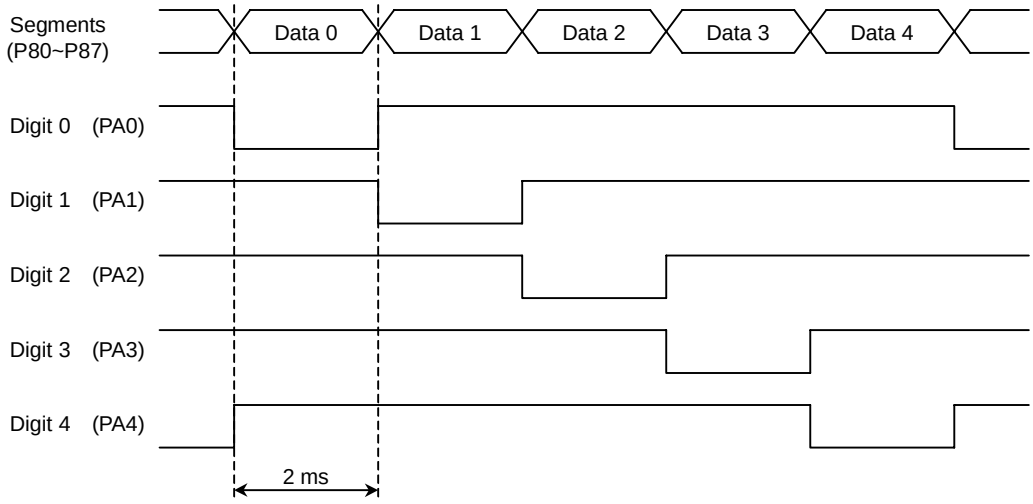


Figure 3.5.5 Correspondence between digits and segments

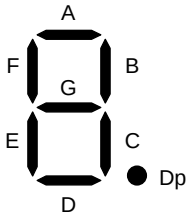
- Figure 3.5.6 shows the timing for the digit and segment outputs.



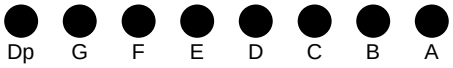
Note: The segment and digit outputs are all active-Low.

Figure 3.5.6 Timing of display digit update

- To light any given display segment, output a Low signal on the corresponding ports. The correspondence between display segments and port outputs is shown in Table 3.5.2.



7SEG-LED



LED

Table 3.5.2 Correspondence between display segments and port outputs

	P87(Dp)	P86(G)	P85(F)	P84(E)	P83(D)	P82(C)	P81(B)	P80(A)
0	*	1	0	0	0	0	0	0
1	*	1	1	1	1	0	0	1
2	*	0	1	0	0	1	0	0
3	*	0	1	1	0	0	0	0
4	*	0	0	1	1	0	0	1
5	*	0	0	1	0	0	1	0
6	*	0	0	0	0	0	1	0
7	*	1	0	1	1	0	0	0
8	*	0	0	0	0	0	0	0
9	*	0	0	1	1	0	0	0
A	*	0	0	0	1	0	0	0
B	*	0	0	0	0	0	1	1
C	*	1	0	0	0	1	1	0
D	*	0	1	0	0	0	0	1
E	*	0	0	0	0	1	1	0
F	*	0	0	0	1	1	1	0
Blank	1	1	1	1	1	1	1	1

Note 1: All segment outputs are active-Low.

Note 2: Set port output * to 0 to display Dp and to 1 to turn Dp off.

3.5.4.2 Control method for 7-segment LED display

■ Initial settings

		7	6	5	4	3	2	1	0	
P8	←	1	1	1	1	1	1	1	1	Set P80~P87 output latches to 1.
P8CR	←	1	1	1	1	1	1	1	1	Set P80~P87 for output.
P8FC	←	0	0	0	0	0	0	0	0	Set P80~P87 to be port.
PA	←	-	-	-	1	1	1	1	1	Set PA0~PA4 output latches to 1.
PACR	←	-	-	-	1	1	1	1	1	Set PA0~PA4 for output.
PAFC	←	-	-	-	0	0	0	0	0	Set PA0~PA4 to be port.

Note: X denotes Don't care; "–" denotes No change.

■ Processing of the display output

Create the display data in the main routine and output display data for each digit to the ports within a 2-ms Interval Timer interrupt. For details of the 2-ms interval timer, please refer to Section 3.5.5.

The brightness of the 7-segment and indicator LEDs can be adjusted by changing the time at which display data is output to the ports. The display output control flow is shown in Figure 3.5.7.

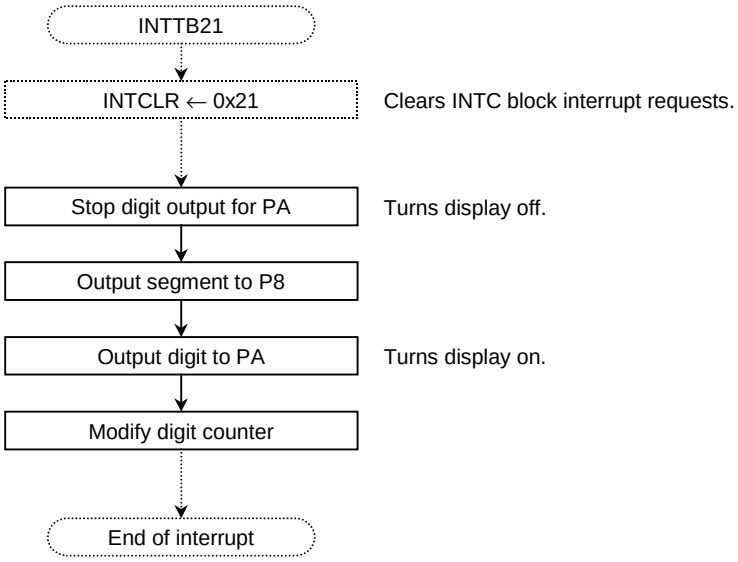


Figure 3.5.7 Display output control flow

3.5.5 2-ms Interval Timer

3.5.5.1 Overview of the 2-ms Interval Timer

- Using the 16-Bit Timer (TMRB), generate an interrupt every 2 ms.
- Set the interval time in the Timer Register TB2RG1 and generate INTTB21.

■ Timer Register calculation

The prescaler output resolutions needed for calculation are shown in Table 3.5.3.

Table 3.5.3 Prescaler output clock resolution

@fc = 32 MHz

Peripheral Clock Selection <FPSEL>	Clock Gear Value <GEAR1:0>	Selection of Prescaler Clock <PRCK1:0>	Resolution of Prescaler Output Clock		
			ϕT1	ϕT4	ϕT16
0 (gear)	00 (fc)	00 (fperiph/4)	fc/2^3 (0.25 μs)	fc/2^5 (1.0 μs)	fc/2^7 (4.0 μs)
		01 (fperiph/2)	fc/2^2 (0.125 μs)	fc/2^4 (0.5 μs)	fc/2^6 (2.0 μs)
		10 (fperiph)	—	fc/2^3 (0.25 μs)	fc/2^5 (1.0 μs)
	01 (fc/2)	00 (fperiph/4)	fc/2^4 (0.5 μs)	fc/2^6 (2.0 μs)	fc/2^8 (8.0 μs)
		01 (fperiph/2)	fc/2^3 (0.25 μs)	fc/2^5 (1.0 μs)	fc/2^7 (4.0 μs)
		10 (fperiph)	—	fc/2^4 (0.5 μs)	fc/2^6 (2.0 μs)
	10 (fc/4)	00 (fperiph/4)	fc/2^5 (1.0 μs)	fc/2^7 (4.0 μs)	fc/2^9 (16 μs)
		01 (fperiph/2)	fc/2^4 (0.5 μs)	fc/2^6 (2.0 μs)	fc/2^8 (8.0 μs)
		10 (fperiph)	—	fc/2^5 (1.0 μs)	fc/2^7 (4.0 μs)
	11 (fc/8)	00 (fperiph/4)	fc/2^6 (2.0 μs)	fc/2^8 (8.0 μs)	fc/2^{10} (32 μs)
		01 (fperiph/2)	fc/2^5 (1.0 μs)	fc/2^7 (4.0 μs)	fc/2^9 (16 μs)
		10 (fperiph)	—	fc/2^6 (2.0 μs)	fc/2^8 (8.0 μs)
1 (fc)	00 (fc)	00 (fperiph/4)	fc/2^3 (0.25 μs)	fc/2^5 (1.0 μs)	fc/2^7 (4.0 μs)
		01 (fperiph/2)	fc/2^2 (0.125 μs)	fc/2^4 (0.5 μs)	fc/2^6 (2.0 μs)
		10 (fperiph)	—	fc/2^3 (0.25 μs)	fc/2^5 (1.0 μs)
	01 (fc/2)	00 (fperiph/4)	fc/2^3 (0.25 μs)	fc/2^5 (1.0 μs)	fc/2^7 (4.0 μs)
		01 (fperiph/2)	—	fc/2^4 (0.5 μs)	fc/2^6 (2.0 μs)
		10 (fperiph)	—	fc/2^3 (0.25 μs)	fc/2^5 (1.0 μs)
	10 (fc/4)	00 (fperiph/4)	—	fc/2^5 (1.0 μs)	fc/2^7 (4.0 μs)
		01 (fperiph/2)	—	fc/2^4 (0.5 μs)	fc/2^6 (2.0 μs)
		10 (fperiph)	—	—	fc/2^5 (1.0 μs)
	11 (fc/8)	00 (fperiph/4)	—	fc/2^5 (1.0 μs)	fc/2^7 (4.0 μs)
		01 (fperiph/2)	—	—	fc/2^6 (2.0 μs)
		10 (fperiph)	—	—	fc/2^5 (1.0 μs)

Note 1: The prescaler's output clock ϕTn must be selected such that the relationship ϕTn < fsys/2 is satisfied (i.e. ϕTn must be slower than fsys/2).
Note 2: Do not change the clock gear value while the timer is operating.
Note 3: The dash character, —, in the table indicates a prohibited setting.

For ϕT1 (fc/2^3)

2000 μs ÷ 0.25 μs = 8000 (1F40H)

Set TB2RG1L to 00H and TB2RG1H to 1FH.

For ϕT4 (fc/2^5)

2000 μs ÷ 1.0 μs = 2000 (07D0H)

Set TB2RG1L to D0H and TB2RG1H to 07H.

For ϕT16 ($fc/2^7$)

2000 μs ÷ 4.0 μs = 500 (01F4H)

Set TB2RG1L to F4H and TB2RG1H to 01H.

3.5.5.2 Method for controlling the 2-ms Interval Timer

■ Initial settings

		7	6	5	4	3	2	1	0		
											Stop TMRB2.
TB2RUN	←	0	0	X	X	0	0	0	0		Disable trigger.
TB2FFCR	←	1	1	0	0	0	0	1	1		Select prescaler output clock as input clock and disable
TB2MOD	←	0	0	1	0	0	1	1	1		capture function.
											Set interval time.
TB2RG1L	←	1	1	1	1	0	1	0	0		Set TB2RG1 to 2000 μs÷ ϕT16 = 500.
TB2RG1H	←	0	0	0	0	0	0	0	1		
IMC8L	←	X	X	-	-	-	-	-	-	L	Set INTTB21 for rising edge and to level 4.
		X	X	1	1	0	0	1	1	H	
TB2RUN	←	0	0	X	X	0	1	0	1		Start TMRB2.

Note 1: X denotes Don't care; "–" denotes No change.

Note 2: The interrupt level can be set to any desired value.

- Always set the eight low-order data bits in the Timer Register first, followed by the eight high-order bits.
- In the sample program chattering-elimination processing for the matrix keys and AD conversion keys and 7-segment LED display output processing is performed using a INTTB21 interrupt.

3.5.6 Sample program

3.5.6.1 Generic flowchart

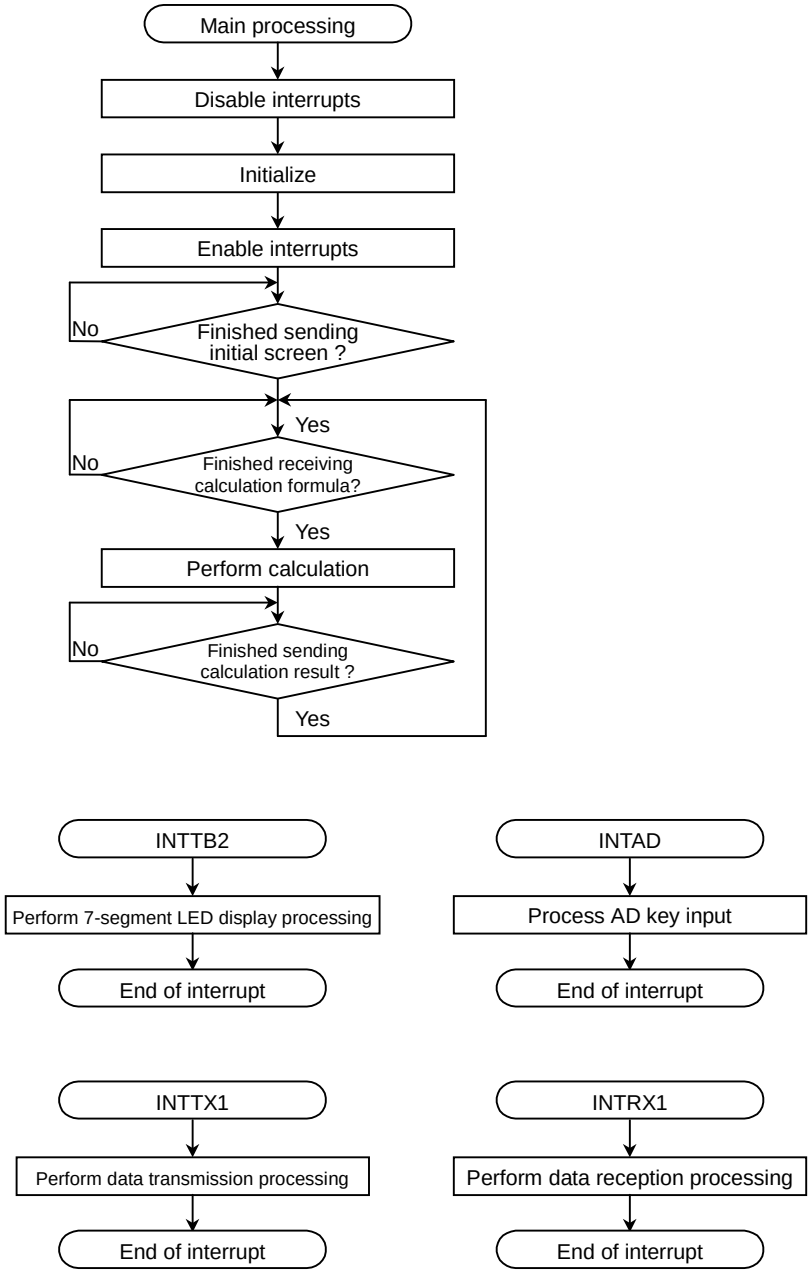


Figure 3.5.8 General flow of operation in PC Communications Mode

3.5.6.2 File configuration

Table 3.5.4 lists the files used in the sample programs.

Table 3.5.4 File configuration

Filename	Contents	Page References
uart.c	Main PC communications section	3-167
func_uart.c	PC Communications Mode functional section	3-174
led.h	7-segment LED display definitions	3-176
ascii.h	ASCII code definitions	3-177
Stc9i16.asm	Start-up routine	3-12
io1940.c	Special function registers	3-51
io1940.h	Special function register definitions	3-28

3.5.6.3 Vector table

The vector table used in the sample programs is shown below. Replace the vector table section in the start-up routine with the vector table given below.

- Interrupt vector table section (start-up routine)

```
_VecTable:
      dw      _Int_dummy      ;0 --- software set
      dw      _Int_dummy      ;1 --- INT[0]
      dw      _Int_dummy      ;2 --- INT[1]
      dw      _Int_dummy      ;3 --- INT[2]
      dw      _Int_dummy      ;4 --- INT[3]
      dw      _Int_dummy      ;5 --- INT[4]
      dw      _Int_dummy      ;6 --- *
      dw      _Int_dummy      ;7 --- *
      dw      _Int_dummy      ;8 --- *
      dw      _Int_dummy      ;9 --- *
      dw      _Int_dummy      ;10--- INT[5]
      dw      _Int_dummy      ;11--- INT[6]
      dw      _Int_dummy      ;12--- INT[7]
      dw      _Int_dummy      ;13--- INT[8]
      dw      _Int_dummy      ;14--- INT[9]
      dw      _Int_dummy      ;15--- INT[A]
      dw      _Int_dummy      ;16--- *
      dw      _Int_dummy      ;17--- *
      dw      _Int_dummy      ;18--- *
      dw      _Int_dummy      ;19--- *
      dw      _Int_dummy      ;20--- INTTA0 : 8bit Timer 0
      dw      _Int_dummy      ;21--- INTTA1 : 8bit Timer 1
      dw      _Int_dummy      ;22--- INTTA2 : 8bit Timer 2
      dw      _Int_dummy      ;23--- INTTA3 : 8bit Timer 3
      dw      _Int_dummy      ;24--- *
      dw      _Int_dummy      ;25--- *
      dw      _Int_dummy      ;26--- *
      dw      _Int_dummy      ;27--- *
      dw      _Int_dummy      ;28--- INTTB00 :16bit Timer 0 (TB0RG0)
      dw      _Int_dummy      ;29--- INTTB01 :16bit Timer 0 (TB0RG1)
      dw      _Int_dummy      ;30--- INTTB10 :16bit Timer 1 (TB1RG0)
      dw      _Int_dummy      ;31--- INTTB11 :16bit Timer 1 (TB1RG1)
      dw      _Int_dummy      ;32--- INTTB20 :16bit Timer 2 (TB2RG0)
      dw      _minttb21       ;33--- INTTB21 :16bit Timer 2 (TB2RG1)
      dw      _Int_dummy      ;34--- INTTN30 :16bit Timer 3 (TB3RG0)
      dw      _Int_dummy      ;35--- INTTB31 :16bit Timer 3 (TB3RG1)
      dw      _Int_dummy      ;36--- *
      dw      _Int_dummy      ;37--- *
      dw      _Int_dummy      ;38--- *
      dw      _Int_dummy      ;39--- *
```

```
dw      _Int_dummy      ;40--- INTTBOF0:16bit Timer 0 (OverFlow)
dw      _Int_dummy      ;41--- INTTBOF1:16bit Timer 1 (OverFlow)
dw      _Int_dummy      ;42--- INTTBOF2:16bit Timer 2 (OverFlow)
dw      _Int_dummy      ;43--- INTTBOF3:16bit Timer 3 (OverFlow)
dw      _Int_dummy      ;44--- *
dw      _Int_dummy      ;45--- *
dw      _Int_dummy      ;46--- *
dw      _Int_dummy      ;47--- *
dw      _Int_dummy      ;48--- INTRX0 :Serial receive (Channel 0)
dw      _Int_dummy      ;49--- INTTX0 :Serial transmit (Channel 0)
dw      _mintrxl        ;50--- INTRX1 :Serial receive (Channel 1)
dw      _mintrxl        ;51--- INTTX1 :Serial transmit (Channel 1)
dw      _Int_dummy      ;52--- INTS2  :Serial Channel 2 interrupt
dw      _Int_dummy      ;53--- *
dw      _Int_dummy      ;54--- INTRX3 :Serial receive (Channel 3)
dw      _Int_dummy      ;55--- INTTX3 :Serial transmit (Channel 3)
dw      _Int_dummy      ;56--- INTRX4 :Serial receive (Channel 4)
dw      _Int_dummy      ;57--- INTTX4 :Serial transmit (Channel 4)
dw      _Int_dummy      ;58--- INTRTC :Timer for RTC interrupt
dw      _Int_dummy      ;59--- INTAD  :AD conversion finished
dw      _Int_dummy      ;60--- INTDMA0:DMA transfer finished (Channel 0)
dw      _Int_dummy      ;61--- INTDMA1:DMA transfer finished (Channel 1)
dw      _Int_dummy      ;62--- INTDMA2:DMA transfer finished (Channel 2)
dw      _Int_dummy      ;63--- INTDMA3:DMA transfer finished (Channel 3)
```

3.5.6.4 Source code

- Filename: uart.c

```
/*
        ************************************************
        **              Application Note              **
        **                                            **
        **             ( U-art ROUTINE )              **
        **                                            **
        **                   MCU                      **
        **                 TX1940                     **
        **                                            **
        **              fc      = 32MHz               **
        **              fsys    = 32MHz               **
        **              fperiph = 32MHz               **
        **              ϕT0     = 8MHz                **
        **                                            **
        **                 2000/2/3                   **
        **                                            **
        ************************************************
        ************************************************
        *      COPYRIGHT(C) 1999 TOSHIBA CORPORATION     *
        *             ALL RIGHTS RESERVED              *
        ************************************************/

/************************************************/
/*              Loading header files            */
/************************************************/
#include "io1940.h"                             /* Special registers */
#include "led.h"                                /* Display definitions */
#include "ascii.h"                              /* ASCII definitions */
#include <stdlib.h>

/************************************************/
/*              Constant definitions            */
/************************************************/
/*--- Mode definitions ---*/
#define cmd_uart_open               0           /* Initial Screen Display Mode */
#define cmd_uart_input              1           /* Calculation Data Input Mode */
#define cmd_uart_output             2           /* Calculation Result Output Mode */
/*--- Transmission/Receive definitions ---*/
#define ctx_data_max                32          /* Maximum volume of transmission data */
#define crx_data_max                32          /* Maximum volume of receive data */

/************************************************/
/*              External declaration            */
/************************************************/
extern void             p7seg_disp(void);

extern unsigned char    g7seg_data[4];
extern unsigned char    gled_data;

/************************************************/
/*                      RAM                     */
/************************************************/
unsigned char   gmd_uart;                       /* Mode data */
unsigned char   guart_rx_size;                  /* Receive data size */
unsigned char   guart_tx_size;                  /* Transmission data size */
unsigned char   guart_tx_point;                 /* Transmission data position */
         char   guart_rx_data[crx_data_max];    /* Stored receive data */
         char   guart_tx_data[ctx_data_max];    /* Stored transmission data */

/************************************************/
```

```
/*                      CODE                        */
/************************************************/

/*****************************************************************************/
/* Module name  :muart_init                                                  */
/*****************************************************************************/
/* Function     :Initialize PC Communications Mode.                          */
/* Inputs       :None                                                        */
/* Outputs      :SFR                                                         */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/*****************************************************************************/
const unsigned char tuart_open[] = {            /* Initial screen display data */
         cascii_cr,cascii_lf,cascii_lf,'T','B','1','9','4','0',cascii_cr,cascii_lf,
         'C','a','l','c','u','l','a','t','o','r',' ','V','e','r','.','1','.','0',
         cascii_cr,cascii_lf,'>'};
         /* TB1940             */
         /* Calculator Ver.1.0 */

void muart_init(void){

        unsigned char i;

/*--- CG settings ---*/
        IO_SYSCR1    = 0x10;    /* <SYSCK>=0 <FPSEL>=1 <DFOSC>=0 <GEAR>=00 */

/*--- I/O port settings ---*/
        IO_P0        = 0xff;    /* Output High on P00-P07 */
        IO_P0CR      = 0xff;    /* P00-P07 for output */

        IO_P1        = 0xff;    /* Output High on P10-P17 */
        IO_P1CR      = 0xff;    /* Set P10-P17 for output */
        IO_P1FC      = 0x00;    /* Set P10-P17 to be a port  */

        IO_P2        = 0xff;    /* Output High on P20-P27 */
        IO_P2CR      = 0xff;    /* Set P20-P27 for output */
        IO_P2FC      = 0x00;    /* Set P20-P27 to be a port  */

        IO_P3        = 0xff;    /* Output High on P30-P31 and set P32-P37 to be a port */
        IO_P3CR      = 0x00;    /* Set P30-P31 for output and P32-P37 for input */
        IO_P3FC      = 0x00;    /* Set P30-P37 to be a port  */

        IO_P4        = 0x1f;    /* Set P40-P43 to be a port and output High on P44 */
        IO_P4CR      = 0x10;    /* Set P40-P43 for input and P44 for output */
        IO_P4FC      = 0x00;    /* Set P40-P44 to be a port  */

        IO_P7        = 0x65;    /* Output High on P70,P72,P75,P76 */
        IO_P7CR      = 0x67;    /* Set P70,P71,P72,P75,P76 for output and P73,P74,P77 for
                                   input */
        IO_P7FC      = 0x00;    /* Set P70-P77 to be a port  */

        IO_P8        = 0xff;    /* Output High on P80-P87 */
        IO_P8CR      = 0xff;    /* Set P80-P87 for output */
        IO_P8FC      = 0x00;    /* Set P80-P87 to be a port  */

        IO_P9        = 0x3f;    /* Output Low on P96-P97 */
        IO_P9CR      = 0xc8;    /* Set P90-P92,P94-P95 for input and P93,P96-P97 for
                                   output */
        IO_P9FC      = 0x08;    /* Set P93 to be TXD1 and P94 to be RxD1, and set others
                                   to be ports */

        IO_PA        = 0x1f;    /* Output High on PA0-PA4 */
        IO_PACR      = 0x1f;    /* Set PA0-PA4 for output and PA5-PA7 for input */
        IO_PAFC      = 0x00;    /* Set PA0-PA7 to be a port  */

/*--- Timer settings ---*/
```

```
        IO_TB2RUN     = 0x00;    /* <TB2RDE>=0 <I2TB2>=0 <TB2PRUN>=0 <TB2RUN>=0 */
        IO_TB2FFCR    = 0xc3;    /* <TB2C1T1>=0 <TB2C0T1>=0 <TB2E1T1>=0 */
                                 /* <TB2E0T1>=0 <TB2FF0C>=11 */
        IO_TB2MOD     = 0x27;    /* <TB2CP0I>=1 <TB2CPM>=00 <TB2CLE>=1 <TB2CLK>=11 */
        IO_TB2RG1L    = 0xf4;    /* TB2RG1 = 2000 / 4 */
        IO_TB2RG1H    = 0x01;    /*  */
        IO_IMC8L      = 0x3330;  /* ↑ LEVEL3 */

/*--- UART settings ---*/
        IO_SC1MOD0    = 0x09;    /* <TB8>=0 <CTSE>=0 <RXE>=0 <WU>=0 <SM>=10 <SC>=01 */
        IO_SC1MOD1    = 0X00;    /* <I2SO>=0 <FDPX0>=0 */
        IO_SC1CR      = 0x00;    /* <EVEN>=0 <PE>=0 <SCLKS>=0 <IOC>=0 */
        IO_BR1CR      = 0x1d;    /* 9600bps */
        IO_IMCCH      = 0x3435;  /* (Receive) ↑ LEVEL5 (Transmission) ↑ LEVEL4 */

/*--- Data initialization ---*/
        g7seg_data[0] = c7seg_c;
        g7seg_data[1] = c7seg_p;
        g7seg_data[2] = c7seg_spc;
        g7seg_data[3] = c7seg_spc;
        gled_data     = cled_all_off;
        for (i=0 ;i<32 ;i++) {
                     guart_tx_data[i] = tuart_open[i];
                     guart_tx_size++;
         }

/*--- Timer start ---*/
        IO_TB2RUN     = 0x05;    /* <TB2RDE>=0 <I2TB2>=0 <TB2PRUN>=1 <TB2RUN>=1 */

/*--- Transmission start ---*/
        guart_tx_point = 1;
        IO_SC1BUF      = guart_tx_data[0];
}

/*****************************************************************************/
/* Module name  :puart_answer                                                */
/*****************************************************************************/
/* Function     :Generates calculation result                                */
/* Inputs       :None                                                        */
/* Outputs      :guart_tx_size,guart_tx_point,guart_tx_data,SC1BUF           */
/* Parameter    :Calculation result                                          */
/* Return value :None                                                        */
/*****************************************************************************/
void puart_answer(int ldata) {

         unsigned char ldata_count ,i;
         unsigned char ldata_work[10];

/*--- Generating new-line data ---*/
         guart_tx_size = 0;
         guart_tx_point = 1;
         guart_tx_data[guart_tx_size++] = cascii_cr;
         guart_tx_data[guart_tx_size++] = cascii_lf;

/*--- Positive number ---*/
         if (ldata >= 0) {
                 for (ldata_count=0 ;ldata_count<10 ;ldata_count++) {
                         ldata_work[ldata_count] = '0' + (ldata % 10);
                         ldata /= 10;
                         if (ldata == 0) break;
                 }
         }
/*--- Negative number ---*/
         else {
                 guart_tx_data[guart_tx_size++] = '-';
                 for (ldata_count=0 ;ldata_count<10 ;ldata_count++) {
```

```
                        ldata_work[ldata_count] = '0' - (ldata % 10);
                        ldata /= 10;
                        if (ldata == 0) break;
                }
        }

/*--- Generating calculation result data ---*/
        for (i=0 ;i<(ldata_count+1) ;i++) {
                guart_tx_data[guart_tx_size++] = ldata_work[ldata_count-i];
        }

/*--- Generating new-line data ---*/
        guart_tx_data[guart_tx_size++] = cascii_cr;
        guart_tx_data[guart_tx_size++] = cascii_lf;
        guart_tx_data[guart_tx_size++] = cascii_lf;
        guart_tx_data[guart_tx_size++] = '>';

/*--- Start of transmission ---*/
        IO_SC1BUF = guart_tx_data[0];
}

/*****************************************************************************/
/* Module name  :puart_error                                                 */
/*****************************************************************************/
/* Function     :Generates error data                                        */
/* Inputs       :None                                                        */
/* Outputs      :guart_tx_data,guart_tx_size,guart_tx_point,SC1BUF           */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/*****************************************************************************/
const unsigned char tuart_error[] = {
        cascii_cr,cascii_lf,'E','r','r','o','r',
        cascii_cr,cascii_lf,cascii_lf,'>'};

void puart_error(void){

        unsigned char i;

/*--- Generating error data ---*/
        guart_tx_size = 0;
        for (i=0 ;i<11 ;i++) {
                guart_tx_data[i] = tuart_error[i];
                guart_tx_size++;
        }

/*--- Start of transmission ---*/
        guart_tx_point = 1;
        IO_SC1BUF = guart_tx_data[0];
}

/*****************************************************************************/
/* Module name  :puart_cal                                                   */
/*****************************************************************************/
/* Function     :Perform calculation processing                              */
/* Inputs       :guart_rx_data                                               */
/* Outputs      :None                                                        */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/*****************************************************************************/
void puart_cal(void) {

        unsigned int ldata_left, ldata_right, lwork;
        unsigned char lpoint ,lcal ,fcal_error;
        int ldata_ans;

/*--- Data initialization ---*/
```

```
        lpoint     = fcal_error = 0;
        ldata_left = ldata_right = 0;

/*--- Generating left-side data ----*/
        lwork = guart_rx_data[lpoint];  /* Read data */
        if (lwork>='0' && lwork<='9') {  /* Check that data is numeric */
                for(;;) {
                        if (lwork>='0' && lwork<='9') {   /* Check that data is numeric */
                                ldata_left = ldata_left*10 + (lwork-'0');
                                /* Check for overflow */
                                if (ldata_left > 0x7fffffff) {
                                        fcal_error = 1;
                                        break;
                                }
                                /* Read next data item */
                                else lwork = guart_rx_data[++lpoint];
                        }
                        else break;
                }
        }
        else fcal_error = 1;

/*--- Generating operator ---*/
        if (fcal_error == 0) {
                switch (lwork) {
                case cascii_add:
                case cascii_sub:
                case cascii_mul:
                case cascii_div:
                        lcal = lwork;           /* Store operator */
                        break;
                default:
                        fcal_error = 1;
                        break;
                }
                lpoint++;
        }

/*--- Generating right-side data ---*/
        if (fcal_error == 0) {
                lwork = guart_rx_data[lpoint];  /* Read data */
                if (lwork>='0' && lwork<='9') { /* Check that data is numeric */
                        for(;;) {
                                if (lwork>='0' && lwork<='9') {  /* Check that data is
                                                                    numeric */
                                        ldata_right = ldata_right*10 + (lwork-'0');
                                        /* Check for overflow */
                                        if (ldata_right > 0x7fffffff) {
                                               fcal_error = 1;
                                               break;
                                        }
                                        /* Read next data item */
                                        else lwork = guart_rx_data[++lpoint];
                                }
                                else break;
                        }
                }
                else fcal_error = 1;
        }

/*--- Check for end of data ---*/
        if (fcal_error == 0) {
                if (lwork != cascii_cr) fcal_error = 1;
        }

/*--- Calculation ---*/
```

```
        if (fcal_error == 0) {
                switch (lcal) {
                case cascii_add:                        /* Add */
                        ldata_ans = (int)ldata_left + (int)ldata_right;
                        puart_answer(ldata_ans);        /* Generate answer data */
                        break;
                case cascii_sub:                        /* Subtract */
                        ldata_ans = (int)ldata_left - (int)ldata_right;
                        puart_answer(ldata_ans);        /* Generate answer data */
                        break;
                case cascii_mul:                        /* Multiply */
                        ldata_ans = (int)ldata_left * (int)ldata_right;
                        puart_answer(ldata_ans);        /* Generate answer data */
                        break;
                case cascii_div:/* Divide */
                        if (ldata_right != 0) {         /* Check whether right side is 0 */
                                ldata_ans = (int)ldata_left / (int)ldata_right;
                                puart_answer(ldata_ans);   /* Generate answer data */
                        }
                        else puart_error();             /* Generate error data */
                        break;
                }
        }
        else puart_error();     /* Generate error data */
}

/*****************************************************************************/
/* Module name  :main                                                        */
/*****************************************************************************/
/* Function     :Main processing for PC Communications Mode                  */
/* Inputs       :gmd_uart,guart_tx_point,guart_tx_size,guart_rx_data,guart_rx_size */
/* Outputs      :SC1MOD0,gmd_uart,guart_rx_size                              */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/*****************************************************************************/
void main(void){

        __DI();

        muart_init();           /* Initialization */

        __EI();

/*************************************/
/*                                   */
/*              Main loop            */
/*                                   */
/*************************************/
for(;;){
        switch (gmd_uart) {
        case cmd_uart_open:
                if (guart_tx_point == guart_tx_size) { /* Check for end of transmission */
                        IO_SC1MOD0 = 0x29;              /* Enable reception */
                        gmd_uart = cmd_uart_input;
                }
                break;
        case cmd_uart_input:
                if (guart_rx_data[guart_rx_size-1] == cascii_cr) { /* Check for end of
                                                                      reception */
                        puart_cal();                    /* Perform calculation */
                        gmd_uart = cmd_uart_output;
                }
                break;
        case cmd_uart_output:
                if (guart_tx_point == guart_tx_size) { /* Check for end of transmission */
                        guart_rx_size = 0;
```

```
                        IO_SC1MOD0 = 0x29;              /* Enable reception */
                        gmd_uart = cmd_uart_input;
                }
                break;
        }
}
}

/*****************************************************************************/
/* Module name  :minttx1                                                     */
/*****************************************************************************/
/* Function     :Transmits data (using Serial Transmission interrupt)        */
/* Inputs       :guart_tx_point,guart_tx_size,guart_tx_data                  */
/* Outputs      :SC1BUF,SC1MOD0                                              */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/*****************************************************************************/
void __interrupt minttx1(void){

        IO_INTCLR = 0x33;               /* Clear interrupt latch */

        switch (gmd_uart) {
        case cmd_uart_open:
        case cmd_uart_output:
                if (guart_tx_point < guart_tx_size) {
                        /* Set transmission data */
                        IO_SC1BUF = guart_tx_data[guart_tx_point++];
                }
                break;
        case cmd_uart_input:
                IO_SC1MOD0 = 0x29;      /* Enable reception */
                break;
        }
}

/*****************************************************************************/
/* Module name  :mintrx1                                                     */
/*****************************************************************************/
/* Function     :Receive data (using Serial Receive interrupt)               */
/* Inputs       :SC1BUF,guart_rx_size,guart_rx_data                          */
/* Outputs      :SC1MOD0,SC1BUF                                              */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/*****************************************************************************/
void __interrupt mintrx1(void){

        unsigned char lrx_data;

        IO_INTCLR = 0x32;           /* Clear interrupt latch */

        lrx_data = IO_SC1BUF;       /* Read received data */

/*--- Dividing up received data ---*/
        switch (lrx_data) {
        case cascii_0:
        case cascii_1:
        case cascii_2:
        case cascii_3:
        case cascii_4:
        case cascii_5:
        case cascii_6:
        case cascii_7:
        case cascii_8:
        case cascii_9:
        case cascii_mul:
        case cascii_add:
```

```
        case cascii_sub:
        case cascii_div:
                IO_SC1MOD0 = 0x09;    /* Disable reception */
                if (guart_rx_size < (crx_data_max-1)) { /* Check amount of received data */
                        guart_rx_data[guart_rx_size++] = lrx_data;
                        /* Store received data */
                }
                IO_SC1BUF = lrx_data; /* Set transmission data */
                break;
        case cascii_cr:
                IO_SC1MOD0 = 0x09;    /* Disable reception */
                guart_rx_data[guart_rx_size++] = cascii_cr; /* Store received data */
                break;
        }
}

/*****************************************************************************/
/* Module name :minttb21                                                     */
/*****************************************************************************/
/* Function    :2-ms Interval Timer interrupt                                */
/* Inputs      :None                                                         */
/* Outputs     :None                                                         */
/*****************************************************************************/
void __interrupt minttb21(void){

        IO_INTCLR = 0x21;          /* Clear interrupt latch */

        p7seg_disp();
}
```

- Filename: func_uart.c

```
/*
        ************************************************
        **              Application Note               **
        **                                             **
        **          ( URAT FUNCTION ROUTINE )          **
        **                                             **
        **                    MCU                      **
        **                   TX1940                    **
        **                                             **
        **              fc      = 32MHz                **
        **              fsys    = 32MHz                **
        **              fperiph = 32MHz                **
        **              φT0     = 8MHz                 **
        **                                             **
        **                 1999/9/20                   **
        **                                             **
        ************************************************
        ************************************************
        *      COPYRIGHT(C) 1999 TOSHIBA CORPORATIO      *
        *               ALL RIGHTS RESERVED              *
        ************************************************/

/***********************************************/
/*              Loading header file            */
/***********************************************/
#include "io1940.h"

/***********************************************/
/*              Constant definitions           */
/***********************************************/
/*--- Port output definitions (for 7-segment LED digits) ---*/
```

```
#define  c7seg_digit_out0       0x1e            /* PA (00011110) */
#define  c7seg_digit_out1       0x1d            /* PA (00011101) */
#define  c7seg_digit_out2       0x1b            /* PA (00011011) */
#define  c7seg_digit_out3       0x17            /* PA (00010111) */
#define  c7seg_digit_out4       0x0f            /* PA (00001111) */

/*********************************************/
/*                    RAM                    */
/*********************************************/
unsigned char            g7seg_data[4];         /* 7-segment LED display data */
unsigned char            gled_data;             /* LED display data */

/*********************************************/
/*                    CODE                   */
/*********************************************/

/*****************************************************************************/
/* Module name  :p7seg_disp                                                  */
/*****************************************************************************/
/* Function     :7-segment LED display output (using 2-ms interrupt)         */
/* Inputs       :gled_data,g7seg_data                                        */
/* Outputs      :P8,PA                                                       */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/*****************************************************************************/
void p7seg_disp(void){

        static unsigned char ldisp_digit = 0;

/*--- Turning display off ---*/
        IO_P8 = 0xff;

/*--- Display output ---*/
        switch (ldisp_digit) {
        case 0:
                IO_PA = c7seg_digit_out0;      /* Digit output */
                IO_P8 = gled_data;             /* LED output */
                break;
        case 1:
                IO_PA = c7seg_digit_out1;      /* Digit output */
                IO_P8 = g7seg_data[0];         /* 7-segment LED output */
                break;
        case 2:
                IO_PA = c7seg_digit_out2;      /* Digit output */
                IO_P8 = g7seg_data[1];         /* 7-segment LED output */
                break;
        case 3:
                IO_PA = c7seg_digit_out3;      /* Digit output */
                IO_P8 = g7seg_data[2];         /* 7-segment LED output */
                break;
        case 4:
                IO_PA = c7seg_digit_out4;      /* Digit output */
                IO_P8 = g7seg_data[3];         /* 7-segment LED output */
                break;
        }

/*--- Digit count ---*/
        ldisp_digit++;
        if (ldisp_digit > 4) ldisp_digit = 0;
}
```

- Filename: led.h

```
/*
        ************************************************
        **              TOSHIBA CORPORATION           **
        **                                            **
        **                ( LED HEADER )              **
        **                                            **
        **                     MCU                    **
        **                    TX1940                  **
        **                                            **
        **                  1999/9/20                 **
        **                                            **
        ************************************************
        ************************************************
        *      COPYRIGHT(C) 1999 TOSHIBA CORPORATION     *
        *              ALL RIGHTS RESERVED               *
        ************************************************/

/*** 7-segment LED definitions ***/
        #define         c7seg_0                 0xc0    /* 0 */
        #define         c7seg_1                 0xf9    /* 1 */
        #define         c7seg_2                 0xa4    /* 2 */
        #define         c7seg_3                 0xb0    /* 3 */
        #define         c7seg_4                 0x99    /* 4 */
        #define         c7seg_5                 0x92    /* 5 */
        #define         c7seg_6                 0x82    /* 6 */
        #define         c7seg_7                 0xd8    /* 7 */
        #define         c7seg_8                 0x80    /* 8 */
        #define         c7seg_9                 0x98    /* 9 */
        #define         c7seg_a                 0x88    /* A */
        #define         c7seg_b                 0x83    /* B */
        #define         c7seg_c                 0xc6    /* C */
        #define         c7seg_d                 0xa1    /* D */
        #define         c7seg_e                 0x86    /* E */
        #define         c7seg_f                 0x8e    /* F */
        #define         c7seg_p                 0x8c    /* P */
        #define         c7seg_spc               0xff    /* Blank */

/*** LED definitions ***/
        #define         cled_all_on             0x00    /* Turn all LEDs ON */
        #define         cled_all_off            0xff    /* Turn all LEDs OFF */
```

- Filename: ascii.h

```
/*
        ************************************************
        **              TOSHIBA CORPORATION           **
        **                                            **
        **               ( ASCII HEADER )             **
        **                                            **
        **                    MCU                     **
        **                   TX1940                   **
        **                                            **
        **                  1999/9/20                 **
        **                                            **
        ************************************************
        ************************************************
        *      COPYRIGHT(C) 1999 TOSHIBA CORPORATION     *
        *              ALL RIGHTS RESERVED               *
        ************************************************/

/*** ASCII code definitions ***/
        #define cascii_0            0x30
        #define cascii_1            0x31
        #define cascii_2            0x32
        #define cascii_3            0x33
        #define cascii_4            0x34
        #define cascii_5            0x35
        #define cascii_6            0x36
        #define cascii_7            0x37
        #define cascii_8            0x38
        #define cascii_9            0x39
        #define cascii_mul          0x2a
        #define cascii_add          0x2b
        #define cascii_sub          0x2d
        #define cascii_div          0x2f
        #define cascii_cr           0x0d
        #define cascii_lf           0x0a
```

3.6 Processing Analog Inputs

This feature allows the change of the microphone output voltage level and the photo-interrupter output voltage level to be displayed.

3.6.1 Functional specifications

3.6.1.1 How to use this feature

Connect a microphone to the MIC terminal of the TB1940 and talk into it. The volume level corresponding to the input voltage is displayed on the 7-segment LEDs as a series of 0s. The higher the level, the more zeroes are displayed.

In the same way, the amount of light from the photo-interrupter on the TB1940 which impinges on the photodetector is displayed on the 7-segment LEDs as a series of 0s. Partially blocking the gap in the photo-interrupter causes the amount of light hitting the photodetector to decrease. This in turn causes some of the LEDs to turn off.

3.6.1.2 Basic specifications

- Microphone
  - A microphone is connected to the MIC terminal.
  - The output level for the microphone is represented by the number of 0s displayed on the 7-segment LEDs. The correspondence between the microphone output level and the 7-segment LED display is shown in Table 3.6.1.
  - The microphone output is amplified by the microphone amp IC on the TB1940 before being fed to the AN0 input on the TX1940. The AN0 input is converted into a digital signal by the AD converter.
- Photo-interrupter
  - Partially blocking the gap in the photo-interrupter reduces the amount of light propagating from the light-emitting part to the light-receiving part, thus reducing the photodetector's output level.
  - The correspondence between the photo-interrupter's output voltage level and the number of LEDs which are lit is shown in Table 3.6.2.
  - The photo-interrupter output is connected to the AN1 input on the TX1940. The AN1 input value is converted into a digital signal by the AD converter.

Table 3.6.1 7-segment LED display output

Vdd = 3.3 V

Microphone Amp IC Output Voltage Range [V]	7-Segment LED Display Output
(5/10)Vdd < V ≤ (6/10)*Vdd	[ ] [ ] [ ] [ ]
(6/10)*Vdd < V ≤ (7/10)*Vdd	[0] [ ] [ ] [ ]
(7/10)*Vdd < V ≤ (8/10)*Vdd	[0] [0] [ ] [ ]
(8/10)*Vdd < V ≤ (9/10)*Vdd	[0] [0] [0] [ ]
(9/10)*Vdd < V ≤ Vdd	[0] [0] [0] [0]

V: Microphone amp IC output voltage.
[ ]: No segments are turned on.
[0]: Segments display 0.

Table 3.6.2 LED turn-on conditions

Vdd = 3.3 V

Photo-interrupter Output Voltage Range [V]	LED State
0 ≤ V ≤ (1/9)*Vdd	❍❍❍❍❍❍❍❍
(1/9)*Vdd < V ≤ (2/9)*Vdd	●❍❍❍❍❍❍❍
(2/9)*Vdd < V ≤ (3/9)*Vdd	●●❍❍❍❍❍❍
(3/9)*Vdd < V ≤ (4/9)*Vdd	●●●❍❍❍❍❍
(4/9)*Vdd < V ≤ (5/9)*Vdd	●●●●❍❍❍❍
(5/9)*Vdd < V ≤ (6/9)*Vdd	●●●●●❍❍❍
(6/9)*Vdd < V ≤ (7/9)*Vdd	●●●●●●❍❍
(7/9)*Vdd < V ≤ (8/9)*Vdd	●●●●●●●❍
(8/9)*Vdd < V ≤ Vdd	●●●●●●●●

V : Photo-interrupter output voltage;

●: LED turned on

❍: LED turned off

3.6.2 Functional block diagram

The functional blocks used by this sample program are shown in Figure 3.6.1.

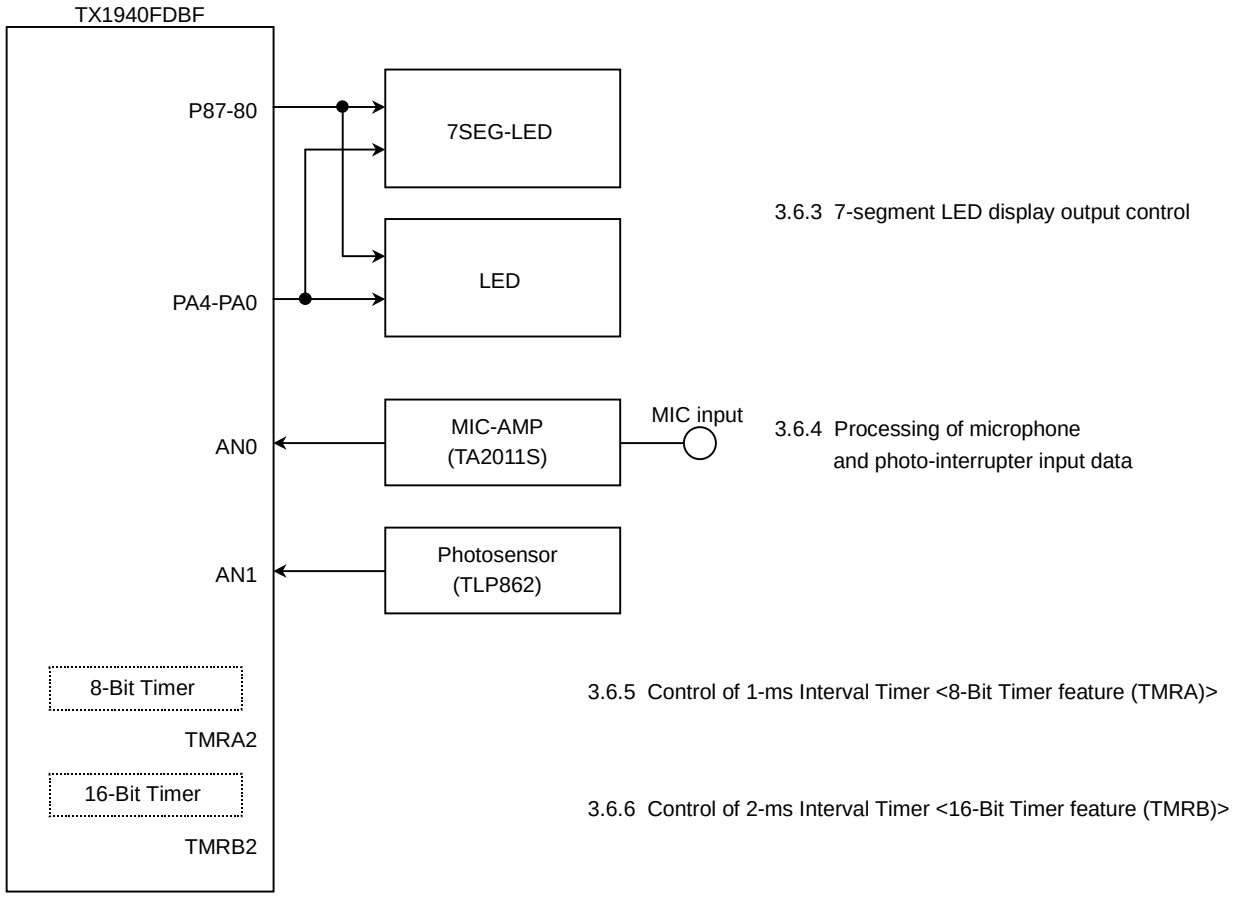


Figure 3.6.1 Functional block diagram of analog input-processing section

3.6.3 Control of 7-segment LED display output

3.6.3.1 Overview of 7-segment LED display

- The display data is dynamically displayed on the 7-segment LEDs and updated every 2 ms.

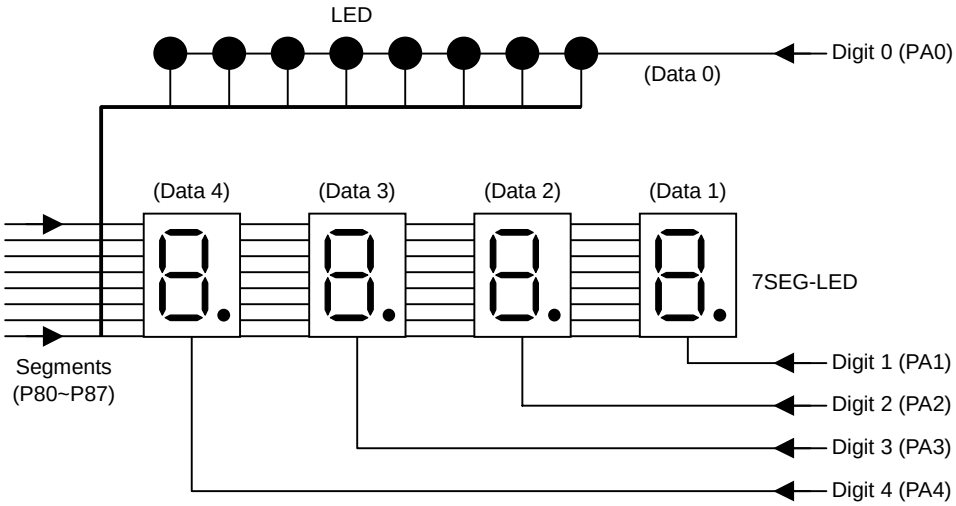
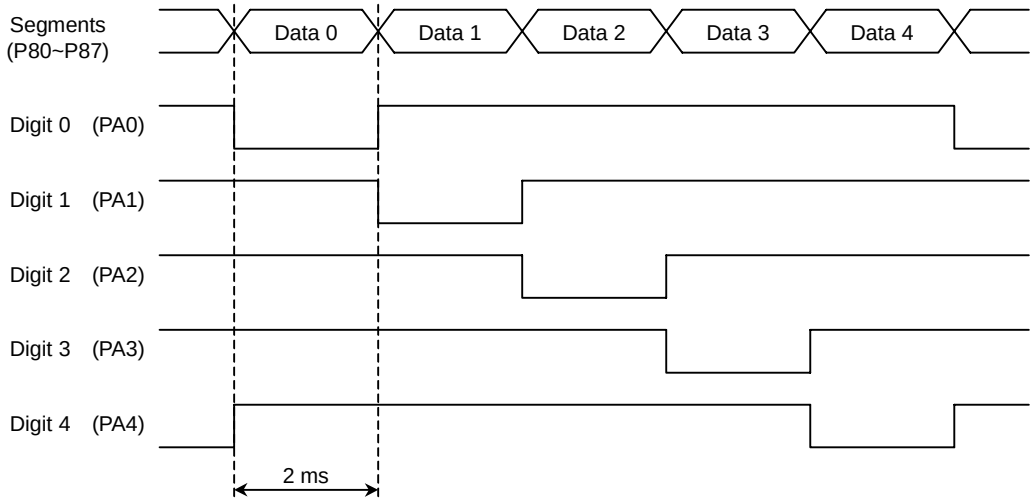


Figure 3.6.2 Correspondence between digits and segments

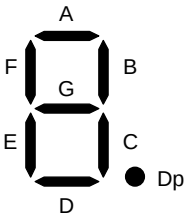
- Figure 3.6.3 shows the timing for the digit and segment outputs.



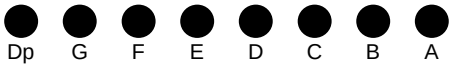
Note: The segment and digit outputs are all active-Low.

Figure 3.6.3 Timing of display digit update

- To light any given display segment, output a Low signal on the corresponding ports. The correspondence between display segments and port outputs is shown in Table 3.6.3.



7SEG-LED



LED

Table 3.6.3 Correspondence between display segments and port outputs

	P87(Dp)	P86(G)	P85(F)	P84(E)	P83(D)	P82(C)	P81(B)	P80(A)
0	*	1	0	0	0	0	0	0
1	*	1	1	1	1	0	0	1
2	*	0	1	0	0	1	0	0
3	*	0	1	1	0	0	0	0
4	*	0	0	1	1	0	0	1
5	*	0	0	1	0	0	1	0
6	*	0	0	0	0	0	1	0
7	*	1	0	1	1	0	0	0
8	*	0	0	0	0	0	0	0
9	*	0	0	1	1	0	0	0
A	*	0	0	0	1	0	0	0
B	*	0	0	0	0	0	1	1
C	*	1	0	0	0	1	1	0
D	*	0	1	0	0	0	0	1
E	*	0	0	0	0	1	1	0
F	*	0	0	0	1	1	1	0
Blank	1	1	1	1	1	1	1	1

Note 1: All segment outputs are active-Low.

Note 2: Set port output * to 0 to display Dp and to 1 to turn Dp off.

3.6.3.2 Control method for 7-segment LED display

■ Initial settings

		7	6	5	4	3	2	1	0	
P8	←	1	1	1	1	1	1	1	1	Set P80~P87 output latches to 1.
P8CR	←	1	1	1	1	1	1	1	1	Set P80~P87 for output.
P8FC	←	0	0	0	0	0	0	0	0	Set P80~P87 to be port.
PA	←	-	-	-	1	1	1	1	1	Set PA0~PA4 output latches to 1.
PACR	←		-	-	1	1	1	1	1	Set PA0~PA4 for output.
PAFC	←	-	-	-	0	0	0	0	0	Set PA0~PA4 to be port.

Note: X denotes Don't care; "–" denotes No change.

■ Processing of the display output

Create the display data in the main routine and output display data for each digit to the ports within a 2-ms Interval Timer interrupt. For details of the 2-ms interval timer, please refer to Section 3.6.6.

The brightness of the 7-segment and indicator LEDs can be adjusted by changing the time at which display data is output to the ports. The display output control flow is shown in Figure 3.6.4.

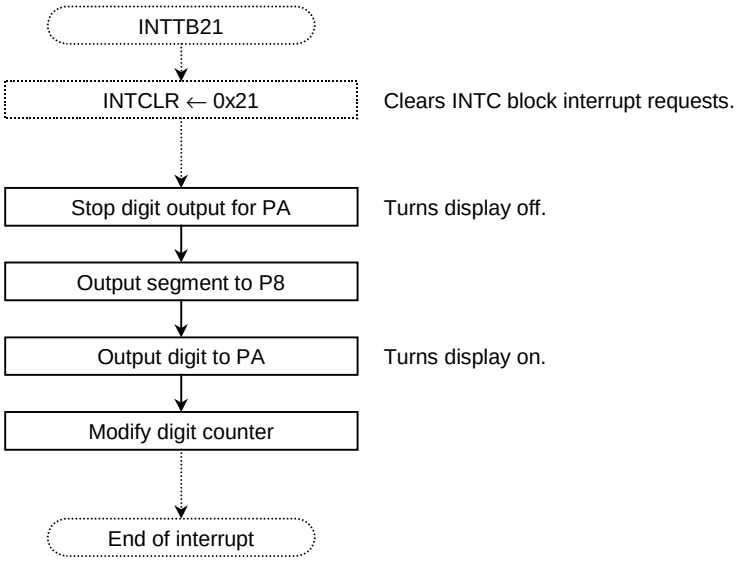


Figure 3.6.4 Display output control flow

3.6.4 Processing of microphone and photo-interrupter input data

3.6.4.1 Overview of photo-interrupter and microphone output data display processing

- The output stage of the microphone amp IC is connected to the AN0 terminal and the photo-interrupter output is connected to the AN1 terminal.
- AD conversion is performed in Channel Scan Single-Conversion Mode.
- AD conversion is performed every 1 ms. For details of the setting method of the 1-ms Interval Timer, please refer to Section 3.6.5.

3.6.4.2 Analog data control method

■ Initial settings

Settings made in the main routine

		7	6	5	4	3	2	1	0		
ADCCLK	←	X	X	X	X	X	X	0	1		Set AD conversion time to 10.75 µs (when fsys = 32 MHz).
ADMOD0	←	X	X	0	0	0	0	1	0		Select Channel Scan Mode.
ADMOD1	←	1	-	X	X	0	0	0	1		Apply V_{ref}, disable external trigger, scan (AN0 → AN1).
IMC8L	←	X	X	1	1	0	1	0	1	High-order	Set for rising edge and set interrupt level to 5.
		X	X	X	X	X	X	X	X	Low-order	Set the interrupt level to any desired value.

Note 1: X denotes Don't care.
Note 2: The interrupt level can be set to any desired value.

Settings made by 2-ms Interval interrupt

		7	6	5	4	3	2	1	0	
ADMOD0	←	X	X	0	0	0	0	1	1	Start conversion in Channel Scan Single-Conversion Mode.

Note: X denotes Don't care.

■ Latching AD-converted data

- AD Conversion-Finished interrupt processing consists of writing data from the AD Conversion Result Register to memory.
- Figure 3.6.5 shows the flow of control for when AD-converted data is latched into memory.

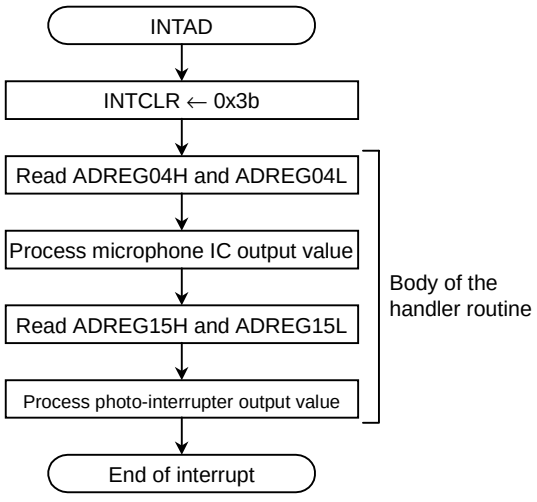


Figure 3.6.5 Flow of control for INTAD interrupt

■ Method for calculating the average of the AD-converted values

- INTAD interrupt processing consists of calculating the average of the AD conversion results in software.
- The maximum AD conversion value from the preceding 32 data entries is taken as the microphone amp IC output value and is displayed on the 7-segment LEDs.Figure 3.6.6 shows how the maximum value is derived.
- Figure 3.6.7 shows the flow of control.
- The average AD conversion value of the preceding eight entries of data is taken as the photo-interrupter output value and is displayed on the 7-segment LEDs. Figure 3.6.8 shows how the maximum value is derived. Figure 3.6.9 shows the flow of control.
- The microphone output is converted into one of five value levels which is then represented on the 7-segment LEDs using the corresponding number of 0s. (See Table 3.6.1 for 7-segment LED displays.)
- The photo-interrupter output is converted into one of nine value levels which is then represented on the 7-segment LEDs using the corresponding number of 0s. (See Figure 3.6.2 for 7-segment LED displays.)

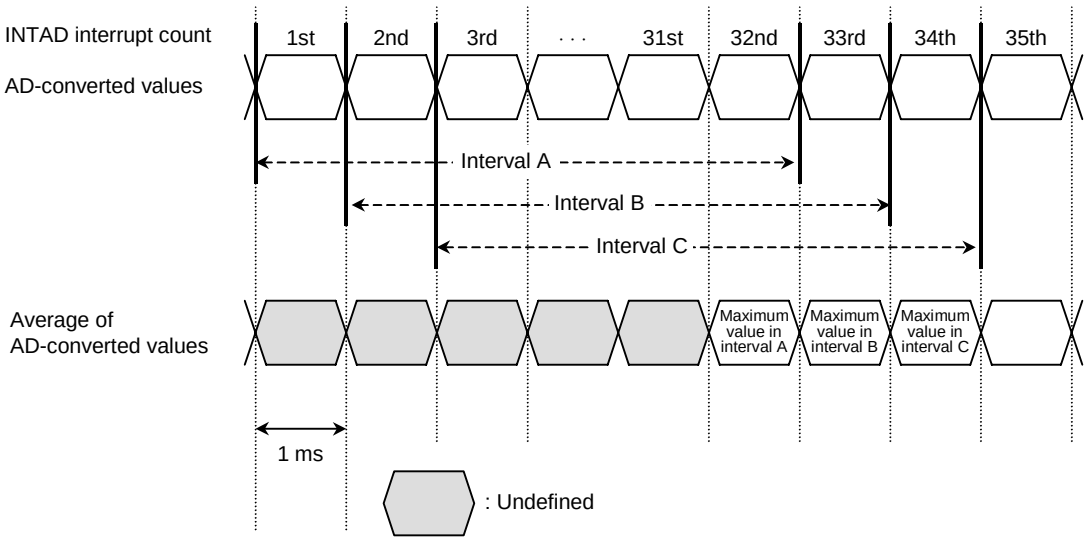


Figure 3.6.6 Calculation method for maximum value of microphone IC output

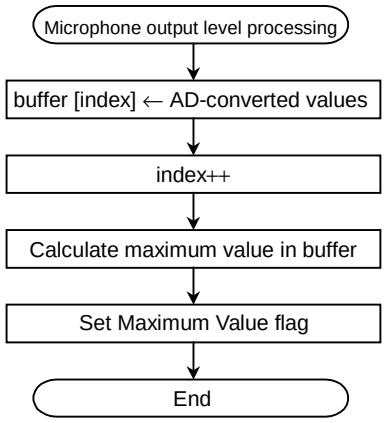


Figure 3.6.7 Flow of control for calculation of microphone IC output display value

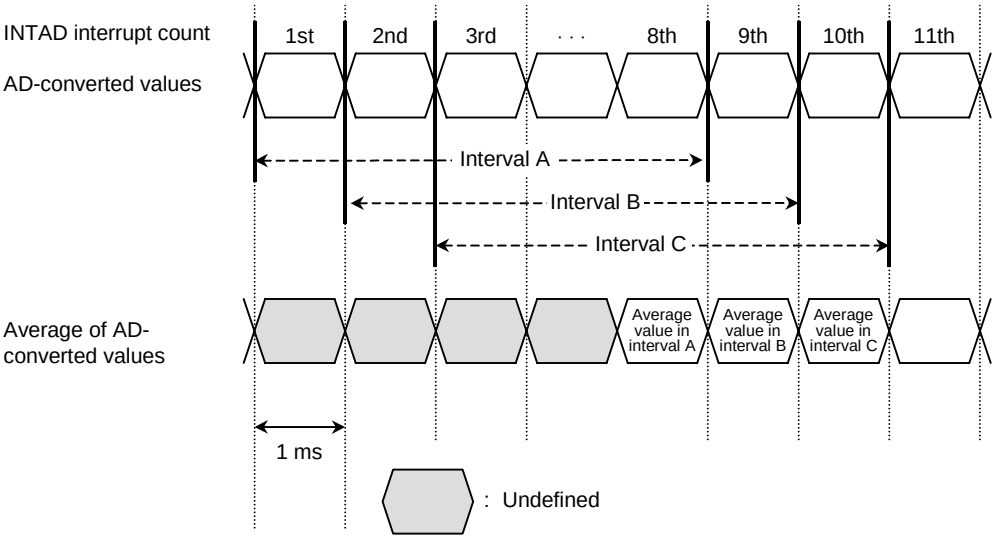


Figure 3.6.8 Derivation method for average value of photo-interrupter output

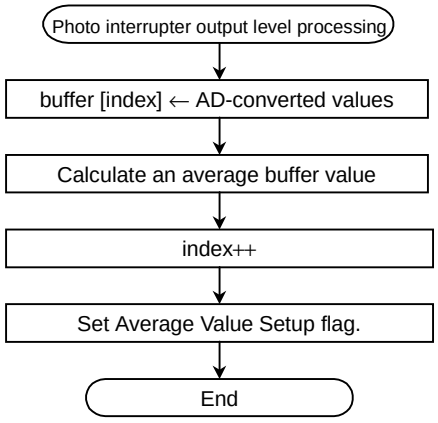


Figure 3.6.9 Photo-interrupter output value display processing

3.6.5 Control of 1-ms Interval Timer <8-Bit Timer feature (TMRA)>

3.6.5.1 Overview of the 1-ms Interval Timer

- Using the 8-Bit Timer feature (TMRA), generate an interrupt every 1 ms.
- Set the time interval at which to generate INTTA2 in the timer register TA2REG.

3.6.5.2 Control method for the 1-ms Interval Timer

■ Initial settings

		7	6	5	4	3	2	1	0		
TA23RUN	←	-	X	X	X	-	-	X	0		Stop TMRA2 and clear it to 0.
TA23MOD	←	0	0	X	X	X	X	1	1		Select 8-Bit Timer and ϕT16 (4.0 μs).
TA2REG	←	125									TA2RUN = 1000 ÷ 4
INTCLR=22											
IMC5H	←	X	X	1	1	0	1	1	0	High-order	Set for rising edge and set interrupt level to 6.
		X	X	X	X	X	X	X	X	Low-order	
TA2RUN	←	0	0	X	X	–	1	X	1		Start TMRB0.

Note: The interrupt level can be set to any desired value.

■ Interrupt processing

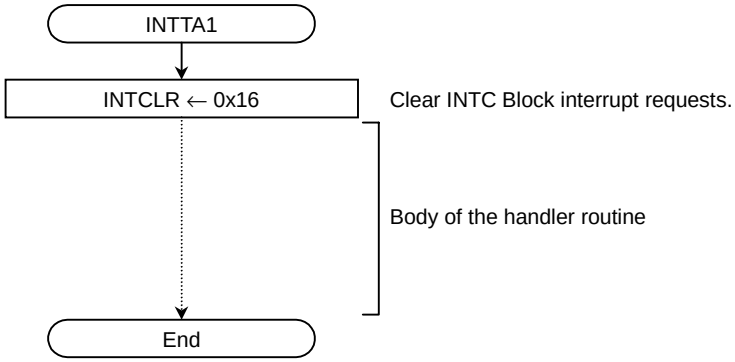


Figure 3.6.10 Flow of control for INTTA2 interrupt

3.6.6 Control of 2-ms Interval Timer <16-Bit Timer feature (TMRB)>

3.6.6.1 Overview of the 2-ms Interval Timer

- Using the 16-Bit Timer feature (TMRB), generate an interrupt every 2 ms.
- Set the time interval at which to generate INTTB21 in the timer register TB2RG1.

■ Timer Register calculation

The prescaler output resolutions needed for calculation are shown in Table 3.6.4.

Table 3.6.4 Prescaler output clock resolution

@fc = 32 MHz

Peripheral Clock Selection <FPSEL>	Clock Gear Value <GEAR1:0>	Selection of Prescaler Clock <PRCK1:0>	Resolution of Prescaler Output Clock		
			ϕT1	ϕT4	ϕT16
0 (gear)	00 (fc)	00 (fperiph/4)	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)
		01 (fperiph/2)	$fc/2^2$ (0.125 μs)	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)
		10 (fperiph)	—	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)
	01 (fc/2)	00 (fperiph/4)	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)	$fc/2^8$ (8.0 μs)
		01 (fperiph/2)	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)
		10 (fperiph)	—	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)
	10 (fc/4)	00 (fperiph/4)	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)	$fc/2^9$ (16 μs)
		01 (fperiph/2)	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)	$fc/2^8$ (8.0 μs)
		10 (fperiph)	—	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)
	11 (fc/8)	00 (fperiph/4)	$fc/2^6$ (2.0 μs)	$fc/2^8$ (8.0 μs)	$fc/2^{10}$ (32 μs)
		01 (fperiph/2)	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)	$fc/2^9$ (16 μs)
		10 (fperiph)	—	$fc/2^6$ (2.0 μs)	$fc/2^8$ (8.0 μs)
1 (fc)	00 (fc)	00 (fperiph/4)	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)
		01 (fperiph/2)	$fc/2^2$ (0.125 μs)	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)
		10 (fperiph)	—	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)
	01 (fc/2)	00 (fperiph/4)	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)
		01 (fperiph/2)	—	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)
		10 (fperiph)	—	$fc/2^3$ (0.25 μs)	$fc/2^5$ (1.0 μs)
	10 (fc/4)	00 (fperiph/4)	—	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)
		01 (fperiph/2)	—	$fc/2^4$ (0.5 μs)	$fc/2^6$ (2.0 μs)
		10 (fperiph)	—	—	$fc/2^5$ (1.0 μs)
	11 (fc/8)	00 (fperiph/4)	—	$fc/2^5$ (1.0 μs)	$fc/2^7$ (4.0 μs)
		01 (fperiph/2)	—	—	$fc/2^6$ (2.0 μs)
		10 (fperiph)	—	—	$fc/2^5$ (1.0 μs)

Note 1: The prescaler's output clock ϕTn must be selected such that the relationship ϕTn < fsys/2 is satisfied (i.e. ϕTn must be slower than fsys/2).
Note 2: Do not change the clock gear value while the timer is operating.
Note 3: The dash character, —, in the table indicates a prohibited setting.

For ϕT1 ($fc/2^3$)

2000 μs ÷ 0.25 μs = 8000 (1F40H)

Set TB2RG1L to 00H and TB2RG1H to 1FH.

For ϕT4 ($fc/2^5$)

2000 μs ÷ 1.0 μs = 2000 (07D0H)

Set TB2RG1L to D0H and TB2RG1H to 07H.

For φT16 ($fc/2^7$)

2000 μs ÷ 4.0 μs = 500 (01F4H)

Set TB2RG1L to F4H and TB2RG1H to 01H.

3.6.6.2 Control method for the 2-ms Interval Timer

■ Initial settings

		7	6	5	4	3	2	1	0		
											Stop TMRB2.
TB2RUN	←	0	0	X	X	0	0	0	0		Disable trigger.
TB2FFCR	←	1	1	0	0	0	0	1	1		Select prescaler output clock as input clock and disable
TB2MOD	←	0	0	1	0	0	1	1	1		capture function.
											Set interval time.
TB2RG1L	←	1	1	1	1	0	1	0	0		Set TB2RG1 to 2000 μs÷ φT16 = 500.
TB2RG1H	←	0	0	0	0	0	0	0	1		
IMC8L	←	X	X	-	-	-	-	-	-	L	Set INTTB21 for rising edge and to level 4.
		X	X	1	1	0	0	1	1	H	
TB2RUN	←	0	0	X	X	0	1	0	1		Start TMRB2.

Note 1: X denotes Don't care; "–" denotes No change.

Note 2: The interrupt level can be set to any desired value.

■ Interrupt processing

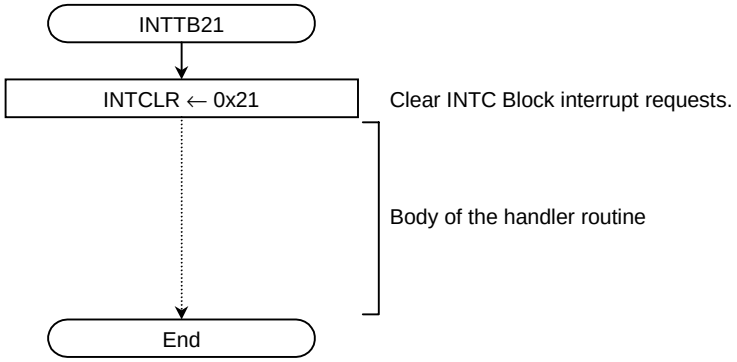


Figure 3.6.11 Flow of control for INTTB21 interrupt

3.6.7 Sample programs

3.6.7.1 Generic flowchart

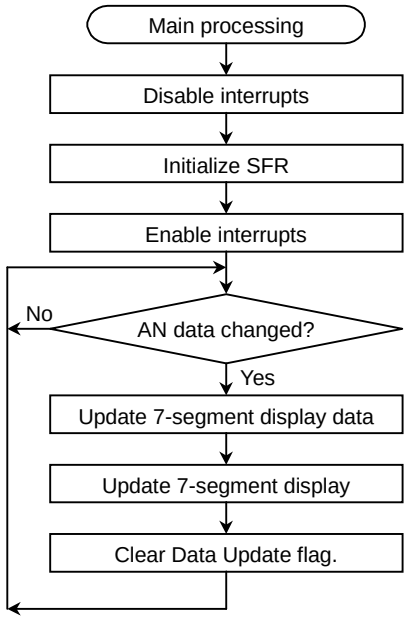


Figure 3.6.12 Main processing

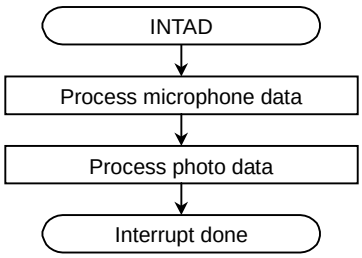


Figure 3.6.13 INTAD interrupt processing

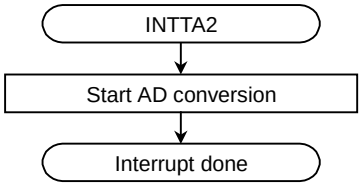


Figure 3.6.14 INTTA2 interrupt processing

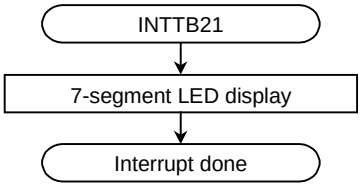


Figure 3.6.15 INTTB21 interrupt processing

3.6.7.2 File configuration

Table 3.6.5 lists the files used in the sample programs.

Table 3.6.5 File configuration

Filename	Contents	Page References
Stc9i16_anlg.asm	Start-up routine	3-12
io1940.c	Special function register definitions	3-51
analog_proc.c	Main photo- and mike-processing section	3-193
intrpt_analog.c	Interrupt-processing section	3-196
func_analog.c	Section for individual functions	3-197
io1940.h	Special function register declarations	3-28
ram_analog.h	External variable declarations	3-200

3.6.7.3 Vector table

The vector table used in the sample programs is shown below. Replace the vector table section in the start-up routine with the vector table given below.

- Interrupt vector table

```
_VecTable:
      dw      _Int_dummy      ;0 --- software set
      dw      _Int_dummy      ;1 --- INT[0]
      dw      _Int_dummy      ;2 --- INT[1]
      dw      _Int_dummy      ;3 --- INT[2]
      dw      _Int_dummy      ;4 --- INT[3]
      dw      _Int_dummy      ;5 --- INT[4]
      dw      _Int_dummy      ;6 --- *
      dw      _Int_dummy      ;7 --- *
      dw      _Int_dummy      ;8 --- *
      dw      _Int_dummy      ;9 --- *
      dw      _Int_dummy      ;10--- INT[5]
      dw      _Int_dummy      ;11--- INT[6]
      dw      _Int_dummy      ;12--- INT[7]
      dw      _Int_dummy      ;13--- INT[8]
      dw      _Int_dummy      ;14--- INT[9]
      dw      _Int_dummy      ;15--- INT[A]
      dw      _Int_dummy      ;16--- *
      dw      _Int_dummy      ;17--- *
      dw      _Int_dummy      ;18--- *
      dw      _Int_dummy      ;19--- *
      dw      _Int_dummy      ;20--- INTTA0 : 8bit Timer 0
      dw      _Int_dummy      ;21--- INTTA1 : 8bit Timer 1
      dw      _mintta2        ;22--- INTTA2 : 8bit Timer 2
      dw      _Int_dummy      ;23--- INTTA3 : 8bit Timer 3
      dw      _Int_dummy      ;24--- *
      dw      _Int_dummy      ;25--- *
      dw      _Int_dummy      ;26--- *
      dw      _Int_dummy      ;27--- *
      dw      _Int_dummy      ;28--- INTTB00 :16bit Timer 0 (TB0RG0)
      dw      _Int_dummy      ;29--- INTTB01 :16bit Timer 0 (TB0RG1)
      dw      _Int_dummy      ;30--- INTTB10 :16bit Timer 1 (TB1RG0)
      dw      _Int_dummy      ;31--- INTTB11 :16bit Timer 1 (TB1RG1)
      dw      _Int_dummy      ;32--- INTTB20 :16bit Timer 2 (TB2RG0)
      dw      _minttb21       ;33--- INTTB21 :16bit Timer 2 (TB2RG1)
      dw      _Int_dummy      ;34--- INTTN30 :16bit Timer 3 (TB3RG0)
      dw      _Int_dummy      ;35--- INTTB31 :16bit Timer 3 (TB3RG1)
      dw      _Int_dummy      ;36--- *
      dw      _Int_dummy      ;37--- *
```

```
dw      _Int_dummy      ;38--- *
dw      _Int_dummy      ;39--- *
dw      _Int_dummy      ;40--- INTTBOF0:16bit Timer 0 (OverFlow)
dw      _Int_dummy      ;41--- INTTBOF1:16bit Timer 1 (OverFlow)
dw      _Int_dummy      ;42--- INTTBOF2:16bit Timer 2 (OverFlow)
dw      _Int_dummy      ;43--- INTTBOF3:16bit Timer 3 (OverFlow)
dw      _Int_dummy      ;44--- *
dw      _Int_dummy      ;45--- *
dw      _Int_dummy      ;46--- *
dw      _Int_dummy      ;47--- *
dw      _Int_dummy      ;48--- INTRX0 :Serial receive (Channel 0)
dw      _Int_dummy      ;49--- INTTX0 :Serial transmit (Channel 0)
dw      _Int_dummy      ;50--- INTRX1 :Serial receive (Channel 1)
dw      _Int_dummy      ;51--- INTTX1 :Serial transmit (Channel 1)
dw      _Int_dummy      ;52--- INTS2  :Serial Channel 2 interrupt
dw      _Int_dummy      ;53--- *
dw      _Int_dummy      ;54--- INTRX3 :Serial receive (Channel 3)
dw      _Int_dummy      ;55--- INTTX3 :Serial transmit (Channel 3)
dw      _Int_dummy      ;56--- INTRX4 :Serial receive (Channel 4)
dw      _Int_dummy      ;57--- INTTX4 :Serial transmit (Channel 4)
dw      _Int_dummy      ;58--- INTRTC :Timer for RTC interrupt
dw      _mintad         ;59--- INTAD  :AD conversion finished
dw      _Int_dummy      ;60--- INTDMA0:DMA transfer finished (Channel 0)
dw      _Int_dummy      ;61--- INTDMA1:DMA transfer finished (Channel 1)
dw      _Int_dummy      ;62--- INTDMA2:DMA transfer finished (Channel 2)
dw      _Int_dummy      ;63--- INTDMA3:DMA transfer finished (Channel 3)
```

3.6.7.4 Source code

- Filename: analog_proc.c

```
/*
        ************************************************
        **              Application Note              **
        **                                            **
        **              ( MAIN ROUTINE )              **
        **                                            **
        **                    MCU                     **
        **                  TX1940                    **
        **                                            **
        **              fc      = 32MHz               **
        **              fsys    = 32MHz               **
        **              fperiph = 32MHz               **
        **                                            **
        **                                            **
        ************************************************
        ************************************************
        *      COPYRIGHT(C) 1999 TOSHIBA CORPORATION     *
        *              ALL RIGHTS RESERVED               *
        ************************************************/


/***********************************************/
/*              Loading header files           */
/***********************************************/
#include "io1940.h"
#include "ram_analog.h"
#include <stdlib.h>


/*****************************************************************************/
/* Module name  :                                                            */
/*****************************************************************************/
/* Function    :Initializes photo-interrupter and microphone processing      */
/* Inputs      :                                                             */
/* Outputs     :                                                             */
/* Parameters  :None                                                         */
/* Return value:None                                                         */
/*****************************************************************************/
void manalog_init(void){
/*--- CG settings ---*/
        IO_SYSCR0   = 0xf0;         /* <XEN>=1 <XTEN>=1 <RXEN>=1 <RXTEN>=1 */
                                    /* <RSYSCR>=0 <WUEF>=0 <PRCK>=00 */
        IO_SYSCR1   = 0x10;         /* <SYSCR>=0 <FPSEL>=1 <DF0SC>=0 <GEAR>=0 */
        IO_ADCCLK   = 0x01;         /* <ADCCLK>=01 */

/*--- I/O port settings  ---*/
        IO_P4       = 0x10;         /* Output High on P44 */
        IO_P4CR     = 0x1f;         /* Set P44-P40 for output */
        IO_P4FC     = 0x0f;         /* Set P43-P40 to be *CS */

        IO_P7       = 0x6d;         /* Output High on P76,P75,P73,P72,P70 */
        IO_P7CR     = 0x6f;         /* Set P76,P75,P74,P73,P72,P71,P70 for output and
                                       P77,P74 for input */
        IO_P7FC     = 0x00;         /* Set P77-P70 to be a port */

        IO_P8       = 0xff;         /* Output High on P87-P80 */
        IO_P8CR     = 0xff;         /* Set P87-P80 for output  */
        IO_P8FC     = 0x00;         /* Set P87-P80 to be a port */

        IO_P9CR     = 0xc9;         /* Set P97,P96,P93,P90 for output and P95,P94,P92,P91
```

```
                                        for input */
        IO_P9FC     = 0x09;         /* Set P93 to be TXD1, P90 to be TXD0 and others to be
                                        ports */

        IO_PA       = 0x1f;         /* Output High on PA4-PA0 */
        IO_PACR     = 0x3f;         /* Set PA5-PA0 for output and PA7-PA6 for input */
        IO_PAFC     = 0x00;         /* Set PA7-PA0 to be a port*/

/*--- AD conversion settings ---*/
        IO_ADMOD0   = 0x02;         /* <ITM0>=0 <REPEAT>=0 <SCAN>=1 <ADS>=0 */
        IO_ADMOD1   = 0x81;         /* <VREFN>=1 <12AD>=0 <ADTREG>=0 <ADCH>=001 */
        IO_IMCEH    = 0x3530;       /* Rising edge, interrupt level 5 */

/*--- 1-ms Interval Timer settings ---*/
/* TMRA2 */
        IO_TA23RUN  = 0x00;         /* <TA2RDE>=0 <I2TA23>=0 <TA23PRUN>=0 <TA3RUN>=0
                                        <TA2RUN>=0 */
        IO_TA23MOD  = 0x03;         /* <TA23M>=00 <PWM>=00 <TA2CLK>=11 */
        IO_TA2REG   =  250;         /* TA2REG = 1000/4 */
        IO_IMC5H    = 0x3036;       /* Rising edge, interrupt level 6 */

/*--- 2-ms Interval Timer settings ---*/
/* TMRB2 */
        IO_TB2RUN   = 0x00;         /* <TB2RDE>=0 <I2TB2>=0 <TB2PRUN>=0 <TB2RUN>=0 */
        IO_TB2FFCR  = 0xc3;         /* <TB2C1T1>=0 <TB2C0T1>=0 <TB2E1T1>=0 */
                                    /* <TB2E0T1>=0 <TB2FF0C>=11 */
        IO_TB2MOD   = 0x27;         /* <TB2CP0I>=1 <TB2CPM>=00 <TB2CLE>=1 <TB2CLK>=11 */
                                    /* ϕT16=4.0usec*/
        IO_TB2RG1L  = 0xf4;         /* TB2RG1 = 2000/4 */
        IO_TB2RG1H  = 0x01;         /*                   */
        IO_IMC8L    = 0x3430;       /* Rising edge, interrupt level 4 */

/*--- Timer start ---*/
        IO_TA23RUN  = 0x05;         /* <TA2RDE>=0 <I2TA23>=0 <TA23PRUN>=1 <TA3RUN>=0
                                        <TA2RUN>=1 */
        IO_TB2RUN   = 0x05;         /* <TB2RDE>=0 <I2TB2>=0 <TB2PRUN>=1 <TB2RUN>=1 */
}


/************************************************************************/
/* Module name  :main                                                   */
/************************************************************************/
/* Function     :Main processing                                        */
/* Inputs       :fmic_disp,gmaxbuffer,fphoto_disp,gphoto_ave            */
/* Outputs      :fmic_disp,gphoto_level,gled_data,fphoto_disp           */
/* Parameters   :None                                                   */
/* Return value :None                                                   */
/************************************************************************/
void main(void){

        __DI();

/*--- Initializing SFR ---*/
        manalog_init();

        __EI();

        for(;;){
        /*--- 7SEG-LED display data (microphone output level) change ---*/
                switch(fan_data){
                case    0:
                        break;
                case    1:
                        if(gmaxbuffer <= Vref && gmaxbuffer >= c90_per){
                                g7seg_data[3] = c7seg_0;
                                g7seg_data[2] = c7seg_0;
```

```
                                        g7seg_data[1] = c7seg_0;
                                        g7seg_data[0] = c7seg_0;
                                }
                                else if(gmaxbuffer < c90_per && gmaxbuffer >= c80_per){
                                        g7seg_data[3] = c7seg_0;
                                        g7seg_data[2] = c7seg_0;
                                        g7seg_data[1] = c7seg_0;
                                        g7seg_data[0] = c7seg_all_clear;
                                }
                                else if(gmaxbuffer < c80_per && gmaxbuffer >= c70_per){
                                        g7seg_data[3] = c7seg_0;
                                        g7seg_data[2] = c7seg_0;
                                        g7seg_data[1] = c7seg_all_clear;
                                        g7seg_data[0] = c7seg_all_clear;
                                }
                                else if(gmaxbuffer < c70_per && gmaxbuffer >= c60_per){
                                        g7seg_data[3] = c7seg_0;
                                        g7seg_data[2] = c7seg_all_clear;
                                        g7seg_data[1] = c7seg_all_clear;
                                        g7seg_data[0] = c7seg_all_clear;
                                }
                                else if(gmaxbuffer < c60_per && gmaxbuffer >= c50_per){
                                        g7seg_data[3] = c7seg_all_clear;
                                        g7seg_data[2] = c7seg_all_clear;
                                        g7seg_data[1] = c7seg_all_clear;
                                        g7seg_data[0] = c7seg_all_clear;
                                }
                        /*--- 7SEG-LED display data (photo-interrupter output level) change ---*/
                                if(gphoto_ave<=1023 && gphoto_ave>= 911)        gphoto_level=8;
                                else if(gphoto_ave <= 910 && gphoto_ave >= 797) gphoto_level=7;
                                else if(gphoto_ave <= 796 && gphoto_ave >= 683) gphoto_level=6;
                                else if(gphoto_ave <= 682 && gphoto_ave >= 570) gphoto_level=5;
                                else if(gphoto_ave <= 569 && gphoto_ave >= 456) gphoto_level=4;
                                else if(gphoto_ave <= 455 && gphoto_ave >= 342) gphoto_level=3;
                                else if(gphoto_ave <= 341 && gphoto_ave >= 279) gphoto_level=2;
                                else if(gphoto_ave <= 278 && gphoto_ave >= 116) gphoto_level=1;
                                else if(gphoto_ave <= 114 && gphoto_ave >= 000) gphoto_level=0;
                                gled_data = tled_ch[gphoto_level];
                        /*--- Clear analog data change flag ---*/
                                fan_data = 0;
                                break;
                        }

                }
        }
```

- Filename: intrpt_analog.c

```
/*
        ************************************************
        **              Application Note              **
        **                                            **
        **           ( INTERRUPT ROUTINE )            **
        **                                            **
        **                  MCU                       **
        **                TX1940                      **
        **                                            **
        **              fc      = 32MHz               **
        **              fsys    = 32MHz               **
        **              fperiph = 32MHz               **
        **                                            **
        **                                            **
        ************************************************
        ************************************************
        *      COPYRIGHT(C) 1999 TOSHIBA CORPORATION     *
        *              ALL RIGHTS RESERVED               *
        ************************************************/



/************************************************/
/*              Loading header files            */
/************************************************/
#include "io1940.h"
#include "ram_analog.h"


/************************************************/
/*         External variable declarations       */
/************************************************/
unsigned short  gad0_data,gad1_data;            /* AD conversion result (Ch.0,Ch1) */
volatile  unsigned char  fan_data = 1;          /* AN Data Change flag */

/*****************************************************************************/
/* Module name :minttb21                                                     */
/*****************************************************************************/
/* Function    :AD Conversion-Finished interrupt (interrupt)                 */
/* Inputs      :ADREG04H,ADREG04L,ADREG15H,ADREG15L                          */
/* Outputs     :INTCLR,gad0_data,gad1_data                                   */
/*****************************************************************************/
void __interrupt mintad(void){

        IO_INTCLR = 0x3b;                       /* Clear interrupt latch */

/*--- AN0 (microphone IC output) processing ---*/
        gad0_data = (IO_ADREG04H<<2)+(IO_ADREG04L>>6);
        mmic_proc();
/*--- AN1 (photo-interrupter output) processing ---*/
        gad1_data = (IO_ADREG15H<<2)+(IO_ADREG15L>>6);
        mphoto_proc();
/*--- Set Analog Data Change flag ---*/
        fan_data = 1;
}


/*****************************************************************************/
/* Module name  :mintta2                                                     */
/*****************************************************************************/
/* Function     :1-ms Interval Timer (interrupt)                             */
/* Inputs       :None                                                        */
/* Outputs      :INTCLR,ADMOD0                                               */
```

```
/*************************************************************************/
void __interrupt mintta2(void){

        IO_INTCLR = 0x16;                       /* Clear interrupt latch */
        IO_ADMOD0 = 0x03;
}

/*************************************************************************/
/* Module name :minttb21                                                 */
/*************************************************************************/
/* Function    :2-ms Interval Timer (interrupt)                          */
/* Inputs      :None                                                     */
/* utputs      :INTCLR                                                   */
/*************************************************************************/
void __interrupt minttb21(void){

        IO_INTCLR = 0x21;                       /* Clear interrupt latch */
        p7seg_disp();
}
```

- Filename: func_analog.c

```
/*
        ************************************************
        **              Application Note              **
        **                                            **
        **            ( FUNCTION ROUTINE )            **
        **                                            **
        **                   MCU                      **
        **                 TX1940                     **
        **                                            **
        **              fc      = 32MHz               **
        **              fsys    = 32MHz               **
        **              fperiph = 32MHz               **
        **                                            **
        **                                            **
        ************************************************
        ************************************************
        *      COPYRIGHT(C) 1999 TOSHIBA CORPORATION     *
        *               ALL RIGHTS RESERVED              *
        ************************************************/

/***********************************************/
/*              Loading header files           */
/***********************************************/
#include "io1940.h"
#include "ram_analog.h"


/***********************************************/
/*              Constant definitions           */
/***********************************************/
/*--- Definitions of port output (7SEG-LED digit) ---*/
#define  c7seg_digit_out0           0x1e        /* PA (00011110) */
#define  c7seg_digit_out1           0x1d        /* PA (00011101) */
#define  c7seg_digit_out2           0x1b        /* PA (00011011) */
#define  c7seg_digit_out3           0x17        /* PA (00010111) */
#define  c7seg_digit_out4           0x0f        /* PA (00001111) */
/*--- Number of analog data storage buffers ---*/
#define  LIMITM                     32          /* Number of microphone output values
                                                   stored */
#define  LIMITP                     8           /* photo-interrupter Number of photo
                                                   interrupt output values stored */
```

```
#define  digit(pos)      ((pos+1)%5)
#define  nextm(pos)      ((pos+1)%LIMITM)
#define  nextp(pos)      ((pos+1)%LIMITP)


/*********************************************/
/*        External variable declarations         */
/*********************************************/
unsigned char        g7seg_data[4];              /* 7SEG-LED display data */
unsigned char        gled_data;                  /* LED display data */
unsigned short       gmaxbuffer  = 0;            /* Maximum microphone output value */
unsigned short       gphoto_ave;                 /* photo-interrupter average data */
unsigned char        gphoto_level;               /* photo-interrupter display level data */


/*****************************************************************************/
/* Module name   :p7seg_disp                                                 */
/*****************************************************************************/
/* Function      :7-segment LED display output (using 2-ms Interval Timer interrupt)*/
/* Inputs        :gled_data,g7seg_data                                       */
/* Outputs       :P8,PA                                                      */
/* Parameters    :None                                                       */
/* Return value  :None                                                       */
/*****************************************************************************/
void p7seg_disp(void){
         static unsigned char ldisp_digit = 0;

/*** Turning display off ***/
         IO_P8 = 0xff;

/*** Display output ***/
         switch (ldisp_digit) {
         case 0:
                  IO_PA = c7seg_digit_out0;       /* Update digits */
                  IO_P8 = gled_data;              /* Output on 7-segment display */
                  break;
         case 1:
                  IO_PA = c7seg_digit_out1;       /* Update digits */
                  IO_P8 = g7seg_data[0];          /* Output on 7-segment display */
                  break;
         case 2:
                  IO_PA = c7seg_digit_out2;       /* Update digits */
                  IO_P8 = g7seg_data[1];          /* Output on 7-segment display */
                  break;
         case 3:
                  IO_PA = c7seg_digit_out3;       /* Update digits */
                  IO_P8 = g7seg_data[2];          /* Output on 7-segment display */
                  break;
         case 4:
                  IO_PA = c7seg_digit_out4;       /* Update digits */
                  IO_P8 = g7seg_data[3];          /* Output on 7-segment display */
                  break;
         }

/*** Counting digits  ***/
         ldisp_digit = digit(ldisp_digit);
}

/*********************************************/
/*              Display data table              */
/*********************************************/
const unsigned char tled_ch[] = {
         0xff,0x7f,0x3f,0x1f,0x0f,0x07,0x03,0x01,0x00
};
```

```
/*****************************************************************************/
/* Module name  :mmic_proc                                                   */
/*****************************************************************************/
/* Function     :Processes microphone output signal                          */
/* Inputs       :gad0_data                                                   */
/* Outputs      :gmaxbuffer,fmic_disp                                        */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/*****************************************************************************/
void mmic_proc(void){

        static unsigned char        index = 0,i = 0;
        static unsigned short       buffer[LIMITM];

/*--- Allocating area for microphone output data in buffer ---*/
        buffer[index] = gad0_data;
        index = nextm(index);                          /* Increment array index */
/*--- Allocating area for maximum value in buffer ---*/
        gmaxbuffer = 0;
        for(i=0;i<LIMITM;i++){
                if( gmaxbuffer < buffer[i] )
                        gmaxbuffer = buffer[i];
        }
}


/*****************************************************************************/
/* Module name  :mphoto_proc                                                 */
/*****************************************************************************/
/* Function     :Processes photo-interrupter output signal                   */
/* Inputs       :gad1_data                                                   */
/* Outputs      :gphoto_ave,fphoto_disp                                      */
/* Parameters   :None                                                        */
/* Return value :None                                                        */
/*****************************************************************************/
void mphoto_proc(void){

        static unsigned int         sum = 0;
        static unsigned char        index = 0;
        static unsigned short       buffer[LIMITP];

/*--- Allocating area for photo-interrupter output data in buffer ---*/
        sum  -= buffer[index];
        buffer[index]    = gad1_data;   /* Allocate area for current microphone output
                                           value */
        sum  += buffer[index];
/*--- Storing average ---*/
        gphoto_ave   = sum>>3;
/*--- Increment array index ---*/
        index = nextp(index);
}
```

- Filename: ram_analog.h

```
/*--- Declarations of external variables ---*/
extern     unsigned char           g7seg_data[4];
extern     unsigned char           gled_data;
extern     unsigned short          gad0_data;
volatile   extern unsigned char    fan_data;
extern     unsigned short          gad1_data;
extern     unsigned short          gmaxbuffer;
extern     unsigned short          gphoto_ave;
extern     unsigned char           gphoto_level;


/*--- Constant definitions ---*/
/* Microphone output ranges */
#define    Vref                    1023
#define    c50_per                 5*Vref/10
#define    c60_per                 6*Vref/10
#define    c70_per                 7*Vref/10
#define    c80_per                 8*Vref/10
#define    c90_per                 9*Vref/10
/* 7SEG-LED data table */
#define    c7seg_0                 0xc0
#define    c7seg_all_clear         0xff

/*--- CONST constant ---*/
extern     const unsigned char     tled_ch[];
```

3.7 Other Facilities

3.7.1 Watchdog Timer feature

3.7.1.1 Overview of the Watchdog Timer

The Watchdog Timer is used for detecting CPU runaway. If the CPU starts operating erratically for some reason, e.g. because of noise, the Watchdog Timer detects the erroneous condition and generates a non-maskable interrupt, notifying the CPU of the runaway condition and instructing it to return to its normal state.

The Watchdog Timer can also be connected to the chip's internal reset input, enabling the processor to be reset automatically when runaway occurs.

- Setting the Watchdog Timer detection time

Table 3.7.1 Watchdog Timer detection time

System Clock Selection <SYSCK>	SYSCR1 Clock Gear Value <GEAR1:0>	Watchdog Timer Detection Time			
		WDMOD<WDTP1, 0>			
		00 $2^{16}/f_{SYS}$	01 $2^{18}/f_{SYS}$	10 $2^{20}/f_{SYS}$	11 $2^{22}/f_{SYS}$
1 (fs)	xxx	2.0 s	8.0 s	32.0 s	128.0 s
0 (gear)	00 (fc)	2.048 ms	8.192 ms	32.768 ms	131.072 ms
	01 ($fc/_2$)	4.096 ms	16.384 ms	65.536 ms	262.144 ms
	10 ($fc/_4$)	8.192 ms	32.768 ms	131.072 ms	524.288 ms
	11 ($fc/_8$)	16.384 ms	65.536 ms	262.144 ms	1048.576 ms

@ fc = 32 MHz, fs = 32.768 kHz

In the sample program 2^{22}/fsys (WDTP = '11') is used and a runaway condition is detected in 131.072 ms.

- Clearing the binary counter

Ordinarily the binary counter should be cleared at several points within the main processing routine, not just at a single point. The binary counter is not normally cleared during interrupt processing.

WDCR ← 4Eh Clear code is written.

- Interrupt processing

When an INTWD occurs, the interrupt handler should branch to a non-maskable exception. This should transfer control to a runaway correction program which will restore normal conditions. The details of the runaway correction program are not covered here.

3.7.1.2 Watchdog Timer control method

Figure 3.7.1 illustrates the flow of control for the Watchdog Timer.

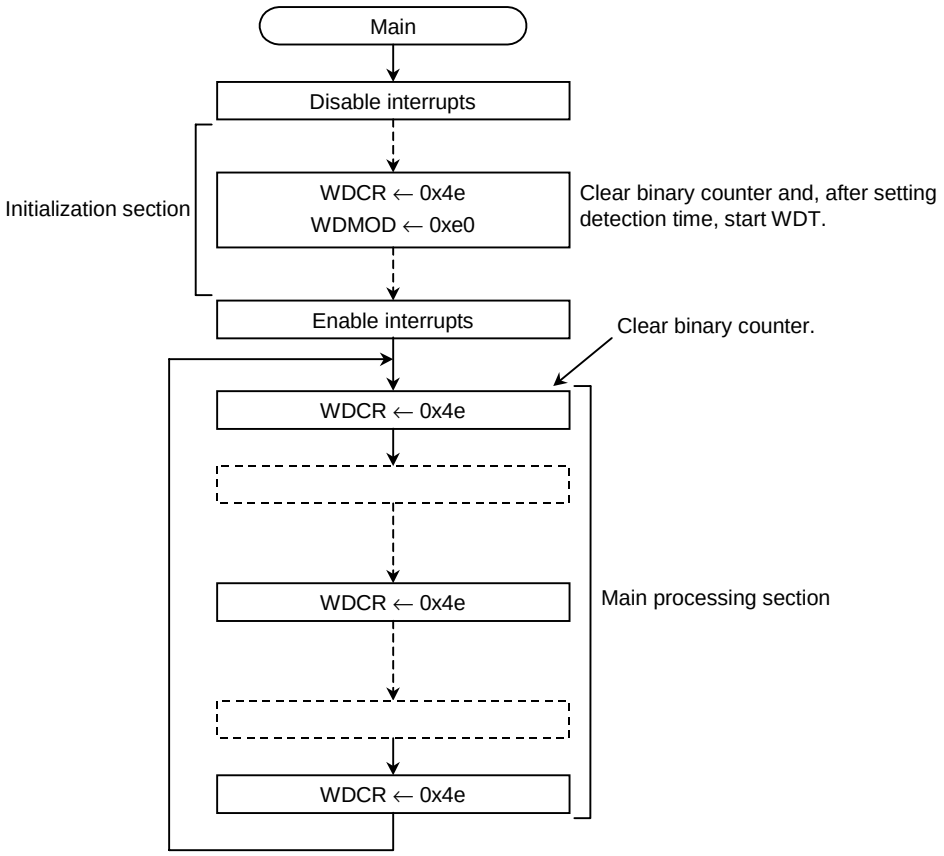


Figure 3.7.1 Flow of control for Watchdog Timer

3.7.2 Example of how to use the DMA controller

This section presents an example of how to set the parameters for DMA transfer in Internal Transfer Request Mode. In this example the source and destination memory addresses for the transfer are specified.

3.7.2.1 Set-up parameters for sample program

The following settings are the main parameters used for performing DMA transfer in this program. Other settings vary according to the functions which are used. Please refer to the relevant section for further details (in most cases the CCRn registers are used to make these other settings).

Table 3.7.2 Settings for DMA transfer

Parameter	Setting
DMA channel used	Channel 0
Transfer request mode	Internal Transfer Request Mode
Snoop function	Used
Source device for transfer	Memory (internal RAM)
Destination device for transfer	Memory (internal RAM)
Address mode	Dual-Address Mode (this mode is only available on the TX1940FDBF)
Normal termination	Interrupts not used
Abnormal termination	Interrupts not used

3.7.2.2 Generating physical addresses

- Generation of physical addresses

SAR0 and DAR0 must be set to hold physical addresses. Virtual addresses must be converted to physical addresses in software. Figure 3.7.2 shows the address map for the TX19L. Figure 3.7.3 shows the memory map for the TX1940FDBF. Based on these maps Table 3.7.3 shows the correspondence between the virtual addresses and the physical addresses.

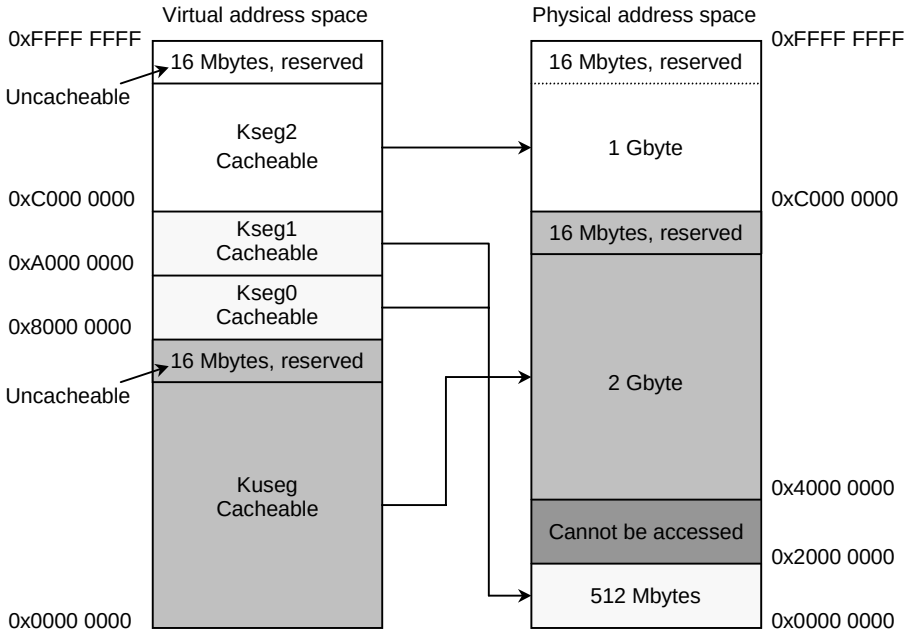
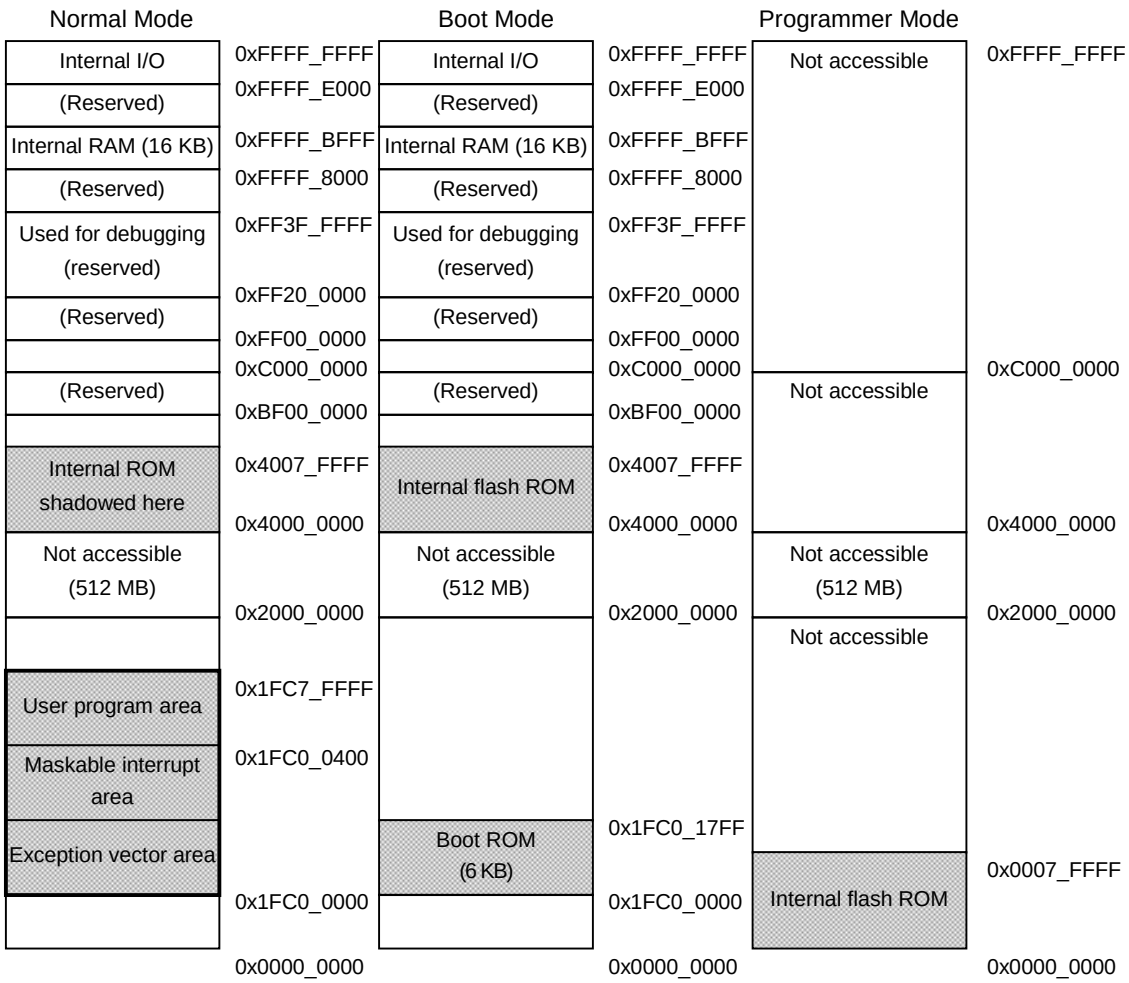


Figure 3.7.2 Memory map of TX19L



Note 1: The addresses shown above are physical addresses.

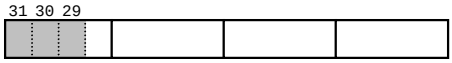
Figure 3.7.3 TX1940FDBF memory map for each mode

Table 3.7.3 Correspondence between virtual addresses and physical addresses

Virtual Addresses	Physical Addresses
0x0000_0000 ~ 0x7FFF_FFFF	0x4000_0000 ~ 0xBFFF_FFFF
0x8000_0000 ~ 0x9FFF_FFFF	0x0000_0000 ~ 0x1FFF_FFFF
0xA000_0000 ~ 0xBFFF_FFFF	0x0000_0000 ~ 0x1FFF_FFFF
0xC000_0000 ~ 0xFFFF_FFFF	0xc000_0000 ~ 0xFFFF_FFFF

As illustrated by Table 3.7.3, in the application shown here the conversion of a virtual address to a physical address involves changing only the three high-order bits. This is shown in Figure 3.7.4.

■ Address conversion



The three high-order bits are looked up in the conversion table.

Virtual Address	<3 high-order bits>	Physical Address	<3 high-order bits>	Virtual Address	<3 high-order bits>	Physical Address	<3 high-order bits>
0000_0000 ↓ 1FFF_FFFF	31 30 29 0 0 0		31 30 29 0 1 0	8000_0000 ↓ 9FFF_FFFF	31 30 29 1 0 0		31 30 29 0 0 0
2000_0000 ↓ 3FFF_FFFF	31 30 29 0 0 1		31 30 29 0 1 1	A000_0000 ↓ BFFF_FFFF	31 30 29 1 0 1		31 30 29 0 0 1
4000_0000 ↓ 5FFF_FFFF	31 30 29 0 1 0		31 30 29 1 0 0	C000_0000 ↓ DFFF_FFFF	31 30 29 1 1 0		31 30 29 1 1 0
6000_0000 ↓ 7FFF_FFFF	31 30 29 0 1 1		31 30 29 1 0 1	E000_0000 ↓ FFFF_FFFF	31 30 29 1 1 1		31 30 29 1 1 1

Figure 3.7.4 Address conversion

- Conversion of virtual addresses to physical addresses involves changing only the three high-order bits.
- In the sample program the virtual addresses are divided into eight discrete areas as shown in Figure 3.7.4. The data that should be converted the address can be found in the table.
- The sample program uses the conversion table VtoP_Table[ ] and the virtual-to-physical address conversion function mVtoP( ).

3.7.2.3 Functional description

■ mFillArray

This function transfers data from a fixed address to a dynamic address one byte at a time. The first parameter indicates the destination address of the transfer, the second parameter indicates the source address of the transfer and the third parameter indicates the number of bytes to be transferred. The source of the data transfer is gFillData (1 byte) and the destination is ucArray_A[ ] or ucArray_B[ ] (1 byte × 256). Since the data size for the transfer source is 1 byte, the device port size and the unit of transfer are both set to 1 byte.

Table 3.7.4 Set-up of mFillArray

Set-up Parameter	Settings
Source address count	Address fixed
Destination address count	Address incremented
Unit of transfer	1 byte
Device port size	1 byte
Destination address counter	From bit 0
Source address counter	From bit 0

■ mCopyArray

This function transfers data from a dynamic address to a dynamic address four bytes at a time. The arrays ucArray_A[ ] and ucArray_B[ ] can both be specified as either the source or the destination of the data transfer. The first parameter indicates the destination address of the transfer, the second parameter indicates the source address of the transfer and the third parameter indicates the number of bytes to be transferred. Since the transfer size and device port size are both 4 bytes, the Byte Count Register and the number of array elements must both be multiples of 4.

Table 3.7.5 Set-up of mCopyArray

Set-up Parameter	Settings
Source address count	Address incremented
Destination address count	Address incremented
Unit of transfer	4 bytes
Device port size	4 bytes
Destination address counter	From bit 0
Source address counter	From bit 0

3.7.2.4 Sample program

3.7.2.4.1 Generic flowchart

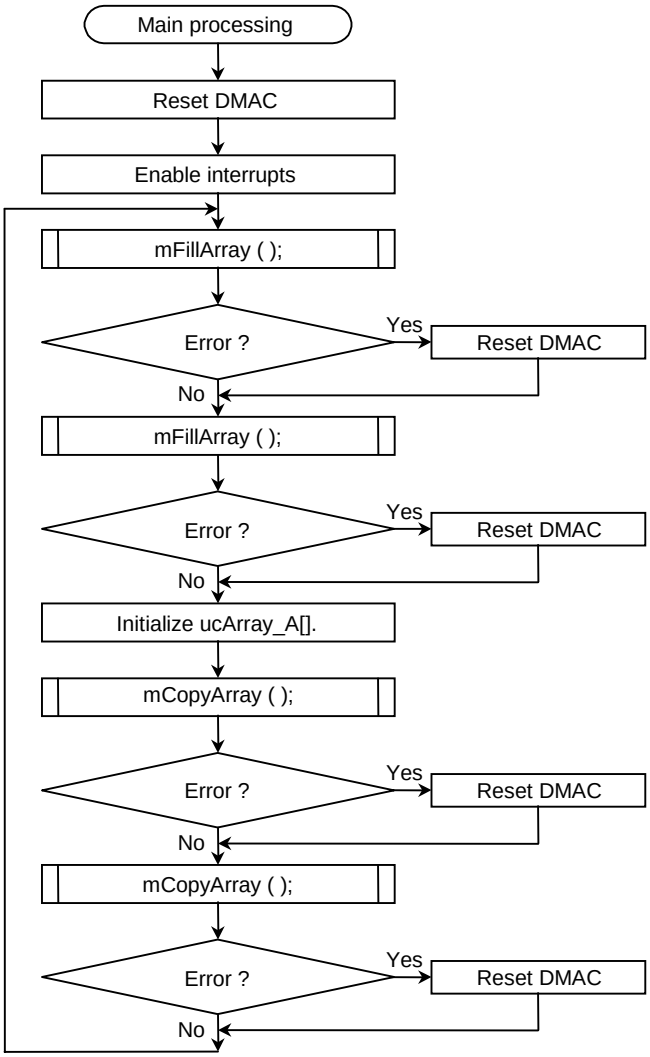


Figure 3.7.6 Main processing

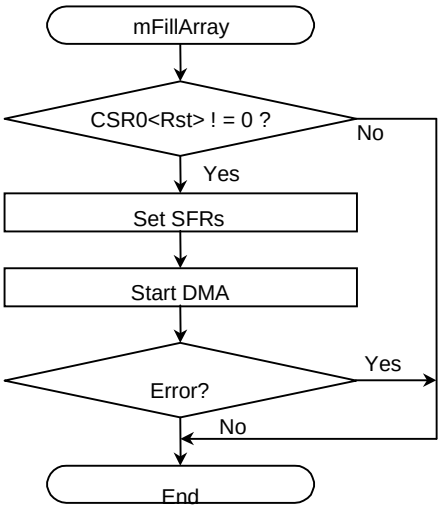


Figure 3.7.5 Function processing

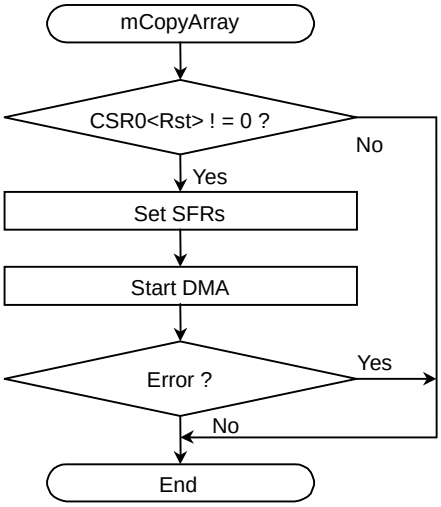


Figure 3.7.7 Function processing

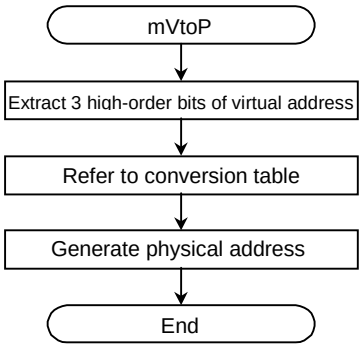


Figure 3.7.8 Address conversion

3.7.2.4.2 File configuration

Table 3.7.6 list the files used in the sample program.

Table 3.7.6 File configuration

Filename	Contents	Page References
Stc9i16.asm	Start-up routine	3-12
io1940.c	Special function register declarations	3-51
Dma_src.c	DMAC program	3-208
io1940.h	Special function register definitions	3-28

3.7.2.4.3 Source code

- Filename: DMA_src.c

```
/*
      ************************************************
      **           Application Note                 **
      **                                            **
      **             ( FUNCTION ROUTINE )           **
      **                                            **
      **           MCU                              **
      **               TX1940                       **
      **                                            **
      **            fc      = 32MHz                 **
      **            fsys    = 32MHz                 **
      **            fperiph = 32MHz                 **
      **                                            **
      **                                            **
      ************************************************
      ************************************************
      *      COPYRIGHT(C) 1999 TOSHIBA CORPORATION     *
      *               ALL RIGHTS RESERVED              *
      ************************************************/


/***********************************************/
/*              Loading header files           */
/***********************************************/
#include "io1940.h"
#include <stdlib.h>


/***********************************************/
/*              Constant definitions           */
/***********************************************/
#define      NUM      256
#define      FALSE    (0==1)
#define      TRUE     (0==0)

/***********************************************/
/*                Type definitions             */
/***********************************************/
typedef      int      Boolean;


/***********************************************/
/*         External variable definitions       */
/***********************************************/
/* Position arrays in global memory */
unsigned char      __tiny ucArray_A[NUM],
                   __tiny ucArray_B[NUM],
```

```
                  __tiny gFillData;


/* ************************************************************************** */
/* Module name  : mVtoP                                                       */
/* ************************************************************************** */
/* Function     : Converts virtual addresses to physical addresses            */
/* Inputs       : None                                                        */
/* Outputs      : None                                                        */
/* Parameter    : Address                                                     */
/* Return value : physical address                                           */
/* ************************************************************************** */
/**/
const unsigned char      VtoP_Table[8] = {2,3,4,5,0,0,6,7};
unsigned long mVtoP( unsigned long Address){
      unsigned long      work1,
                         work2;
      work1 = Address & 0x1fffffff;
      work2 = (Address >> 29);
      return( ( VtoP_Table[ work2 ]<<29 ) + work1 );
}


/* ************************************************************************** */
/* Module name  : mFillArray                                                  */
/* ************************************************************************** */
/* Function     : Transfers data from fixed address to dynamic address using DMA */
/* Inputs       : CSR0                                                        */
/* Outputs      : SAR0,DAR0,BCR0,CCR0,CSR0                                    */
/* Parameters   : *Array,iCount,ucData                                        */
/* Return value : Boolean                                                     */
/* ************************************************************************** */
/**/
Boolean     mFillArray( unsigned char *Array, unsigned char ucData, unsigned int iCount ){
      unsigned int      dma_status;

      dma_status = IO_CSR0;
      if ((dma_status & 0x80000000) != 0x0){
            /* The value of this bit is 1 after DMA has been set up but before transfer. */
            /* Hence, the value is 0 before DMA has been set up */
            return FALSE;
      }
      gFillData = ucData;
      IO_DTCR0 = 0;
      IO_SAR0  = mVtoP((unsigned long)&gFillData);/* Set source address */
      IO_DAR0  = mVtoP((unsigned long)Array);     /* Set destination address */
      IO_BCR0  = iCount;                        /* Set number of bytes to be transferred */
      IO_CCR0  = 0x8000090F;                       /* <Str>=1 <NIEn>=0 <AbEn>=0 <Big>=0 */
                                          /* <ExR>=0 <PosE>=0(Ignored) <Lev>=0(Ignored) */
                                          /* <SReq>=1 <RelEn>=0(Ignored) <SIO>=0 */
                                          /* <SAC>=10 <DIO>=0 <DAC>=00 <Trsiz>=11 */
                                          /* <DPS>=11 */

      while( (IO_CSR0 & 0x80000000) != 0x00 ){
      }
      dma_status = IO_CSR0;
      IO_CSR0 = 0;
      if ((dma_status & 0x005C0000) != 0){ /* <AbC> <BES> <BED> <Conf> */
            return FALSE;                  /* Error if any field is 1 */
      }
      return      TRUE;
}


/* ************************************************************************** */
/* Module name  : mCopyArray                                                   */
```

```
/* ************************************************************************** */
/* Function     : Transfer data from dynamic address to dynamic address using DMA */
/* Inputs       : CSR0                                                        */
/* Outputs      : SAR0,DAR0,BCR0,CCR0,CSR0                                    */
/* Parameters   : *SrcArray,*DstArray,iCount                                  */
/* Return value : Boolean                                                     */
/* ************************************************************************** */
/**/
Boolean    mCopyArray( unsigned char *DstArray,unsigned char *SrcArray, unsigned int iCount ){
      unsigned int      dma_status,dummy;

      dma_status = IO_CSR0;
      if ((dma_status & 0x80000000) != 0x0){
            return FALSE;
      }
      IO_DTCR0 = 0;
      IO_SAR0  = mVtoP((unsigned long)SrcArray);      /* Set source address */
      IO_DAR0  = mVtoP((unsigned long)DstArray);      /* Set destination address */
      IO_BCR0  = iCount;                  /* Set number of bytes to be transferred */
      IO_CCR0  = 0x80000800;              /* <Str>=1 <NIEn>=0 <AbEn>=0 <Big>=0 */
                                          /* <ExR>=0 <PosE>=0(Ignored) <Lev>=0(Ignored) */
                                          /* <SReq>=1 <RelEn>=0(Ignored) <SIO>=0 */
                                          /* <SAC>=00 <DIO>= <DAC>=00 <Trsiz>=11 */
                                          /* <DPS>=11 */
      while( (CSR0 & 0x80000000) == 0x00 ){
      }
      IO_dma_status = IO_CSR0;
      IO_CSR0 = 0;
      if ( (dma_status & 0x005C0000) != 0){  /* <AbC> <BES> <BED> <Conf> */
            return FALSE;                    /* Error if any field is 1 */
      }
      return      TRUE;
}


/* ************************************************************************** */
/* Module name  : main                                                        */
/* ************************************************************************** */
/* Function     : Main processing                                             */
/* Inputs       : mFillArray,mCopyArray                                       */
/* Outputs      : DCR                                                         */
/* Parameters   : None                                                        */
/* Return value : None                                                        */
/* ************************************************************************** */
/**/
void      main(void){
      Boolean      status;
      int      i;

      /*--- Software reset --- */
      IO_DCR = 0x80000000;                /* <Rst>=0 */

      __EI();

      for(;;){
            status = mFillArray( ucArray_A,0x33,NUM );
            if (status == FALSE){
            /*--- Software reset ---*/
                  IO_DCR = 0x80000000;      /* <Rst>=1 */
            }

            status = mFillArray( ucArray_B,0x55,NUM );
            if (status == FALSE){
            /*--- Software reset ---*/
                  IO_DCR = 0x80000000;      /* <Rst>=1 */
            }
```

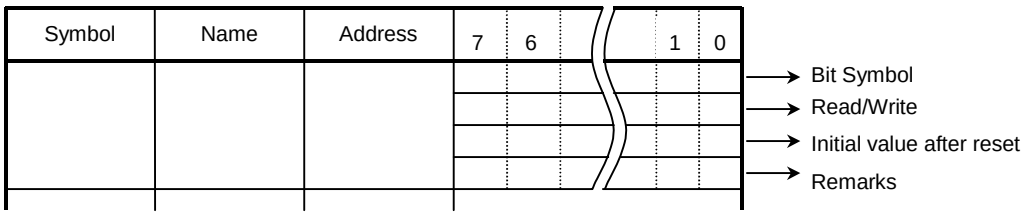
```
                for(i=0;i<NUM;i++){
                      ucArray_A[i] = i;
                }

                status = mCopyArray(ucArray_A,ucArray_B,NUM);
                if (status == FALSE){
                /*--- Software reset ---*/
                      IO_DCR = 0x80000000;      /* <Rst>=1 */
                }

                for(i=0;i<NUM;i++){
                      ucArray_A[i] = i;
                }

                status = mCopyArray(ucArray_B,ucArray_A,NUM);
                if (status == FALSE){
                /*--- Software reset ---*/
                      IO_DCR = 0x80000000;      /* <Rst>=1 */
                }
        }
}
```

3.8 List of special function registers used

A list of the special function registers used in the sample programs is shown below. For information regarding any other special function registers, please refer to the "32-Bit TX System RISC TX19 Family TX1940."

(1) Input/Output Port Register
(2) Input/Output Port Control Register
(3) Interrupt Register
(4) Clock Generator Register
(5) DMA Controller Register
(6) 8-Bit Timer Register
(7) 16-Bit Timer Register
(8) UART/Serial Channel Register
(9) I^2C Bus / Serial Channel Register
(10) AD Converter Register
(11) Watchdog Timer Register
(12) Real-Time Counter Register

Table structure



Symbol meanings

R/W: Registers can be both read from and written to.
R: Registers can only be read from.
W: Registers can only be written to.
W*: Registers can be both read from and written to.
(However, the register is always read as 1.)

(1) Input/Output ports

Symbol	Name	Address	7	6	5	4	3	2	1	0
P0	PORT0	FFFF F000H	P07	P06	P05	P04	P03	P02	P01	P00
			R/W							
			Undefined							
			Input Mode							
P1	PORT1	FFFF F001H	P17	P16	P15	P14	P13	P12	P11	P10
			R/W							
			Input Mode							
P2	PORT2	FFFF F012H	P27	P26	P25	P24	P23	P22	P21	P20
			R/W							
			1	1	1	1	1	1	1	1
			Input Mode							
P3	PORT3	FFFF F018H	P37	P36	P35	P34	P33	P32	P31	P30
			R/W							
			1	1	1	1	1	1	1	1
			Input Mode						Output Mode	
P4	PORT4	FFFF F01EH				P44	P43	P42	P41	P40
						R/W				
						1	1	1	1	1
						Input Mode				
P5	PORT5	FFFF F025H	P57	P56	P55	P54	P53	P52	P51	P50
			R							
			Input Mode							
P7	PORT7	FFFF F02BH	P77	P76	P75	P74	P73	P72	P71	P70
			R/W							
			1	1	1	1	1	1	1	1
			Input Mode							
P8	PORT8	FFFF F030H	P87	P86	P85	P84	P83	P82	P81	P80
			R/W							
			1	1	1	1	1	1	1	1
			Input Mode							
P9	PORT9	FFFF F031H	P97	P96	P95	P94	P93	P92	P91	P90
			R/W							
			1	1	1	1	1	1	1	1
			Output Mode		Input Mode					
PA	PORTA	FFFF F036H	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
			R/W							
			1	1	1	1	1	1	1	1
			Input Mode							

(2) Input/Output control (1/2)

Symbol	Name	Address	7	6	5	4	3	2	1	0
P0CR	PORT0 Control	FFFF F002H	P07C	P06C	P05C	P04C	P03C	P02C	P01C	P00C
			W							
			0	0	0	0	0	0	0	0
			0: IN, 1: OUT							
P1CR	PORT1 Control	FFFF F004H	P17C	P16C	P15C	P14C	P13C	P12C	P11C	P10C
			W							
			(See P1FC column)							
P1FC	PORT1 Function	FFFF F005H	P17F	P16F	P15F	P14F	P13F	P12F	P11F	P10F
			W							
			0	0	0	0	0	0	0	0
			P1FC/P1CR = 00: IN, 01: OUT, 10: AD15~AD8, 11: A15~8							
P2CR	PORT2 Control	FFFF F014H	P27C	P26C	P25C	P24C	P23C	P22C	P21C	P20C
			W							
			0	0	0	0	0	0	0	0
			(See P2FC column)							
P2FC	PORT2 Function	FFFF F015H	P27F	P26F	P25F	P24F	P23F	P22F	P21F	P20F
			W							
			0	0	0	0	0	0	0	0
			P2FC/P2CR = 00: IN, 01: OUT, 10: A0~A7, 11: A23~A16							
P3CR	PORT3 Control	FFFF F01AH	P37C	P36C	P35C	P34C	P33C	P32C		
			W							
			0	0	0	0	0	0		
			0: IN, 1: OUT							
P3FC	PORT3 Function	FFFF F01BH		P36F	P35F	P34F		P32F	P31F	P30F
				W				W		
				0	0	0		0	0	0
				0: PORT 1: R/$\overline{W}$ output	0: PORT 1: $\overline{BUSAK}$ output	0: PORT 1: $\overline{BUSRQ}$ input		0: PORT 1: $\overline{HWR}$ output	0: PORT 1: $\overline{WR}$ output	0: PORT 1: $\overline{RD}$ output
P4CR	PORT4 Control	FFFF F020H				P44C	P43C	P42C	P41C	P40C
						W				
						0	0	0	0	0
						0: IN, 1: OUT				
P4FC	PORT4 Function	FFFF F021H				P44F	P43F	P42F	P41F	P40F
						W				
						0	0	0	0	0
						0: PORT 1: SCOUT output	0: PORT 1: $\overline{CS3}$ output	0: PORT 1: $\overline{CS2}$ output	0: PORT 1: $\overline{CS1}$ output	0: PORT 1: $\overline{CS0}$ output
P7CR	PORT7 Control	FFFF F02EH	P77C	P76C	P75C	P74C	P73C	P72C	P71C	P70C
			W							
			0	0	0	0	0	0	0	0
			0: IN 1: OUT							
P7FC	PORT7 Function	FFFF F02FH	P77F	P76F	P75F	P74F	P73F	P72F	P71F	P70F
			0	0	0	0	0	0	0	0
			0: PORT 1: INT0 exit input	0: PORT 1: TB0OUT output	0: PORT 1: TB0IN1 input	0: PORT 1: TB0IN0 input	0: PORT 1: TA3OUT output RXD4 input	0: PORT 1: TA2IN input TXD4 output	0: PORT 1: TA1OUT output RXD3 input	0: PORT 1: TA0IN input TXD3 output

Note: P77F must be set to 1 when INT0 is used as an input to cause the device to exit from STOP Mode (when SYSCR2 <DRIVE> = 0).

Input/Output control (2/2)

Symbol	Name	Address	7	6	5	4	3	2	1	0
P8CR	PORT8 Control	FFFF F032H	P87C	P86C	P85C	P84C	P83C	P82C	P81C	P80C
			W							
			0	0	0	0	0	0	0	0
			0: IN, 1: OUT							
P8FC	PORT8 Function	FFFF F033H	—	P86F	P85F	P84F	P83F	P82F	P81F	P80F
			W							
			0	0	0	0	0	0	0	0
			Must always be set to 0	0: PORT 1: TB3OUT output	0: PORT 1: TB2OUT output	0: PORT 1: TB2IN1 input	0: PORT 1: TB2IN0 input	0: PORT 1: TB1OUT output	0: PORT 1: TB1IN1 input	0: PORT 1: TB1IN0 input
P9CR	PORT9 Control	FFFF F034H	P97C	P96C	P95C	P94C	P93C	P92C	P91C	P90C
			W							
			0	0	0	0	0	0	0	0
			0: IN, 1: OUT							
P9FC	PORT9 Function	FFFF F035H			P95F		P93F	P92F		P90F
					W		W			W
					0		0	0		0
					0: PORT 1: SCLK1 output $\overline{\text{CTS1}}$ / SCLK1 input		0: PORT 1: TXD1 output	0: PORT 1: SCLK0 output $\overline{\text{CTS0}}$ / SCLK0 input		0: PORT 1: TXD0 output
PACR	PORTA Control	FFFF F038H	PA7C	PA6C	PA5C	PA4C	PA3C	PA2C	PA1C	PA0C
			W							
			0	0	0	0	0	0	0	0
			0: IN, 1: OUT							
PAFC	PORTA Function	FFFF F039H	PA7F	PA6F	PA5F	—	PA3F	PA2F	PA1F	PA0F
			W							
			0	0	0	0	0	0	0	0
			0: PORT SCL output	0: PORT 1: SDA/SO output	0: PORT SCK output	Must always be set to 0	0: PORT 1: INT4 exit input	0: PORT 1: INT3 exit input	0: PORT 1: INT2 exit input	0: PORT 1: INT1 exit input

Note: PA0F~PA3F must be set to 1 when INT1~INT4 are used as an input to cause the device to exit from STOP Mode (when SYSCR2 <DRIVE> = 0).

(3) Interrupt control (1/12)

Symbol	Name	Address	7	6	5	4	3	2	1	0
IMC0L	Interrupt Mask control Register 0L	FFFF E000H			EIM01	EIM00	DM0	IL02	IL01	IL00
					R/W					
					0	0	0	0	0	0
					00: Low Must always be set to "Low".		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM0 = 0, interrupt level for Interrupt 0 (software) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM0 = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
			15	14	13	12	11	10	9	8
					EIM11	EIM10	DM1	IL12	IL11	IL10
					R/W					
					0	0	0	0	0	0
					Sets active state for interrupt request 00: Low 01: High 10: Falling edge 11: Rising edge		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM1 = 0, interrupt level for interrupt 1 (INT0) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM1 = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
IMC0H	Interrupt Mask controlRegister 0H	FFFF E002H	23	22	21	20	19	18	17	16
					EIM21	EIM20	DM2	IL22	IL21	IL20
					R/W					
					0	0	0	0	0	0
					Sets active state for interrupt request 00: Low 01: High 10: Falling edge 11: Rising edge		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	IF DM2 = 0, interrupt level for interrupt 2 (INT1) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM2 = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Setting inhibited		
			31	30	29	28	27	26	25	24
					EIM31	EIM30	DM3	IL32	IL31	IL30
					R/W					
					0	0	0	0	0	0
					Sets active state for interrupt request 00: Low 01: High 10: Falling edge 11: Rising edge		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM3 = 0, interrupt level for interrupt 3 (INT2) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM3 = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		

Note: When using INT0~INT4 as an input to cause the device to exit from SLEEP/STOP Mode, set the active state using the IMCGxx register in the CG block and set <EIMx1:EIMx0> of the IMCxx register to 01 (High level) in the INTC block.

Interrupt control (2/12)

Symbol	Name	Address	7	6	5	4	3	2	1	0
IMC1L	Interrupt Mask controlRegister 1L	FFFF E004H			EIM41	EIM40	DM4	IL42	IL41	IL40
					R/W					
					0	0	0	0	0	0
					Sets active state for interrupt request 00: Low 01: High 10: Falling edge 11: Rising edge		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM4 = 0, interrupt level for interrupt 4 (INT3) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM4 = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
			15	14	13	12	11	10	9	8
					EIM51	EIM50	DM5	IL52	IL51	IL50
					R/W					
					0	0	0	0	0	0
					Sets active state for interrupt request 00: Low 01: High 10: Falling edge 11: Rising edge		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM5 = 0, interrupt level for interrupt 5 (INT4) is set. 000: Disables interrupt 001~111: Sets levels 1~7 If DM5 = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
IMC2H	Interrupt Mask control Register 2H	FFFF E00AH	23	22	21	20	19	18	17	16
					EIMA1	EIMA0	DMA	ILA2	ILA1	ILA0
					R/W					
					0	0	0	0	0	0
					Sets active state for interrupt request 00: Low 01: High 10: Falling edge 11: Rising edge		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DMA = 0, interrupt level for interrupt 10 (INT5) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DMA = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
			31	30	29	28	27	26	25	24
					EIMB1	EIMB0	DMB	ILB2	ILB1	ILB0
					R/W					
					0	0	0	0	0	0
					Sets active state for interrupt request 00: Low 01: High 10: Falling edge 11: Rising edge		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DMB = 0, interrupt level for interrupt 11 (INT6) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DMB = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		

Interrupt control (3/12)

Symbol	Name	Address	7	6	5	4	3	2	1	0
IMC3L	Interrupt Mask control Register 3L	FFFF E00CH			EIMC1	EIMC0	DMC	ILC2	ILC1	ILC0
					R/W					
					0	0	0	0	0	0
					Sets active state for interrupt request 00: Low 01: High 10: Falling edge 11: Rising edge		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DMC = 0, interrupt level for interrupt 12 (INT7) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DMC = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
			15	14	13	12	11	10	9	8
					EIMD1	EIMD0	DMD	ILD2	ILD1	ILD0
					R/W					
					0	0	0	0	0	0
					Sets active state for interrupt request 00: Low 01: High 10: Falling edge 11: Rising edge		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DMD = 0, interrupt level for interrupt 13 (INT8) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DMD = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
IMC3H	Interrupt Mask control Register 3H	FFFF E00EH	23	22	21	20	19	18	17	16
					EIME1	EIME0	DME	ILE2	ILE1	ILE0
					R/W					
					1	0	0	0	0	0
					Sets active state for interrupt request 00: Low 01: High 10: Falling edge 11: Rising edge		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DME = 0, interrupt level for interrupt 14 (INT9) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DME = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
			31	30	29	28	27	26	25	24
					EIMF1	EIMF0	DMF	ILF2	ILF1	ILF0
					R/W					
					0	0	0	0	0	0
					Sets active state for interrupt request 00: Low 01: High 10: Falling edge 11: Rising edge		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DMF = 0, interrupt level for interrupt 15 (INTA) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DMF = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		

Interrupt control (4/12)

Symbol	Name	Address	7	6	5	4	3	2	1	0
IMC5L	Interrupt Mask control Register 5L	FFFF E014H			EIM141	EIM140	DM14	IL142	IL141	IL140
					R/W					
					0	0	0	0	0	0
					Must always be set to 11.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM14 = 0, interrupt level for interrupt 20 (INTTA0) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM14 = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
			15	14	13	12	11	10	9	8
					EIM151	EIM150	DM15	IL152	IL151	IL150
					R/W					
					0	0	0	0	0	0
					Must always be set to 11.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM15 = 0, interrupt level for interrupt 21 (INTTA1) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM15 = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
IMC5H	Interrupt Mask control Register 5H	FFFF E016H	23	22	21	20	19	18	17	16
					EIM161	EIM160	DM16	IL162	IL161	IL160
					R/W					
					0	0	0	0	0	0
					Must always be set to 11.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM16 = 0, interrupt level for interrupt 22 (INTTA2) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM16 = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
			31	30	29	28	27	26	25	24
					EIM171	EIM170	DM17	IL172	IL171	IL170
					R/W					
					0	0	0	0	0	0
					Must always be set to 11.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM17 = 0, interrupt level for interrupt 23 (INTTA3) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM17 = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		

Interrupt control (5/12)

Symbol	Name	Address	7	6	5	4	3	2	1	0
IMC7L	Interrupt Mask control Register 7L	FFFF E01CH			EIM1C1	EIM1C0	DM1C	IL1C2	IL1C1	IL1C0
					R/W					
					0	0	0	0	0	0
					Must always be set to 11.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM1C = 0, interrupt level for interrupt 28 (INTTB00) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM1C = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
			15	14	13	12	11	10	9	8
					EIM1D1	EIM1D0	DM1D	IL1D2	IL1D1	IL1D0
					R/W					
					0	0	0	0	0	0
					Must always be set to 11.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM1D = 0, interrupt level for interrupt 29 (INTTB01) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM1D = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
IMC7H	Interrupt Mask control Register 7H	FFFF E01EH	23	22	21	20	19	18	17	16
					EIM1E1	EIM1E0	DM1E	IL1E2	IL1E1	IL1E0
					R/W					
					0	0	0	0	0	0
					Must always be set to 11.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM1E = 0, interrupt level for interrupt 30 (INTTB10) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM1E = 1, DMAC channel is selected 000~011: Selects channels 0~3 100~111: Invalid setting		
			31	30	29	28	27	26	25	24
					EIM1F1	EIM1F0	DM1F	IL1F2	IL1F1	IL1F0
					R/W					
					0	0	0	0	0	0
					Must always be set to 11.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM1F = 0, interrupt level for interrupt 31 (INTTB11) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM1F = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		

Interrupt control (6/12)

Symbol	Name	Address	7	6	5	4	3	2	1	0
IMC8L	Interrupt Mask control Register 8L	FFFF E020H			EIM201	EIM200	DM20	IL202	IL201	IL200
					R/W					
					0	0	0	0	0	0
					Must always be set to 11.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM20 = 0, interrupt level for interrupt 32 (INTTB20) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM20 = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
			15	14	13	12	11	10	9	8
					EIM211	EIM210	DM21	IL212	IL211	IL210
					R/W					
					0	0	0	0	0	0
					Must always be set to 11.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM21 = 0, interrupt level for interrupt 33 (INTTB21) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM21 = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
IMC8H	Interrupt Mask control Register 8H	FFFF E022H	23	22	21	20	19	18	17	16
					EIM221	EIM220	DM22	IL222	IL221	IL220
					R/W					
					0	0	0	0	0	0
					Must always be set to 11.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM22 = 0, interrupt level for interrupt 34 (INTTB30) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM22 = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
			31	30	29	28	27	26	25	24
					EIM231	EIM230	DM23	IL232	IL231	IL230
					R/W					
					0	0	0	0	0	0
					Must always be set to 11.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM23 = 0, interrupt level for interrupt 35 (INTTB31) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM23 = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		

Interrupt control (7/12)

Symbol	Name	Address	7	6	5	4	3	2	1	0
IMCAL	Interrupt Mask control Register AL	FFFF E028H			EIM281	EIM280	DM28	IL282	IL281	IL280
					R/W					
					0	0	0	0	0	0
					Must always be set to 11.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM28 = 0, interrupt level for interrupt 40 (INTTBOF0) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM28 = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
			15	14	13	12	11	10	9	8
					EIM291	EIM290	DM29	IL292	IL291	IL290
					R/W					
					0	0	0	0	0	0
					Must always be set to 11.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM29 = 0, interrupt level for interrupt 41 (INTTBOF1) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM29 = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
IMCAH	Interrupt Mask control Register AH	FFFF E02AH	23	22	21	20	19	18	17	16
					EIM2A1	EIM2A0	DM2A	IL2A2	IL2A1	IL2A0
					R/W					
					0	0	0	0	0	0
					Must always be set to 11.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM2A = 0, interrupt level for interrupt 42 (INTTBOF2) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM2A = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
			31	30	29	28	27	26	25	24
					EIM2B1	EIM2B0	DM2B	IL2B2	IL2B1	IL2B0
					R/W					
					0	0	0	0	0	0
					Must always be set to 11.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM2B = 0, interrupt level for interrupt 43 (INTTBOF3) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM2B = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		

Interrupt control (8/12)

Symbol	Name	Address	7	6	5	4	3	2	1	0
IMCCL	Interrupt Mask control Register CL	FFFF E030H			EIM301	EIM300	DM30	IL302	IL301	IL300
					R/W					
					0	0	0	0	0	0
					Must always be set to 11.		Sets DMAC trigger 0: Not set 1: Uses interrupt to be DMAC trigger	If DM30 = 0, interrupt level for interrupt 48 (INTRX0) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM30 = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
			15	14	13	12	11	10	9	8
					EIM311	EIM310	DM31	IL312	IL311	IL310
					R/W					
					0	0	0	0	0	0
					Must always be set to 11.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM31 = 0, interrupt level for interrupt 49 (INTTX0) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM31 = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
IMCCH	Interrupt Mask control Register CH	FFFF E032H	23	22	21	20	19	18	17	16
					EIM321	EIM320	DM32	IL322	IL321	IL320
					R/W					
					0	0	0	0	0	0
					Must always be set to 11		Sets DMAC trigge 0: Not set 1: Sets interrupt to be DMAC trigger	If DM32 = 0, interrupt level for interrupt 50 (INTRX1) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM32 = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
			31	30	29	28	27	26	25	24
					EIM331	EIM330	DM33	IL332	IL331	IL330
					R/W					
					0	0	0	0	0	0
					Must always be set to 11.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM33 = 0, interrupt level for interrupt 51 (INTTX1) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM33 = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		

Interrupt control (9/12)

Symbol	Name	Address	7	6	5	4	3	2	1	0
IMCDL	Interrupt Mask control Register DL	FFFF E034H			EIM341	EIM340	DM34	IL342	IL341	IL340
					R/W					
					0	0	0	0	0	0
					Must always be set to 11.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM34 = 0, interrupt level for interrupt 52 (INTS2) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM34 = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
IMCDH	Interrupt Mask control Register DH	FFFF E036H	23	22	21	20	19	18	17	16
					EIM361	EIM360	DM36	IL362	IL361	IL360
					R/W					
					0	0	0	0	0	0
					Must always be set to 11.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM36 = 0, interrupt level for interrupt 54 (INTRX3) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM36 = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
			31	30	29	28	27	26	25	24
					EIM371	EIM370	DM37	IL372	IL371	IL370
					R/W					
					0	0	0	0	0	0
					Must always be set to 11.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM37 = 0, interrupt level for interrupt 55 (INTTX3) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM37 = 1, selects DMAC channel 000~011: Selects Channels 0~3 100~111: Invalid setting		

Interrupt control (10/12)

Symbol	Name	Address	7	6	5	4	3	2	1	0
IMCEL	Interrupt Mask control Register EL	FFFF E038H			EIM381	EIM380	DM38	IL382	IL381	IL380
					R/W					
					0	0	0	0	0	0
					Must always be set to 11.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM38 = 0, interrupt level for interrupt 56 (INTRX4) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM38 = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
			15	14	13	12	11	10	9	8
					EIM391	EIM390	DM39	IL392	IL391	IL390
					R/W					
					0	0	0	0	0	0
					Must always be set to 11.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM39 = 0, interrupt level for interrupt 57 (INTTX4) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM39 = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
IMCEH	Interrupt Mask control Register EH	FFFF E03AH	23	22	21	20	19	18	17	16
					EIM3A1	EIM3A0	DM3A	IL3A2	IL3A1	IL3A0
					R/W					
					0	0	0	0	0	0
					Must always be set to 01.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM3A = 0, interrupt level for interrupt 58 (INTRTC) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM3A = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
			31	30	29	28	27	26	25	24
					EIM3B1	EIM3B0	DM3B	IL3B2	IL3B1	IL3B0
					R/W					
					0	0	0	0	0	0
					Must always be set to 11.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM3B = 0, interrupt level for interrupt 59 (INTAD) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM3B = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		

Interrupt control (11/12)

Symbol	Name	Address	7	6	5	4	3	2	1	0
IMCFL	Interrupt Mask control Register FL	FFFF E03CH			EIM3C1	EIM3C0	DM3C	IL3C2	IL3C1	IL3C0
					R/W					
					0	0	0	0	0	0
					Must always be set to 10.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM3C = 0, interrupt level for interrupt 60 (INTDMA0) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM3C = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
			15	14	13	12	11	10	9	8
					EIM3D1	EIM3D0	DM3D	IL3D2	IL3D1	IL3D0
					R/W					
					0	0	0	0	0	0
					Must always be set to 10.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM3D = 0, interrupt level for interrupt 61 (INTDMA1) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM3D = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
IMCFH	Interrupt Mask control Register FH	FFFF E03EH	23	22	21	20	19	18	17	16
					EIM3E1	EIM3E0	DM3E	IL3E2	IL3E1	IL3E0
					R/W					
					0	0	0	0	0	0
					Must always be set to 10.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM3E = 0, interrupt level for interrupt 62 (INTDMA2) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM3E = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		
			31	30	29	28	27	26	25	24
					EIM3F1	EIM3F0	DM3F	IL3F2	IL3F1	IL3F0
					R/W					
					0	0	0	0	0	0
					Must always be set to 10.		Sets DMAC trigger 0: Not set 1: Sets interrupt to be DMAC trigger	If DM3F = 0, interrupt level for interrupt 63 (INTDMA3) is set 000: Disables interrupt 001~111: Sets levels 1~7 If DM3F = 1, DMAC channel is selected 000~011: Selects Channels 0~3 100~111: Invalid setting		

Interrupt Control (12/12)

Symbol	Name	Address	7	6	5	4	3	2	1	0
IVR	Interrupt Vector address register	FFFF E040H	IVRL							
			R							
			0	0	0	0	0	0	0	0
			Bits 9~4 hold the vector for the interrupt generated.							
			15	14	13	12	11	10	9	8
			IVRH						IVRL	
			R/W						R	
			0	0	0	0	0	0	0	0
									Bits 9~4 hold the vector for the interrupt generated.	
			23	22	21	20	19	18	17	16
			IVRH							
			R/W							
			0	0	0	0	0	0	0	0
			31	30	29	28	27	26	25	24
			IVRH							
			R/W							
			0	0	0	0	0	0	0	0
INTCLR	Interrupt request clear Register	FFFF E060H	7	6	5	4	3	2	1	0
					EICLR5	EICLR4	EICLR3	EICLR2	EICLR1	EICLR0
				W						
				Sets IVR<9:4> value that corresponds to the interrupt whose request is to be cleared.						

(4) Chip Select/Wait controller (1/4)

Symbol	Name	Address	7	6	5	4	3	2	1	0
BMA0	Base/Mask Address Register	FFFF E400H	MA0							
			R/W							
			1	1	1	1	1	1	1	1
			Bits 9~0 specify the address bits (A23~A14) which are to be masked. 0: Not masked 1: Masked							
			15	14	13	12	11	10	9	8
			MA0							
			R/W							
			0	0	0	0	0	0	0	0
			Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.	Mask address 0: Not masked 1: Masked	
			23	22	21	20	19	18	17	16
			BA0							
			R/W							
			0	0	0	0	0	0	0	0
			Specifies base address of CS0 space (bits 31~16 correspond to A31~A16 respectively.)							
			31	30	29	28	27	26	25	24
			BA0							
			R/W							
			0	0	0	0	0	0	0	0
			Specifies base address of CS0 space (bits 31~16 correspond to A31~A16 respectively.)							
BMA1	Base/Mask Address Register	FFFF E404H	7	6	5	4	3	2	1	0
			MA1							
			R/W							
			1	1	1	1	1	1	1	1
			Bits 9~0 specify the address bits (A23~A14) which are to be masked. 0: Not masked 1: Masked							
			15	14	13	12	11	10	9	8
			MA1							
			R/W							
			0	0	0	0	0	0	0	0
			Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.	Mask address 0: Not masked 1: Masked	
			23	22	21	20	19	18	17	16
			BA1							
			R/W							
			0	0	0	0	0	0	0	0
			Specifies base address of CS1 space (bits 31~16 correspond to A31~A16 respectively.)							
			31	30	29	28	27	26	25	24
			BA1							
			R/W							
			0	0	0	0	0	0	0	0
			Specifies base address of CS1 space (bits 31~16 correspond to A31~A16 respectively.)							

Chip Select/Wait controller (2/4)

Symbol	Name	Address	7	6	5	4	3	2	1	0
BMA2	Base/Mask Address Register	FFFF E408H	MA2							
			R/W							
			1	1	1	1	1	1	1	1
			Bits 8~0 specify the address bits (A23~A15) which are to be masked 0: Not masked 1: Masked							
			15	14	13	12	11	10	9	8
			MA2							
			R/W							
			0	0	0	0	0	0	0	0
			Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.	Mask address 0: Not masked 1: Masked
			23	22	21	20	19	18	17	16
			BA2							
			R/W							
			0	0	0	0	0	0	0	0
			Specifies base address of CS0 space (bits 31~16 correspond to A31~A16 respectively.)							
			31	30	29	28	27	26	25	24
			BA2							
			R/W							
			0	0	0	0	0	0	0	0
			Specifies base address of CS0 space (bits 31~16 correspond to A31~A16 respectively.)							
BMA3	Base/Mask Address Register	FFFF E40CH	7	6	5	4	3	2	1	0
			MA3							
			R/W							
			1	1	1	1	1	1	1	1
			Bits 8~0 specify the address bits (A23~A15) which are to be masked 0: Not masked 1: Masked							
			15	14	13	12	11	10	9	8
			MA3							
			R/W							
			0	0	0	0	0	0	0	0
			Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.	Mask address 0: Not masked 1: Masked
			23	22	21	20	19	18	17	16
			BA3							
			R/W							
			0	0	0	0	0	0	0	0
			Specifies base address of CS1 space (bits 31~16 correspond to A31~A16 respectively.)							
			31	30	29	28	27	26	25	24
			BA3							
			R/W							
			0	0	0	0	0	0	0	0
			Specifies base address of CS1 space (bits 31~16 correspond to A31~A16 respectively.)							

Chip Select/Wait controller (3/4)

Symbol	Name	Address	7	6	5	4	3	2	1	0
B01CS	Chipselect /wait control Register	FFFF E480H	B0OM			B0BUS	B0W			
			W			W				
			0	0		0	0	1	0	1
			Specifies chip select output waveform 00: Used for ROM/SRAM Other: Invalid setting			Specifies data bus width 0: 16 bit 1: 8 bit	Specifies number of wait states 0000: 0 wait states; 0001: 1 wait state 0010: 2 wait states; 0011: 3 wait states 0100: 4 wait states; 0101: 5 wait states 0110: 6 wait states; 0111: 7 wait states 1111: (1+N) wait states; Other: Invalid setting			
			15	14	13	12	11	10	9	8
							B0E		B0RCV	
							W		W	
							0		0	0
							CS enable 0: Disabled 1: Enabled		Specifies the number of dummy cycles to be inserted 00: 2 cycles 01: 1 cycle 10: None 11: Invalid setting	
			23	22	21	20	19	18	17	16
			B1OM			B1BUS	B1W			
			W			W				
			0	0		0	0	1	0	1
			Specifies chip select output waveform 00: Used for ROM/SRAM Other: Invalid setting			Specifies data bus width 0: 16 bit 1: 8 bit	Specifies number of wait states 0000: 0 wait states; 0001: 1 wait state 0010: 2 wait states; 0011: 3 wait states 0100: 4 wait states; 0101: 5 wait states 0110: 6 wait states; 0111: 7 wait states 1111: (1 + N) wait states; Other: Invalid setting			
			31	30	29	28	27	26	25	24
							B1E		B1RCV	
							W		W	
							0		0	0
							CS enable 0: Disabled 1: Enabled		Specifies the number of dummy cycles to be inserted 00: 2 cycles 01: 1 cycle 10: None 11: Invalid setting	

Chip Select/Wait controller (4/4)

Symbol	Name	Address	7	6	5	4	3	2	1	0
B23CS	Chipselect /wait control Register	FFFF E484H	B2OM			B2BUS	B2W			
			W			W				
			0	0		0	0	1	0	1
			Specifies chip select output waveform 00: Used for ROM/SRAM Other: Invalid setting			Specifies data bus width 0: 16 bit 1: 8 bit	Specifies number of wait states 0000: 0 wait states; 0001: 1 wait state 0010: 2 wait states; 0011: 3 wait states 0100: 4 wait states; 0101: 5 wait states 0110: 6 wait states; 0111: 7 wait states 1111: (1 + N) wait states; Other: Invalid setting			
			15	14	13	12	11	10	9	8
							B2E	B2M	B2RCV	
							W	W	W	
							1	0	0	0
							CS enable 0: Disabled 1: Enabled	Specifies CS2 space 0: 4-Gbyte space 1: CS space	Specifies the number of dummy cycles to be inserted 00: 2 cycles 01: 1 cycle 10: None 11: Invalid setting	
			23	22	21	20	19	18	17	16
			B3OM			B3BUS	B3W			
			W			W				
			0	0		0	0	1	0	1
			Specifies chip select output waveform 00: Used for ROM/SRAM Other: Invalid setting			Specifies data bus width 0: 16 bit 1: 8 bit	Specifies number of wait states 0000: 0 wait states; 0001: 1 wait state 0010: 2 wait states; 0011: 3 wait states 0100: 4 wait states; 0101: 5 wait states 0110: 6 wait states; 0111: 7 wait states 1111: (1 + N) wait states; Other: Invalid setting			
			31	30	29	28	27	26	25	24
							B3E		B3RCV	
							W		W	
							0		0	0
							CS enable 0: Disabled 1: Enabled		Specifies the number of dummy cycles to be inserted 00: 2 cycles 01: 1 cycle 10: None 11: Invalid setting	
BEXCS	Chipselect /wait control Register	FFFF E488H	7	6	5	4	3	2	1	0
			BEXOM			BEXBUS	BEXW			
			W			W				
			0	0		0	0	1	0	1
			Specifies chip select output waveform 00: Used for ROM/SRAM Other: Invalid setting			Specifies data bus width 0: 16 bit 1: 8 bit	Specifies number of wait states 0000: 0 wait states; 0001: 1 wait state 0010: 2 wait states; 0011: 2 wait states 0100: 4 wait states; 0101: 5 wait states 0110: 6 wait states; 0111: 7 wait states 1111: (1 + N) wait states; Other: Invalid setting			
			15	14	13	12	11	10	9	8
									BEXRCV	
									W	
									0	0
									Specifies the number of dummy cycles to be inserted 00: 2 cycles 01: 1 cycle 10: None 11: Invalid setting	

(5) Clock control (1/2)

Symbol	Name	Address	7	6	5	4	3	2	1	0
SYSCR0	System Clock Control Register 0	FFFF EE00H	XEN	XTEN	RXEN	RXTEN	RSYSCK	WUEF	PRCK1	PRCK0
			R/W							
			1	0	1	0	0	0	0	0
			High-speed oscillator 0: Stopped 1: Oscillates	Low-speed oscillator 0: Stopped 1: Oscillates	High-speed oscillator after device has exited STOP Mode 0: Stopped 1: Oscillates	Low-speed oscillator after device has exitied STOP Mode 0: Stopped 1: Oscillates	Clock to be used after device has exited STOP Mode 0: High-speed 1: Low-speed	Warm-up (WUP) Writing 0: Don't care Writing 1: Starts timer Read as 0: WUP finished Read as 1: WUP in progress	Specifies prescaler clock frequency 00: fperiph/4 01: fperiph/2 10: fperiph 11: Reserved Note: Do not set a clock frequency greater than 12.5 MHz.	
SYSCR1	System Clock Control Register 1	FFFF EE01H			SYSCK	FPSEL	DFOSC		GEAR1	GEAR0
					R/W				R/W	
					0	0	0		1	1
					Specifies system clock frequency 0: High-speed (fc) 1: Low-speed (fs)	Specifies fperiph clock frequency 0: fgear 1: fc	Specifies high-speed oscillator divisor 0: 2 1: 1		Specifies high-speed clock gear 00: fc 01: fc /2 10: fc /4 11: fc /8	
SYSCR2	System Clock Control Register 2	FFFF EE02H	DRVOSCH	DRVOSCL	WUPT1	WUPT0	STBY1	STBY0		DRVE
			R/W	R/W	R/W	R/W	R/W	R/W		R/W
			0	0	1	0	1	1		0
			High-speed oscillator driving capability control 0: Normal 1: Weak	Low-speed oscillator driving capability control 0: Normal 1: Weak	Specifies oscillator WUP time 00: Reserved 01: 2^8/ input frequency 10: 2^{14} 11: 2^{16}		00: Reserved 01: STOP Mode 10: SLEEP Mode 11: IDLE Mode			1: Pins also driven in STOP Mode
SYSCR3	System Clock Control Register 3	FFFF EE03H		SCOSEL		ALESEL			LUPFG	LUPTM
				R/W		R/W			R/W	
				0		1			0	0
				SCOUT output 0: fs 1: fsys		Specifies ALE output pulse width 0: fsys x 0.5 1: fsys x 1.5			Lock-up (LUP) flag 0: LUP finished 1: LUP in progress	Specifies lock-up time 0: 2^{16}/ input frequency 1: 2^{12}/ input frequency
ADCCLK	ADC conversion clock register	FFFF EE04H							ADCCK1	ADCCK0
									R/W	R/W
									0	0
									Specifies ADC conversion clock frequency 00: fsys/2 01: fsys/4 10: fsys/8 11 Reserved	
IMCGA0	Interrupt mask conntrol Register A0	FFFF EE10H			EMCG01	EMCG00				INT0EN
					R/W					R/W
					1	0				0
					Specifies active state for INT0 standby exit request 00: Low 01: High 10: Falling edge 11: Rising edge					INT0 exit input 0: Disabled 1: Enabled
IMCGA1	Interrupt mask conntrol Register A1	FFFF EE11H			EMCG11	EMCG10				INT1EN
					R/W					R/W
					1	0				0
					Specifies active state for INT1 standby exit request 00: Low 01: High 10: Falling edge 11: Rising edge					INT1 exit input 0: Disabled 1: Enabled

Clock control (2/2)

Symbol	Name	Address	7	6	5	4	3	2	1	0
IMCGA2	Interrupt mask conntrol Register A2	FFFF EE12H			EMCG201	EMCG20				INT2EN
					R/W					R/W
					1	0				0
					Specifies active state for INT2 standby exit request 00: Low 01: High 10: Falling edge 11: Rising edge					INT2 exit input 0: Disabled 1: Enabled
IMCGB1	Interrupt mask conntrol Register B1	FFFF EE15								
					1	0				0
					These bits should always be set to 10.					These bits should always be set to 0.
IMCGB2	Interrupt mask conntrol Register B2	FFFF EE16								
					1	0				0
					These bits should always be set to 10.					These bits should always be set to 0.
IMCGA3	Interrupt mask conntrol Register A3	FFFF EE13H			EMCG31	EMCG30				INT3EN
					R/W					R/W
					1	0				0
					Specifies active state for INT3 standby exit request 00: Low 01: High 10: Falling edge 11: Rising edge					INT3 exit input 0: Disabled 1: Enabled
IMCGB0	Interrupt mask conntrol Register B0	FFFF EE14H			EMCG41	EMCG40				INT4EN
					R/W					R/W
					1	0				0
					Specifies active state for INT4 standby exit request 00: Low 01: High 10: Falling edge 11: Rising edge					INT4 exit input 0: Disabled 1: Enabled
IMCGB1	Interrupt mask conntrol Register B1	FFFF EE15								
					1	0				0
					Must always be set to 10.					Must always be set to 0.
IMCGB2	Interrupt mask conntrol Register B2	FFFF EE16								
					1	0				0
					Must always be set to 11.					Must always be set to 0.
IMCGB3	Interrupt mask control Register B3	FFFF EE17H			EMCG71	EMCG72				INTRTCEN
					R/W					R/W
					1	0				0
					Must always be set to 11.					INTRTC exit input 0: Disabled 1: Enabled
EICRCG	Interrupt request clear Register	FFFF EE20H						ICRCG2	ICRCG1	ICRCG0
								W		
								0	0	0
								Clears interrupt request (Effective only when exit input is enabled) 000: INT0 001: INT1 010: INT2 011: INT3 100: INT4 101: reserved 110: reserved 111: INTRTC		

(6) DMA control (1/13)

Symbol	Name	Address	7	6	5	4	3	2	1	0
CCR0	DMA channel Control Register 0	FFFF E200H	SAC0	DIO	DAC1	DAC0	TrSiz1	TrSiz0	DPS1	DPS0
			R/W							
			0	0	0	0	0	0	0	0
			Bits 8 and 7 specify how source address will be changed 00: Incre-mented 01: Decre-mented 1x: Fixed	Specifies destination device 0: Memory 1: I/O	Specifies how destination address will be changed 00: Incremented 01: Decremented 1x: Fixed		Specifies amount of data to be transferred in a single operation 0x: 32 bits 10: 16 bits 11: 8 bits		Specifies I/O device bus width 0x: 32 bits 10: 16 bits 11: 8 bits	
			15	14	13	12	11	10	9	8
			—	ExR	PosE	Lev	SReq	RelEn	SIO	SAC1
			R/W							
			0	0	0	0	0	0	0	0
			Write 0 to this bit.	Specifies transfer request mode 1: Initiated by interrupt request 0: Internal request	Write 0 to this bit.	Write 1 to this bit.	Enables Snoop function 0: Disabled 1: Enabled	Enables request for surrender of bus control 0: Disabled 1: Enabled	Specifies source device 0: Memory 1: I/O	Bits 8 and 7 specify how source address will be changed 00: Incre-mented 01: Decre-mented 1x: Fixed
			23	22	21	20	19	18	17	16
			NIEn	AbIEn	—	—	—	—	Big	—
			R/W							
			1	1	1	0	0	0	1	0
			Enables interrupt on normal termination 0: Disabled 1: Enabled	Enables interrupt on abnormal termination 0: Disabled 1: Enabled	Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.
			31	30	29	28	27	26	25	24
			Str							—
			W							W
			0	0	0	0	0	0	0	0
			1: Starts DMA Channel 0							Write 0 to this bit.
CSR0	DMA Channel status Register 0	FFFF E204H	7	6	5	4	3	2	1	0
								—	—	—
								R/W		
			0	0	0	0	0	0	0	0
								Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.
			15	14	13	12	11	10	9	8
			0	0	0	0	0	0	0	0
			23	22	21	20	19	18	17	16
			NC	AbC	—	BES	BED	Conf		
				R/W			R			
			0	0	0	0	0	0	0	0
			1: Normal Termi-nation Status flag	1: Abnormal Termi-nation Status flag	Write 0 to this bit.	1: Bus error in source address	1: Bus error in desti-nation address	1: Configu-ration error		
			31	30	29	28	27	26	25	24
			Act							
			R							
			0	0	0	0	0	0	0	0
			1: DMA Channel 0 in standby state							

DMA control (2/13)

Symbol	Name	Address	7	6	5	4	3	2	1	0
SAR0	DMA Source address Register 0	FFFF E208H	SAddr7	SAddr6	SAddr5	SAddr4	SAddr3	SAddr2	SAddr1	SAddr0
			R/W							
			Undefined							
			15	14	13	12	11	10	9	8
			SAddr15	SAddr14	SAddr13	SAddr12	SAddr11	SAddr10	SAddr9	SAddr8
			R/W							
			Undefined							
			23	22	21	20	19	18	17	16
			SAddr23	SAddr22	SAddr21	SAddr20	SAddr19	SAddr18	SAddr17	SAddr16
			R/W							
			Undefined							
			31	30	29	28	27	26	25	24
			SAddr31	SAddr30	SAddr29	SAddr28	SAddr27	SAddr26	SAddr25	SAddr24
			R/W							
			Undefined							
DAR0	DMA Destination address Register 0	FFFF E20CH	7	6	5	4	3	2	1	0
			DAddr7	DAddr6	DAddr5	DAddr4	DAddr3	DAddr2	DAddr1	DAddr0
			R/W							
			Undefined							
			15	14	13	12	11	10	9	8
			DAddr15	DAddr14	DAddr13	DAddr12	DAddr11	DAddr10	DAddr9	DAddr8
			R/W							
			Undefined							
			23	22	21	20	19	18	17	16
			DAddr23	DAddr22	DAddr21	DAddr20	DAddr19	DAddr18	DAddr17	DAddr16
			R/W							
			Undefined							
			31	30	29	28	27	26	25	24
			DAddr31	DAddr30	DAddr29	DAddr28	DAddr27	DAddr26	DAddr25	DAddr24
			R/W							
			Undefined							
BCR0	DMA Byte Count Register 0	FFFF E210H	7	6	5	4	3	2	1	0
			BC7	BC6	BC5	BC4	BC3	BC2	BC1	BC0
			R/W							
			Undefined							
			15	14	13	12	11	10	9	8
			BC15	BC14	BC13	BC12	BC11	BC10	BC9	BC8
			R/W							
			Undefined							
			23	22	21	20	19	18	17	16
			BC23	BC22	BC21	BC20	BC19	BC18	BC17	BC16
			R/W							
			Undefined							
			31	30	29	28	27	26	25	24
			0	0	0	0	0	0	0	0

DMA control (3/13)

Symbol	Name	Address	7	6	5	4	3	2	1	0
DTCR0	DMA Transfer control Register 0	FFFF E218H			DACM2	DACM1	DACM0	SACM2	SACM1	SACM0
					R/W					
			0	0	0	0	0	0	0	0
					Bit position at which destination address starts 000: Bit 0 001: Counts beginning at bit 4 of the address counter 010: Counts beginning at bit 8 of the address counter 011: Counts beginning at bit 12 of the address counter 100: Counts beginning at bit 16 of the address counter 101: Reserved 110: Reserved 111: Reserved			Bit position at which source address starts 000: Counts beginning at bit 0 of the address counter 001: Counts beginning at bit 4 of the address counter 010: Counts beginning at bit 8 of the address counter 011: Counts beginning at bit 12 of the address counter 100: Counts beginning at bit 16 of the address counter 101: Reserved 110: Reserved 111: Reserved		
			15	14	13	12	11	10	9	8
			0	0	0	0	0	0	0	0
			23	22	21	20	19	18	17	16
			0	0	0	0	0	0	0	0
			31	30	29	28	27	26	25	24
			0	0	0	0	0	0	0	0

DMA control (4/13)

Symbol	Name	Address	7	6	5	4	3	2	1	0
CCR1	DMA channel Control Register 1	FFFF E220H	SAC0	DIO	DAC1	DAC0	TrSiz1	TrSiz0	DPS1	DPS0
			R/W							
			0	0	0	0	0	0	0	0
			Bits 8 and 7 specify how source address will be changed 00: Incre-mented 01: Decre-mented 1x: Fixed	Specifies destination device 0: Memory 1: I/O	Specifies how destination address will be changed 00: Incremented 01: Decremented 1x: Fixed		Specifies amount of data to be transferred in a single operation 0x: 32 bits 10: 16 bits 11: 8 bits		Specifies I/O device bus width 0x: 32 bits 10: 16 bits 11: 8 bits	
			15	14	13	12	11	10	9	8
			—	ExR	PosE	Lev	SReq	RelEn	SIO	SAC1
			R/W							
			0	0	0	0	0	0	0	0
			Write 0 to this bit.	Specifies transfer request method 1: Initiated by interrupt request 0: Internal request	Write 0 to this bit.	Write 1 to this bit.	Selects Snoop function 0: Disabled 1: Enabled	Enables request for surrender of bus control 0: Disabled 1: Enabled	Specifies source device 0: Memory 1: I/O	Bits 8 and 7 specify how source address will be changed 00: Incre-mented 01: Decre-mented 1x: Fixed
			23	22	21	20	19	18	17	16
			NIEn	AbIEn	—	—	—	—	Big	—
			R/W							
			1	1	1	0	0	0	1	0
			Enables interrupt on normal termination 0: Disabled 1: Enabled	Enables interrupt on abnormal termination 0: Disabled 1: Enabled	Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.
			31	30	29	28	27	26	25	24
			Str							—
			W							W
			0	0	0	0	0	0	0	0
			1: Starts DMA Channel 1							Write 0 to this bit.
CSR1	DMA Channel status Register 1	FFFF E224H	7	6	5	4	3	2	1	0
								—	—	—
								R/W		
			0	0	0	0	0	0	0	0
								Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.
			15	14	13	12	11	10	9	8
			0	0	0	0	0	0	0	0
			23	22	21	20	19	18	17	16
			NC	AbC	—	BES	BED	Conf		
			R/W			R				
			0	0	0	0	0	0	0	0
			1: Normal Termination Status flag	1: Abnormal Termination Status flag	Write 0 to this bit.	1: Bus error in source address	1: Bus error in desti-nation address	1: Configu-ration error		
			31	30	29	28	27	26	25	24
			Act							
			R							
			0	0	0	0	0	0	0	0
			1: DMA Channel 1 in standby state							

DMA control (5/13)

Symbol	Name	Address	7	6	5	4	3	2	1	0
SAR1	DMA Source address Register 1	FFFF E228H	SAddr7	SAddr6	SAddr5	SAddr4	SAddr3	SAddr2	SAddr1	SAddr0
			R/W							
			Undefined							
			15	14	13	12	11	10	9	8
			SAddr15	SAddr14	SAddr13	SAddr12	SAddr11	SAddr10	SAddr9	SAddr8
			R/W							
			Undefined							
			23	22	21	20	19	18	17	16
			SAddr23	SAddr22	SAddr21	SAddr20	SAddr19	SAddr18	SAddr17	SAddr16
			R/W							
			Undefined							
			31	30	29	28	27	26	25	24
			SAddr31	SAddr30	SAddr29	SAddr28	SAddr27	SAddr26	SAddr25	SAddr24
			R/W							
			Undefined							
DAR1	DMA Destination address Register 1	FFFF E22CH	7	6	5	4	3	2	1	0
			DAddr7	DAddr6	DAddr5	DAddr4	DAddr3	DAddr2	DAddr1	DAddr0
			R/W							
			Undefined							
			15	14	13	12	11	10	9	8
			DAddr15	DAddr14	DAddr13	DAddr12	DAddr11	DAddr10	DAddr9	DAddr8
			R/W							
			Undefined							
			23	22	21	20	19	18	17	16
			DAddr23	DAddr22	DAddr21	DAddr20	DAddr19	DAddr18	DAddr17	DAddr16
			R/W							
			Undefined							
			31	30	29	28	27	26	25	24
			DAddr31	DAddr30	DAddr29	DAddr28	DAddr27	DAddr26	DAddr25	DAddr24
			R/W							
			Undefined							
BCR1	DMA Byte Count Register 1	FFFF E230H	7	6	5	4	3	2	1	0
			BC7	BC6	BC5	BC4	BC3	BC2	BC1	BC0
			R/W							
			Undefined							
			15	14	13	12	11	10	9	8
			BC15	BC14	BC13	BC12	BC11	BC10	BC9	BC8
			R/W							
			Undefined							
			23	22	21	20	19	18	17	16
			BC23	BC22	BC21	BC20	BC19	BC18	BC17	BC16
			R/W							
			Undefined							
			31	30	29	28	27	26	25	24
			0	0	0	0	0	0	0	0

DMA control (6/13)

Symbol	Name	Address	7	6	5	4	3	2	1	0
DTCR1	DMA Transfer control Register 1	FFFF E238H			DACM2	DACM1	DACM0	SACM2	SACM1	SACM0
					R/W					
			0	0	0	0	0	0	0	0
					Bit position at which destination address starts 000: Counts beginning at bit 0 of the address counter 001: Counts beginning at bit 4 of the address counter 010: Counts beginning at bit 8 of the address counter 011: Counts beginning at bit 12 of the address counter 100: Counts beginning at bit 16 of the address counter 101: Reserved 110: Reserved 111: Reserved			Bit position at which source address starts 000: Counts beginning at bit 0 of the address counter 001: Counts beginning at bit 4 of the address counter 010: Counts beginning at bit 8 of the address counter 011: Counts beginning at bit 12 of the address counter 100: Counts beginning at bit 16 of the address counter 101: Reserved 110: Reserved 111: Reserved		
			15	14	13	12	11	10	9	8
			0	0	0	0	0	0	0	0
			23	22	21	20	19	18	17	16
			0	0	0	0	0	0	0	0
			31	30	29	28	27	26	25	24
			0	0	0	0	0	0	0	0

DMA control (7/13)

Symbol	Name	Address	7	6	5	4	3	2	1	0
CCR2	DMA channel Control Register 2	FFFF E240H	SAC0	DIO	DAC1	DAC0	TrSiz1	TrSiz0	DPS1	DPS0
			R/W							
			0	0	0	0	0	0	0	0
			Bits 8 and 7 specify how source address will be changed 00: Incre-mented 01: Decre-mented 1x: Fixed	Specifies destination device 0: Memory 1: I/O	Specifies how destination address will be changed 00: Incremented 01: Decremented 1x: Fixed		Specifies amount of data to be transferred in a single operation 0x: 32 bits 10: 16 bits 11: 8 bits		Specifies I/O device bus width 0x: 32 bits 10: 16 bits 11: 8 bits	
			15	14	13	12	11	10	9	8
			—	ExR	PosE	Lev	SReq	RelEn	SIO	SAC1
			R/W							
			0	0	0	0	0	0	0	0
			Write 0 to this bit	Specifies transfer request method 1: Initiated by inter-rupt request 0: Internal request	Write 0 to this bit.	Write 1 to this bit.	Selects Snoop function 0: Disabled 1: Enabled	Enables request for surrender of bus control 0: Disabled 1: Enabled	Specifies source device 0: Memory 1: I/O	Bits 8 and 7 specify how source address will be changed 00: Incre-mented 01: Decre-mented 1x: Fixed
			23	22	21	20	19	18	17	16
			NIEn	AbIEn	—	—	—	—	Big	—
			R/W							
			1	1	1	0	0	0	1	0
			Enables interrupt on normal termination 0: Disabled 1: Enabled	Enables interrupt on abnormal termination 0: Disabled 1: Enabled	Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.
			31	30	29	28	27	26	25	24
			Str							—
			W							W
			0	0	0	0	0	0	0	0
			1: Starts DMA Channel 2							Write 0 to this bit.
CSR2	DMA Channel status Register 2	FFFF E244H	7	6	5	4	3	2	1	0
								—	—	—
								R/W		
			0	0	0	0	0	0	0	0
								Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.
			15	14	13	12	11	10	9	8
			0	0	0	0	0	0	0	0
			23	22	21	20	19	18	17	16
			NC	AbC	—	BES	BED	Conf		
				R/W			R			
			0	0	0	0	0	0	0	0
			1: Normal Termi-nation Status flag	1: Abnormal Termi-nation Status flag	Write 0 to this bit	1: Bus error in source address	1: Bus error in desti-nation address	1: Configu-ration error		
			31	30	29	28	27	26	25	24
			Act							
			R							
			0	0	0	0	0	0	0	0
			1: DMA Channel 2 in standby state							

DMA control (8/13)

Symbol	Name	Address	7	6	5	4	3	2	1	0
SAR2	DMA Source address Register 2	FFFF E248H	SAddr7	SAddr6	SAddr5	SAddr4	SAddr3	SAddr2	SAddr1	SAddr0
			R/W							
			Undefined							
			15	14	13	12	11	10	9	8
			SAddr15	SAddr14	SAddr13	SAddr12	SAddr11	SAddr10	SAddr9	SAddr8
			R/W							
			Undefined							
			23	22	21	20	19	18	17	16
			SAddr23	SAddr22	SAddr21	SAddr20	SAddr19	SAddr18	SAddr17	SAddr16
			R/W							
			Undefined							
			31	30	29	28	27	26	25	24
			SAddr31	SAddr30	SAddr29	SAddr28	SAddr27	SAddr26	SAddr25	SAddr24
			R/W							
			Undefined							
DAR2	DMA Destination address Register 2	FFFF E24CH	7	6	5	4	3	2	1	0
			DAddr7	DAddr6	DAddr5	DAddr4	DAddr3	DAddr2	DAddr1	DAddr0
			R/W							
			Undefined							
			15	14	13	12	11	10	9	8
			DAddr15	DAddr14	DAddr13	DAddr12	DAddr11	DAddr10	DAddr9	DAddr8
			R/W							
			Undefined							
			23	22	21	20	19	18	17	16
			DAddr23	DAddr22	DAddr21	DAddr20	DAddr19	DAddr18	DAddr17	DAddr16
			R/W							
			Undefined							
			31	30	29	28	27	26	25	24
			DAddr31	DAddr30	DAddr29	DAddr28	DAddr27	DAddr26	DAddr25	DAddr24
			R/W							
			Undefined							
BCR2	DMA Byte Count Register 2	FFFF E250H	7	6	5	4	3	2	1	0
			BC7	BC6	BC5	BC4	BC3	BC2	BC1	BC0
			R/W							
			Undefined							
			15	14	13	12	11	10	9	8
			BC15	BC14	BC13	BC12	BC11	BC10	BC9	BC8
			R/W							
			Undefined							
			23	22	21	20	19	18	17	16
			BC23	BC22	BC21	BC20	BC19	BC18	BC17	BC16
			R/W							
			Undefined							
			31	30	29	28	27	26	25	24
			0	0	0	0	0	0	0	0

DMA control (9/13)

Symbol	Name	Address	7	6	5	4	3	2	1	0
DTCR2	DMA Transfer control Register 2	FFFF E258H			DACM2	DACM1	DACM0	SACM2	SACM1	SACM0
					R/W					
			0	0	0	0	0	0	0	0
					Bit position at which destination address starts 000: Counts beginning at bit 0 of the address counter 001: Counts beginning at bit 4 of the address counter 010: Counts beginning at bit 8 of the address counter 011: Counts beginning at bit 12 of the address counter 100: Counts beginning at bit 16 of the address counter 101: Reserved 110: Reserved 111: Reserved			Bit position at which source address starts 000: Counts beginning at bit 0 of the address counter 001: Counts beginning at bit 4 of the address counter 010: Counts beginning at bit 8 of the address counter 011: Counts beginning at bit 12 of the address counter 100: Counts beginning at bit 16 of the address counter 101: Reserved 110: Reserved 111: Reserved		
			15	14	13	12	11	10	9	8
			0	0	0	0	0	0	0	0
			23	22	21	20	19	18	17	16
			0	0	0	0	0	0	0	0
			31	30	29	28	27	26	25	24
			0	0	0	0	0	0	0	0

DMA control (10/13)

Symbol	Name	Address	7	6	5	4	3	2	1	0
CCR3	DMA channel Control Register 3	FFFF E260H	SAC0	DIO	DAC1	DAC0	TrSiz1	TrSiz0	DPS1	DPS0
			R/W							
			0	0	0	0	0	0	0	0
			Bits 8 and 7 specify how source address will be changed 00: Incre-mented 01: Decre-mented 1x: Fixed	Specifies destination device 0: Memory 1: I/O	Specifies how destination address will be changed 00: Incremented 01: Decremented 1x: Fixed		Specifies amount of data to be transferred in a single operation 0x: 32 bits 10: 16 bits 11: 8 bits		Specifies I/O device bus width 0x: 32 bits 10: 16 bits 11: 8 bits	
			15	14	13	12	11	10	9	8
			—	ExR	PosE	Lev	SReq	RelEn	SIO	SAC1
			R/W							
			0	0	0	0	0	0	0	0
			Write 0 to this bit.	Specifies transfer request method 1: Initiated by interrupt request 0: Internal request	Write 0 to this bit.	Write 1 to this bit.	Selects Snoop function 0: Disabled 1: Enabled	Enables request for surrender of bus control 0: Disabled 1: Enabled	Specifies source device 0: Memory 1: I/O	Bits 8 and 7 specify how source address will be changed 00: Incre-mented 01: Decre-mented 1x: Fixed
			23	22	21	20	19	18	17	16
			NIEn	AbIEn	—	—	—	—	Big	—
			R/W							
			1	1	1	0	0	0	1	0
			Enables interrupt on normal termination 0: Disabled 1: Enabled	Enables interrupt on abnormal termination 0: Disabled 1: Enabled	Write 0 to this bit	Write 0 to this bit	Write 0 to this bit	Write 0 to this bit	Write 0 to this bit	Write 0 to this bit
			31	30	29	28	27	26	25	24
			Str							—
			W							W
			0	0	0	0	0	0	0	0
			1: Starts DMA Channel 3							Write 0 to this bit.
CSR3	DMA Channel status Register 3	FFFF E264H	7	6	5	4	3	2	1	0
								—	—	—
								R/W		
			0	0	0	0	0	0	0	0
								Write 0 to this bit.	Write 0 to this bit.	Write 0 to this bit.
			15	14	13	12	11	10	9	8
			0	0	0	0	0	0	0	0
			23	22	21	20	19	18	17	16
			NC	AbC	—	BES	BED	Conf		
				R/W			R			
			0	0	0	0	0	0	0	0
			1: Normal Termi-nation Status flag	1: Abnormal Termi-nation Status flag	Write 0 to this bit	1: Bus error in source address	1: Bus error in desti-nation address	1: Configu-ration error		
			31	30	29	28	27	26	25	24
			Act							
			R							
			0	0	0	0	0	0	0	0
			1: DMA Channel 3 in standby state							

DMA control (11/13)

Symbol	Name	Address	7	6	5	4	3	2	1	0
SAR3	DMA Source address Register 3	FFFF E268H	SAddr7	SAddr6	SAddr5	SAddr4	SAddr3	SAddr2	SAddr1	SAddr0
			R/W							
			Undefined							
			15	14	13	12	11	10	9	8
			SAddr15	SAddr14	SAddr13	SAddr12	SAddr11	SAddr10	SAddr9	SAddr8
			R/W							
			Undefined							
			23	22	21	20	19	18	17	16
			SAddr23	SAddr22	SAddr21	SAddr20	SAddr19	SAddr18	SAddr17	SAddr16
			R/W							
			Undefined							
			31	30	29	28	27	26	25	24
			SAddr31	SAddr30	SAddr29	SAddr28	SAddr27	SAddr26	SAddr25	SAddr24
			R/W							
			Undefined							
DAR3	DMA Destination address Register 3	FFFF E26CH	7	6	5	4	3	2	1	0
			DAddr7	DAddr6	DAddr5	DAddr4	DAddr3	DAddr2	DAddr1	DAddr0
			R/W							
			Undefined							
			15	14	13	12	11	10	9	8
			DAddr15	DAddr14	DAddr13	DAddr12	DAddr11	DAddr10	DAddr9	DAddr8
			R/W							
			Undefined							
			23	22	21	20	19	18	17	16
			DAddr23	DAddr22	DAddr21	DAddr20	DAddr19	DAddr18	DAddr17	DAddr16
			R/W							
			Undefined							
			31	30	29	28	27	26	25	24
			DAddr31	DAddr30	DAddr29	DAddr28	DAddr27	DAddr26	DAddr25	DAddr24
			R/W							
			Undefined							
BCR3	DMA Byte Count Register 3	FFFF E270H	7	6	5	4	3	2	1	0
			BC7	BC6	BC5	BC4	BC3	BC2	BC1	BC0
			R/W							
			Undefined							
			15	14	13	12	11	10	9	8
			BC15	BC14	BC13	BC12	BC11	BC10	BC9	BC8
			R/W							
			Undefined							
			23	22	21	20	19	18	17	16
			BC23	BC22	BC21	BC20	BC19	BC18	BC17	BC16
			R/W							
			Undefined							
			31	30	29	28	27	26	25	24
			0	0	0	0	0	0	0	0

DMA control (12/13)

Symbol	Name	Address	7	6	5	4	3	2	1	0
DTCR3	DMA Transfer control Register 3	FFFF E278H			DACM2	DACM1	DACM0	SACM2	SACM1	SACM0
					R/W					
			0	0	0	0	0	0	0	0
					Bit position at which destination address starts 000: Counts beginning at bit 0 of the address counter 001: Counts beginning at bit 4 of the address counter 010: Counts beginning at bit 8 of the address counter 011: Counts beginning at bit 12 of the address counter 100: Counts beginning at bit 16 of the address counter 101: Reserved 110: Reserved 111: Reserved			Bit position at which source address starts 000: Counts beginning at bit 0 of the address counter 001: Counts beginning at bit 4 of the address counter 010: Counts beginning at bit 8 of the address counter 011: Counts beginning at bit 12 of the address counter 100: Counts beginning at bit 16 of the address counter 101: Reserved 110: Reserved 111: Reserved		
			15	14	13	12	11	10	9	8
			0	0	0	0	0	0	0	0
			23	22	21	20	19	18	17	16
			0	0	0	0	0	0	0	0
			31	30	29	28	27	26	25	24
			0	0	0	0	0	0	0	0

DMA control (13/13)

Symbol	Name	Address	7	6	5	4	3	2	1	0
DCR	DMA Control Register	FFFF E280H								
			0	0	0	0	0	0	0	0
			15	14	13	12	11	10	9	8
			0	0	0	0	0	0	0	0
			23	22	21	20	19	18	17	16
			0	0	0	0	0	0	0	0
			31	30	29	28	27	26	25	24
			Rst							
			W							
			0	0	0	0	0	0	0	0
			1: Initializes DMA							
DHR	DMA Data Holding Register	FFFF E28CH	7	6	5	4	3	2	1	0
			DOT7	DOT6	DOT5	DOT4	DOT3	DOT2	DOT1	DOT0
			R/W							
			Undefined							
			15	14	13	12	11	10	9	8
			DOT15	DOT14	DOT13	DOT12	DOT11	DOT10	DOT9	DOT8
			R/W							
			Undefined							
			23	22	21	20	19	18	17	16
			DOT23	DOT22	DOT21	DOT20	DOT19	DOT18	DOT17	DOT16
			R/W							
			Undefined							
			31	30	29	28	27	26	25	24
			DOT31	DOT30	DOT29	DOT28	DOT27	DOT26	DOT25	DOT24
			R/W							
			Undefined							

(7) 8-Bit Timer control

Symbol	Name	Address	7	6	5	4	3	2	1	0
TA01-RUN	TMRA01 RUN	FFFF F100H	TA0RDE				I2TA01	TA01PRUN	TA1RUN	TA0RUN
			R/W				R/W			
			0				0	0	0	0
			Double Buffer 0: Disabled 1: Enabled				IDLE 0: Idle 1: Operate	Timer Run/Stop Control 0: Stop & Clear 1: Run (Count up)		
TA23-RUN	TMRA23 RUN	FFFF F108H	TA2RDE				I2TA23	TA23PRUN	TA3RUN	TA2RUN
			R/W				R/W			
			0				0	0	0	0
			Double Buffer 0: Disabled 1: Enabled				IDLE 0: Idle 1: Operate	Timer Run/Stop Control 0: Stop & Clear 1: Run (Count up)		
TA01-MOD	TMRA01	FFFF F104H	TA01M1	TA01M0	PWM01	PWM00	TA1CLK1	TA1CLK0	TA0CLK1	TA0CLK0
			R/W							
			0	0	0	0	0	0	0	0
			Operation mode 00: 8-Bit Timer Mode 01: 16-Bit Timer Mode 10: 8-Bit PPG Mode 11: 8-Bit PWM Mode		PWM period 00: Reserved 01: 2^6-1 10: 2^7-1 11: 2^8-1		TMRA1 source clock 00: TA0TRG 01: ϕT1 10: ϕT16 11: ϕT256		TMRA0 source clock 00: TA0IN pin input 01: ϕT1 10: ϕT4 11: ϕT16	
TA23-MOD	TMRA23	FFFF F10CH	TA23M1	TA23M0	PWM21	PWM20	TA3CLK1	TA3CLK0	TA2CLK1	TA2CLK0
			R/W							
			0	0	0	0	0	0	0	0
			Operation mode 00: 8-Bit Timer Mode 01: 16-Bit Timer Mode 10: 8-Bit PPG Mode 11: 8-Bit PWM Mode		PWM period 00: Reserved 01: 2^6-1 10: 2^7-1 11: 2^8-1		TMRA3 source clock 00: TA2TRG 01: ϕT1 10: ϕT16 11: ϕT256		TMRA2 source clock 00: TA2IN 01: ϕT1 10: ϕT4 11: ϕT16	
TA1FFCR	TMRA1	FFFF F105H					TAFF1C1	TAFF1C0	TAFF1IE	TAFF1IS
							R/W			
							1	1	0	0
							00: Invert TA1FF 01: Set TA1FF 10: Clear TA1FF 11: Don't care		TA1FF inversion control 0: Disabled 1: Enabled	Specifies TA1FF-inverting signal 0: TMRA0 1: TMRA1
TA3FFCR	TMRA3	FFFF F10DH					TAFF3C1	TAFF3C0	TAFF3IE	TAFF3IS
							R/W			
							1	1	0	0
							00: Invert TA3FF 01: Set TA3FF 10: Clear TA3FF 11: Don't care		TA3FF inversion control 0: Disabled 1: Enabled	Specifies TA3FF-inverting signal 0: TMRA2 1: TMRA3

(8) 16-Bit Timer control (1/3)

Symbol	Name	Address	7	6	5	4	3	2	1	0
TB0RUN	TMRB0 RUN	FFFF F180H	TB0RDE	—			I2TB0	TB0PRUN		TB0RUN
			R/W				R/W			R/W
			0	0			0	0		0
			Double Buffer 0: Disabled 1: Enabled	Write 0 to this bit.			IDLE 0: Idle 1: Operate	Timer Run/Stop Control 0: Stop & Clear 1: Run (Count up)		
TB1RUN	TMRB1 RUN	FFFF F190H	TB1RDE	—			I2TB1	TB1PRUN		TB1RUN
			R/W				R/W			R/W
			0	0			0	0		0
			Double Buffer 0: Disabled 1: Enabled	Write 0 to this bit.			IDLE 0: Idle 1: Operate	16 Bit Timer Run/Stop Control 0: Stop & Clear 1: Run (Count up)		
TB2RUN	TMRB0 RUN	FFFF F1A0H	TB2RDE	—			I2TB2	TB2PRUN		TB2RUN
			R/W				R/W			R/W
			0	0			0	0		0
			Double Buffer 0: Disabled 1: Enabled	Write 0 to this bit.			IDLE 0: Idle 1: Operate	Timer Run/Stop Control 0: Stop & Clear 1: Run (Count up)		
TB3RUN	TMRB1 RUN	FFFF F1B0H	TB3RDE	—			I2TB3	TB3PRUN		TB3RUN
			R/W				R/W			R/W
			0	0			0	0		0
			Double Buffer 0: Disabled 1: Enabled	Write 0 to this bit.			IDLE 0: Idle 1: Operate	16 Bit Timer Run/Stop Control 0: Stop & Clear 1: Run (Count up)		
TB0MOD	TMRB0	FFFF F182H	—	—	TB0CP0	TB0CPM1	TB0CPM0	TB0CLE	TB0CLK1	TB0CLK0
					W*	R/W				
			0	0	1	0	0	0	0	0
			Must always be set to 00.		Software capture control 0: Software capture 1: Don't care	Capture timing 00: Disabled 01: TB0IN0 ↑ TB0IN1 ↑ 10: TB0IN0 ↑ TB0IN1 ↓ 11: TA1OUT ↑ TA1OUT ↓		Up-counter control 0: Clear disabled 1: Clear enabled	Specifies source clock 00: TB0IN0 pin input 01: φT1 10: φT4 11: φT16	

(8) 16-Bit Timer control (2/3)

Symbol	Name	Address	7	6	5	4	3	2	1	0
TB1MOD	TMRB1	FFFF F192H	—	—	TB1CP0	TB1CPM1	TB1CPM0	TB1CLE	TB1CLK1	TB1CLK0
			R/W		W*	R/W				
			0	0	1	0	0	0	0	0
			Must always be set to 00.		Software capture control 0: Software capture 1: Don't care	Capture timing 00: Disabled 01: TB1IN0 ↑ TB1IN1 ↑ 10:TB1IN0 ↑ TB1IN1 ↓ 11: TA1OUT ↑ TA1OUT ↓		Up-counter control 0: Clear disabled 1: Clear enabled	Specifies source clock 00: TB1IN0 pin input 01: φT1 10: φT4 11: φT16	
TB2MOD	TMRB2	FFFF F1A2H	—	—	TB2CP0	TB2CPM1	TB2CPM0	TB2CLE	TB2CLK1	TB2CLK0
					W*	R/W				
			0	0	1	0	0	0	0	0
			Must always be set to 00.		Software capture control 0: Software capture 1: Don't care	Capture timing 00: Disabled 01: TB2IN0 ↑ TB2IN1 ↑ 10:TB2IN0 ↑ TB2IN1 ↓ 11: TA1OUT ↑ TA1OUT ↓		Up-counter control 0: Clear disabled 1: Clear enabled	Specifies source clock 00: TB0IN0 pin input 01: φT1 10: φT4 11: φT16	
TB3MOD	TMRB3	FFFF F1B2H	—	—	TB3CP0	TB3CPM1	TB3CPM0	TB3CLE	TB3CLK1	TB3CLK0
					W*	R/W				
			0	0	1	0	0	0	0	0
			Must always be set to 00.		Software capture control 0: Software capture 1: Don't care	Capture timing 00: Disabled 01: Disabled 10: Disabled 11: TA1OUT ↑ TA1OUT ↓		Up-counter control 0:Clear disabled 1: Clear enabled	Specifies source clock 00: TB3IN0 pin input 01: φT1 10: φT4 11: φT16	
TB0FFCR	TMRB0	FFFF F183H	—	—	TB0C1T1	TB0C0T1	TB0E1T1	TB0E0T1	TB0FF0C1	TB0FF0C0
			W*		R/W				W*	
			1	1	0	0	0	0	1	1
			Must always be set to 11. * Always read as 11.		TB0FF0 inversion trigger 0: Disables trigger 1: Enables trigger				Controls TB0FF0 00: Invert 01: Set 10: Clear 11: Don't care * Always read as 11.	
					When capturing up-counter value into TB0CP1	When capturing up-counter value into TB0CP0	When up-counter and TB0RG1 match	When up-counter and TB0RG0 match		

(8) 16-Bit Timer control (3/3)

Symbol	Name	Address	7	6	5	4	3	2	1	0
TB1FFCR	TMRB1	FFFF F193H	—	—	TB1C1T1	TB1C0T1	TB1E1T1	TB1E0T1	TB1FF0C1	TB1FF0C0
			W*		R/W				W*	
			1	1	0	0	0	0	1	1
			Must always be set to 11. * Always read as 11.		TB1FF0 inversion trigger 0: Disables trigger 1: Enables trigger				Controls TB1F0 00: Invert 01: Set 10: Clear 11: Don't care * Always read as 11.	
					When capturing up-counter value into TB1CP1	When capturing up-counter value into TB1CP0	When up-counter and TB1RG1 match	When up-counter and TB1RG0 match		
TB2FFCR	TMRB2	FFFF F1A3H	—	—	TB2C1T1	TB2C0T1	TB2E1T1	TB2E0T1	TB2FF0C1	TB2FF0C0
			W*		R/W				W*	
			1	1	0	0	0	0	1	1
			Must always be set to 11. * Always read as 11.		TB2FF0 inversion trigger 0: Disables trigger 1: Enables trigger				Controls TB2FF0 00: Invert 01: Set 10: Clear 11: Don't care * Always read as 11.	
					When capturing up-counter value into TB2CP1	When capturing up-counter value into TB2CP0	When up-counter and TB2RG1 match	When up-counter and TB2RG0 match		
TB3FFCR	TMRB3	FFFF F1B3H	—	—	TB3C1T1	TB3C0T1	TB3E1T1	TB3E0T1	TB3FF0C1	TB3FF0C0
			W*		R/W				W*	
			1	1	0	0	0	0	1	1
			Must always be set to 11. * Always read as 11.		TB3FF0 inversion trigger 0: Disables trigger 1: Enables trigger				Controls TB3FF0 00: Invert 01: Set 10: Clear 11: Don't care * Always read as 11.	
					When capturing up-counter value into TB3CP1	When capturing up-counter value into TB3CP0	When up-counter and TB3RG1 match	When up-counter and TB3RG0 match		

(9) UART/Serial channel (1/4)

Symbol	Name	Address	7	6	5	4	3	2	1	0
SC0CR	Serial Channel 0 Control	FFFF F201H	RB8	EVEN	PE	OERR	PERR	FERR	SCLKS	IOC
			R	R/W		R (Cleared to 0 by reading)			R/W	
			0	0	0	0	0	0	0	0
			Bit 8 of received data	Parity type 0: Odd 1: Even	Parity 0: Disabled 1: Enabled	1: Error			0:SCLK0↑ 1:SCLK0↓	0: Baud rate generator 1: SCLK0 pin input
						Overrun	Parity	Framing		
SC0-MOD0	Serial Channel 0 Mode0	FFFF F202H	TB8	CTSE	RXE	WU	SM1	SM0	SC1	SC0
			R/W							
			0	0	0	0	0	0	0	0
			Bit 8 of transmission data	Controls hand-shaking function 0: Disables CTS 1: Enables CTS	Controls reception 0: Disables reception 1: Enables reception	Wake-up function 0: Disabled 1: Enabled	Serial transfer mode 00: I/O Interface Mode 01: 7-Bit UART Mode 10: 8-Bit UART Mode 11: 9-Bit UART Mode		Serial transfer clock (for UART) 00: Timer TA0TRG 01: Baud rate generator 10: Internal clock $f_{SYS}/2$ 11: External clock (SCLK0 input)	
BR0CR	Baud Rate Control	FFFF F203H	—	BR0ADDE	BR0CK1	BR0CK0	BR0S3	BR0S2	BR0S1	BR0S0
			R/W							
			0	0	0	0	0	0	0	0
			Write 0 to this bit	N+ (16–K)/16 Division function 0: Disabled 1: Enabled	00: φT0 01: φT2 10: φT8 11: φT32		Specifies value of divisor N			
BR0-ADD	Baud Rate Control	FFFF F204H					BR0K3	BR0K2	BR0K1	BR0K0
							R/W			
							0	0	0	0
							Specifies K value for N + (16 – K) / 16 Division			
SC0-MOD1	Serial Channel 0 Mode1	FFFF F205H	I2S0	FDPX0						
			R/W	R/W						
			0	0						
			IDLE 0: Idle 1: Operate	Synchro-nous operation 1: Full-duplex 0: Half-duplex						

UART / Serial channel (2/4)

Symbol	Name	Address	7	6	5	4	3	2	1	0
SC1CR	Serial Channel 1 Control	FFFF F209H	RB8	EVEN	PE	OERR	PERR	FERR	SCLKS	IOC
			R	R/W		R (Cleared to 0 by reading)			R/W	
			0	0	0	0	0	0	0	0
			Bit 8 of received data	Parity type 0: Odd 1: Even	Parity 0: Disabled 1: Enabled	1: Error			0: SCLK1↑ 1: SCLK1↓	0: Baud rate generator 1: SCLK0 pin input
						Overrun	Parity	Framing		
SC1-MOD0	Serial Channel 1 Mode	FFFF F20AH	TB8	CTSE	RXE	WU	SM1	SM0	SC1	SC0
			R/W							
			0	0	0	0	0	0	0	0
			Bit 8 pof transmission data	Controls hand-shaking function 0: Disables CTS 1: Enables CTS	Controls reception 0: Disables reception 1: Enables reception	Wake-up function 0: Disabled 1: Enabled	Serial transfer mode 00: I/O Interface Mode 01: 7-Bit UART Mode 10: 8-Bit UART Mode 11: 9-Bit UART Mode		Serial transfer clock (for UART) 00: Timer TA0TRG 01: Baud rate generator 10: Internal clock $f_{SYS}/2$ 11: External clock (SCLK1 input)	
BR1CR	Baud Rate Control	FFFF F20BH	—	BR1ADDE	BR1CK1	BR1CK0	BR1S3	BR1S2	BR1S1	BR1S0
			R/W							
			0	0	0	0	0	0	0	0
			Write 0 to this bit.	N+ (16–K)/16 Division function 0: Disabled 1: Enabled	00: ϕT0 01: ϕT2 10: ϕT8 11: ϕT32		Specifies value of divisor N			
BR1-ADD	Baud Rate Control	FFFF F20CH					BRK1K3	BRK1K2	BRK1K1	BRK1K0
							R/W			
							0	0	0	0
							Specifies K value for N + (16 – K) / 16 Division			
SC1-MOD1	Serial Channel 1 Mode1	FFFF F20DH	I2S0	FDPX0						
			R/W							
			0	0						
			IDLE 0: Idle 1: Operate	Synchro-nous operation 1: Full-duplex 0: Half-duplex						

UART/Serial channel (3/4)

Symbol	Name	Address	7	6	5	4	3	2	1	0
SC3CR	Serial Channel 3 Control	FFFF F281H	RB8	EVEN	PE	OERR	PERR	FERR	—	—
			R	R/W		R (Cleared to 0 by reading)			R/W	
			0	0	0	0	0	0	0	0
			Bit 8 of received data	Parity type 0: Odd 1: Even	Parity 0: Disabled 1: Enabled	1: Error			Must always be set to 0.	
						Overrun	Parity	Framing		
SC3-MOD0	Serial Channel 3 Mode	FFFF F282H	TB8	CTSE	RXE	WU	SM1	SM0	SC1	SC0
			R/W							
			0	0	0	0	0	0	0	0
			Bit 8 of transmission data	Must always be set to 0.	Controls reception 0: Disables reception 1: Enables reception	Wake-up function 0: Disabled 1: Enabled	Serial transfer mode 00: Reserved 01: 7-Bit UART Mode 10: 8-Bit UART Mode 11: 9-Bit UART Mode		Serial transfer clock (for UART) 00: Timer TA0TRG 01: Baud rate generator 10: Internal clock $f_{SYS}/2$ 11: Don't care	
BR3CR	Baud Rate Control	FFFF F283H	—	BR3ADDE	BR3CK1	BR3CK0	BR3S3	BR3S2	BR3S1	BR3S0
			R/W							
			0	0	0	0	0	0	0	0
			Write 0 to this bit.	N+ (16–K)/16 Division function 0: Disabled 1: Enabled	00: φT0 01: φT2 10: φT8 11: φT32		Specifies value of divisor N			
BR3-ADD	Baud Rate Control	FFFF F284H					BR3K3	BR3K2	BR3K1	BR3K0
							R/W			
							0	0	0	0
							Specifies K value for N + (16 – K) / 16 Division			
SC3-MOD1	Serial Channel 3 Mode1	FFFF F285H	I2S0							
			R/W							
			0							
			IDLE 0: Idle 1: Operate							

UART/Serial channel (4/4)

(9-4) UART only Channel4

Symbol	Name	Address	7	6	5	4	3	2	1	0
SC4CR	Serial Channel 4 Control	FFFF F289H	RB8	EVEN	PE	OERR	PERR	FERR	—	—
			R	R/W		R (Cleared to 0 by reading)			R/W	
			0	0	0	0	0	0	0	0
			Bit 8 of received data	Parity type 0: Odd 1: Even	Parity 0: Disabled 1: Enabled	1: Error			Must always be set to 0.	
						Overrun	Parity	Framing		
SC4-MOD0	Serial Channel 4 Mode	FFFF F28AH	TB8	CTSE	RXE	WU	SM1	SM0	SC1	SC0
			R/W							
			0	0	0	0	0	0	0	0
			Bit 8 of transmission data	Must always be set to 0.	Controls reception 0: Disables reception 1: Enables reception	Wake-up function 0: Disabled 1: Enabled	Serial transfer mode 00: Reserved 01: 7-Bit UART Mode 10: 8-Bit UART Mode 11: 9-Bit UART Mode		Serial transfer clock (for UART) 00: Timer TA0TRG 01: Baud rate generator 10: Internal clock $f_{SYS}/2$ 11: Don't care	
BR4CR	Baud Rate Control	FFFF F28BH	—	BR4ADDE	BR4CK1	BR4CK0	BR4S3	BR4S2	BR4S1	BR4S0
			R/W							
			0	0	0	0	0	0	0	0
			Write 0 to this bit.	N+ (16–K)/16 Division function 0: Disabled 1: Enabled	00: ϕT0 01: ϕT2 10: ϕT8 11: ϕT32		Specifies value of divisor N			
BR4-ADD	Baud Rate Control	FFFF F28CH					BR4K3	BR4K2	BR4K1	BR4K0
							R/W			
							0	0	0	0
							Specifies K value for N + (16 – K) / 16 Division			
SC4-MOD1	Serial Channel 4 Mode1	FFFF F28DH	I2S0							
			R/W							
			0							
			IDLE 0: Idle 1: Operate							

(10) I²C bus / Serial channel control

Symbol	Name	Address	7	6	5	4	3	2	1	0/
SBI0CR1	Serial Bus Interface Control Register 1	FFFF F240H (I²C Bus Mode)	BC2	BC1	BC0	ACK		SCK2	SCK1	SCK0 SWRMON
			W			R/W		W	W	R/W
			0	0	0	0		0	0	1
			Specifies number of bits to be transferred (when <ACK> = 0) 000: 8, 001: 1, 010: 2 011: 3, 100: 4, 101: 5 110: 6, 111: 7			Acknowledgment clock 0: Not generated 1: Generated		Specifies internal SCL output clock frequency and reflects reset state (when writing) 000: 4, 001: 5, 010: 6 011: 7, 100: 8, 101: 9 110: 10, 111: Reserved		
		FFFF F240H (SIO Mode)	SIOS	SIOINH	SIOM1	SIOM0		SCK2	SCK1	SCK0
			W					W		R/W
			0	0	0	0		0	0	1
			Controls transfer 0: Finishes transfer 1: Starts transfer	Forcibly stops transfer 0: Transfer continues 1: Transfer stops	Specifies transfer mode 00: Transmit Mode 01: Reserved 10: Transmit/Receive Mode 11: Receive Mode			Specifies serial clock frequency (when writing) 000 : 3, 001 : 4, 010 : 5 011 : 6, 100 : 7, 101 : 8 110 : 9, 111 : external clock		
SBI0-DBR	SBI Buffer Register	FFFF F241H	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
			R (Receiving)/W (Transmission)							
			Undefined							
I2C0AR	I²C bus Address Register	FFFF F242H	SA6	SA5	SA4	SA3	SA2	SA1	SA0	ALS
			W							
			0	0	0	0	0	0	0	0
			Specifies slave address when device is operating as slave device							Address recognition 0 : Recognize 1 : Do not recognize
SBI0CR2 when writing SBI0SR when reading	Serial Bus Interface Control Register 2	FFFF F243H (I²C Bus Mode)	MST	TRX	BB	PIN	SBIM1	SBIM0	SWRST1	SWRST0
			W							
			0	0	0	1	0	0	0	0
			Specifies master/slave	Specifies transmit/ receive	Generates start/stop state	Clears INTSBSI interrupt request	Serial bus interface operation mode 00: Port Mode 01: SIO Mode 10: I²C Bus Mode 11: Reserved		Generates software reset. A reset is generated by writing 10 and then 01.	
			MST	TRX	BB	PIN	AL	AAS	AD0	LRB
			R							
			0	0	0	1	0	0	0	0
			Reflects master/slave selection	Transmit/ Receive selection	Reflects I²C bus state	Monitors INTS2 interrupt request	Detects arbitration-lost condition 0: — 1: Detect	Detects slave address 0: — 1: Detect	Detects general call 0: — 1: Detect	Reflects last bit received 0: "0" 1: "1"
		FFFF F243H (SIO Mode)	—	—	—	—	SIOF	SEF	—	—
							R			
							0	0		
							Reflects transfer state 0: Finished 1: Transfer in progress	Reflects status of shift operation 0: Finished 1: Shift in progress		
SBI0-BR0	Serial Bus Interface Control Register 0	FFFF F244H		I2SBI0						
				R/W						W
				0						0
				IDLE 0: Idle 1: Operate						Must always be set to 0.
SBI0-BR1	Serial Bus Interface Control Register 1	FFFF F245H	P4EN							
			R/W							
			0							
			Internal clock 0: Idle 1: Operate							

(11) AD converter control

Symbol	Name	Address	7	6	5	4	3	2	1	0
ADMOD0	A/D MODE Reg0	FFFF F310H	EOCF	ADBF	—	—	ITM0	REPEAT	SCAN	ADS
			R		R/W					
			0	0	0	0	0	0	0	0
			AD Conversion End flag 0: Conversion in progress 1: Finished	AD Conversion Busy flag 0: Conversion stopped 1: Conversion in progress	Write 0 to this bit	Write 0 to this bit	INT timing during repeat mode	1: Repeat	1: Scan	1: Start
ADMOD1	A/D MODE Reg1	FFFF F311H	VREFON	I2AD			ADTRGE	ADCH2	ADCH1	ADCH0
			R/W				R/W			
			0	0			0	0	0	0
			Controls application of VREF 0: OFF 1: ON	IDLE 0: Stops application of VREF 1: Applies VREF			Controls external AD trigger 0: Disabled 1: Enabled	Selects analog input channel 000: AN0 AN0 001: AN1 AN0 →AN1 010: AN2 AN0 → AN1 → AN2 011: AN3 AN0 → AN1 → AN2 → AN3 100: AN4 AN4 101: AN5 AN4 → AN5 110: AN6 AN4 → AN5 → AN6 111: AN7 AN4 → AN5 → AN6 → AN7		
AD REG04L	AD Result Reg 0/4 low	FFFF F300H	ADR01	ADR00						ADR0RF
			R							R
			Undefined							0
AD REG04H	AD Result Reg 0/4 high	FFFF F301H	ADR09	ADR08	ADR07	ADR06	ADR05	ADR04	ADR03	ADR02
			R							
			Undefined							
AD REG15L	AD Result Reg 1/5 low	FFFF F302H	ADR11	ADR10						ADR1RF
			R							R
			Undefined							0
AD REG15H	AD Result Reg 1/5 high	FFFF F303H	ADR19	ADR18	ADR17	ADR16	ADR15	ADR14	ADR13	ADR12
			R							
			Undefined							
AD REG26L	AD Result Reg 2/6 low	FFFF F304H	ADR21	ADR20						ADR2RF
			R							R
			Undefined							0
AD REG26H	AD Result Reg 2/6 high	FFFF F305H	ADR29	ADR28	ADR27	ADR26	ADR25	ADR24	ADR23	ADR22
			R							
			Undefined							
AD REG37L	AD Result Reg 3/7 low	FFFF F306H	ADR31	ADR30	—	—	—	—	—	ADR3RF
			R							R
			Undefined							0
AD REG37H	AD Result Reg 3/7 high	FFFF F307H	ADR39	ADR38	ADR37	ADR36	ADR35	ADR34	ADR33	ADR32
			R							
			Undefined							
ADCCLK	AD MODE Clock Reg	FFFF EE04H							ADCCK1	ADCCK0
									R/W	
									0	0
									Selects AD conversion clock 00: fsys/2 01: fsys/4 10: fsys/8 11:Reserved	

(12) Watchdog Timer

Symbol	Name	Address	7	6	5	4	3	2	1	0
WDMOD	WDT MODE Reg	FFFF F090H	WDTE	WDTP1	WDTP0			I2WDT	RESCR	—
			R/W	R/W				R/W		
			1	0	0			0	0	0
			1: WDT control	00: $2^{16}/f_{sys}$ 01: $2^{18}/f_{sys}$ 10: $2^{20}/f_{sys}$ 11: $2^{22}/f_{sys}$				IDLE 0: Idle 1: Operate	1: Internal connection to RESET pin enabled	Write 0 to this bit.
WDCR	WD Control	FFFF F091H	—							
			W							
			—							
			B1H: WDT disable code; 4EH: WDT clear code							

(13) Timer for Real-Time Clock

Symbol	Name	Address	7	6	5	4	3	2	1	0
RTCCR	Timekeeping Timer Control Register	FFFF F0A0H	—				RTCRCLR	RTCSEL1	RTCSEL0	RTCRUN
			R/W				R/W	R/W		R/W
			0				0	0	0	0
			Write 0 to this bit.				0: Clears Accumu-lator	00: 2^{14}/fs 01: 2^{13}/fs 10: 2^{12}/fs 11: 2^{11}/fs		0: STOP & CLR 1: RUN
RTCREG	Accumulator	FFFF F0A4H	RUI7	RUI6	RUI5	RUI4	RUI3	RUI2	RUI1	RUI0
			R							
			0	0	0	0	0	0	0	0

(14) Flash control/status

Symbol	Name	Address	7	6	5	4	3	2	1	0
SEQMOD	Security Mode Reg	FFFFE510								SEQON
										R/W
										1
										1: Security on 0: Security off
SEQCNT	Security Control Reg	FFFFE514								
			W							
			Must be written as 0x0000_00C5.							
			W							
			Must be written as 0x0000_00C5.							
			W							
			Must be written as 0x0000_00C5.							
			W							
			Must be written as 0x0000_00C5.							
FLCS	Flash Control/ Status Reg	FFFF E520H					LVDD	RDY/BSY	—	FSE
							R/W	R	R/W	R/W
							0	1	0	0
							0: Normal (R) Cleared (W) Abnormal	0: Busy 1: Ready	Must be written as “0”.	0: Access main logic. 1: Access security logic.

Note: This register is a 32-bit register, which must be accessed in 32-bit units.

3.9 TX1940 Internal Circuits Used

The tables below list the TX1940 internal circuits which are used in these application notes.

(1) External interrupts

Interrupt Source	Purpose of Use in Application Notes
INT0	Starts/Stops DC motors; starts stopwatch
INT1	Unused
INT2	Unused
INT3	Unused
INT4	Unused
INT5	Unused
INT6	Unused
INT7	Unused
INT8	Unused
INT9	Unused
INTA	Unused
NMI	Takes stopwatch out of standby mode

(2) Port functions

Port		Purpose of Use in Application Notes
Port 0		Unused
Port 1		Unused
Port 2		Unused
Port 3		Unused
Port 4	P40	Unused
	P41	
	P42	
	P43	
	P44	DIP switch digit output
Port 5	P50	Unused
	P51	
	P52	
	P53	
	P54	Reads matrix key
	P55	Reads matrix key
	P56	Reads matrix key
	P57	Reads matrix key
Port 7	P70	Matrix key digit output
	P71	Unused
	P72	Matrix key digit output
	P73	Unused
	P74	Unused
	P75	Matrix key digit output
	P76	Matrix key digit output
	P77	Unused
Port 8	P87-P80	Outputs 7-segment LED and indicator LED display data
Port 9	P97-P90	Unused
Port A	PA0	Switches display digit between 7-segment LEDs and indicator LEDs
	PA1	Switches display digit between 7-segment LEDs and indicator LEDs
	PA2	Switches display digit between 7-segment LEDs and indicator LEDs
	PA3	Switches display digit between 7-segment LEDs and indicator LEDs
	PA4	Switches display digit between 7-segment LEDs and indicator LEDs
	PA5	Unused
	PA6	Unused
	PA7	Unused

(3) Chip select and wait controller

Block	Purpose of Use in Application Notes
CS0	Unused
CS1	Unused
CS2	Unused
CS3	Unused

(4) DMA controller (DMAC)

Channel Used	Purpose of Use in Application Notes	Remarks
Channel 0	Please refer to Section 3.7, Other Features	Memory-to-memory transfer
Channel 1	Unused	
Channel 2	Unused	
Channel 3	Unused	

(5) 8-Bit timers (TMRA)

Module	Mode Used	Channel Used	Purpose of Use in Application Notes
TMRA01	8-Bit Interval Timer	TMRA1(TA1FF)	Beep tone output (output)
	8-Bit Interval Timer	TMRA1	Combined with TMRB0
TMRA23	8-Bit Interval Timer	TMRA2	Analog data sample period
	8-bit PWM	TMRA2 (TA3FF)	DC motor drive (output)

(6) 16-bit timers (TMRB)

Channel Used	Mode Used	Purpose of Use in Application Notes
TMRB0	Capture feature	Motor encoder pulse input
TMRB1	Unused	Unused
TMRB2	16-Bit Interval Timer Mode	2-ms Interval Timer interrupt
TMRB3	Unused	Unused

(7) Serial channels (SIO)

Channel	Mode Used	Purpose of Use in Application Notes
SIO0	Unused	Unused
SIO1	UART Mode	PC communications (via RS-232C)
SIO2	Unused	Unused
SIO3	Unused	Unused

(8) Serial bus interface (SBI)

Channel	Mode Used	Purpose of Use in Application Notes
SBI	I^2C Bus Mode (multimasters)	E^2PROM access

(9) Analog-to-digital converter

Mode Used	Channel Used	Conversion Mode	Purpose of Use in Application Notes
Channel fixed	AN2	Single-Conversion Mode	AD conversion key input
	AN3	Single-Conversion Mode	Motor revolution control
Channel scan	AN0→AN1	Single-Conversion Mode	Microphone input, photo-interrupter input

(10) Watchdog Timer

Control	Purpose of Use in Application Notes
Enable examples	Please refer to Section 3.7, Other Features
Disable examples	Sample programs

(11) Real-Time Counter

Interrupt Interval	Purpose of Use in Application Notes
0.5 sec	Time count control