AN1310

Using the MC68332 Microcontroller for AC Induction Motor Control

Prepared by: Jeff Baum and Ken Berringer
Discrete Applications and Systems Engineering

ABSTRACT

AC induction motors offer a low cost variable speed solution for many motor control applications. The most common method for controlling an AC induction motor uses a three phase voltage source inverter with sine wave pulse width modulation (PWM) voltage control. One must have the means to produce sine waves of variable voltage and frequency. Specifically, sine wave frequency and amplitude are used to control motor speed and acceleration. A single microcontroller can be employed to generate sine wave modulated PWM waveforms and provide control functions.

Motorola's MC68332 ('332) is particularly well suited for the control of AC induction motors. The '332 is built around the CPU 32, a new 32 bit core. The CPU 32 utilizes an instruction set which is almost identical to that of MC68020. One new instruction, Table Look-up and Interpolation (TBL), provides for linear interpolation between points in a look-up table. In this paper, this instruction is used to generate very precise sine waves from a relatively small table. The '332 also contains the Time Processor Unit (TPU). This module is a microcoded processor dedicated to handling time function tasks. One of the microcoded primitives, Synchronous Pulse Width Modulation (SPWM), is utilized to generate three PWM waveforms for the AC induction motor. The TPU and SPWM primitive are configured to produce three waveforms with a common period, independently varying pulse widths, and well controlled time relationships between phases. The TPU limitations for minimum and maximum pulse width, minimum offset between waveforms, worst case latency, and update coherency are discussed.

An algorithm for generating variable voltage and frequency sine waves is presented. The CPU handles all calculations and periodically updates the TPU output waveforms. The relationships between the parameters of this algorithm, such as PWM frequency and sine wave voltage resolution, are also presented.

HISTORY

AC induction motors have been used for many years in a variety of applications. Three phase AC induction motors are readily available in a wide range of sizes from 1/4 HP up to 1000 HP. With the advent of large bipolar power transistors in the 1960's, it became possible to build variable speed systems using conventional AC induction motors. The complexity and cost of these drives has generally limited the application of variable speed AC motors to industrial applications where variable speed is required. Also, the slow switching speeds of the larger power devices limits operation to the audible range. While the noise associated with large variable speed drives is usually acceptable in noisy industrial environments, it is undesirable for consumer applications.

MARKET

The Japanese have adapted industrial AC drive principles to the consumer air conditioning market. Most of the variable speed air conditioners in Japan are small single room or split rack units and utilize 250 volt MOSFETs. The United States consumer air conditioning market requires larger units due to the larger average size home. Also, the U.S. standard house wiring of 230 VAC for larger appliances requires the use of 500–600 volt transistors. The recent development of high speed Insulated Gate Bipolar Transistors (IGBTs) will permit the operation of a large variable speed air conditioner at 20 kHz.

Another potentially large consumer market of variable speed AC induction motors is electric vehicles. The California legislature has mandated the production of electric vehicles by 1998. While the exact configuration of motor type, trans-axle, and battery voltage is still subject to much debate, other requirements are quite clear. A practical electric vehicle will have an inverter or inverters capable of supplying 50–100 kW at high switching frequencies. One of the most promising approaches is a high-speed AC induction motor and a three phase IGBT inverter.

AC INDUCTION MOTORS

An AC induction motor requires three nearly sinusoidal voltages, each 120° apart. Originally, these motors had been designed to operate at constant frequency and constant voltage (eg. 60 Hz and 230 VAC). These are not synchronous motors, and the actual speed of the motor is determined by the load and the resulting slip frequency. Induction motors are relatively inexpensive to manufacture. The rotor is a copper squirrel cage within laminations and does not require expensive permanent magnets. Three phase induction motors are ideal for the transmission of large amounts of power because they have a constant, nonpulsating total power and present a balanced inductive load to the AC mains.

In order to run an induction motor at variable speeds it is necessary to vary the sine wave frequency. In order to maintain control over the motor torque and current, the voltage applied to the motor must also be varied. Two common methods of controlling the motor speed are constant voltage over frequency control (constant V/f) and vector control. Much has been written on the subject of vector control. Vector control has some advantages for servo applications and applications where efficiency is of primary importance. Due to the complexity of vector control, the remainder of the paper will deal exclusively with simpler control methods.

The simplest form of AC motor speed control is the open loop constant V/f. A conventional AC motor is usually rated at a specific voltage and frequency, such as 230 VAC and 60 Hz. If the voltage and frequency are maintained at this ratio, the motor will produce a constant maximum torque. For example, if a 230 VAC 60 Hz motor is operated at 115 VAC and 30 Hz, it will produce the same torque and about half the speed. Because the system functions open loop without any speed feedback, this system does allow slip. Usually the slip frequency is fairly low, thus open loop control may be used for many variable speed applications. If the load torque is a well defined function of speed, the actual motor speed may be characterized as a function of the applied voltage and frequency.

Many constant V/f controllers also utilize a constant power operation region in order to improve speed range. Once the voltage reaches the maximum output voltage of the system the frequency can be further increased. Because the voltage is not increased, the maximum torque will decrease; however, maximum output power remains constant. Thus, a motor can produce half as much torque at twice the speed. A plot of voltage and frequency for a constant V/f controller with a constant power region is shown in Figure 1. This curve characterizes an example motor which is rated at 3/4 HP, and 1800 RPM, at nominal line voltages of 230 VAC, and 60 Hz. The speed range of a conventional AC induction motor should not be extended above the rated speed without assuring that the motor is physically capable of high speed operation without causing damage to the rotor or bearings. Motors can be designed for high speed operation thus, resulting in a smaller, less expensive motor. Electric vehicles may utilize a small high speed motor in order to reduce weight, size, and motor costs.

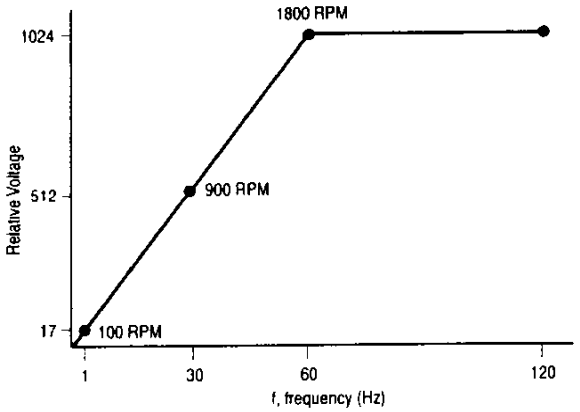

Figure 1. Constant V/F Control with Constant Power Region

PWM INVERTERS

The most common AC motor inverter today is a hard-switched, three-phase, pulse width modulated (PWM) inverter. The basic principle of a PWM drive is to apply a high-voltage pulse train to the motor at a high frequency and vary the duty cycle, or equivalently, the pulse high time. The average resulting voltage over each cycle is then the peak-to-peak voltage times the duty cycle. The duty of each phase is then "pulse-width-modulated" by both a variable frequency sine wave and a variable amplitude. This simple principle allows the generation of variable amplitude, three phase sine waves. This is a "textbook" example of a pulse-width-modulation communication system.

The actual voltage applied to the motor is a constantly changing train of high voltage pulses. However, the motor currents are nearly sinusoidal. The inductance of the motor is very important to the operation of the inverter. The inverter depends on the motor inductance and the back-EMF of the motor in order to limit the high frequency ripple current. In effect, the motor inductance integrates the applied voltage minus the motor back-EMF.

$$V = L \frac{dI}{dt}$$

$$I = \frac{1}{L} \int (V - EMF)\, dt$$

Integrating over one PWM period will produce the high frequency ripple current. The motor resistance also has some effect on the motor current, especially at low speeds.

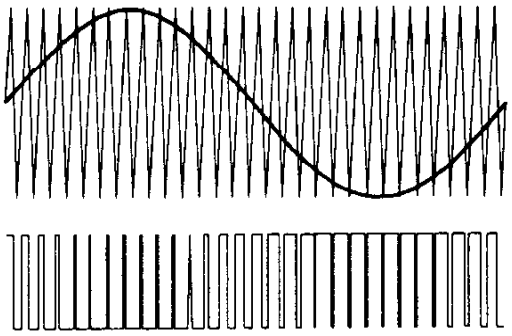

Figure 2. Sine Wave PWM

The generation of sine wave PWM waveforms is best understood by superimposing a sine wave signal with a triangle or sawtooth carrier. When the carrier amplitude is higher than the sine wave's, the output level is high. This is illustrated in Figure 2.

The generation of these PWM waveforms may be accomplished by analog methods using a variable frequency sine wave oscillator, a fixed frequency carrier wave oscillator, an analog multiplier, and a comparator.

DIRECT MICROPROCESSOR PWM

A more elegant approach is to do all calculations in the discrete time domain and avoid the analog sine wave. This approach provides the most accurate control over the effective sine wave amplitude and frequency. A single microprocessor may be used to generate all three PWM signals as well as provide control functions, communications, and a user interface. In this paper we will demonstrate an AC drive system using Motorola's MC68332 microcontroller.

PWM GENERATION

Microcontrollers (MCU's) are capable of producing logic-level pulses. By varying the duty cycle of a square waveform, a microcontroller can implement pulse-width modulation. Typical 8-bit MCU's, such as Motorola's MC68HC11, use interrupt driven or polling schemes to cause the logic-level transitions which produce specified pulse widths and periods for a PWM waveform on a given output pin. On occurrence of a predetermined timer match condition, the pin state makes the appropriate transition and the central processing unit (CPU) calculates the next low-high and high-low transition times. This waveform generation method requires CPU overhead to create each rising and falling edge on the desired signal. Since AC induction motor control requires three PWM signals, the service time imposed by the above technique severely limits the maximum PWM frequency one can produce. Such software latency is highly undesirable since it results in PWM frequencies within the audible range.

The MC68332 is a 32-bit, 68000-based MCU which is ideally suited for many motor control applications. The 68332 has a modular architecture which combines the high performance data manipulation of a 32-bit CPU with intelligent on-chip peripheral subsystems. The features of this MCU which are critical to our motor drive system are: the Time Processor Unit (TPU), the Table Look-Up and Interpolate (TBL) instruction, and the system's ability to operate at 16.78 MHz.

The TPU is a 16 channel dedicated time function processor. The 16 TPU channels, TP0–TP15, are independent, orthogonal channels (i.e. any channel can perform any of the TPU's time functions). The TPU also contains microcoded primitives for performing complex time related functions, such as motor control and pulse-width modulation. Once the necessary parameters for a given time function algorithm are written to a specific TPU channel by the CPU, the TPU runs autonomously. CPU intervention is only needed in order to alter a previously written parameter.

For the PWM example given above, one simply assigns a desired high time, period, and some reference addresses to a TPU channel parameter block, and the corresponding pin for that channel will output the desired waveform. To perform PWM, one can have the TPU cause an interrupt to the CPU to calculate and write a new high time (i.e. pulse width) to a given TPU channel parameter block. Typically, a MCU must interrupt the CPU to generate each logic transition of a PWM signal. In contrast, The TPU calculates all of the rising and falling transition times and affects the output pin accordingly without CPU intervention. CPU overhead is only needed to alter the current duty cycle (i.e. write new high times). Besides avoiding interrupt and polling latencies, the TPU offers additional speed, since the primitives are implemented in microcode. In addition, the 68332's 16.78 MHz system clock provides 240 ns timing resolution. This is twice the resolution of the HC11's general purpose timer. The effects of such timing resolution will be discussed in a subsequent section.

The TBL instruction can be used with signed or unsigned data (i.e. TBLS or TBLU, respectively). This instruction allows one to use a data table of up to 257 points and get 65,536 values of the function represented by the actual data points. In other words, this instruction performs an 8-bit table look-up and an 8-bit interpolation between consecutive data entries.

TPU CONFIGURATION

Five TPU channels have been used to realize this sine wave generation algorithm. A single channel configured to output a 50% duty cycle square wave with a period of 50 µs (20 kHz) is used as a "master" timing channel. Each sample of the resultant sine waves has a finite duration before the next sample is produced. This "master" channel is used to control the sine sample update rate, as well as for temporal alignment of signals on other TPU channels. Another channel uses the Input Transition Counter (ITC) primitive. This channel is programmed to count rising logic-level transitions that occur at the channel's associated input pin. This channel, in conjunction with the "master" timing channel

described above, provides the means for updating the sine wave samples at a designated time interval.

Three TPU channels are defined to execute the Synchronous PWM (SPWM) primitive. This TPU protocol allows multiple PWM channels to be configured with specified timing relationships between the channels. By synchronizing each PWM waveform to the "master" timing channel, coherent updating of PWM parameters and other benefits are obtainable. The specific details of how each TPU channel is used to implement our overall sine wave generation/motor control algorithm will be presented in the following sections.

INITIAL ALGORITHM

Originally, we chose to represent a sine wave using 21 samples of a sine function per wave cycle. Each sample of the sine function translates into a particular high time for the PWM waveforms. The pulse widths (high times) for each of the three PWM phases are determined by the following equations.

$$PWMA = \left(\frac{V \sin \theta + 1}{2} \right) PER$$

$$PWMB = \left(\frac{V \sin (\theta + 240^\circ) + 1}{2} \right) PER$$

$$PWMC = \left(\frac{V \sin (\theta + 120^\circ) + 1}{2} \right) PER$$

where;

V is the voltage amplitude of the sine wave,

θ is the angle of the sine function,

PER is the period of the PWM waveforms.

For a fixed number of 21 sine samples per sine wave cycle,

$$\theta = i \left(\frac{360^\circ}{21} \right), \text{ where } i = 0,1,2,3,\ldots,20$$

The 21 values of the sine function are stored as data in a look-up table (LUT) within the MCU. The rate at which we step through the values of the LUT and compute new pulse width values determines the frequency of the resulting sine wave. The algorithm sine modulates the high times of the three PWM signals, which drive their corresponding transistor bridges. The motor then integrates the voltages on it to produce sine wave currents.

This simple method has been modified to better drive the motor over the desired voltage and frequency range. The ability to vary the number of samples per sine cycle, update all PWM parameters coherently, and use a simple interface for speed control are some of the enhancements which have been examined to improve the above algorithm.

SINE WAVE APPROXIMATION

Since MCU's operate in the digital domain, sine waves must be constructed from discrete values of a sinusoidal function. Thus, a "stair-case" approximation to a sinusoid is the actual goal we wish to accomplish. As a result, one must consider the effect that sample duration, or "step" size, has on motor performance.

The simple algorithm above used a constant number of samples (21) per sine cycle and varied the duration of each step to create sine waves of different frequencies. Twenty-one steps of approximately 48 ms and 400 µs each must be produced (per cycle) for the lowest and highest sine frequencies, respectively. One design "rule-of-thumb" requires the step size to be less than or equal to one-half of the stator time constant of the motor. As determined by the inductance of the motor we are driving, the stator time constant is estimated at 500 µs. Therefore, it is desirable to maintain a step size which does not exceed 250 µs. Even for the best case example above (400 µs for a sine frequency of 120 Hz), the 21 sample method will not result in efficient motor control. One would need to have a data table of at least 4000 values in order to use a constant number of samples with an acceptable step size to produce the frequencies of interest. In addition to the undesirable LUT length demanded by this scenario, producing a frequency of 120 Hz (requiring four-thousand 2 µs steps) may not be feasible.

A more desirable approach to sine wave frequency control is the use of a constant step duration of 250 µs, or less, and varying the number of samples used for sine wave construction. A sine-frequency dependent number of sine samples could be implemented by having a different length LUT for each sine frequency to be produced. For frequencies at one Hertz increments (over the range of 1–120 Hz), 120 LUT's with lengths ranging from 33 to 4000 values would need to be derived and stored in memory. This would be both an inefficient use of memory and a tedious data entry task. One alternative to creating multiple LUT's is to use a single LUT of reasonable length (257 values or less) in conjunction with the '332's TBLS instruction.

The argument of the TBLS instruction is a word-size operand in which the upper byte is used as a pointer to an entry in the LUT, while the lower byte determines how far to interpolate between the entry specified by the upper byte and the next entry in the table. By clever manipulation of the TBLS operand, one can effectively create "virtual" LUT's of varying size from a single data table of fixed length.

SINE WAVE GENERATION

The code that has been developed for this task consists of a main program which simply initializes some parameters, branches to two subroutines to configure the TPU and Queued Serial Module (QSM) subsystems of the '332, and loads the interrupt vectors for the ITC and Serial Communications Interface (SCI) Interrupt Service Routines (ISR's). This sine wave generation scheme is interrupt driven. As mentioned previously, the "master" timing channel outputs a 20 kHz square wave. This channel serves as the input to another TPU channel which has been initialized as an input transition counter. The ITC parameters are configured to generate an interrupt to the CPU after detecting every fifth rising edge of the timing channel. Since the timing

channel has a 50 µs period, ITC interrupts will occur at 250 µs intervals (one-half stator time constant). Each time the interrupt is serviced, the CPU will write a new high time to the three SPWM outputs. By making the "CPU-provided" high times be values of a sine function, one can produce a sine-modulated PWM signal. Thus, the ITC ISR , combined with the sine function LUT, is the "heart" of this algorithm. By producing three such PWM waveforms with 120° phase shifts to each other, three-phase power can then be realized.

Upon entering the ISR, the new high times for all three phases are written to their respective TPU PWM channels. These values were calculated during the previous ITC interrupt service. The calculations which occur in the ISR implement the equations shown in the Initial Algorithm section. The flowchart in Figure 3 illustrates the computations which generate the duty cycles that effect sine-modulated PWM waveforms. The entries in the LUT are 256 scaled values of the sine function. The numbers are scaled by 1024 so that all operations can be done using integer arithmetic. The actual table values have been halved to satisfy the pulse-width equations, while avoiding an extra division operation in the ISR.

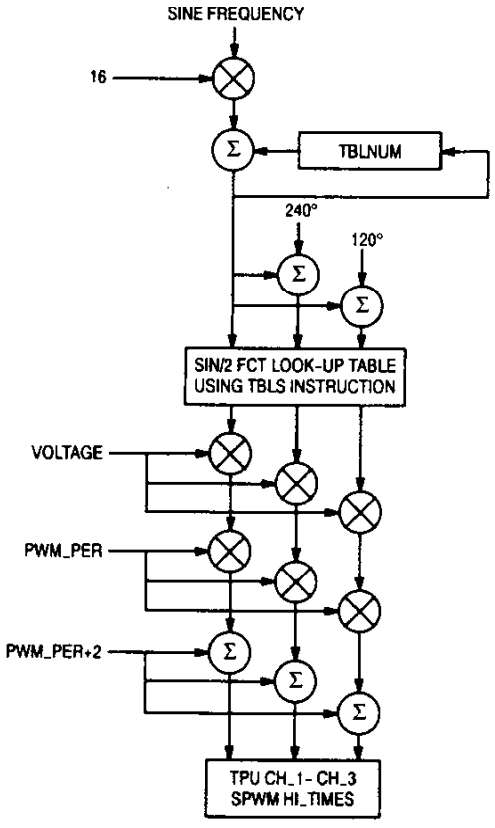

Figure 3. Three Phase Sine Wave Generation Algorithm

RESULTS, LIMITATIONS, AND FUTURE DIRECTIONS

A set of three-phase sine waves which were generated for a desired speed of 3600 RPM's are shown in Figure 4. These signals are indeed at full voltage and 120 Hz, as predicted by the speed characteristic of Figure 1. High quality sine wave voltages can be observed through direct integration (i.e. low-pass filtering) of the SPWM outputs. A simple user interface was developed to demonstrate speed control. The user could vary the voltage and frequency of the resulting sine waves by specifying "f" for faster, or "s" for slower. The protocol echoes the correct speed (in RPM's) of the demonstration motor to the screen. The sine wave frequency changes occur smoothly without any abrupt voltage changes.

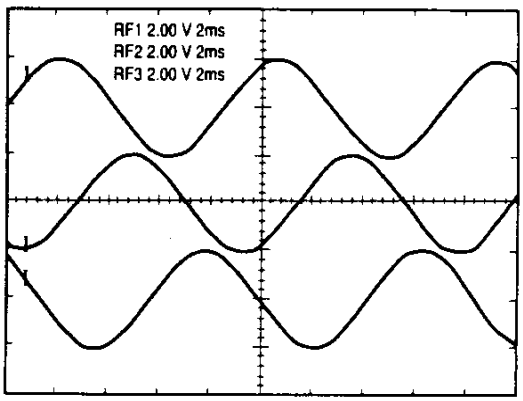

Figure 4. Three Phase Sine Waves

Hardware was designed and developed for AC motor control, and a single phase of the circuit was tested with the '332 microcontroller and an inductive load. The circuit employs the use of an analog dead time circuit, a high-voltage IC gate drive, and high speed IGBT's. A complete three phase inverter printed circuit board layout will be completed shortly.

High resolution three-phase sine waves were accurately generated; however, there were several limitations of this system. TPU microcode service time imposes a constraint on the minimum and maximum pulse-widths the SPWM primitive can produce. Approximately 1900 ns and 1300 ns were the minimum high time and low time, respectively. This limits the maximum duty cycle to approximately 94%. In order to prevent the possibility of inaccurate high times, which could occur as a result of TPU latency, each SPWM TPU channel was offset from the previous channel by 2.4 µs.

The voltage resolution of the sine waves is determined by the PWM frequency chosen. For a 20 kHz PWM frequency, the period is 50 µs or 210 counts of the TPU time base (240 ns/count). Since PWM duty cycle corresponds to sine wave voltage amplitude, the voltage resolution is limited to 210 levels. Thus, voltage resolution is approximately 0.5% of the full-scale voltage. Higher resolution would provide

better low level sine waves at very low motor speeds. However, the bandwidth of the entire inverter must be higher than 4 MHz in order to utilize higher resolution.

Future improvements to the sine wave algorithm may be implemented in custom microcode. The use of two adjacent channels per phase allows the rising edge to be updated prior to the occurrence of the falling edge, and vise versa. This will permit minimum high and low times of 240 ns (one timer count). Centering the pulses with respect to each other may also be accomplished in microcode. This would allow the high frequency ripple current to be further reduced, particularly important in low inductance motors. Coherent updating of all TPU variables can be implemented at the microcode level. Lastly, simple open loop, constant V/F control could be replaced by more complex vector control.

This algorithm for generating variable voltage and frequency sine waves can be downloaded from the Motorola MCU Freeware Line by dialing (512) 891-FREE (3733). The Directory is MCU332 and the file is acinddriv.asm.

ACKNOWLEDGEMENTS

We would like to thank Dr. Allan Plunkett for his contributions to the final algorithm. His inputs were essential for a thorough understanding of the details involved in sine wave PWM generation. We also thank Peter Pinewski for his assistance throughout the development of the '332 code.

GLOSSARY OF ACRONYMS

CPU	— Central Processor Unit
IGBT	— Insulated Gate Bipolar Transistor
ISR	— Interrupt Service Routine
LUT	— Look Up Table
MCU	— Microcontroller Unit
PWM	— Pulse Width Modulation
SPWM	— Synchronous PWM
TPU	— Time Processor Unit
V/f	— Constant Voltage over Frequency

```
*******************************************************************************
*     MC68332 AC INDUCTION MOTOR THREE PHASE SINEWAVE GENERATION PROGRAM      *
*******************************************************************************

************************************************************
** Jeff Baum                                              **
** Discrete Systems and Applications Engineering          **
** Motorola, Inc.                                         **
** 5005 E. McDowell                                       **
** Phoenix, Arizona 85008                                 **
** 602-952-4360  MD 56-116                                **
************************************************************

* This code realizes a novel method for generating high-quality three-phase
* sinewaves intended for driving the electronics which in-turn drive a
* 3/4 HP AC Induction Motor.  This motor has a rating of 1800 RPM at 230 V
* and 60 Hz.  The simple user interface only recognizes the inputs "F" or "S"
* for faster or slower, respectively (case insensitive). The number of RPM's
* that the resultant sinewaves will produce for this specific motor will be
* echoed to the screen. Communications is via the 332 QSM (SCI).

* All EQUate statements using only 6 hex digit addressing will require the
* the use of a ".L" extension when using such labels as an operand.

*******************************************************************************************************

*TPU module registers

TMCR       EQU       $FFFE00
TICR       EQU       $FFFE08
CIER       EQU       $FFFE0A
CFSR2      EQU       $FFFE10
CFSR3      EQU       $FFFE12
CPR1       EQU       $FFFE1E
HSQR1      EQU       $FFFE16
HSRR1      EQU       $FFFE1A
CISR       EQU       $FFFE20

*QSM/SCI control registers

QMCR       EQU       $FFFC00
QILVR      EQU       $FFFC04
SCCR0      EQU       $FFFC08
SCCR1      EQU       $FFFC0A
SCSR       EQU       $FFFC0C
SCDR       EQU       $FFFC0E

* TPU parameter registers

*master SPWM timing channel used for PWM alignment and sine function update

CHLCTL_0   EQU       $FFFF00
HIPER_0    EQU       $FFFF04
LNKREF1_0  EQU       $FFFF08
DELAY_0    EQU       $FFFF0A

*TP1-TP3 are phase A, phase B, phase C PWM outputs

CHLCTL_1   EQU       $FFFF10
HITIME_1   EQU       $FFFF14
DELAY_1    EQU       $FFFF16
REFADS_1   EQU       $FFFF18

CHLCTL_2   EQU       $FFFF20
```

```
HITIME_2  EQU       $FFFF24
DELAY_2   EQU       $FFFF26
REFADS_2  EQU       $FFFF28

CHLCTL_3  EQU       $FFFF30
HITIME_3  EQU       $FFFF34
DELAY_3   EQU       $FFFF36
REFADS_3  EQU       $FFFF38

*TP4 is ITC-input transition counter channel used for counting transitions on TP0 for
*update interval

CHLCTL_4  EQU       $FFFF40
BANKAD_4  EQU       $FFFF42
MTCOUNTS_4EQU       $FFFF44

*ISR-interrupt service routine location vectors

VOFF2     EQU       $000110           *IC ISR vector
VOFF3     EQU       $000140           *SCI ISR vector

*main program labels

V         EQU       $9000             *voltage value
F         EQU       $9002             *sine frequency
PER       EQU       $9004             *sine period
PER2      EQU       $9006             *one-half sine period
DIG1      EQU       $9008             *storage for SCI RPM digits to display
DIG2      EQU       $900A
DIG3      EQU       $900C
DIG4      EQU       $900E
TBLNUM    EQU       $9010

*****************************************************************************************

*MAIN PROGRAM

          ORG       $3000             *put code at address $3000
          MOVE.W    #$2000,SR         *supervisor mode, int mask 5
          MOVE.L    #$5500,VOFF2.L    *load ISR vector starting address
          MOVE.L    #$5800,VOFF3.L    *load ISR vector starting address

          BSR       SCI_CNFG          *configure SCI

          MOVE.L    #PWTBL,A0         *load starting address of lut
          CLR.W     V.L               *initialize voltage
          CLR.W     F.L               *initialize frequency
          CLR.W     TBLNUM.L
          MOVE.W    #$00d2,PER.L      *initialize sine freq.
          AND.W     #$FFEF,CISR.L     *clear TP4 interrupt flag

          MOVE.W    #$0069,D1         *initialize high times to 50% duty cycle
          MOVE.W    #$0069,D2
          MOVE.W    #$0069,D3

          BSR       TPU_CONFG         *configure TPU and initialize TPU channel
*                                      parameters
          BRA       *

*****************************************************************************************
```

```
TPU_CONFG MOVE.W    #$00CF,TMCR.L       *divide by 4 clock,T2CG,IARB is F
          MOVE.W    #$0640,TICR.L       *TPU IRQ level 6,vector base $40
          MOVE.W    #$0010,CIER.L       *enable TP4 interrupts
          MOVE.W    #$000A,CFSR2.L      *TP4 ITC
          MOVE.W    #$7777,CFSR3.L      *TP0,TP1,TP2, TP3 are SPWM
          MOVE.W    #$0156,HSQR1.L      *sequence bits for modes

*         TP0 RAM   PWM phase 0
          MOVE.W    #$0092,CHLCTL_0.L   *TCR1, force pin low
          MOVE.L    #$006900d2,HIPER_0.L*50% duty cycle, 20 kHz
          MOVE.W    #$1300,LNKREF1_0.L
          MOVE.W    #$0000,DELAY_0.L    *no delay

*         TP1 RAM   PWM phase A
          MOVE.W    #$0092,CHLCTL_1.L   *TCR1, force pin low
          MOVE.W    #$0069,HITIME_1.L   *50% duty cycle
          MOVE.W    #$000A,DELAY_1.L    *2.4 µs delay
          MOVE.W    #$0200,REFADS_1.L   *nextrise/lastrise

*         TP2 RAM   PWM phase B
          MOVE.W    #$0092,CHLCTL_2.L   *TCR1, force pin low
          MOVE.W    #$0069,HITIME_2.L   *50% duty cycle
          MOVE.W    #$0014,DELAY_2.L    *4.8 µs delay
          MOVE.W    #$0200,REFADS_2.L   *nextrise/lastrise

*         TP3 RAM   PWM phase C
          MOVE.W    #$0092,CHLCTL_3.L   *TCR1, force pin low
          MOVE.W    #$0069,HITIME_3.L   *50% duty cycle
          MOVE.W    #$001E,DELAY_3.L    *7.2 µs delay
          MOVE.W    #$0200,REFADS_3.L   *nextrise/lastrise

*         TP4 RAM   IC synchronous update interrupt
          MOVE.W    #$0007,CHLCTL_4.L   *detects rising edges
          MOVE.W    #$000E,BANKAD_4.L   *non-existent address
          MOVE.L    #$00050000,MTCOUNTS_4.L       *MAXCOUNT=5, every fifth edge

*TPU INITIALIZATION

          MOVE.W    #$01AA,HSRR1.L      *channel initialization
          MOVE.W    #$01FF,CPR1.L       *channel priority
          RTS

*****************************************************************************************

*SCI configuration subroutine

SCI_CNFG  MOVE.W    #$0081,QMCR.L       *sup mode,iarb 1
          MOVE.W    #$0150,QILVR.L      *intlvl 1, vector $50
          MOVE.W    #$0037,SCCR0.L      *9600 baud
          MOVE.W    #$002C,SCCR1.L      *RIE, TE, RE
          RTS

*****************************************************************************************
```

```
*ITC ISR-interrupt service routine

          ORG       $5500                *locate ITC ISR
          MOVE.W    D1,HITIME_1.L        *write new high times to all
          MOVE.W    D2,HITIME_2.L        *three PWM phases
          MOVE.W    D3,HITIME_3.L

          MOVE.W    PER.L,PER2.L         *divide PER by 2 and store in PER2
          LSR.W     PER2.L
          MOVE.W    TBLNUM.L,D1          *load previous TBLNUM in D1
          MOVE.W    F.L,D2               *put sine freq in D2
          MULU.W    #$10,D2              *multiply sine frequency by 16 (min jmp)
          ADD.W     D2,D1                *add delta TBLNUM to old TBLNUM
          MOVE.W    D1,TBLNUM.L          *save new TBLNUM
          MOVE.W    D1,D2                *copy new TBLNUM to D2
          ADD.W     #$AAAA,D2            *add 2/3 phase shift (+240 or -120 degrees)
          MOVE.W    D1,D3                *copy new TBLNUM to D3
          ADD.W     #$5555,D3            *add 1/3 phase shift (+120 or -240 degrees)

          DC.W      $F810,$1940,$F810,$2940,$F810,$3940                *TBLS statements

          MULS.W    V.L,D1               *multiply sine samples by
          MULS.W    V.L,D2               *Vx1024 values in V LUT
          MULS.W    V.L,D3

          ASR.L     #8,D1                *divide V*sin (f*t +phi)/2 values by 1024
          ASR.L     #2,D1
          ASR.L     #8,D2
          ASR.L     #2,D2
          ASR.L     #8,D3
          ASR.L     #2,D3

          MULS.W    PER.L,D1             *multiply sine samples by
          MULS.W    PER.L,D2             *PER values
          MULS.W    PER.L,D3

          ASR.L     #8,D1                *divide by 1024
          ASR.L     #2,D1
          ASR.L     #8,D2
          ASR.L     #2,D2
          ASR.L     #8,D3
          ASR.L     #2,D3

          ADD.W     PER2.L,D1            *add period/2 to all three phases'
          ADD.W     PER2.L,D2            *new pulse width calculations
          ADD.W     PER2.L,D3

          AND.W     #$FFEF,CISR.L        *clear TP4 interrupt flag
          RTE                            *return from interrupt

*****************************************************************************************
```

```
*QSM/SCI ISR-interrupt service routine

          ORG       $5800
          MOVE.W    SCSR.L,D7           *read SCSR
          MOVE.W    SCDR.L,D4           *receive character
          CMPI.B    #$46,D4             *check for 'F'aster
          BEQ       INCRPM
          CMPI.B    #$66,D4             *check for 'f'aster
          BEQ       INCRPM
          CMPI.B    #$53,D4             *check for 'S'lower
          BEQ       DECRPM
          CMPI.B    #$73,D4             *check for 's'lower
          BEQ       DECRPM
          BRA       ENDSVC              *illegal character, no changes

INCRPM    CMPI.W    #$78,F.L            *do not exceed 120 Hz
          BGE       COMPVOLT            *compute voltage
          ADDQ.W    #$1,F.L             *increment frequency by 1 Hz
          BRA       COMPVOLT            *compute voltage

DECRPM    CMPI.W    #$0,F.L             *do not go negative in freq
          BLE       COMPVOLT            *compute voltage
          SUBQ.W    #$1,F.L             *decrement frequency by 1 Hz


COMPVOLT  CMPI.W    #$35,F.L            *check if greater or equal to 60 Hz
          BGE       CONSTVOLT           *maintain full voltage, do not exceed $400
          MOVE.W    F.L,D5
          MULU.W    #$11,D5             *volts=freq*17
          MOVE.W    D5,V.L              *store new voltage
          BRA       ENDSVC              *go calculate rpm's

CONSTVOLT MOVE.W    #$385,V.L           *max voltage=$400 (1024)


ENDSVC    MOVE.W    F.L,D5
          MULU.W    #$1E,D5             *rpm=30*frequency

XMIT1     BTST      #$0,SCSR.L          *poll bit 8 of SCSR until clear, ready to XMIT
          BEQ       XMIT1
          MOVE.W    #$0D,SCDR.L         *output carriage return to screen

          MOVE.W    #$0030,DIG1.L       *refresh digits to blank characters
          MOVE.W    #$0030,DIG2.L
          MOVE.W    #$0030,DIG3.L
          MOVE.W    #$0030,DIG4.L

          MOVE.L    #DIG4,A1            *point to address to store digits
          ADDQ.L    #$2,A1

BCDCONV   DIVU.W    #$A,D5              *divide rpm hex by 10
          MOVE.W    D5,D6               *copy D5low (quotient) to D6
          SWAP      D5                  *put D5up (remainder) in D5low
          ADDI.B    #$30,D5             *add $30 to remainder to get ASCII
          MOVE.W    D5,-(A1)
          CLR.L     D5                  *clear entire D5
          MOVE.W    D6,D5               *restore previous quotient to D5
          CMPI.W    #$0,D5              *check for quotient zero, done converting
          BNE       BCDCONV             *loop to next conversion
          LEA       DIG1.L,A1           *point to first digit to transmit
          MOVE.W    #3,D7               *reset transmit digit counter
XMIT9     BTST      #$0,SCSR.L  *bit 8 of SCSR    *poll transmit complete flag
          BEQ       XMIT9
          MOVE.W    (A1)+,SCDR.L                  *write SCDR to transmit digit
          DBF       D7,XMIT9            *transmit next digit
          RTE
```

```
*****************************************************************************************
*LOOK-UP-TABLE (LUT)

          ORG       $6000
PWTBL     DC.W      $0000,$000d,$0019,$0026,$0032,$003f,$004b,$0058
          DC.W      $0064,$0070,$007c,$0089,$0095,$00a1,$00ac,$00b8
          DC.W      $00c4,$00cf,$00db,$00e6,$00f1,$00fc,$0107,$0112
          DC.W      $011c,$0127,$0131,$013b,$0145,$014e,$0158,$0161
          DC.W      $016a,$0173,$017b,$0184,$018c,$0194,$019b,$01a3
          DC.W      $01aa,$01b1,$01b7,$01bd,$01c4,$01c9,$01cf,$01d4
          DC.W      $01d9,$01de,$01e2,$01e6,$01ea,$01ed,$01f1,$01f4
          DC.W      $01f6,$01f8,$01fa,$01fc,$01fe,$01ff,$01ff,$0200
          DC.W      $0200,$0200,$01ff,$01ff,$01fe,$01fc,$01fa,$01f8
          DC.W      $01f6,$01f4,$01f1,$01ed,$01ea,$01e6,$01e2,$01de
          DC.W      $01d9,$01d4,$01cf,$01c9,$01c4,$01bd,$01b7,$01b1
          DC.W      $01aa,$01a3,$019b,$0194,$018c,$0184,$017b,$0173
          DC.W      $016a,$0161,$0158,$014e,$0145,$013b,$0131,$0127
          DC.W      $011c,$0112,$0107,$00fc,$00f1,$00e6,$00db,$00cf
          DC.W      $00c4,$00b8,$00ac,$00a1,$0095,$0089,$007c,$0070
          DC.W      $0064,$0058,$004b,$003f,$0032,$0026,$0019,$000d
          DC.W      $0000,$fff3,$ffe7,$ffda,$ffce,$ffc1,$ffb5,$ffa8
          DC.W      $ff9c,$ff90,$ff84,$ff77,$ff6b,$ff5f,$ff54,$ff48
          DC.W      $ff3c,$ff31,$ff25,$ff1a,$ff0f,$ff04,$fef9,$feee
          DC.W      $fee4,$fed9,$fecf,$fec5,$febb,$feb2,$fea8,$fe9f
          DC.W      $fe96,$fe8d,$fe85,$fe7c,$fe74,$fe6c,$fe65,$fe5d
          DC.W      $fe56,$fe4f,$fe49,$fe43,$fe3c,$fe37,$fe31,$fe2c
          DC.W      $fe27,$fe22,$fe1e,$fe1a,$fe16,$fe13,$fe0f,$fe0c
          DC.W      $fe0a,$fe08,$fe06,$fe04,$fe02,$fe01,$fe01,$fe00
          DC.W      $fe00,$fe00,$fe01,$fe01,$fe02,$fe04,$fe06,$fe08
          DC.W      $fe0a,$fe0c,$fe0f,$fe13,$fe16,$fe1a,$fe1e,$fe22
          DC.W      $fe27,$fe2c,$fe31,$fe37,$fe3c,$fe43,$fe49,$fe4f
          DC.W      $fe56,$fe5d,$fe65,$fe6c,$fe74,$fe7c,$fe85,$fe8d
          DC.W      $fe96,$fe9f,$fea8,$feb2,$febb,$fec5,$fecf,$fed9
          DC.W      $fee4,$feee,$fef9,$ff04,$ff0f,$ff1a,$ff25,$ff31
          DC.W      $ff3c,$ff48,$ff54,$ff5f,$ff6b,$ff77,$ff84,$ff90
          DC.W      $ff9c,$ffa8,$ffb5,$ffc1,$ffce,$ffda,$ffe7,$fff3
          DC.W      $0000

*****************************************************************************************
```