CYPRESS SEMICONDUCTOR

CY7C601A

32-Bit RISC Processor

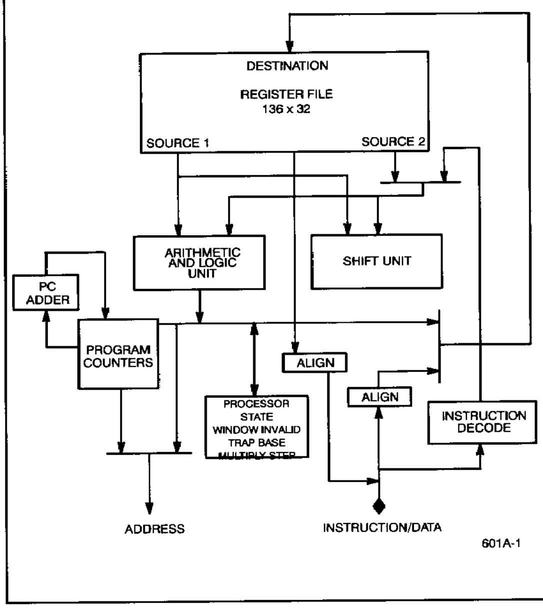
Features

- **Reduced Instruction Set Computer (RISC) Architecture**
  - **— Simple format instructions**
  - **— Most instructions execute in a single cycle**
- **Very high performance**
  - **— 25-, 33-, and 40-MHz clock speeds yield 18, 24, and 29 MIPS sustained throughput respectively**
  - **— Very fast interrupt response**
  - **— Four-stage pipeline**
- **Large windowed register file**
  - **— 136 general-purpose 32-bit registers**
  - **— Registers can be used as eight windows of 24 registers each for low procedure overhead**
  - **— Registers can also be used as register banks for fast context switching**
- **Multiprocessing support**
- **Large virtual address space**
  - **— 32-bit virtual address bus**
  - **— 8-bit address space identifier bus**
- **Hardware pipeline interlocks**
- **Multitasking support**
  - **— User/supervisor modes**
  - **— Privileged instructions**
- **Artificial intelligence support**
- **High-performance coprocessor interface for user-defined coprocessor**
- **FPU interface allows concurrent execution of floating-point instructions**
- **0.8-micron CMOS technology**
- **207-pin grid array package**

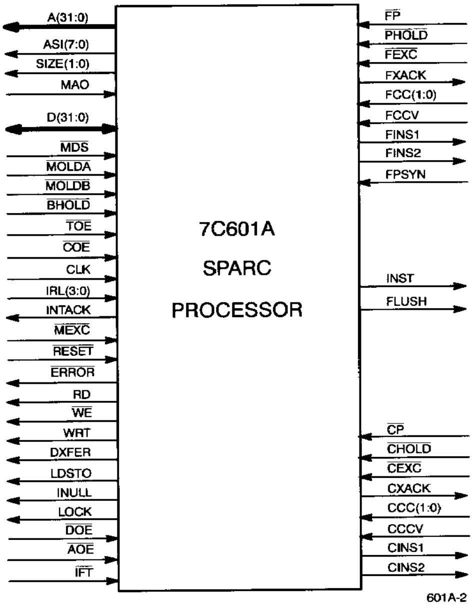
Overview

The CY7C601A integer unit is a high-speed CMOS implementation of the SPARC® 32-bit RISC processor. The RISC architecture makes possible the creation of a processor that can execute instructions at a rate of one instruction per processorclock. The CY7C601A supports a tightly coupled floating-point interface and coprocessor interface that allows concurrent execution of floating-point, coprocessor, and integer instructions.

Logic Block Diagram



Pin Configuration



Selection Guide

	7C601A-40	7C601A-33	7C601A-25
Maximum Operating Current (mA)	650	600	600

SPARC is a registered trademark of SPARC International, Inc.

Overview (continued)

The CY7C601A SPARC processor provides the following features:

Simple instruction format. All instructions are 32-bits wide and aligned on 32-bit boundaries in memory. The three basic instruction formats feature uniform placement of opcode and address fields.

Register intensive architecture. Most instructions operate on either two registers or one register and a constant, and place the result in a third register. Only load and store instructions access off-chip memory.

Large windowed register file. The processor has 136 on-chip 32-bit registers configured as eight overlapping sets of 24 registers each and eight global registers. This scheme allows compilers to cache local values across subroutine calls and provides a register-based parameter passing mechanism.

Delayed control transfer. The processor always fetches the next instruction after a control transfer, and either executes it or annuls it depending on the state of a bit in the control transfer instruction. This feature allows compilers to rearrange code to place a useful instruction after a delayed control transfer and thereby take better advantage of the processor pipeline.

Concurrent floating-point. Floating-point instructions can execute concurrently with each other and with non-floating-point instructions.

Fast interrupt response. Interrupt inputs are sampled on every clock cycle and can be acknowledged in one to three cycles. The first instruction of an interrupt service routine can be executed within 6 to 8 cycles of receiving the interrupt request.

The 7C600 Family

The SPARC processor family consists of a CY7C601A integer unit to perform all non-floating-point operations and a CY7C602A floating-point unit (FPU) to perform floating-point arithmetic concurrent with the CY7C601A. Support is also provided for a second generic coprocessor interface. The CY7C601A communicates with external memory via a 32-bit address bus and a 32-bit data/instruction bus. In typical data processing applications, the CY7C601A and CY7C602A are combined with a high-performance CY7C604A memory management unit and cache controller and a cache memory implemented with CY7C157A 16-Kbyte x 16 cache RAMS. In many dedicated controller applications the CY7C601A can function by itself with only high-speed local memory.

Coprocessor Interface

The CY7C601A is the basic processing engine that executes all of the instruction set except for floating-point operations. The CY7C601A and CY7C602A operate concurrently. The CY7C602A recognizes floating-point instructions and places them in a queue while the CY7C601A continues to execute non-floating-point instructions. If the CY7C602A encounters an instruction that will not fit in its queue, the CY7C602A holds the CY7C601A until the instruction can be stored. The CY7C602A contains its own set of registers on which it operates. The contents of these registers are transferred to and from external memory under control of the CY7C601A via floating-point load/store instructions. Processor interlock hardware hides floating-point concurrency from the compiler or assembly language programmer. A program containing floating-point computations generates the same results as if instructions were executed sequentially.

Registers

The CY7C601A contains a large 136 x 32 triple-port register file which is divided into 8 windows, each with 24 working registers and each having access to the same 8 global registers. A current window pointer (CWP) field in the processor state register keeps track of which window is currently active. The CWP is decremented when the processor calls a subroutine and is incremented when the processor returns. The registers in each window are divided into ins, outs, and locals. The eight global registers are shared by all windows and appear as registers 0–7 in each window. Registers 8–15 serve as outs, registers 16–23 as locals, and 24–31 serve as ins. Each window shares its ins and outs with adjacent windows. The outs of the previous window are the ins of the current window, and the outs of the current window are the ins of the next window. The globals are equally available to all windows and the locals are unique to each window. The windows are joined together in a circular stack where the outs of window 7 are the ins of window 0.

Multitasking Support

The CY7C601A supports a multitasking operating system by providing user and supervisor modes. Some instructions are privileged and can only be executed while the processor is in supervisor mode. Changing from user to supervisor mode requires taking a hardware interrupt or executing a trap instruction.

Interrupts and Traps

The CY7C601A supports both asynchronous traps (interrupts) and synchronous traps (error conditions and trap instructions). Traps transfer control to an offset within a table. The base address of the table is specified by a trap base register and the offset is a function of the trap type. Traps are taken before the current instruction causes any changes visible to the programmer and can therefore be considered to occur between instructions.

Instruction Set Summary

Instructions fall into five basic categories as follows:

1. **Load and store instructions.** Load and store are the only instructions which access external memory. They use two CY7C601A registers or one CY7C601A register and a signed immediate value to generate the memory address. The instruction destination field specifies either an CY7C601A register, a CY7C602A register, or a coprocessor register as the destination for a load or source for a store. Integer load and store instructions support 8-, 16-, 32-, and 64-bit transfers while floating-point and coprocessor instructions support 32- and 64-bit accesses.

2. **Arithmetic/logical/shift.** These instructions compute a result that is a function of two source operands and write the result into a destination register or discard it. They perform arithmetic, tagged arithmetic, logical, and shift operations. An instruction SETHI, useful in creating 32-bit constants in two instructions, writes a 22-bit constant into the high order bits of a register and zeroes the remaining bits. The contents of any register can be shifted left or right any number of bits in one clock cycle as specified by a register or the instruction itself. The tagged instructions are useful in artificial intelligence applications.

3. **Control transfer.** Control transfer instructions include jumps, calls, traps and branches. Control transfer is usually delayed so that the instruction immediately following the control transfer (called the delay instruction) is executed before control is transferred to the target location. The delay instruction is always

Instruction Set Summary (continued)

fetched, however, a bit in the control transfer instruction can cause the delay instruction to be nullified if the branch is not taken. This flexibility increases the likelihood that a useful instruction can be placed after the control transfer thereby filling an otherwise unused hole in the processors pipeline. Branch and call instructions use program counter relative displacements. A jump and link instruction uses a register indirect displacement computing its target address as either the sum of two registers or the sum of a register and a 13-bit signed immediate value. The branch instruction provides a displacement plus or minus 8 megabytes, and the call instructions 30-bit displacement allows transfer to almost any address.

4. **Read/write control registers**. The processor provides special instructions to read and write the contents of the various control registers within the machine. These registers include the multiply step register, processor state register, window invalid mask register, and trap base register.

5. **Floating-point/coprocessor instructions**. These instructions include all floating-point conversion and arithmetic operations as well as future coprocessor instructions. These instructions involve operations only on the contents of the register file internal to the CY7C602A or coprocessor.

The instruction set of the processor is summarized in *Table 1*.

Registers

The following sections provide an overview of the CY7C601A registers. The CY7C601A has two types of registers; working registers (r registers), and control registers. The r registers provide storage for processes, and the control registers keep track of and control the state of the CY7C601A.

r Registers. The r registers (*Figure 1*) consist of eight 32-bit global registers, and 8 windows, each having twenty-four 32-bit registers. Each two adjacent windows are overlapped in eight registers. This results in a total of 136 32-bit general purpose registers on the chip.

CY7C601A Control Registers. The CY7C601A control registers contain various addresses and pointers used by the system to control its internal state. They include the program counters (PC and nPC), the processor state register (PSR), the window invalid mask register (WIM), the trap base register (TBR), and the Y register. The following paragraphs briefly describe each:

Processor Status Register (PSR). The processor status register contains fields that describe and control the state of the CY7C601A (see *Figure 2*).

IU Implementation and IU Version Numbers (IMPL field, PSR<31:28>; VER field, PSR<27:24>). These are read-only fields in the PSR. The version number and the implementation number are each set to "0001".

Integer Condition Codes (PSR<23:20>). The integer condition codes consist of four flags: negative, zero, overflow, and carry. These flags are set by the conditions occurring during integer logic and arithmetic operations.

Enable Coprocessor (EC bit, PSR<13>). This bit is used to enable the coprocessor. If a coprocessor operation (CPop) is encountered and the EC bit is cleared (i.e., coprocessor disabled), a coprocessor disabled trap is generated.

Enable Floating Point Unit (EF bit, PSR<12>). This bit is used to enable the floating point unit. If a floating point operation (FPop) is encountered and the EF bit is cleared (i.e., FPU disabled), a floating point disabled trap is generated.

Processor Interrupt Level (PIL field, PSR<11:8>). This four bit field sets the CY7C601A interrupt level. The CY7C601A will only acknowledge interrupts greater than the level indicated by the PIL field. Bit 11 is the MSB; bit 8 is the LSB.

Supervisor Mode (S bit, PSR<7>). S = 1 indicates that the CY7C601A is in supervisor mode. Supervisor mode can only be entered by a software or hardware trap.

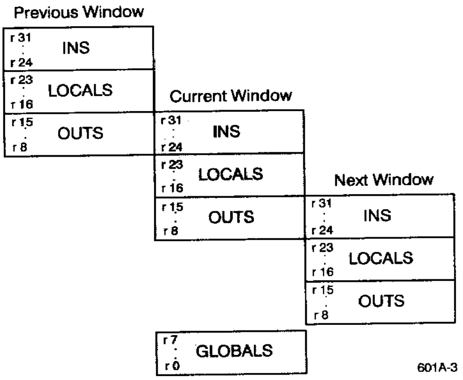


Figure 1. Register Windows

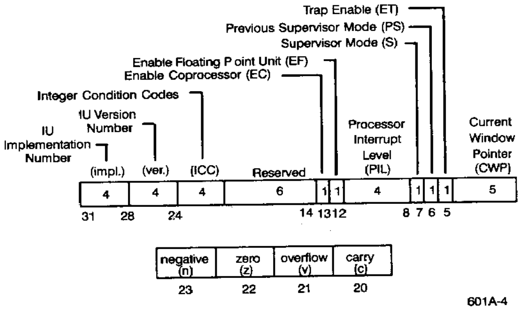


Figure 2. Processor State Register

Table 1. Instruction Set Summary

Inputs		Operation		Cycles
Load and Store Instructions	LDSB(LDSBA*)	Load Signed Byte	(from Alternate Space)	2
	LDSH(LDSHA*)	Load Signed Halfword	(from Alternate Space)	2
	LDUB(LDUBA*)	Load Unsigned Byte	(from Alternate Space)	2
	LDUH(LDUHA*)	Load Unsigned Halfword	(from Alternate Space)	2
	LD(LDA*)	Load Word	(from Alternate Space)	2
	LDD(LDDA*)	Load Doubleword	(from Alternate Space)	3
	LDF	Load Floating Point		2
	LDDF	Load Double Floating Point		3
	LDFSR	Load Floating Point State Register		2
	LDC	LoadCoprocessor		2
	LDDC	Load Double Coprocessor		3
	LDCSR	Load Coprocessor State Register		2
	STB(STBA*)	Store Byte	(into Alternate Space)	3
	STH(STHA*)	Store Halfword	(into Alternate Space)	3
	ST(STA*)	Store Word	(into Alternate Space)	3
	STD(STDA*)	Store Doubleword	(into Alternate Space)	4
	STF	Store Floating Point		3
	STDF	Store Double Floating Point		4
	STFSR	Store Floating Point State Register		3
	STDFQ*	Store Double Floating Point Queue		4
	STC	StoreCoprocessor		3
	STDC	Store Double Coprocessor		4
	STCSR	Store Coprocessor State Register		3
	STDCO*	Store Double Coprocessor Queue		4
	LDSTUB(LDSTUBA*)	Atomic Load/Store Unsigned Byte	(in Alternate Space)	4
	SWAP(SWAPA*)	Swap r Register with Memory	(in Alternate Space)	4
Arithmetic/Logical/Shift	ADD(ADDcc)	Add	(modify icc)	1
	ADDX(ADDXcc)	Add with Carry	(modify icc)	1
	TADDcc(TADDccTV)	Tagged Add and modify icc	(and Trap on overflow)	1
	SUB(SUBcc)	Subtract	(modify icc)	1
	SUBX(SUBXcc)	Subtract with Carry	(modify icc)	1
	TSUBcc(TSUBccTV)	Tagged Subtract and modify icc	(and Trap on overflow)	1
	MULScc	Multiply Step and modify icc		1
	AND(ANDcc)	And	(and modify icc)	1
	ANDN(ANDNcc)	And Not	(and modify icc)	1
	OR(ORcc)	Inclusive Or	(and modify icc)	1
	ORN(ORNcc)	Inclusive Or Not	(and modify icc)	1
	XOR(XORcc)	Exclusive Or	(and modify icc)	1
	XNOR(XNORcc)	Exclusive Nor	(and modify icc)	1
	SLL	Shift Left Logical		1
	SRL	Shift Right Logical		1
	SRA	Shift Right Arithmetic		1
	SETHI	Set High 22 Bits of r Register		1
	SAVE	Save Caller's window		1
	RESTORE	Restore Caller's window		1
Control Transfer	Bicc	Branch on Integer Condition Codes		1**
	FBicc	Branch on Floating Point Condition Codes		1**
	CBccc	Branch on Coprocessor Condition Codes		1**
	CALL	Call		1**
	JMPL	Jump and Link		2**
	RETT	Return from Trap		2**
	Ticc	Trap on Integer Condition Codes		1 (4 if Taken)

Table 1. Instruction Set Summary (continued)

Inputs		Operation	Cycles
Read/Write Control Registers	RDY RDPSR RDWIM RDTBR	Read Y Register Read Processor State Register Read Window Invalid Mask Read Trap Base Register	1 1 1 1
	WRY WRPSR* WRWIM* WRTBR*	Write Y Register Write Processor State Register Write Window Invalid Mask Write Trap Base Register	1 1 1 1
	UNIMP	UnimplementedInstruction	1
	IFLUSH	Instruction Cache Flush	1
FP (CP) Ops	FPop CPop	Floating Point Unit Operations CoprocessorOperations	1 to Launch 1 to Launch

* Privileged instruction.

** Assuming delay slot is filled with useful instruction.

Processor Status Register (continued)

Previous Supervisor Mode (PS bit, PSR<6>). This bit indicates the state of the supervisor bit before the most recent trap.

Trap Enable (ET bit, PSR<5>). This bit enables or disables the CY7C601A traps. This bit is automatically set to 0 (traps disabled) upon entering a trap. When ET = 0, all asynchronous traps are ignored. If a synchronous trap occurs when ET = 0, the CY7C601A enters error mode.

Current Window Pointer (CWP field, PSR<4:0>). The r registers are addressed by the current window pointer (CWP), a field of the processor status register (PSR), which points to the 24 active local registers. It is incremented by a RESTORE instruction and decremented by a SAVE instruction. Note that the globals are always accessible regardless of the CWP. In the overlapping configuration each window shares its ins and outs with adjacent windows. The outs from a previous window (CWP +1) are the ins of the current window, and the outs from the current window are the ins for the next window (CWP -1). In both the windowed and register bank configurations globals are equally available and the locals are unique to each window.

Program Counters (PC and nPC). The program counter (PC) holds the address of the instruction being executed, and the next program counter (nPC) holds the address of the next instruction to be executed.

Trap Base Register (TBR). The trap base register contains the base address of the trap table and a field that provides a pointer into the trap table.

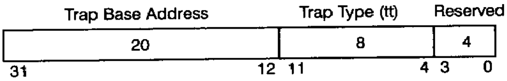


Figure 3. Trap Base Register

Window Invalid Mask Register (WIM). The window invalid mask register determines which windows are valid and which window accesses cause window_overflow and window_underflow traps.

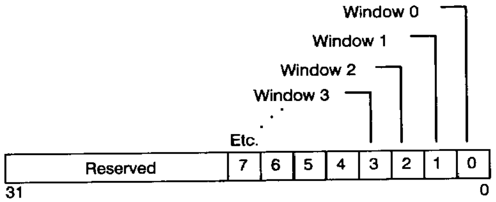


Figure 4. Window Invalid Mask

Y register. The Y register is used to hold the partial product during execution of the multiply-step instruction (MULSCC).

Pin Description

The integer unit's external signals fall into three categories: (1) memory subsystem interface signals, (2) floating-point unit/coprocessor interface signals, and (3) miscellaneous I/O signals. These are described in the following sections. Paragraphsafter the tables describe each signal. Signals that are active LOW are markedwith an overcomer; all others are active HIGH. Forexample, $\overline{\text{WE}}$ is active LOW, while RD is active HIGH.

Memory Subsystem Interface Signals

A[31:0]. These 32 bits are the addresses of instructions or data and they are sent out "unlatched" by the integer unit. Assertion of the MAO signal during a cache miss will force the integer unit to put the previous (missed) address on the address bus. A[31:0] pins are three-stated if the $\overline{\text{AOE}}$ or $\overline{\text{TOE}}$ signal is deasserted.

ASI[7:0]. These 8 bits are the address space identifier for an instruction or data access to the memory. ASI[7:0] are sent out "unlatched" by the integer unit. The value on these pins during any given cycle is the address space identifier corresponding to the memory address on the A[31:0] pins at that cycle. Assertion of the MAO signal during a cache miss will force the integer unit to put the previous address space identifier on the ASI[7:0] pins. ASI[7:0] pins are three-stated if the $\overline{\text{AOE}}$ or $\overline{\text{TOE}}$ signal is deasserted. Normally, the encoding of the ASI bits is as shown in *Table 2.* The remaining codes are software generated.

Table 2. ASI Bit Assignment

Address Space Identifier (ASI)	Address Space
00001000	User Instruction
00001010	User Data
00001001	Supervisor Instruction
00001011	Supervisor Data

D[31:0]. D[31:0] is the bidirectional data bus to and from the integer unit. The data bus is driven by the integer unit during the execution of integer store instructions and the store cycle of atomic load/store instructions. Similarly, the data bus is driven by the floating-point unit only during the execution of floating-point store instructions. The store data is sent out unlatched and must be latched externally before it is used. Once latched, store data is valid during the second data cycle of a store single access, the second and third data cycle of a store double access, and the third data cycle of an atomic load store access. The alignment for load and store instructions is done inside the processor. A double word is aligned on an 8-byte boundary, a word is aligned on a 4-byte boundary, and a half word is aligned on a 2-byte boundary. D(31) corresponds to the most significant bit of the least significant byte of the 32-bit word. If a double word, word, or half word load or store instruction generates an improperly aligned address, a memory address not aligned trap will occur. Instructions and operands are always expected to be fetched from a 32-bit wide memory.

SIZE[1:0]. These two bits specify the data size associated with a data or instruction fetch. Size bits are sent out "unlatched" by the integer unit. The value on these pins at any given cycle is the data size corresponding to the memory address on the A[31:0] pins at that cycle. SIZE[1:0] remains valid on the bus during all data cycles of loads, stores, load_doubles, store_doubles and atomic load stores. Since all instructions are 32-bits long, SIZE[1:0] is set to "10" during all instruction fetch cycles. Encoding of the SIZE[1:0] bits is shown in *Table 3.*

Table 3. Size Bit Assignment

Size 1	Size 0	Data Transfer Type
0	0	Byte
0	1	Halfword
1	0	Word
1	1	Word (Load/Store Double)

MHOLDA and MHOLDB. The processor pipeline will be frozen while MHOLDA or MHOLDB is asserted and the CY7C601A outputs will revert to and maintain the value they had at the rising edge of the clock in the cycle before MHOLDA or MHOLDB was asserted. MHOLDA/B is used to freeze the clock to both the integer and floating point units during a cache miss (for systems with cache) or when a slow memory is accessed. This signal must be presented to the processor chip at the beginning of each processor clock cycle and be stable during the high time of the processor clock. Either MHOLDA or MHOLDB can be used for stopping the processor during a cache miss or memory exception. MHOLDB has the same definition as MHOLDA. The processor hardware uses the logical "OR" of all hold signals (i.e., MHOLDA, MHOLDB and BHOLD) to generate a final hold signal for freezing the processor pipeline. All HOLD signals are latched (transparent latch) in the CY7C601A before they are used.

BHOLD. BHOLD is asserted by the I/O controller when an external bus master requests the data bus. Assertion of this signal will freeze the processor pipeline. External logic should guarantee that after deassertion of BHOLD, the data at all inputs to the chip is the same as what it was before BHOLD was asserted. This signal must be presented to the processor chip at the beginning of each processor clock cycle and be stable during the high time of the processor clock since the CY7C601A processes the BHOLD input through a transparent latch before it is used. BHOLD should be used only for bus access requests by an external device since the MDS and MEXC signals are not recognized while this input is active. BHOLD should not be deasserted while LOCK is asserted.

MDS. Assertion of this signal will enable the clock input to the on-chip instruction register (during an instruction fetch) or to the load result register (during a data fetch). In a system with cache, MDS is used to signal the processor when the missed data (cache miss) is ready on the bus. In a system with slow memories, MDS is used to signal the processor when the read data is available on the bus. MDS must be asserted only while the processor is frozen by either the MHOLDA or MHOLDB input signals. The CY7C601A samples the MDS signal via an on-chip transparent latch before it is used. The MDS signal is also used for strobing memory exceptions. In other words, MDS should be asserted whenever MEXC is asserted (see MEXC definition).

MEXC. This signal is asserted by the memory (or cache) controller to initiate an instruction (or data) exception trap. MEXC is latched in the processor at the rising edge of CLK and is used in the following cycle. If MEXC is asserted during an instruction fetch cycle an instruction access exception is generated, and if MEXC is asserted during a data fetch cycle, a data access exception trap is generated. The MEXC signal is used during (MHOLD) in conjunction with the MDS signal to indicate to the CY7C601A that the memory system was unable to supply valid instruction or data. If MDS is applied without MEXC, the CY7C601A accepts the contents of the data bus as valid information but when MDS is applied with MEXC an exception trap is generated and the contents of the data bus is ignored by the CY7C601A (i.e., MHOLD and MDS must be low when MEXC is asserted). MEXC must be deasserted in the same clock cycle in which MHOLD is released.

AOE. Deassertion of this signal will three-state all output drivers associated with A[31:0] and ASI[7:0] outputs. AOE is connected directly to the output drivers of the address and ASI signals and must be asserted during normal operations. This signal should be deasserted only when the bus is granted to another bus master (i.e., when either BHOLD, MHOLDA or MHOLDB is asserted).

DOE. Deassertion of this signal will three-state all output drivers of the data D[31:0] bus. DOE is connected directly to the data bus output drivers and must be asserted during normal operations. This signal should be deasserted only when the bus is granted to another bus master (i.e., when either BHOLD, MHOLDA or MHOLDB is asserted).

COE. Deassertion of this signal will three-state all output drivers associated with SIZE[1:0], RD, WE, WRT, LOCK, LDSTO and DXFER outputs. COE is connected directly to the output drivers and must be asserted during normal operations. This signal should be deasserted only when the bus is granted to another bus master (i.e., when either BHOLD, MHOLDA, or MHOLDB is asserted).

RD. This signal specifies whether the current memory access is a read or write operation. It is sent out "unlatched" by the integer unit and must be latched externally before it is used. RD is set to "0" only during address cycles of store instructions including the store cycles of atomic load store instructions. This signal when used in conjunction with SIZE[1:0], ASI[7:0], and LDSTO, can be used to check access rights of bus transactions. In addition, the RD signal may be used to turn off the output drivers of data RAMs during a store operation. For atomic load store instructions the RD signal is "1" during the first address cycle (read cycle) and "0" during the second and third address cycles (write cycle).

$\overline{\text{WE}}$. This signal is asserted by the integer unit during the second address cycle of store single instructions, the second and third address cycles of store double instructions, and the third address cycle of atomic load/store instructions. The $\overline{\text{WE}}$ signal is sent out "unlatched" and must be latched externally before it is used. The $\overline{\text{WE}}$ signal may be externally qualified by HOLD signals (i.e., $\overline{\text{MHOLDA}}$ and $\overline{\text{MHOLDB}}$) to avoid writing into the memory during memory exceptions.

WRT. This signal is asserted (set to "1") by the processor during the first address cycle of single or double integer store instructions, the first address cycle of single or double floating-point store instructions, and the second address cycle of atomic load/ store instructions. WRT is sent out "unlatched" and must be latched externally before it is used.

LDSTO. This signal is asserted by the integer unit during the data cycles of atomic load store operations. LDSTO is sent out "unlatched" by the integer unit and must be latched externally before it is used.

LOCK. This signal is set to "1" when the processor needs the bus for multiple cycle transactions such as atomic load/store, double loads and double stores. LOCK signal is sent "unlatched" and should be latched externally before it is used. The bus may not be granted to another bus master as long as LOCK signal is asserted (i.e., $\overline{\text{BHOLD}}$ should not be asserted in the following processor clock cycle when LOCK=1).

DXFER. This signal is asserted by the processor at the beginning of all bus data transfer cycles. DXFER is "unlatched" and DXFER = 1 indicates a data cycle.

INULL. Assertion of INULL indicates that the current memory access (whose address is held in an external latch) is to be nullified by the processor. INULL is intended to be used to disable cache misses (in systems with cache) and to disable memory exception generation for the current memory access (i.e., $\overline{\text{MDS}}$ and $\overline{\text{MEXC}}$ should not be asserted for a memory access when INULL=1). INULL is a latched output and is active during the same cycle as the address, which it nullifies (the address is not on the bus, but is latched externally). INULL is asserted under the following conditions: During the second cycle of a store instruction, or whenever the CY7C601A address is invalid due to an external or internal exception. If a floating-point unit or coprocessor unit is present in the system, INULL should be ORed with the FNULL and CNULL signals from these units.

$\overline{\text{IFT}}$. The state of this pin determines the behavior of the IFLUSH instruction. If $\overline{\text{IFT}}$=1, then IFLUSH executes like a NOP with no side effects. If $\overline{\text{IFT}}$=0, then IFLUSH causes an unimplemented instruction trap.

Floating-Point/Coprocessor Interface Signals

$\overline{\text{FP}}$. This signal indicates whether or not a floating-point unit exists in the system. The $\overline{\text{FP}}$ signal is normally pulled up to VDD by a resistor. It is grounded when the FPU chip is present. The integer unit generates a floating-point disable trap if $\overline{\text{FP}}$ = 1 during the execution of a floating-point instruction, FBfcc instruction or floating-point load, and store instructions.

$\overline{\text{CP}}$. This signal indicates whether or not a coprocessor exists in the system. The $\overline{\text{CP}}$ signal is normally pulled up to VDD by a resistor. It is grounded when the coprocessor chip is present. The integer unit generates a coprocessor disable trap if $\overline{\text{CP}}$ = 1 during the execution of a coprocessor instruction, CBccc instruction or coprocessor load and store instructions.

FCC[1:0]. These bits are taken as the current condition code bits of the FPU. They are considered valid if FCCV=1. During the execution of the FBfcc instruction, the processor uses these bits to determine whether the branch should be taken or not. FCC[1:0] are latched by the processor before they are used.

CCC[1:0]. These bits are taken as the current condition code bits of the coprocessor. They are considered valid if CCCV=1. During the execution of the CBccc instruction, the processor uses these bits to determine whether the branch should be taken or not. CCC[1:0] are latched by the processor before they are used.

FCCV. This signal should be asserted only when the FCC[1:0] bits are valid. The floating-point unit deasserts FCCV if pending floating-point compare instructions exist in the floating-point queue. FCCV is reasserted when the compare instruction is completed and the floating-point condition codes FCC[1:0] are valid. The integer unit will enter a wait state if FCCV is deasserted (i.e., FCCV = "0"). The FCCV signal is latched (transparent latch) in the CY7C601A before it is used.

CCCV. This signal should be asserted only when the CCC[1:0] bits are valid. The coprocessor deasserts CCCV if pending coprocessor compare instructions exist in the coprocessor queue. CCCV is reasserted when the compare instruction is completed and the coprocessor condition codes CCC[1:0] are valid. The integer unit will enter a wait state if CCCV is deasserted (i.e., CCCV = "0"). The CCCV signal is latched (transparent latch) in the CY7C601A before it is used.

$\overline{\text{FHOLD}}$. This signal is asserted by the floating-point unit if a situation arises in which the FPU cannot continue execution. The floating-point unit checks all dependencies in the decode stage of the instruction and asserts $\overline{\text{FHOLD}}$ (if necessary) in the next cycle. This signal is used by the integer unit to freeze the instruction pipeline in the same cycle. The FPU must eventually deassert $\overline{\text{FHOLD}}$ in order to unfreeze the integer unit's pipeline. The $\overline{\text{FHOLD}}$ signal is latched (transparent latch) in the CY7C601A before it is used.

$\overline{\text{CHOLD}}$. This signal is asserted by the coprocessor if a situation arises in which the coprocessor cannot continue execution. The coprocessor checks all dependencies in the decode stage of the instruction and asserts $\overline{\text{CHOLD}}$ (if necessary) in the next cycle. This signal is used by the integer unit to freeze the instruction pipeline in the same cycle. The coprocessor must eventually deassert $\overline{\text{CHOLD}}$ in order to unfreeze the integer unit's pipeline. The $\overline{\text{CHOLD}}$ signal is latched (transparent latch) in the CY7C601A before it is used.

$\overline{\text{FEXC}}$. Assertion of this signal indicates that a floating-point exception has occurred. $\overline{\text{FEXC}}$ must remain asserted until the integer unit takes the trap and acknowledges the FPU via FXACK signal. Floating-point exceptions are taken only during the execution of floating-point instructions, FBfcc instruction and floating-point load, and store instructions. $\overline{\text{FEXC}}$ is latched in the integer unit before it is used. The FPU should deassert $\overline{\text{FHOLD}}$ if it detects an exception while $\overline{\text{FHOLD}}$ is asserted. In this case $\overline{\text{FEXC}}$ should be asserted a cycle before $\overline{\text{FHOLD}}$ is deasserted.

$\overline{\text{CEXC}}$. Assertion of this signal indicates that a coprocessor exception has occurred. This signal must remain asserted until the integer unit takes the trap and acknowledges the coprocessor via CXACK signal. Coprocessor exceptions are taken only during the execution of coprocessor instructions, CBccc instruction and coprocessor load and store instructions. $\overline{\text{CEXC}}$ is latched in the integer unit before it is used. The coprocessor should deassert $\overline{\text{CHOLD}}$ if it detects an exception while $\overline{\text{CHOLD}}$ is asserted. In this case $\overline{\text{CEXC}}$ should be asserted a cycle before $\overline{\text{CHOLD}}$ is deasserted.

INST. This signal is asserted by the integer unit whenever a new instruction is being fetched. It is used by the FPU or coprocessor to latch the instruction on the D[31:0] bus into the FPU or coprocessor instruction buffer. The FPU (or coprocessor) needs two instruction buffers (D1 and D2) to save the last two fetched instructions. When INST is asserted a new instruction enters into the D1 buffer and the old instruction in D1 enters into the D2 buffer.

FLUSH. This signal is asserted by the integer unit and is used by the FPU or coprocessor to flush the instructions in its instruction registers. This may happen when a trap is taken by the integer unit. Instructions that have entered into the floating-point (or coprocessor) queue may continue their execution if FLUSH is raised as a result of a trap or exception other than floating-point (or coprocessor) exceptions.

FINS1. This signal is asserted by the integer unit during the decode stage of an FPU instruction if the instruction is in the D1 buffer of the FPU chip. The FPU uses this signal to latch the instruction in D1 buffer into its execute stage instruction register.

FINS2. This signal is asserted by the integer unit during the decode stage of an FPU instruction if the instruction is in the D2 buffer of the FPU chip. The FPU uses this signal to latch the instruction in D2 buffer into its execute stage instruction register.

CINS1. This signal is asserted by the integer unit during the decode stage of a coprocessor instruction if the instruction is in the D1 buffer of the coprocessor chip. The coprocessor uses this signal to latch the instruction in D1 buffer into its execute stage instruction register.

CINS2. This signal is asserted by the integer unit during the decode stage of a coprocessor instruction if the instruction is in the D2 buffer of the coprocessor chip. The coprocessor uses this signal to latch the instruction in D2 buffer into its execute stage instruction register.

FXACK. This signal is asserted by the integer unit in order to acknowledge to the FPU that the current $\overline{\text{FEXC}}$ trap is taken. The FPU must deassert $\overline{\text{FEXC}}$ after it receives an asserted level of FXACK signal so that the next floating-point instruction does not cause a "repeated" floating-point exception trap.

CXACK. This signal is asserted by the integer unit in order to acknowledge to the coprocessor that the current $\overline{\text{CEXC}}$ trap is taken. The coprocessor must deassert $\overline{\text{CEXC}}$ after it receives an asserted level of CXACK signal so that the next coprocessor instruction does not cause a "repeated" coprocessor exception trap.

Miscellaneous I/O Signals

IRL[3:0]. The data on these pins defines the external interrupt level. IRL[3:0]=0000 indicates that no external interrupts are pending. The integer unit uses two on-chip synchronizing latches to sample these signals on the rising edge of CLK. A given interrupt level must remain valid for at least two consecutive cycles to be recognized by the integer unit. IRL[3:0]=1111 signifies an non-maskable interrupt. All other interrupt levels are maskable by the PIL field of the processor state register (PSR). External interrupts should be latched and prioritized by the external logic before they are passed to the integer unit. The external interrupt latches should keep the interrupts pending until they are taken (and acknowledged) by the integer unit. External interrupts can be acknowledged by software or by the Interrupt Acknowledge (INTACK) output.

INTACK. This signal is asserted by the integer unit when an external interrupt is taken.

$\overline{\text{RESET}}$. Assertion of this pin will reset the integer unit. The $\overline{\text{RESET}}$ signal must be asserted for a minimum of eight processor clock cycles. After a reset, the integer unit will start fetching from address 0. The $\overline{\text{RESET}}$ signal is latched by the integer unit before it is used.

$\overline{\text{ERROR}}$. This signal is asserted by the integer unit when a trap is encountered while traps are disabled via the ET bit in the PSR. In this situation the integer unit saves the PC and nPC registers, sets the tt value in the TBR, enters into an error state, asserts the $\overline{\text{ERROR}}$ signal and then halts. The only way to restart the processor trapped in the error state, is to trigger a reset by asserting the $\overline{\text{RESET}}$ signal.

$\overline{\text{TOE}}$. This signal is used to force all output drivers of the processor chip into a high-impedance state. It is used to isolate the chip from the rest of the system for debugging purposes.

FPSYN. This pin is a mode pin which is used to allow execution of additional instructions in future designs. It should be normally kept deasserted (FPSYN=0) to disable the execution of these instructions.

CLK. CLK is a 50% duty-cycle clock used for clocking the CY7C601A's pipeline registers. It is HIGH during the first half of the processor cycle, and LOW during the second half. The rising edge of CLK defines the beginning of each pipeline stage in the CY7C601A chip.