Features

- SPARC V8 High Performance Low-power 32-bit Architecture
  - 8 Register Windows
- Advanced Architecture:
  - On-chip Amba Bus
  - 5 Stage Pipeline
  - 32 KB 4-way associative Instruction Cache
  - 16 KB 2-way associative Data Cache
- On-chip Peripherals:
  - Memory Interface
    - PROM Controller
    - SRAM Controller
    - SDRAM Controller
  - Timers
    - Two 32-bit Timers
    - Watchdog 32-bit Timer
  - Two 8-bit UARTs
  - Interrupt Controller with 8 External Programmable Inputs
  - 32 Parallel I/O Interface
  - 33MHz PCI Interface Compliant with 2.2 PCI Specification
- Integrated 32/64-bit IEEE 754 Floating-point Unit
- Fault Tolerance by Design
  - Full Triple Modular Redundancy (TMR)
  - EDAC Protection
  - Parity Protection
- Debug and Test Facilities
  - Debug Support Unit (DSU) for Trace and Debug
  - IEEE 1149.1 JTAG Interface
  - Four Hardware Watchpoints
- 8 and 32-bit boot-PROM Interface Possibilities with EDAC
- Operating range
  - Voltages
    - 3.3V ± 0.30V for I/O
    - 1.8V ± 0.15V for Core
  - Temperature
    - -55°C to 125°C
- Clock: 0 MHz up to 100 MHz
- Power consumption: 1 W at 100 MHz
- Performance:
  - 86 MIPS (Dhrystone 2.1)
  - 23 MFLOPS (Whetstone)
- Radiation Performance
  - Tested up to a total dose of 300 krad (Si) according to the MIL-STD883 method 1019
  - SEU error rate better than 1 E-5 error/device/day
  - No Single Event Latchup below a LET threshold of 70 MeV.cm^2/mg
- MCGA-349 (9g) and MQFP-256 packages
- Development Kit Including
  - AT697 Evaluation Board
  - AT697F Sample

Rad-Hard 32 bit SPARC V8 Processor

AT697F

Rev. 7703E–AERO–08/11

Description

The AT697F is a highly integrated, high-performance 32-bit RISC embedded processor based on the SPARC V8 architecture. The implementation is based on the European Space Agency (ESA) LEON2 fault tolerant model. By executing powerful instructions in a single clock cycle, the AT697F achieves throughputs approaching 1MIPS per MHz, allowing the system designer to optimize power consumption versus processing speed.

The AT697F is designed to be used as a building block in computers for on-board embedded real-time applications. It brings up-to-date functionality and performance for space application.

The AT697F only requires memory and application specific peripherals to be added to form a complete on-board computer.

The AT697F contains an Integer Unit (IU), a Floating Point Unit (FPU), separate instruction and data caches, a hardware multiplier and divider, an interrupt controller, two 32-bit timers and a watchdog, two UARTs, a general-purpose I/O interfaces, a PCI Interface and a flexible memory controller. The design is highly testable with support from an embedded Debug Support Unit (DSU) with trace buffers and a JTAG interface for boundary scan.

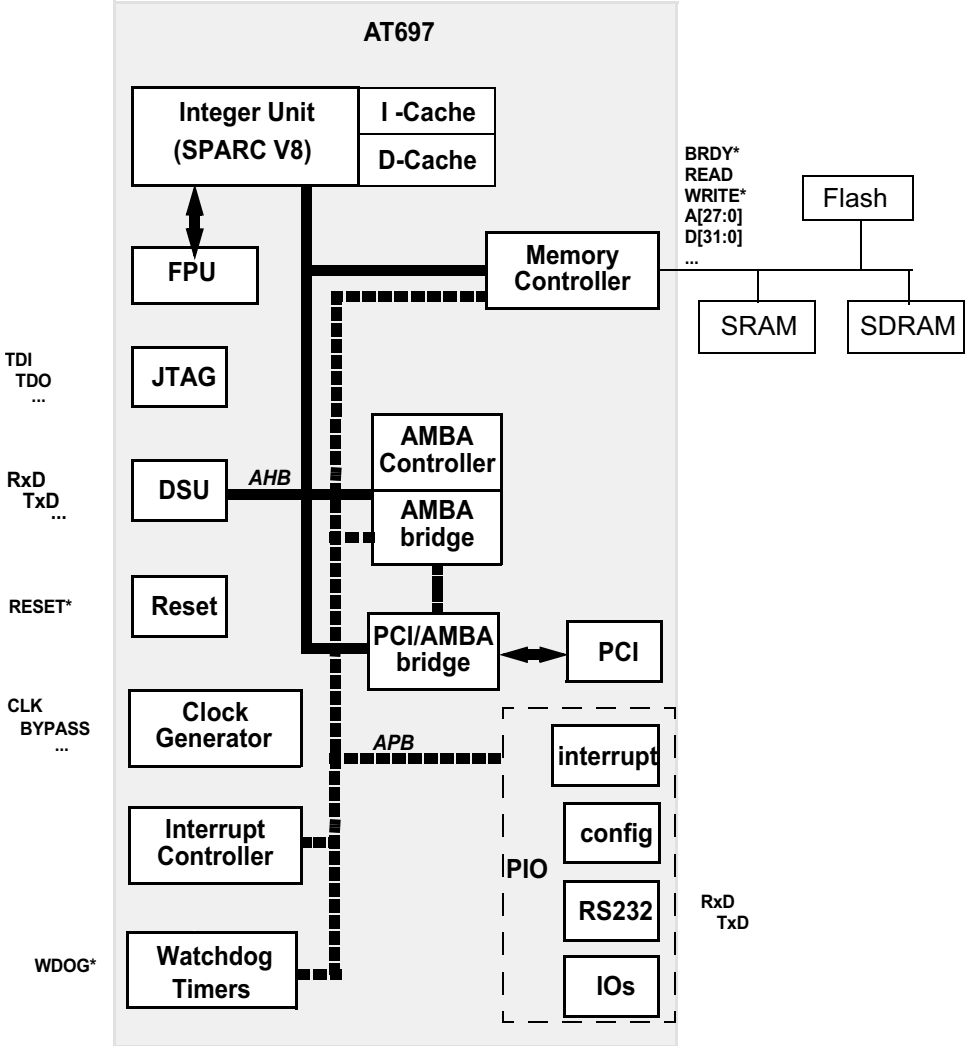
An idle mode holds the processor pipeline and allows Timer/Counter, Serial ports and Interrupt system to continue functioning.

The processor is manufactured using the Atmel ATC18RHA CMOS process. It has been especially designed for space by implementing on-chip concurrent transient and permanent error detection and correction.

The AT697F is pinout compatible with the AT697E.

Refer to section “Differences between AT697F and AT697E” for detailed description of the differences between AT697F and AT697E.

Figure 1. AT697 Block Diagram



Pin Description

A signal name ending with a ‘’ (e.g OE*) designates an active-low signal.*

A bit field in a register are represented in a "dot" notation: `MCFG2.ramwws` *is read as the* `ramwws` *field in register* `MCFG2`*.*

System Interface

`RESET*` - Processor reset (input)

When asserted, this asynchronous active low input immediately halts and resets the processor and all on-chip peripherals. The processor restarts execution after the 5th rising edge of the clock after **`RESET*`** was de-asserted.

`ERROR*` - Processor error (open-drain output with pull-up)

This active low output is asserted when the processor is halted in error mode.

`WDOG*` - Watchdog timeout (open-drain output with pull-up)

This active low output is asserted when the watchdog timer has expired and remains asserted until the watchdog timer is reloaded with a non-null value.

`BEXC*` - Bus exception (input)

This active low input is sampled simultaneously with the data during an access to the external memory. If asserted, a memory error is generated.

Clock Interface

`CLK` - Reference clock (input)

This input provides a reference to generate the internal clock used by the processor and the internal peripherals.

`BYPASS` - PLL bypass (input with pull-down)

This active high input is used to bypass the internal PLL. When asserted, the processor is directly clocked from the external reference clock. When de-asserted, the processor receives its clock from the internal PLL.

Caution: This signal shall be kept static and free from glitches while the processor is operating, as it is not sampled internally. Changing the signal shall only be performed while the processor is under reset otherwise the processor's behavior is not predictable.

`LOCK` - PLL lock (output)

When asserted, this active high output indicates the PLL is locked at a frequency corresponding to four times the frequency of the external reference clock. **Caution: this signal is de-asserted as soon as the PLL unlocks.**

`SKEW[1:0]` - Clock tree skew (input with pull-down)

These input signals are used to programme the skew on the internal triplicated clock trees.

Caution: These signals shall be kept static and free from glitches while the processor is operating, as they are not sampled internally. Changing these signals shall only be performed while the processor is under reset otherwise the processor's behavior is not predictable.

Memory Interface

`A[27:0]` - Address bus (output)

The lower 28 bits of the 32 bit address bus carry instruction or data addresses during a fetch or a load/store operation to the external memory. The address of the last external memory access remains on the address bus whenever the current access can be made out of the internal cache.

`D[31:0]` - Data bus (bi-directional)

The 32-bit bi-directional data bus serves as the interface between the processor and the external memory. The data bus is only driven by the processor during the execution of integer & floating-point store instructions and the store cycle of atomic-load-store instructions. It is kept in high impedance otherwise. However:

- only `D[31:24]` are used during an access to an 8-bit area
- `D[15:0]` are used as part of the general-purpose I/O interface whenever all the memory areas (ROM, SRAM & I/O) are 8-bit wide and the SDRAM interface is not enabled

`CB[7:0]` - Check bits (bi-directional)

These signals carry the EDAC checkbits[1] during a write access to the external memory and are kept in high impedance otherwise. This applies whatever the EDAC activation or not.

Note: 1. While only 7 bits are useful for EDAC protection, **CB[7]** is implemented to enable programming of FLASH memories and takes the value of MCFG3.tcb[7].

OE* - Output enable (output)

This active low output is asserted during a read access to the external memory. It can be used as an output enable signal when accessing PROM & I/O devices.

READ - Read enable (output)

This active high output is asserted during a read access to the external memory. It can be used as a read enable signal when accessing PROM & I/O devices.

WRITE* - Write enable (output)

This active low output is asserted during a write-access to the external memory. It can be used as a write enable signal when accessing PROM & I/O devices.

RWE*[3:0] - PROM & SRAM byte write-enable (output)

These active low outputs provide individual write strobes for each byte-lane on the data bus: **RWE*[0]** controls **D[31:24]**, **RWE*[1]** controls **D[23:16]**, **RWE*[2]** controls **D[15:8]** and **RWE*[3]** controls **D[7:0]**, and they are set according to the transaction width (word/half-word/byte) and the bus width set for the respective area.

BRDY* - Bus ready (input)

When driven low, this input indicates to the processor that the current memory access can be terminated on the next rising clock edge. When driven high, this input indicates to the processor that it must wait and not end the current access.

PROM

ROMS*[1:0] - PROM chip-select (output)

These active low outputs provide the chip-select signals for decoding the PROM area. **ROMS*[0]** is asserted when the lower half of the PROM area is accessed (0x00000000 - 0x0FFFFFFF), while **ROMS*[1]** is asserted when the upper half is accessed (0x10000000 - 0x1FFFFFFF).

SRAM

RAMS*[4:0] - SRAM chip-select (output)

These active low outputs provide the chip-select signals for decoding five SRAM banks.

RAMOE*[4:0] - SRAM output enable (output)

These active low signals provide an output enable signal for each SRAM bank.

I/O

IOS* - I/O select (output)

This active low output provides the chip-select signal for decoding the memory mapped I/O area.

SDRAM

SDCLK - SDRAM clock (output)

This signal provides a reference clock for SDRAM memories. It is a copy of the processor internal clock.

SDCS*[1:0] - SDRAM chip select (output)

These active low outputs provide the chip select signals for decoding two SDRAM banks.

SDRAS*- SDRAM row address strobe (output)

This active low output provides the RAS signal (Row Access Strobe) for SDRAM devices.

SDCAS* - SDRAM column address strobe (output)

This active low output provides the CAS signal (Column Access Strobe) for SDRAM devices.

SDWE* - SDRAM write strobe (output)

This active low output provides the write strobe for SDRAM devices.

SDDQM[3:0] - SDRAM data mask (output)

These active high outputs provide the DQM strobe (Data Mask) for SDRAM devices.

General-Purpose Interface

PIO[15:0] - Parallel I/O port (bi-directional)

These bi-directional signals can be used as general-purpose inputs or outputs to control external devices. Some of these signals have an alternate function and also serve as inputs or outputs for internal peripherals.

PCI interface

System

PCI_CLK - PCI clock (input)

This signal provides timing for all transactions on the PCI bus. All other PCI signals, except **PCI_RST***, are sampled on the rising edge of **PCI_CLK** and all other timing parameters are defined with respect to this edge.

PCI_RST* - PCI Reset (input)

This active low input is used to bring PCI-specific registers, sequencers and signals to a consistent state. When asserted, it immediately halts and resets the PCI interface. The PCI interface resumes execution after the 5th rising edge of the PCI clock after **PCI_RST*** was de-asserted.

SYSEN* - System Enable (input)

This active low input is used to configure the PCI interface as the Host-Bridge for the PCI bus (also called the System-Controller in a CompactPCI-compliant environment). If de-asserted, the PCI interface is configured as a satellite on the PCI bus.

Caution: This signal shall be kept static and free from glitches while the processor is operating, as it is not sampled internally. Changing the signal shall only be performed while the processor is under reset otherwise the processor's behavior is not predictable.

Address & Data

A/D[31:0] - PCI Address Data (bi-directional)

Address and Data are multiplexed on the same PCI pins. During the address phase, **A/D[31:0]** contain a physical address (32 bits). For I/O, this is a byte address; for configuration and memory, it is a 32-bit address. During data phases, **A/D[7:0]** contain the least significant byte and **A/D[31:24]** contain the most significant byte.

C/BE*[3:0] - PCI Bus Command and Byte Enables (bi-directional)

During the address phase of a transaction, **C/BE*[3:0]** define the bus command. During the data phase, **C/BE*[3:0]** are used as Byte Enables. The Byte Enables are valid for the entire data phase.

PAR - Parity (bi-directional)

This signal is even parity across **A/D[31:0]** and **C/BE*[3:0]** (the number of "1"s on **A/D[31:0]**, **C/BE*[3:0]** and **PAR** equals an even number). The master drives **PAR** for address and write data phases; the PCI target drives **PAR** for read data phases.

Interface Control

FRAME* - Cycle Frame (bi-directional)

This signal is driven by the current master to indicate the beginning and duration of an access. **FRAME*** is asserted to indicate a bus transaction is beginning. While **FRAME*** is asserted, data transfers continue. When **FRAME*** is deasserted, the transaction is in the final data phase or has completed.

IRDY* - Initiator Ready (bi-directional)

This signal indicates the initiating agent's ability to complete the current data phase of the transaction. **IRDY*** is used in conjunction with **TRDY***. During a write, **IRDY*** indicates that valid data is present on **A/D[31:0]**. During a read, it indicates the master is prepared to accept data.

TRDY* - Target Ready (bi-directional)

This signal indicates the target agent's (selected device's) ability to complete the current data phase of the transaction. **TRDY*** is used in conjunction with **IRDY***. During a read, **TRDY*** indicates that valid data is present on **A/D[31:0]**. During a write, it indicates the target is prepared to accept data.

STOP* - Stop (bi-directional)

This signal indicates the current target is requesting the master to stop the current transaction.

PCI_LOCK* - Lock (bi-directional)

This signal indicates an atomic operation to a bridge that may require multiple transactions to complete.

IDSEL - Initialization Device Select (input)

Initialization Device Select is used as a chip select during configuration read and write transactions.

DEVSEL* - Device Select (bi-directional)

When actively driven, indicates the driven device has decoded its address as the target of the current access. As an input, **DEVSEL*** indicates whether any device on the bus has been selected.

Arbitration

REQ* - PCI bus request (output)

This signal indicates to the arbiter that this agent desires use of the bus. This is a point-to-point signal. Every master has its own **REQ*** which is tri-stated while PCI reset is asserted.

GNT* - PCI Bus Grant (input)

This signal indicates to the agent that access to the bus has been granted. This is a point-to-point signal. Every master has its own **GNT*** which is ignored while PCI reset is asserted.

Error Reporting

PERR* - Parity Error (bi-directional)

This signal is only for the reporting of data parity errors during all PCI transactions except a Special Cycle. The **PERR*** pin is sustained tri-state and must be driven active by the agent receiving data two clocks following the data when a data parity error is detected. The minimum duration of **PERR*** is one clock for each data phase that a data parity error is detected.

SERR* - System Error (open-drain bi-directional)

This signal is for reporting address parity errors, data parity errors on the special cycle command, or any other system error where the result will be catastrophic. **SERR*** is pure open drain and is actively driven for a single PCI clock by the agent reporting the error.

PCI Arbiter

AREQ*[3:0] - PCI bus request (input)

When asserted, these active low inputs indicate that a PCI agent is requesting the bus.

AGNT*[3:0] - PCI bus grant (PCI-compliant output)

When asserted, these active low outputs indicate that a PCI agent is granted the PCI bus.

DSU Interface

DSUEN - DSU enable (input)

When asserted, this synchronous active high input enables the DSU unit. If de-asserted, the DSU trace buffer will continue to operate but the processor will not enter debug mode.

Caution: This signal is meant for debug purpose and shall be driven low in the final application.

DSURX - DSU receiver (input)

This input provides the serial data stream to the DSU communication link receiver.

Caution: This signal is meant for debug purpose and shall be driven low in the final application.

DSUTX - DSU transmitter (output)

This output provides the serial data stream from the DSU communication link transmitter.

DSUACT - DSU active (output)

This active high output is asserted when the processor is in debug mode and controlled by the DSU.

DSUBRE - DSU break enable (input)

A low-to-high transition on this synchronous input signals a break condition and is used to set the processor into debug mode (see "Debug Support Unit" later in this document for specific use).

Caution: This signal is meant for debug purpose and shall be driven low in the final application.

JTAG Interface

TRST* - Test Reset (input with pull-up)

This asynchronous active low input resets the TAP when asserted.

Caution: This signal is meant for board testing purpose and shall be driven low in the final application.

TCK - Test Clock (input with pull-down)

This input is used to clock state information and test data into and out of the device during operation of the TAP.

TMS - Test Mode select (input with pull-up)

This synchronous input is used to control the state of the TAP controller in the device.

TDI - Test data input (input with pull-up)

This input is used to serially shift test data and test instructions into the device during TAP operation.

TDO - Test data output (tri-statable output with pull-up)

This input is used to serially shift test data and test instructions out of the device during TAP operation.

Power Supply

VCC33 - I/O power (supply)

Power supply for the I/O pins.

VSS33 - I/O ground (supply)

Ground supply for the I/O pins.

VDD18 - Core power (supply)

Power supply for the processor core.

VSS18 - Core ground (supply)

Ground supply for the processor core.

VDD_PLL - PLL power (supply)

Power supply for the PLL.

VSS_PLL - PLL ground (supply)

Ground supply for the PLL.

Summary

Table 1. Signals Properties

Signal	Direction	Active	Reset(1)	Type(5)	Comment
A[27:0]	output		X	CMOS	
A/D[31:0]	bi-directional		Z	PCI	
AGNT*[3:0]	output	Low	1 (2)	PCI	
AREQ*[3:0]	input	Low		CMOS	
BEXC*	input	Low		CMOS	
BRDY*	input	Low		CMOS	
BYPASS	input	High		CMOS	Internal pull-down
C/BE*[3:0]	bi-directional	Low	Z	PCI	
CB[7:0]	bi-directional		Z	CMOS	
CLK	input			CMOS	
DEVSEL*	bi-directional	Low	Z	PCI	
DSUACT	output	High	X / 1 (3)	CMOS	
DSUBRE	input	Rise		CMOS	
DSUEN	input	High		CMOS	
DSURX	input			CMOS	
DSUTX	output		X / 1 (3)	CMOS	
D[31:0]	bi-directional		Z	CMOS	
ERROR*	open-drain output	Low	X / H (3)	CMOS	Internal pull-up
FRAME*	bi-directional	Low	Z	PCI	
GNT*	input	Low		CMOS	
IDSEL	input	High		CMOS	
IOS*	output	Low	1	CMOS	
IRDY*	bi-directional	Low	Z	PCI	
LOCK	output	High	X (4)	CMOS	
OE*	output	Low	1	CMOS	
PAR	bi-directional		Z	PCI	
PCI_CLK	input			CMOS	
PCI_LOCK*	bi-directional	Low	Z	PCI	
PCI_RST*	input	Low		CMOS	
PERR*	bi-directional	Low	Z	PCI	
PIO[15:0]	bi-directional		Z	CMOS	
RAMOE*[4:0]	output	Low	1	CMOS	
RAMS*[4:0]	output	Low	1	CMOS	
READ	output	High	X / 1 (3)	CMOS	
REQ*	output	Low	Z	PCI	

Signal	Direction	Active	Reset[1]	Type[5]	Comment
RESET*	input	Low		CMOS	
ROMS*[1:0]	output	Low	1	CMOS	
RWE*[3:0]	output	Low	X / 1 [3]	CMOS	
SDCAS*	output	Low	X / 1 [3]	CMOS	
SDCLK	output			CMOS	
SDCS*[1:0]	output	Low	1	CMOS	
SDDQM[3:0]	output	High		CMOS	
SDRAS*	output	Low	X / 1 [3]	CMOS	
SDWE*	output	Low	X / 1 [3]	CMOS	
SERR*	bi-directional	Low	Z	PCI	
SKEW[1:0]	input			CMOS	Internal pull-down
STOP*	bi-directional	Low	Z	PCI	
SYSEN*	input	Low		CMOS	
TCK	input			CMOS	Internal pull-down
TDI	input			CMOS	Internal pull-up
TDO	tri-state output			CMOS	Internal pull-up
TMS	input			CMOS	Internal pull-up
TRDY*	bi-directional	Low	Z	PCI	
TRST*	input	Low		CMOS	Internal pull-up
WDOG*	open-drain output	Low	X / H [3]	CMOS	Internal pull-up
WRITE*	output	Low	X / 1 [3]	CMOS	

Notes:
1. Signals on the PCI interface clocked by the **PCI_CLK** pin and reset by the **PCI_RST*** pin. Others signals (internally) clocked by the **SDCLK** pin and reset by the **RESET*** pin.
   Reset values meaning:
   - X: value is a strong high or low level, but cannot be predicted
   - 1: value is a strong high level
   - Z: no level is driven from the device, signal is in high-impedance state
   - H: value is a resistive high level
2. In PCI Host mode, parking granted to PCI agent 0 (**AGNT*[0]**) during reset after the first rising edge of the PCI clock if no request is made to the PCI arbiter at that time (**AREQ*[3:0]** = 1111).
3. First value effective during reset without a running clock while second value effective during reset after the first rising edge of the clock.
4. Value is a strong high or low level, but cannot be predicted during reset without a running clock, then value is a strong low level during reset after the first rising edge of the clock until it becomes a strong high level as soon as the PLL locks (not applicable in PLL bypass mode / **BYPASS** = 1).
5. See "Electrical Characteristics".

Caution: Any unused input pin or bi-directional pin with a tri-stated output driver shall be pulled to a valid high or low level to prevent it from floating as a floating input pin does not have a stable voltage level and is often the cause for additional power-consumption in the input section of the pad because of continuous noise-induced switches.

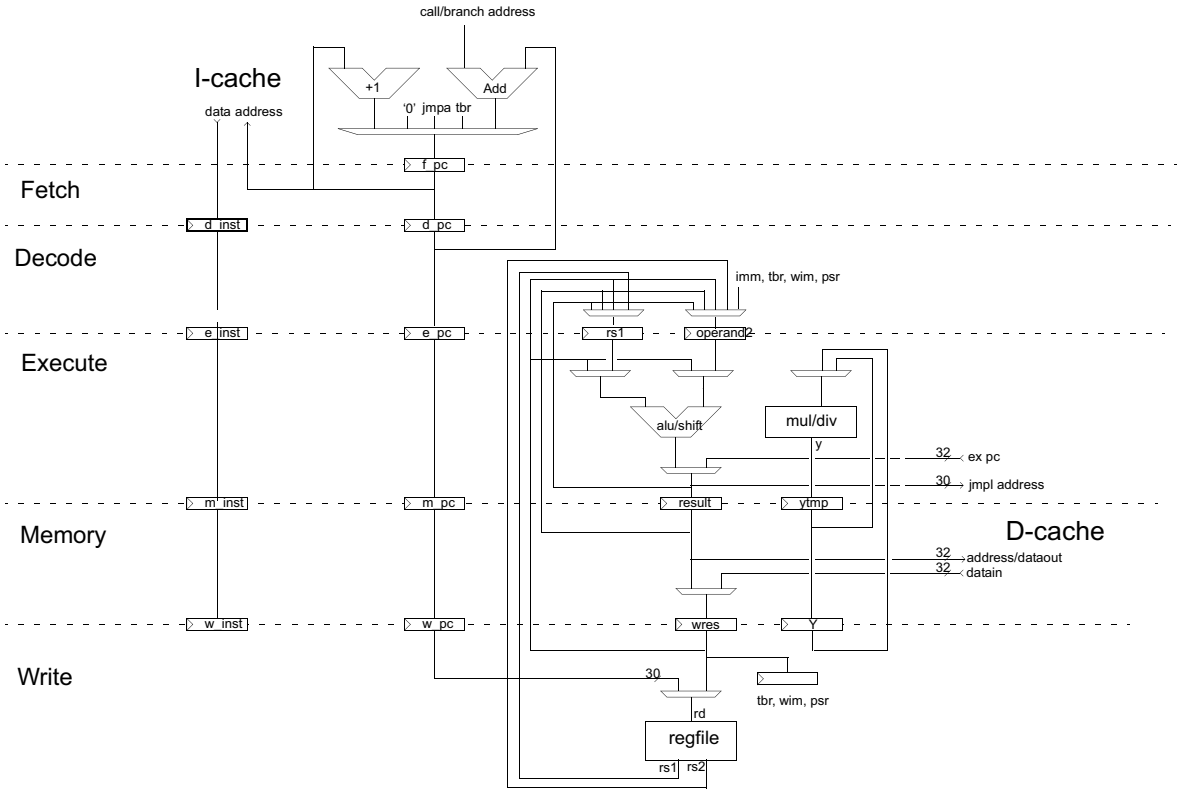
Architecture

The AT697F is a general-purpose processor that can perform intensive calculations, access high-speed memories, control peripherals and handle interrupts. It has been designed for radiation-hardened high-reliability applications by including fault-tolerance features.

Integer Unit

The AT697F integer unit is based on the LEON2-FT architecture, it implements the SPARC integer instruction-set defined in the SPARC Architecture Manual version 8.

Figure 2. Integer Unit Architecture



To execute instructions at a rate approaching one instruction per clock cycle, the IU employs a five-stage instruction pipeline that permits parallel execution of multiple instructions.

- **Fetch**

  The instruction is fetched from the instruction cache is enabled and available or the fetch is forwarded to the memory controller.

- **Decode**

  The instruction is placed in the instruction register and decoded. The operands are read from the register file and/or from immediate data and the next instruction computed `CALL`/`Bicc` target address are generated.

- **Execute**

  Arithmetic, logical and shift operations are performed and the result saved in temporary registers. Memory and `JMPL`/`RETT` target address are generated. Pending traps are prioritized and internal traps are taken, if any.

- **Memory**

  On a memory load instruction, data is read from the data cache if enabled and available or the read is forwarded to the memory controller. On a memory store instruction, store data is always forwarded to the memory controller and any matching data cache entry is invalidated if enabled.

- **Write**

  The result of any arithmetic, logical, shift or memory/cache read operation is written back to the register file.

All five stages operate in parallel, working on up to five different instructions at a time.

A basic "single-cycle" instruction enters the pipeline and completes in five cycles. By the time it reaches the write stage, four more instructions have entered and are moving through the pipeline behind it. So, after the first five cycles, a single-cycle instruction exits the pipeline and a single-cycle instruction enters the pipeline on every cycle.

Of course, a "single-cycle" instruction actually takes five cycles to complete, but they are called single cycle because with this type of instruction the processor can complete one instruction per cycle after the initial five-cycle delay.

Table 2. Cycles per instruction (assuming cache hit and no load interlock)

Instruction	Cycles
Jump and Link (JMPL)	2
Load Double-Word (LDD)	2
Store Single-Word (ST)	2
Store Double-Word (STD)	3
Integer Multiply (SMUL/UMUL/SMULcc/UMULcc)	5
Integer Divide (SDIV/UDIV/SDIVcc/UDIVcc)	35
Taken Trap (Ticc)	4
Atomic Load/Store (LDSTUB)	3
All other IU instructions	1
Single-Precision Multiply (FMULS)	16(1)
Double-Precision Multiply (FMULD)	21(1)
Single-Precision Divide (FDIVS)	20(1)
Double-Precision Divide (FDIVD)	36(1)
Single-Precision Square-Root (FSQRTS)	37(1)
Double-Precision Square-Root (FSQRTD)	65(1)
Single-Precision Absolute Value (FABS)	2(1)
Single-Precision Move (FMOVS)	2(1)
Single-Precision Negate (FNEGS)	2(1)
Convert Single to Double-Precision (FSTOD)	2(1)
All other arithmetic FPU instructions	7(1)

Note: 1. This value is to be considered "typical". As the execution of FPU operations is oper-and-dependent, the true execution time can be lower or higher than mentioned.

Program Counters

The Program Counter (PC) contains the address of the instruction currently being executed by the Integer Unit, and the next Program Counter (nPC) holds the address (PC + 4) of the next instruction to be executed (assuming there is no control transfer and a trap does not occur). The nPC is necessary to implement delayed control transfers, wherein the instruction that immediately follows a control transfer may be executed before control is transferred to the target address.

Having both the PC and nPC available to the trap handler allows a trap handler to choose between retrying the instruction causing the trap (after the trap condition has been eliminated) or resuming program execution after the trap causing instruction.

Windowed Register File

The AT697F contains a 136×32 register file divided into 8 overlapping windows, each window providing a working registers set at a time. Working registers are used for normal operations and are called *r* registers.

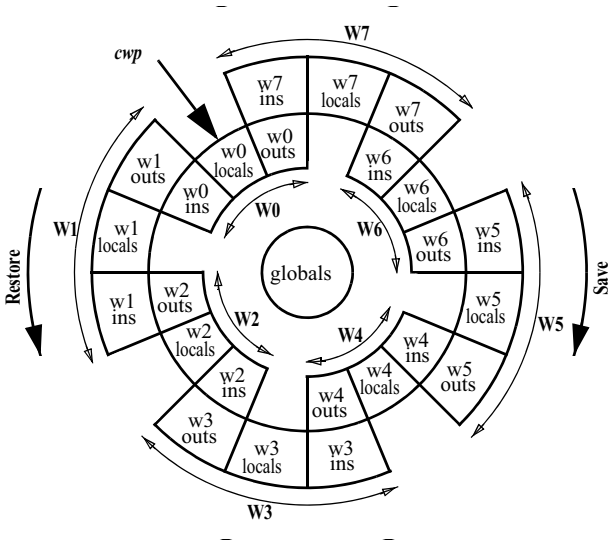
The 136 registers are 32-bits wide and are divided into a set of 8 global registers and a set of 128 window registers grouped into 8 sets of 24 *r* registers called windows. At any given time, a program can access 32 active *r* registers (r0 to r31): 8 (common) global

registers (r0 to r7) and 24 window registers (r8 to r31) that are divided by software convention into 8 *ins*, 8 *locals* and 8 *outs*:

- The first 8 *globals* (r0 to r7) are also called *g* registers (g0 to g7), they usually hold common data to all functions (as a special case, r0/g0 always returns the value 0 when read and discards the value written to it)
- The next 8 *ins* (r8 to r15) are also called *i* registers (i0 to i7), they usually are the input parameters of a function
- The next 8 *locals* (r16 to r23) are also called *l* registers (l0 to l7), they usually are scratch registers that can be used for anything within a function
- The last 8 *outs* (r24 to r31) are also called *o* registers (o0 to o7), they usually are the return parameters of a function

The register file can be viewed as a circular stack, with the highest window joined to the lowest. Note that each window shares its *ins* and *outs* with adjacent windows: *outs* from a previous window are the *ins* of the current window and the *outs* of the current window are the *ins* of the next window.

Figure 3. Circular Stack of Overlapping Windows



The register file implementation is based on two dual-port RAMs. The first dual-port RAM provides the first operand of a SPARC instruction while the second dual-port RAM provides the second operand unless an immediate value is needed. When applicable, the result of the instruction is written back into the register file, so the two dual-port RAMs always have equal contents.

When one function calls another, the calling function can choose to execute a SAVE instruction. This instruction decrements an internal counter, the current window pointer (CWP), shifting the register window downward. The caller's out registers then become the calling function's in registers and the calling function gets a new set of local and out registers for its own use. Only the pointer changes because the registers and return address do not need to be stored on a stack. The RESTORE/RETT instructions acts in the opposite way

The Window Invalid Mask register (WIM) is controlled by supervisor software and is used by the hardware to determine whether a window overflow or underflow trap is to be generated by a SAVE, RESTORE or RETT instruction.

When a SAVE, RESTORE or RETT instruction is executed, the current value of the CWP is compared against the WIM register. If the SAVE, RESTORE, or RETT instruction would cause the CWP to point to an "invalid" register set, a window_overflow or window_underflow trap is caused.

Arithmetic & Logic Unit

The high-performance ALU operates in direct connection with all the 32 working registers. Within a single clock cycle, a 32-bit arithmetic operation between two working registers or between a working register and an immediate value is executed.

State Register

The Processor State Register (PSR) contains fields that report the status of the processor operations or control processor operations.

Instructions that modify its fields include SAVE, RESTORE, Ticc, RETT and any instruction that modifies the condition code fields (*icc*). Any hardware or software action that generates a trap will also modify some of its fields.

A global interrupt management is provided: traps and interrupts (asynchronous traps) can be enabled/disabled and interrupts level response can be fine tuned.

Instruction Set

AT697F SPARC instructions fall into six functional categories: load/store, arithmetic/logical/ shift, control transfer, read/write control register, floating-point and miscellaneous. Please refer to the SPARC V8 Architecture Manual for further details.

Multiply instructions

The AT697F fully supports the SPARC V8 multiply instructions (UMUL, SMUL, UMULcc and SMULcc). The multiply instructions perform a 32×32-bit integer multiply producing a 64-bit result. SMUL and SMULcc perform signed multiply while UMUL and UMULcc performs unsigned multiply. UMULcc and SMULcc also set the condition codes to reflect the result. The Y register holds the most-significant half of the 64-bit result.

Divide Instructions

The AT697F fully supports the SPARC V8 divide instructions (UDIV, SDIV, UDIVcc and SDIVcc). The divide instructions perform a 64×32 bit divide and produce a 32-bit result. SDIV and SDIVcc perform signed multiply while UDIV and UDIVcc performs unsigned divide. UDIVcc and SDIVcc also set the condition codes to reflect the result. Rounding and overflow detection is performed as defined in the SPARC V8 standard. The Y register holds the most-significant half of the 64-bit divided value.

Floating-Point Unit

The AT697F floating-point unit is based on the MEIKO core and implements the SPARC floating-point instruction-set defined in the SPARC Architecture Manual version 8.

The FPU interface is integrated into the IU pipeline and does not implement a floating-point queue[1], so the IU is stopped during the execution of floating-point instructions.

Note: 1. This means the qne bit in the FSR register is always zero and any attempts to execute the STDFQ instruction will generate an FPU_exception trap (0x08).

The AT697F contains a 32×32 register file for floating-point operations, refered to as *f* registers (f0 to f31). These registers are 32-bits wide and can be concatenated to support 64-bits double-words (extended precision instructions and format are not supported). The whole 32 registers set is available at all time for any floating-point operation.

Integer and single-precision data require a single 32-bit *f* register. Double-precision data require 64-bit of storage and occupy an even-odd pair of adjacent registers.

Memory Mapping

The 32-bit addressable memory space is divided into 6 areas, each area being allocated to a specific device type and externally or internally decoded accordingly:

Table 3. Memory Mapping

Address Range	Area	Device Select Signals
0x00000000 - 0x1FFFFFFF	PROM(1)(2)	2
0x20000000 - 0x3FFFFFFF	I/O(1)	1
0x40000000 - 0x7FFFFFFF	RAM(2)	5 (SRAM) + 2 (SDRAM)
0x80000000 - 0x8FFFFFFF	REGISTER(1)	*n/a (internal)*
0x90000000 - 0x9FFFFFFF	DSU(1)	
0xA0000000 - 0xFFFFFFFF	PCI(1)	*see PCI/cPCI specification*

Notes: 1. Area is equally accessible in User & Supervisor mode on read & write access
2. Write protection may apply if enabled on the area

Fault Tolerance

The processor has been especially designed for radiation-hardened applications. To prevent erroneous operations from single event transient (SET) and single event upset (SEU) errors, the AT697F processor implements a set of protection features including:

- Full triple modular redundancy (TMR) architecture
- Programmable skews on the clock trees
- EDAC protection on IU and FPU register files
- EDAC protection on external memory interface
- Parity protection on instruction and data caches

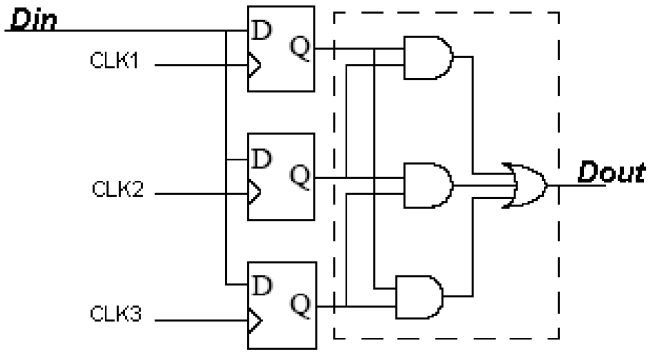
Triple Modular Redundancy

To protect against SEU errors (Single Event Upset), each on-chip register is implemented using triple modular redundancy (TMR). This means that any SEU register error is automatically removed within one clock cycle while the output of the register maintains the correct (glitch-free) value.

Moreover, an independent clock tree is used for each of the three registers making up one TMR module. This feature protects against SET errors (Single Event Transient) in the clock tree, to the expense of increased routing.

The CPU clock and the PCI clock are built as three-clock trees. The same triplication is applied to the CPU reset and to the PCI reset as well.

Figure 4. TMR Register with Separate Clock-Tree



Clock-Tree Skew

Optionally, a skew can be applied between each of the three branches of the clock trees in order to provide additional SET protection.

Register File SEU Protection

To prevent erroneous operations from SEU errors in the IU and FPU register file, each register is protected using a 7-bit EDAC checksum. Checking of the EDAC bits is done every time a fetched register value is used in an instruction. If a correctable error is detected, the erroneous data is corrected before being used. At the same time, the corrected register value is also written back to the register file. A correction operation incurs a delay 4 clock cycles, but has no other software visible impact. If an uncorrectable error is detected, a `register_hardware_error` trap (0x20) is generated.

Cache Parity

The cache parity mechanism is transparent to the user, but in case of a cache parity error, a cache miss is generated and an access to external memory is perfomed to reload the cache entry, implying some delay.

Operating Modes

Reset Mode

The processor asynchronously enters reset mode as soon as the **RESET*** signal is asserted. It synchronously exits reset mode on the 5th rising edge of the **SDCLK** signal after the **RESET*** signal has been deasserted.

While in reset mode, the IU, the FPU and all the peripherals are halted. The processor disables traps (`PSR.et` = `0`), sets the supervisor mode (`PSR.s` = `1`) and sets the program counter to location zero (`PC` = `0` & `nPC` = `4`). Other IU, FPU and peripheral registers are initialized as well (see "Register Description").

On exit from reset mode, the processor enters execute mode.

Execute Mode

In execute mode, the processor fetches instructions and executes them in the IU (Integer Unit) or the FPU (Floating-Point Unit). Please refer to "*The SPARC Architecture Manual - Version 8*" for further information.

Error Mode

The processor enters error mode when a synchronous trap must occur while traps are disabled (`PSR.et` = `0`).

This essentially means that a trap which cannot be ignored occured while another trap is being serviced. In order for that synchronous trap to be serviced, the processor goes through the normal operation of a trap, including setting the trap-type bits (`TBR.tt`) to identify the trap type. It then enters error mode, halts and asserts the **ERROR*** signal.

The only way to leave error mode is to assert the **RESET*** signal, which forces the processor into reset mode. All information placed in the IU, FPU and all peripherals registers from the last execute mode (the trap operation) remains unchanged unless stated otherwise (see "Register Description"). The code executed upon exit from reset mode can examine the trap type of the synchronous trap and the IU, FPU and all peripherals registers and deal with the information they contain accordingly.

Idle Mode

While in execute mode, the processor may enter idle mode in software.

Entry into idle mode is initiated by writing any value to the idle register (`IDLE`) and made effective on the next load instruction(caution). While in idle mode, the IU stops fetching instructions and is kept into a minimal activity. All other peripherals operate as nominal.

Caution: In order to avoid any unwanted side-effect (idle entry not on the foreseen load instruction), the load instruction shall immediately follow the write operation to the idle register,

Idle mode is terminated as soon as an unmasked interrupt is pending (`ITP.ipend`) with an interrupt number of 15 or greater than the current processor interrupt level (`PSR.pil`). Then the processor directly goes through the normal operation of servicing the interrupt.

Debug Mode

Caution: *This section is for information purpose only.*

*As its name clearly states, the Debug Support Unit (DSU) is exclusively meant for debugging purpose. None of the DSU features shall ever be used in the final application where the DSU shall be turned into an inactive state (**DSUEN**, **DSUBRE** and **DSURX** tied to a permanent low level).*

As a special case when the DSU is enabled (**DSUEN** signal asserted), debug mode is entered when specific conditions are met (see "Debug Support Unit" and "Register Description" chapters later in this document).

In debug mode, the processor pipeline is held and the processor is controlled by the DSU so all processor registers, caches memories and even external peripherals can be accessed.

Traps and Interrupts

Overview

The AT697F supports two types of traps:

- synchronous traps
- asynchronous traps also called interrupts.

Synchronous traps are caused by the hardware responding to a particular instruction. They occur during the instruction that caused them. Asynchronous traps occur when an external event interrupts the processor. They are not related to any particular instruction and occur between the execution of instructions.

A trap is a vectored transfer of control to the supervisor through a special trap table that contains the first four instructions of each trap handler. The trap base address (`TBR`) of the table is established by the supervisor and the displacement, within the table, is determined by the trap type.

A trap causes the current window pointer to advance to the next register window and the hardware to write the program counters (`PC` & `nPC`) into two registers of the new window.

Synchronous Traps

The AT697F follows the general SPARC trap model. The table below shows the implemented traps and their individual priority.

Table 4. Trap Overview

Trap	Trap Type	Priority	Description
`reset`	*n/a*(1)	1	Power-on reset
`write_error`	`0x2B`	2	Write buffer error
`instruction_access_exception`	`0x01`	3	Error during instruction fetch Edac uncorrectable error during instruction fetch
`illegal_instruction`	`0x02`	5	`UNIMP` or other un-implemented instruction
`privileged_instruction`	`0x03`	4	Execution of privileged instruction in user mode
`fp_disabled`	`0x04`	6	FP instruction while FPU disabled
`cp_disabled`	`0x24`		Co-processor instruction while co-processor disabled
`watchpoint_detected`	`0x0B`	7	Instruction or data watchpoint match
`window_overflow`	`0x05`	8	SAVE into invalid window
`window_underflow`	`0x06`		RESTORE into invalid window
`register_hardware_error`	`0x20`	9	Register file uncorrectable EDAC error
`mem_address_not_aligned`	`0x07`	10	Memory access to un-aligned address
`fp_exception`	`0x08`	11	FPU exception
`data_access_exception`	`0x09`	13	Access error during load or store instruction
`tag overflow`	`0x0A`	14	Tagged arithmetic overflow

Trap	Trap Type	Priority	Description
divide_exception	0x2A	15	Divide by zero
trap_instruction	0x80 - 0xFF	16	Software trap instruction (Ticc)

Note: 1. The reset trap is a special case of the external asynchronous trap type: the trap type field of the trap base register (TBR.tt) is never modified so if the processor goes to error mode, a *post-mortem* can later be conducted on what caused the error, if any.

Traps Description

- reset - A reset trap occurs immediately after the **RESET*** signal is deasserted.
- write_error - An error exception occurred on a data store to memory.
- instruction_access_exception - A blocking error exception occurred on an instruction access.
- illegal_instruction - An attempt was made to execute an instruction with an unimplemented opcode, or an UNIMP instruction, or an instruction that would result in illegal processor state.
- privileged_instruction - An attempt was made to execute a privileged instruction while not in supervisor mode (PSR.s = 0).
- fp_disabled - An attempt was made to execute an FPU instruction while the FPU is not enabled.
- cp_disabled - An attempt was made to execute a co-processor instruction (there is no co-processor).
- watchpoint_detected - An instruction fetch memory address or load/store data memory address matched the contents of an active watchpoint register.
- window_overflow - A SAVE instruction attempted to cause the current window pointer (CWP) to point to an invalid window in the window invalid mask register (WIM).
- window_underflow - A RESTORE or RETT instruction attempted to cause the current window pointer (CWP) to point to an invalid window in the window invalid mask register (WIM).
- register_hardware_error - An error exception occurred on a read only register access. A register file uncorrectable error was detected.
- mem_address_not_aligned - A load/store instruction would have generated a memory address that was not properly aligned according to the instruction, or a JMPL or RETT instruction would have generated a non-word-aligned address.
- fp_exception - An FPU instruction generated an IEEE-754_exception and its corresponding trap enable mask bit (FSR.tem) was set, or the FPU instruction was unimplemented, or the FPU instruction did not complete, or there was a sequence

or hardware error in the FPU. The type of floating-point exception is encoded in the FPU state register (`FSR.ftt`).

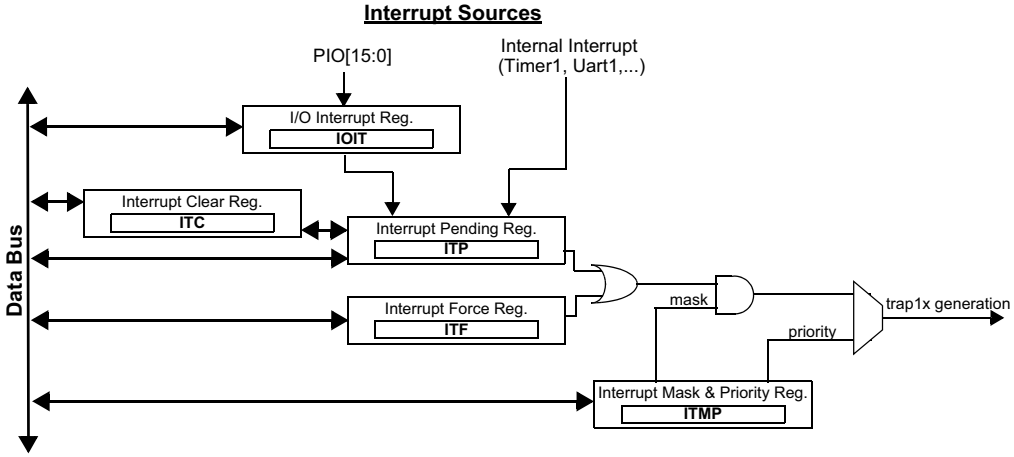
- `data_access_exception` - A blocking error exception occurred on a load/store data access. EDAC uncorrectable error.
- `tag_overflow` - A tagged arithmetic instruction was executed, and either arithmetic overflow occurred or at least one of the tag bits of the operands was non zero.
- `trap_division_by_zero` - An integer divide instruction attempted to divide by zero.
- `trap_instruction` - A software instruction (`Ticc`) was executed and the trap condition evaluated to true.

When multiple synchronous traps occur at the same cycle (i.e hardware errors), the highest priority trap is taken and lower priority traps are ignored.

Asynchronous Traps / Interrupts

The AT697F handles up to 15 interrupts. The interrupt controller is used to prioritize and propagate interrupts requests from internal peripherals and external devices to the IU.

Figure 5. Interrupt Controller Block Diagram



Operation

Interrupts are controlled in the interrupt mask and priority register (`ITMP`):

- interrupts requests can be masked (`ITMP.imask`) so they will not trigger an interrupt
- interrupt requests can have a low/high priority level (`ITMP.ilevel`)

When an interrupt request is generated, the corresponding bit is set in the interrupt pending register (`ITP.ipend[]`) only if the interrupt is not masked (`ITMP.imask[]` = `0`).

Then all pending interrupts are forwarded to the priority selector. The pending interrupt (`ITP.ipend[]`) with the highest number on the high priority level (`ITMP.ilevel[]` = `1`) is selected and forwarded to the IU. If no pending interrupt exists with a high priority level, the pending interrupt with the highest number on the low priority level (`ITMP.ilevel[]` = `0`) is selected and forwarded to the IU.

Interrupts can also be forced by setting the corresponding bit in the interrupt force register (`ITF.iforce[]` = `1`). Forcing is effective only if the corresponding interrupt is not masked (`ITMP.imask[]` = `0`). A forced interrupt never shows up as pending.

Pending interrupts can be cleared by setting the appropriate bit in the interrupt clear register (`ITC.iclear[]` = `1`).

When the IU acknowledges an interrupt, the corresponding pending bit is automatically cleared (`ITP.ipend[]` = `0`) unless it was forced in the interrupt force register (`ITF.iforce[]` = `1`). If the interrupt was forced, the IU acknowledgement rather clears the force bit (`ITF.iforce[]` = `0`).

Interrupts List

The following table presents the interrupts assignment:

Table 5. Interrupt Overview

Interrupt	Trap Type	Source
15	`0x1F`	I/O interrupt 7[(1)]
14	`0x1E`	PCI
13	`0x1D`	I/O interrupt 6
12	`0x1C`	I/O interrupt 5
11	`0x1B`	DSU trace buffer
10	`0x1A`	I/O interrupt 4
9	`0x19`	Timer 2
8	`0x18`	Timer 1
7	`0x17`	I/O interrupt 3
6	`0x16`	I/O interrupt 3
5	`0x15`	I/O interrupt 1
4	`0x14`	I/O interrupt 0
3	`0x13`	UART 1
2	`0x12`	UART 2
1	`0x11`	Hardware error[(2)]

Notes:
1. Interrupt 15 cannot be filtered by the processor interrupt level in the IU (`PSR.pil`) and shall be used with care.
2. Interrupt 1 is triggered each time a new hardware error is reported (`FAILAR.hed`) so the application ultimately knows about any hardware error that was detected (bus exception, write protection error, EDAC correctable and uncorrectable external memory error, PCI initiator error or PCI target error).

I/O Interrupts

As an alternate function of the general purpose interface, the AT697F can handle interrupts from external devices. Up to 8 external interrupts can be programmed at the same time. The external interrupts are assigned to interrupt 4, 5, 6, 7, 10, 12, 13 and 15.

There are two registers for configuring I/O interrupts:

- IO interrupt 0, 1, 2 and 3 are controlled by `IOIT1`
- IO interrupt 4, 5, 6 and 7 are controlled by `IOIT2`

Each I/O interrupt is controlled through 4 parameters in the appropriate configuration register (*n* = 1 & *x* in [0:3] or *n* = 2 & *x* in [4:7]):

- the interrupt can be enabled or disabled (IOITn.enx)
- the interrupt source can be chosen among **PIO[15:0]** and **D[15:0]**[1] (IOITn.iselx)
- the interrupt can be level-triggered[2] or edge-triggered[2] (IOITn.lex)
- the interrupt polarity can be high/low[2] when level-triggered or rising/falling[2] when edge-triggered (IOITn.plx)

Notes: 1. **D[15:0]** can be used as part of the general-purpose I/O interface only when all the memory areas (ROM, SRAM & I/O) are 8-bit wide and the SDRAM interface is not enabled.

2. Whatever the chosen trigger mode, the I/O inputs are sampled on the rising edge of the system clock. If generated synchronously, the signal driving the I/O interrupt shall meet the required setup and hold constraints (see "Electrical Characteristics"). If generated asynchronously, the signal driving the I/O interrupt shall be maintained for a minimum of 1.5 clock cycles so at least 1 active sample can be made.

The following table summarizes the I/O interrupt trigger configurations.

Table 6. I/O Interrupt Trigger Configuration

IOITn.lex	IOITn.plx	Trigger
0	0	low level
0	1	high level
1	0	falling edge
1	1	rising edge

Cache Memories

Overview

The AT697F implements a Harvard architecture with separate instruction and data bus, each connected to an independent cache controller. In order to improve the speed performance of the processor, multi-way caches are used to provide a faster access to instructions and data.

The PROM and RAM areas, which represent most of the external memory areas, can be cached.

Table 7. Caching Capability

Address Range	Area	Cacheability
0x00000000 - 0x1FFFFFFF	PROM	Cacheable
0x20000000 - 0x3FFFFFFF	I/O	Non-cacheable
0x40000000 - 0x7FFFFFFF	RAM	Cacheable
0x80000000 - 0x8FFFFFFF	Registers	Non-cacheable
0x90000000 - 0x9FFFFFFF	DSU	Non-cacheable
0xA0000000 - 0xFFFFFFFF	PCI	Non-cacheable

Operation

An associative cache is organized in sets, each set being divided into cache lines. Each line has a cache tag associated with it consisting of a tag field and a valid field with one valid bit for each 4-byte cache data sub-block.

The cache replacement policy used for both instruction and data caches is based on the LRU algorithm: new entries are allocated in a cache until the cache is full, then least recently used entries are replaced with new entries not already present in the cache.

A cache always operates in one of the three modes: disabled, enabled or frozen.

Disabled Mode

No cache operation is performed and fetch/load requests are passed directly to the memory controller.

Enabled Mode

On a cache miss to a cacheable location, the fetch/load request is passed to the memory controller and the corresponding tag and data line are updated with the retrieved instruction/data. Otherwise, the instruction/data is directly retrieved from the cache and forwarded to the IU/FPU.

Frozen Mode

The cache is accessed as if it was enabled, but no new line is allocated on a cache miss.

If the freeze-on-interrupt feature is activated, the corresponding cache is automatically frozen when an asynchronous interrupt is taken. This can be beneficial in real-time system to allow a more accurate calculation of worst-case execution time for an interrupt service routine (ISR). Execution of the interrupt handler will not evict any cache line so when control is returned to the interrupted task, the cache state is identical to what it was before the interrupt.

If a cache was frozen by an interrupt, it can only be enabled again in software. This is typically done at the end of the interrupt service routine before control is returned to the interrupted task.

Parity Protection

Cache tag/data error protection is implemented using two parity bits per tag and per 4-byte data sub-block. The tag parity is generated from the tag value, the valid bits and the tag address. By including the tag address, it is also possible to detect errors in the cache ram address decoding logic. Similarly, the data subblock parity is derived from the sub-block address and the sub-block data. The parity bits are written simultaneously with the associated tag or sub-block and checked on each access. The two parity bits correspond to the parity of odd and even tag/data bits.

If a tag parity error is detected during a cache access, a cache miss is generated and the tag/data is automatically updated. All valid bits except the one corresponding to the newly loaded data are cleared. Each tag error is reported in the cache tag error counter, which is incremented until the maximum count is reached.

If a data sub-block parity error occurs, a miss is also generated but only the failed sub-block is updated with fetched/loaded data. Each error is reported in the cache data error counter, which is incremented until the maximum count is reached.

Instruction Cache

Overview

The AT697F instruction cache is implemented as a 4-way associative cache of 32 KB, organized in 4 sets of 8 KB. Each instruction cache set is divided into cache lines of 32 bytes (8 instructions). Each line has a cache tag associated with it consisting of a tag field and a valid bit field per instruction.

Cache Control

The instruction cache operation is controlled in the cache control register (`CCR`):

- operating mode is reported and can be changed (`CCR.ics`)
- cache can be frozen on interrupts (`CCR.if`)
- cache flush can be initiated (`CCR.fi`)
- pending cache flush is reported (`CCR.ip`)
- burst fetch is reported and can be activated (`CCR.ib`)
- up to 3 cache tag/data errors are reported in counters (`CCR.ite` and `CCR.ide`), which shall be cleared to register new events

Operation

Instruction Fetch

On an instruction cache fetch-miss to a cacheable location, an instruction (4 bytes) is loaded into the cache from external memory.

Burst Fetch

If instruction burst fetch is enabled in the cache control register (`CCR.ib` = `1`), the cache line is filled from main memory starting at the missed address and until the end of the line.

At the same time, the instructions are forwarded to the IU (streaming). If the IU cannot accept the streamed instructions due to internal dependencies or a multi-cycle instruction, the IU is halted until the line fill is completed. If the IU executes a control transfer instruction during the cache line fill (`Bicc`/`CALL`/ `JMPL`/`RETT`/`Ticc`), the cache line fill is terminated on the next fetch.

If instruction burst fetch is enabled, instruction streaming is enabled even when the cache is disabled. In this case, the fetched instructions are only forwarded to the IU and the cache is not updated.

Cache Flush

The instruction cache is flushed by executing the `FLUSH` instruction, or by activating the instruction cache flush in the cache control register (`CCR.fi` = `1`).

The flush operation takes one clock cycle per cache line and set. The IU is not halted during the cache flush but the cache behaves as if it was disabled. When the flush operation is completed, the cache resumes the state (disabled, enabled or frozen) indicated in the cache control register (`CCR.ics`).

Error reporting

If a memory access error occurs during a line fill with the IU halted, the corresponding valid bit in the cache tag is not set. If the IU later fetches an instruction from the failed address, a cache miss will occur, triggering a new access to the failed address.

If the error remains, an `instruction_access_error` trap (`0x01`) is generated.

Data Cache

Overview

The AT697Fdata cache is implemented as a 2-way associative cache of 16 KB, organized in 2 sets of 8 KB. Each data cache set is divided into cache lines of 16 bytes (4 words). Each line has a cache tag associated with it consisting of a tag field and a valid bit field per word.

Cache Control

The data cache operation is controlled in the cache control register (`CCR`):

- operating mode is reported and can be changed (`CCR.dcs`)
- cache can be frozen on interrupts (`CCR.df`)
- cache flush can be initiated (`CCR.fd`)
- pending cache flush is reported (`CCR.dp`)
- cache snooping can be activated (`CCR.ds`)
- up to 3 cache tag/data errors are reported in counters (`CCR.dte` and `CCR.dde`), which shall be cleared to register new events

Operation

Data Load

On a data cache read-miss to a cacheable location, 1 word of data (4 bytes) is loaded into the cache from external memory.

Data Store

The write policy for stores is write-through with update on write-hit and no-allocate on write-miss. An internal write buffer of three 32-bit words is used to temporarily hold store data until it is effectively written into the external device. For half-word and byte stores, the stored data is replicated into proper byte alignment for writing to a word-addressed device before being loaded into the write buffer.

The write buffer is emptied prior to a load-miss/cache-fill sequence to avoid any stale data from being read into the data cache.

Cache Flush

The data cache is flushed by executing the `FLUSH` instruction, or by activating the data cache flush in the cache control register (`CCR.fd` = `1`).

The flush operation takes one clock cycle per cache line and set. The IU is not halted during the cache flush but the cache behaves as if it was disabled. When the flush operation is completed, the cache resumes the state (disabled, enabled or frozen) indicated in the cache control register (`CCR.dcs`).

Cache Snooping

The data cache can perform snooping on the internal bus. When snooping is enabled (`CCR.ds` = `1`), the data cache controller monitors write accesses performed either by the PCI DMA controller, or by the PCI target controller or by the DSU communication module. When a write access is performed to a cacheable memory location, the corre-

sponding cache line is invalidated in the data cache if present. Cache snooping has no overhead and does not affect performance.

Error Reporting

If a memory access error occurs during a data load, the corresponding valid bit in the cache tag is set and a `data_access_error` trap (`0x09`) is generated.

Since the processor executes in parallel with the write buffer, a write error may not cause an exception to the store instruction. Depending on memory and cache activity, the external memory write access may not occur until several clock cycles after the store instructions has completed. If a write error occurs, the currently executing instruction will take a `write_error` trap (`0x2B`).

Caution: The `write_error` trap handler shall flush the data cache since a write hit would update the cache while the memory would keep the old value due the write error.

Diagnostic Cache Access

Tags and data in the instruction and data cache can be accessed through `ASI` address space by executing `LDA` and `STA` instructions (only the least-significant nibble of the `ASI` field -- `ASI[3:0]` -- is used for mapping to the alternate address space). Address bits making up the cache offset are used to index the tag to be accessed while the least significant bits of the bits making up the address tag are used to index the cache set.

The following table summarizes the `ASI` allocation on the AT697F.

Table 8. ASI Usage

ASI	Usage
0x0, 0x1, 0x2, 0x3	Force cache miss (replace if cacheable)
0x4, 0x7	Force cache miss (update on hit)
0x5	Flush instruction cache
0x6	Flush data cache
0x8	User instruction (replace if cacheable)
0x9	Supervisor instruction (replace if cacheable)
0xA	User data (replace if cacheable)
0xB	Supervisor data (replace if cacheable)
0xC	Instruction cache tags
0xD	Instruction cache data
0xE	Data cache tags
0xF	Data cache data

Note: Please refer to "The SPARC Architecture Manual Version 8" document for detailed information on `ASI` usage.

The tags can be directly read/written by executing an `LDA`/`STA` instruction with `ASI=0xC` for instruction cache tags and `ASI=0xE` for data cache tags. The cache line and the cache set are indexed by the address bits making up the cache offset and the least significant bits of the address bits making up the address tag..

Similarly, the data sub-blocks are read/written by executing an `LDA`/`STA` instruction with `ASI=0xD` for instruction cache data and `ASI=0xF` for data cache data..

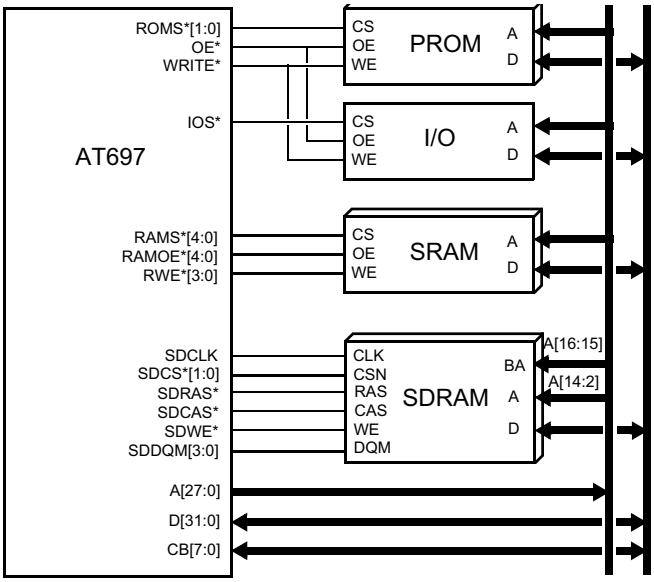
Note: Diagnostic cache access is not possible during a cache flush operation and will cause a `data_exception` trap (`0x09`) if attempted.

Memory Interface

Overview

The AT697F provides a direct memory interface to PROM, memory mapped I/O, asynchronous static ram (SRAM) and synchronous dynamic ram (SDRAM) devices.

Figure 6. Memory Interface Overview(1)



Note: 1. **WRITE*** and **RWE*[3:0]** present redundant information and they can be used in PROM and SRAM areas.

The memory controller decodes a 2 GB address space and performs chip-select decoding for two PROM banks, one I/O bank, five SRAM banks and two SDRAM banks.

Table 9. Memory Controller Address Map

Address Range	Size	Area
0x00000000 - 0x1FFFFFFF	512M	PROM
0x20000000 - 0x3FFFFFFF	512M	I/O
0x40000000 - 0x7FFFFFFF	1G	SRAM and/or SDRAM

The memory data bus width can be configured for either 8-bit or 32-bit access, independently for PROM, memory-mapped I/O and SRAM. A fixed 32-bit wide data bus is required for SDRAM.

EDAC protection is available for PROM, SRAM and SDRAM memories (**CB[7:0]** are always driven on a write access in 32-bit mode even when EDAC is not activated).

To improve the bandwidth of the memory bus, accesses to consecutive addresses can be performed in burst mode. Burst transfers will be generated when the memory controller is accessed using a burst request from the internal bus. These includes instruction cache-line fills, double-word loads and double-word stores.

The memory interface is controlled through 3 memory configuration registers:

- MCFG1 is dedicated to PROM and I/O configuration
- MCFG2 & MCFG3 are dedicated to SRAM and SDRAM configuration

PROM Interface

Overview

The memory controller allows addressing of up to 512 MB of PROM in two banks. PROM memory access control is performed using dedicated chip selects (**ROMS*[1:0]**) and common output enable (**OE***), read (**READ**) and write (**WRITE***) signals.

PROM banks map as follows:

- **ROMS*[0]** decodes the 256 MB lower half of the PROM area (0x00000000 - 0x0FFFFFFF)
- **ROMS*[1]** decodes the 256 MB upper half of the PROM area (0x10000000 - 0x1FFFFFFF)

The PROM interface is controlled in the memory configuration registers (MCFG1 and MCFG3):

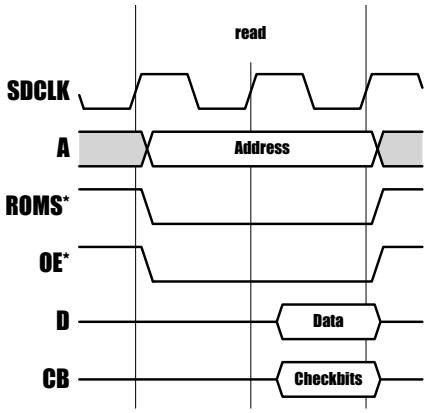
- data bus width(1) can be 8-bit or 32-bit (MCFG1.prwdh)
- wait-states(2) can be programmed for read access (MCFG1.prrws) and write access (MCFG1.prwws)
- write can be enabled/disabled (MCFG1.prwen)
- EDAC protection(3) can be enabled/disabled (MCFG3.pe)
- read/write access can be **BRDY***-controlled (MCFG1.pbrdy) synchronously/asynchronously(4) (MCFG1.abrdy)

Notes:
1. Upon reset, the PROM data bus width is automatically configured from the value read on the **PIO[1:0]** pins. By driving **PIO[1:0]** appropriately during reset, it is possible to set the PROM data bus width on boot.
2. Upon reset, the PROM wait-states are set to the maximum value to allow booting on even the slowest memory.
3. Upon reset, the PROM EDAC protection is automatically configured from the value read on the **PIO[2]** pin.
4. Asynchronous **BRDY***-control feature common to PROM, SRAM and I/O interfaces.

Read Access

A PROM read access consists in two data cycles.

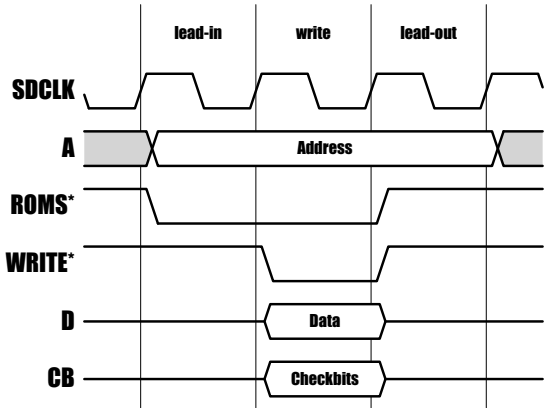
Figure 7. PROM Read Access (no wait-states)



Write Access

A PROM write access consists in an address lead-in cycle, a data write cycle and an address lead-out cycle. The write operation is strobed by the **WRITE*** signal.

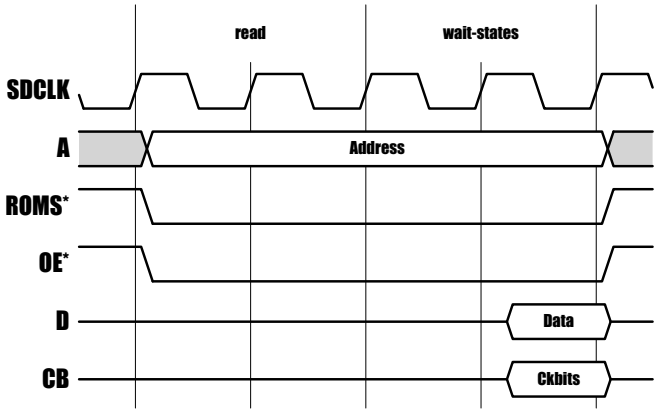
Figure 8. PROM Write Access (no wait-states)



Wait-States

For application accessing slow PROM memories, the memory controller allows to insert wait-states during a PROM read access (MCFG1.prrws) and write access (MCFG1.prwws). Up to 30 wait-states can be inserted in steps of 2 (number of wait-states is twice the programmed value).

Figure 9. PROM Read Access with 2 Wait-States (MCFG1.prrws=1)



PROM read/write access can further be stretched when even more delay is needed (see "**BRDY***-Controlled Access" later in this chapter").

Write Protection

PROM write access is disabled after reset and shall be enabled (MCFG1.prwen = 1) before any write access is attempted. Otherwise the write access is cancelled and a write_error trap (0x2B) is taken.

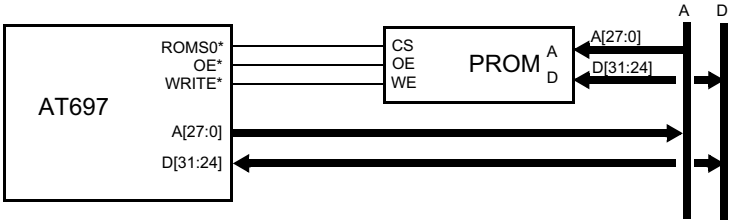
Data Bus Width

When configured for 32-bit PROM (MCFG1.prwdh = 00), **D[31:0]** shall be connected to the memory device(s). **CB[7:0]** shall be connected as well if EDAC protection is activated (MCFG3.pe = 1).

To support applications with limited memory and/or limited performance requirements, the PROM area can be configured for 8-bit operations.

When configured for 8-bit PROM (`MCFG1.prwdh` = `00`), **`D[31:24]`** shall be connected to the memory device(s).

Figure 10. 8-bit PROM Interface



Since an IU load operation is always performed on a word basis (32-bit), read access to 8-bit memory is transformed into a burst of 4 read access to retrieve the 4 bytes. If EDAC protection is activated, a 5th byte read access is performed as well to retrieve the checkbits (see "Error Management - EDAC" section later in this chapter).

During a store operation, only the necessary bytes are written if EDAC protection is not activated.

Caution: When EDAC protection is activated, only a full word write operation shall be performed (5 bytes).

Memory-Mapped I/O

Overview

The memory controller allows addressing a single memory-mapped I/O area. I/O memory access control is performed using a dedicated chip select (**`IOS*`**) and common output enable (**`OE*`**), read (**`READ`**) and write (**`WRITE*`**) signals. No EDAC protection is available in this area.

`IOS*` decodes a fixed 512 MB(1) area (`0x20000000` - `0x3FFFFFFF`).

The I/O interface is controlled in the memory configuration registers (`MCFG1`):

- interface can be enabled/disabled (`MCFG1.ioen`)
- data bus width can be 8-bit or 32-bit (`MCFG1.iowdh`)(2)
- wait-states can be programmed for read and write access (`MCFG1.iows`)
- read/write access can be **`BRDY*`**-controlled (`MCFG1.iobrdy`) synchronously/asynchronously(3) (`MCFG1.abrdy`)

Notes:
1. The upper 256 MB area (`0x30000000` - `0x3FFFFFFF`) is a shadow of the lower 256 MB area (`0x20000000` - `0x2FFFFFFF`) because of the 28 bits address bus limitation.
2. The I/O area shall only be accessed with load/store instructions of a size matching the configured bus width (`LDUB/LDSB/STB` when in 8-bit mode and `LD/ST` when in 32-bit mode).
3. Asynchronous **`BRDY*`**-control feature common to PROM, SRAM and I/O interfaces.

Interface Enable

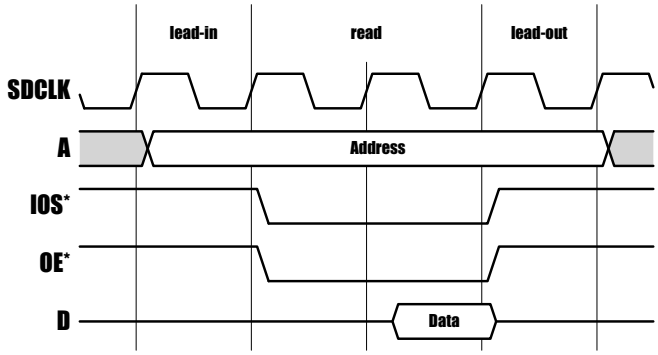
The I/O interface shall be enabled (`MCFG1.ioen`) before any read or write access is attempted, otherwise the access is cancelled and:

- an `instruction_access_exception` trap (`0x01`) is generated if an instruction fetch is in progress
- a `data_access_exception` trap (`0x09`) is generated if a data load is in progress
- a `write_error` trap (`0x2B`) is generated if a data store is in progress

Read Access

An I/O read access consists in a address lead-in cycle (the `IOSEL*` signal is delayed by one clock cycle to provide a stable address for sampling), two data cycles and an address lead-out-cycle.

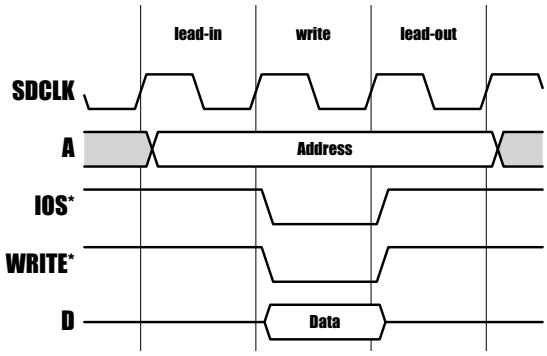
Figure 11. I/O Read Access (no wait-states)



Write Access

An I/O write access consists in an address lead-in cycle, a data write cycle and an address lead-out cycle. The write operation is strobed by the **`WRITE*`** signal.

Figure 12. I/O Write Access (no wait-states)



Wait-States

For application accessing slow I/O devices, the memory controller allows to insert wait-states during an I/O read /write access (`MCFG1.iows`). Up to 15 wait-states can be inserted.

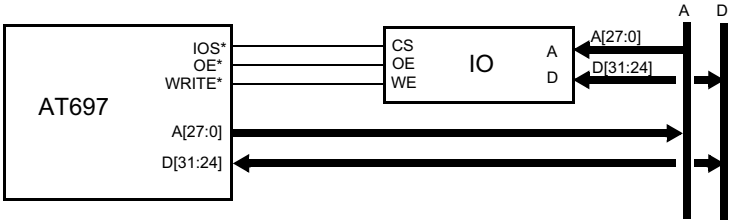
An I/O read/write access can further be stretched when even more delay is needed (see "**`BRDY*`**-Controlled Access" later in this chapter").

Data Bus Width

When configured for 32-bit I/O (`MCFG1.iowdh` = `00`), **`D[31:0]`** shall be connected to the I/O device(s). Only 32-bit load/store instructions (`LD`, `ST`) shall be used then.

When configured for 8-bit I/O (`MCFG1.iowdh` = `00`), **`D[31:24]`** shall be connected to the I/O device(s). Only 8-bit load/store instructions (`LDUB`, `LDSB`, `STB`) shall be used then.

Figure 13. 8-bit I/O interface



.

RAM Interface

The memory controller allows to control up to 1 GB of RAM. The global RAM area supports two RAM types: asynchronous static RAM (SRAM) and synchronous dynamic RAM (SDRAM).

SRAM Interface

Overview

The memory controller allows addressing of up to 768 MB of SRAM in 5 banks. SRAM memory access control is performed using dedicated chip selects (**`RAMS*[4:0]`**), output enables (**`RAMOE*[3:0]`**) and byte-write enables (`RAMWE*[3:0]`) signals.

SRAM banks map as follows:

- **`RAMS*[0]`**, **`RAMS*[1]`**, **`RAMS*[2]`** and **`RAMS*[3]`** decode contiguous banks with a common programmable size (8 KB to 256 MB) at the lower half of the RAM area (from `0x40000000` onwards)
- **`RAMS*[4]`** decodes a fixed 256 MB at the upper half of the RAM area (`0x60000000` - `0x6FFFFFFF`) unless the SDRAM interface is enabled

The SRAM interface is controlled in the memory configuration registers (`MCFG2 and MCFG3`):

- interface can be enabled/disabled (`MCFG2.si`)
- data bus width can be 8-bit or 32-bit (`MCFG2.ramwdh`)
- bank size can be set from 8 KB to 256 MB (`MCFG2.rambs`)
- wait-states can be programmed for read access (`MCFG2.ramrws`) and write access (`MCFG2.ramwws`)
- read-modify-write can be activated for sub-word write operations (`MCFG2.ramrmw`)
- EDAC protection[1] can be enabled/disabled (`MCFG3.re`)
- read/write access can be **`BRDY*`**-controlled (`MCFG2.rambrdy`) synchronously/asynchronously[2] (`MCFG1.abrdy`)

Notes: 1. EDAC protection activation common to SRAM and SDRAM interfaces.
2. Asynchronous **`BRDY*`**-control feature common to PROM, SRAM and I/O interfaces.

Figure 14. SRAM Bank Organization

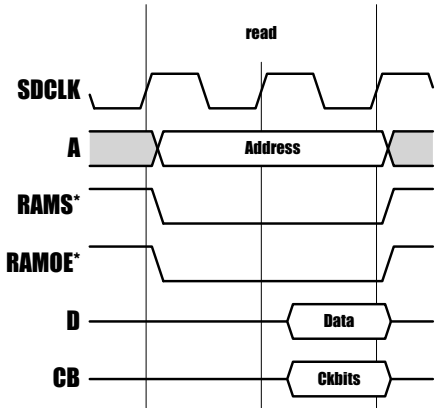
SRAM bank size	256MB	128MB	64MB
Start Address	***Memory assignement***	***Memory assignement***	***Memory assignement***
0x7C000000	**Unused**	**Unused**	**Unused**
0x78000000			
0x74000000			
0x70000000			
0x6C000000	**RAMS*[4][1][2]**	**RAMS*[4][2]**	**RAMS*[4][2]**
0x68000000			
0x64000000			
0x60000000			
0x5C000000	**RAMS*[1]**	**RAMS*[3]**	**Unused**
0x58000000			
0x54000000		**RAMS*[2]**	
0x50000000			
0x4C000000	**RAMS*[0]**	**RAMS*[1]**	**RAMS*[3]**
0x48000000			**RAMS*[2]**
0x44000000		**RAMS*[0]**	**RAMS*[1]**
0x40000000			**RAMS*[0]**

Notes: 1. When SRAM bank size is set to 256 MB, bank 2 and bank 3 overlap with bank 4. Because priority is given to bank 4, bank 2 and bank 3 control signals are never asserted.
2. When SDRAM is enabled, priority is given to the SDRAM over the SRAM. Any memory access above `0x60000000` is assigned to SDRAM and no SRAM control signal is asserted.

Read Access

An SRAM read access consists in two data cycles. A dedicated output enable signal is provided for each SRAM bank (**RAMOE*[]**) and is only asserted when that bank is selected.

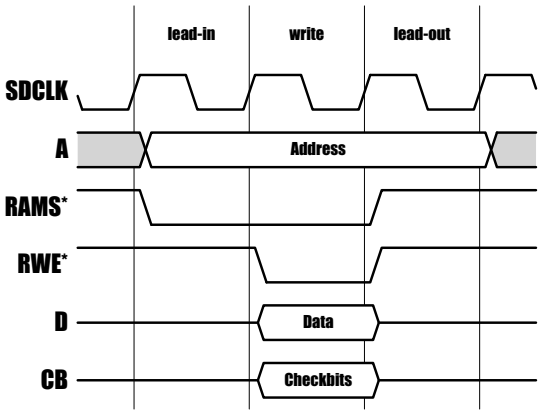
Figure 15. SRAM Read Access (no wait-states)



Write Access

An SRAM write access consists in an address lead-in cycle, a data write cycle and an address lead-out cycle. Each byte lane has an individual write strobe (`RAMWE*[]`) to allow efficient byte and half-word writes.

Figure 16. SRAM Write Access (no wait-states)

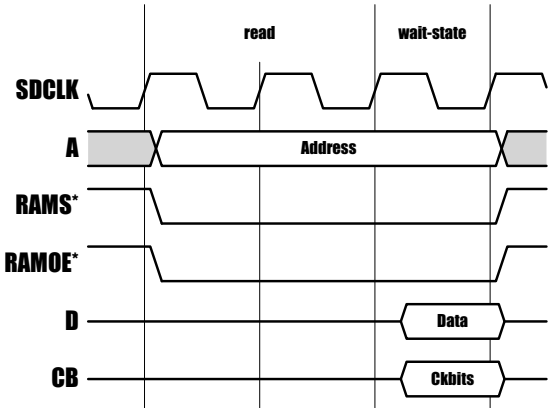


Caution: If EDAC protection is activated on the RAM area or a common write strobe is used for the full 32-bit data, read-modify-write shall be activated (`MCFG2.ramrmw`) so the EDAC checkbits integrity is preserved on sub-word writes.

Wait-States

For application accessing slow SRAM memories, the memory controller allows to insert wait-states during an SRAM read access (`MCFG2.ramrws`) and write access (`MCFG2.ramwws`). Up to 3 wait-states can be inserted.

Figure 17. SRAM Read Access with 1 Wait-State (`MCFG2.ramrws` = `1`)



SRAM read/write access to bank 4 (**`RAMS*[4]`**) can further be stretched when even more delay is needed (see "**`BRDY*`**-Controlled Access" later in this chapter").

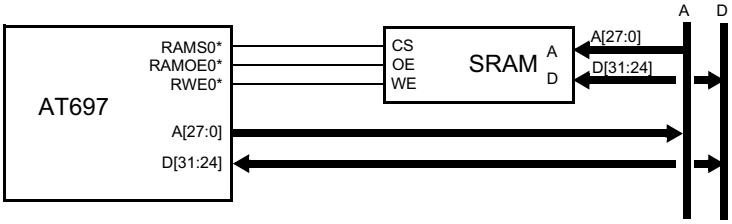
Data Bus Width

When configured for 32-bit SRAM (`MCFG2.ramwdh` = `00`), **`D[31:0]`** shall be connected to the memory device(s). **`CB[7:0]`** shall be connected as well if EDAC protection is activated (`MCFG3.pe` = `1`).

To support applications with limited memory and/or limited performance requirements, the SRAM area can be configured for 8-bit operations.

When configured for 8-bit SRAM (`MCFG2.ramwdh` = `00`), **`D[31:24]`** shall be connected to the memory device(s).

Figure 18. 8-bit SRAM Interface



On an 8-bit memory, 32-bit load/store instructions are always performed as a sequence of 4 consecutive memory accesses. If EDAC protection is activated, a 5th byte read access is performed as well to retrieve the checkbits (see "Error Management - EDAC" section later in this chapter). During a store operation, only the necessary bytes are written if EDAC protection is not activated. When EDAC protection is activated, the processor will always perform a full-word read-modify-write transaction on any sub-word store operation.

SDRAM Interface

Overview

The SDRAM controller allows addressing of up to 1 GB of SDRAM in 2 banks. SDRAM memory access control is performed using dedicated chip selects (**`SDCS*[1:0]`**), data masks (**`SDDQM[3:0]`**), byte-write enables (**`SDWE*[3:0]`**) and clock (**`SDCLK`**) signals.

SDRAM devices shall be 64 Mbit, 256 Mbit or 512 Mbit with 8 to 12 column-address bits, up to 13 row-address bits and exclusively 4 internal data banks. Only 32-bit data bus width is supported.

The SDRAMs address bus shall be connected to **A[14:2]** and the bank address to **A[16:15]**. Devices with less than 13 address pins should only use the less significant bits of **A[14:2]**.

SDRAM banks map as follows:

- **SDCS*[0]** and **SDCS*[1]** decode 2 contiguous banks with a common programmable size (4 MB to 512 MB)
- **SDCS*[1:0]** decode the upper half of the RAM area (0x60000000 - 0x7FFFFFFF) when the SRAM interface is enabled
- **SDCS*[1:0]** decode the lower half of the RAM area (0x40000000 - 0x5FFFFFFF) when the SRAM interface is disabled

The SDRAM interface is controlled in the memory configuration registers (MCFG2 and MCFG3):

- interface can be enabled/disabled (MCFG2.se)
- bank size can be set from 4 MB to 512 MB (MCFG2.sdrbs)
- column size can be set from 256 to 4096 (MCFG2.sdrcls)
- commands can be sent to the devices (MCFG2.sdrcmd)
- timings parameters can be set (MCFG2.sdrcas, MCFG2.trp and MCFG2.trfc)
- auto-refresh can be enabled/disabled (MCFG2.sdrref) and programmed (MCFG3.srcrv)
- EDAC protection[(1)(2)] can be enabled/disabled (MCFG3.re)

Notes: 1. EDAC protection activation common to SRAM and SDRAM interfaces.
2. Read-modify-write on sub-word operations simultaneously activated with EDAC.

Initialization

After reset, the SDRAM controller automatically performs an SDRAM initialization sequence. It consists in a PRECHARGE cycle, two AUTO-REFRESH cycles and a LOAD-MODE-REG cycle on both SDRAM banks simultaneously.

The controller programs the SDRAM to use *page burst* on read and *single location access* on write.

Timing Parameters

The SDRAM interface parameters can be programmed so read/write access to SDRAM devices is optimized:

Table 10. SDRAM Programmable Timing Parameters

Function	Parameter	Range	Unit	Control
CAS latency, RAS/CAS delay	t_{CAS}, t_{RCD}	2 - 3	clocks	MCFG2.sdrcas
Precharge to activate	t_{RP}	2 - 3	clocks	MCFG2.trp
Auto-refresh command period	t_{RFC}	3 - 11	clocks	MCFG2.trfc
Auto-refresh interval		10 - 32768	clocks	MCFG3.srcrv

Refresh

The SDRAM controller embeds a refresh module. When enabled (MCFG2.sdrref = 1), it periodically issues an AUTO-REFRESH command to both SDRAM banks with a programmable period (MCFG3.srcrv).

Depending on the SDRAM device used, the refresh period is typically 7.8 µs or 15.6 µs. The refresh period is calculated as

$$\textit{Refresh period} \quad \frac{\textit{Reload value} + 1}{\text{sdclk}_{\text{freq}}}$$

Commands

The SDRAM controller can issue three SDRAM commands (`MCFG2.sdrcmd`): `PRE-CHARGE`, `AUTO-REFRESH` and `LOAD-MODE-REG`. The command field is cleared after a command has been executed.

If the `LOAD-MODE-REG` command is issued, the programmed CAS delay is used (`MCFG2.sdrcas`) while remaining fields are fixed (*page read burst*, *single location write* and *sequential burst*).

Caution: *A `LOAD-MODE-REG` command shall be issued whenever the programmed CAS delay is updated.*

Read Access

A read access consists in several phases:

- an `ACTIVATE` command to the desired bank and row
- a **`READ`** command after the programmed CAS delay
- data read(s) (single or burst with no idle cycles if requested on the internal bus)
- a `PRE-CHARGE` command to terminate the access (no bank left open)

Write Access

A write access consists in several phases:

- an `ACTIVATE` command to the desired bank and row
- a `WRITE` command after the programmed CAS delay
- data read(s) (single or burst with no idle cycles if requested on the internal bus)
- a `PRE-CHARGE` command to terminate the access (no bank left open)

Write Protection

Two write protection schemes are provided to protect the RAM area against accidental over-writing: the “Start/End Address Scheme” and the “Tag/Mask Address Scheme”.

Start/End Address Scheme

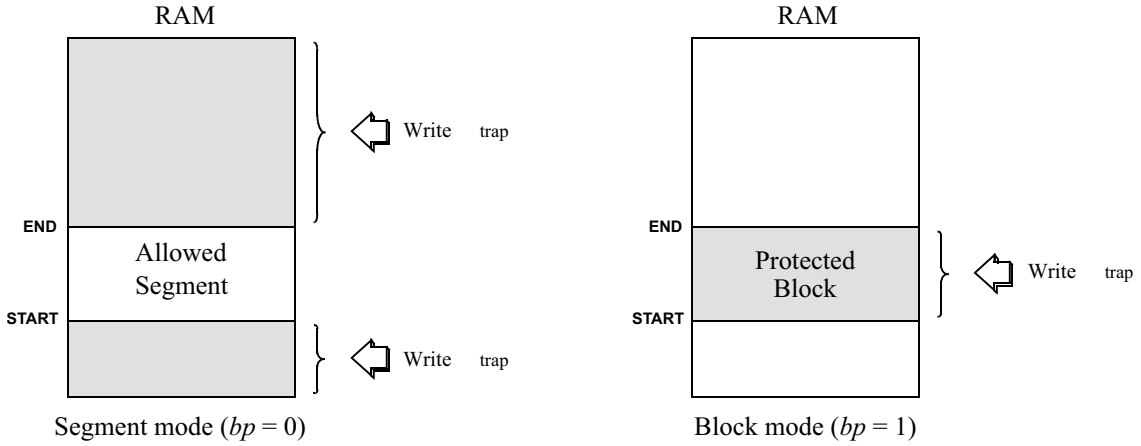
Start/End Address scheme protection is implemented as 2 write protection units capable of each controlling supervisor and/or user write access inside/outside of a memory segment of any arbitrary size.

Each unit *n* is configured by two write protection registers (`WPSTAn` and `WPSTOn`):

- the unit can be enabled/disabled[1] in supervisor mode[2] (`WPSTOn.su`) and in user mode[3] (`WPSTOn.us`)
- the segment is defined by a *START* address[4] (`WPSTAn.start`) and an *END* address[4] (`WPSTOn.end`)
- protection can be performed inside/outside the segment (`WPSTAn.bp`)

Notes:
1. The unit is enabled as soon as one of the two modes is enabled
2. The DSU communication interface has supervisor status when accessing the RAM area
3. The PCI interface (DMA and Target) has user status when accessing the RAM area
4. Address is a 32-bit word-aligned offset from the start of the RAM area (`0x40000000`)

Figure 19. Start/End Address Scheme Protection Overview



The protection is based on a segment of any arbitrary size in the RAM address space (4 bytes to 1 GB):

Table 11. Write Address Comparison

bit num	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	01		Most significant write address bits																												word	

The most significant bits of the write address are simply compared against the *START* and the *END* address of the segment (both boundaries included) to determine if the write address is inside the defined segment or in the block (outside of this segment).

If the write protection unit is enabled for the current IU mode (user or supervisor) and a block or segment protection error is detected, the write access is cancelled and a `write_error` trap (`0x2B`) is generated.

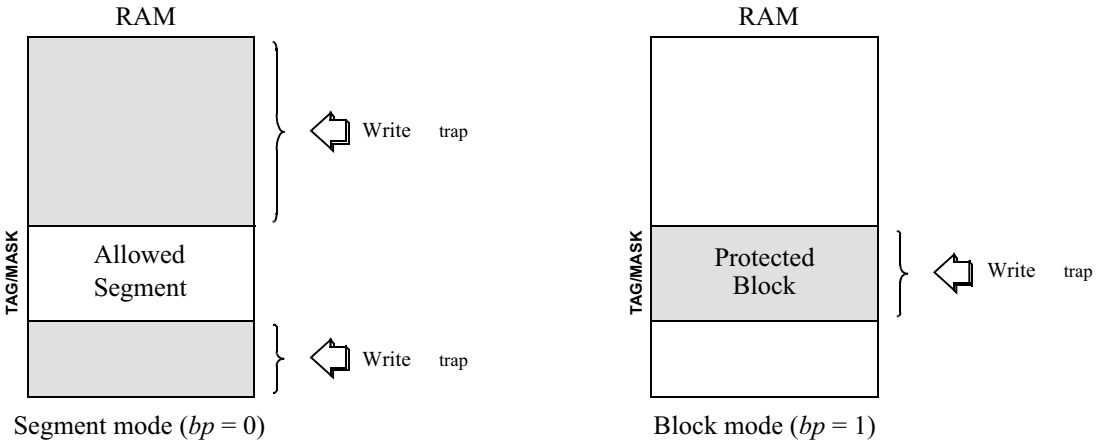
Tag/Mask Address Scheme

Tag/Mask address scheme protection is implemented as two write protection units capable of each controlling write access inside/outside of a binary aligned memory segment in the range of 32 KB - 1 GB.

Each unit *n* is configured by a write protection register (`WPRn`):

- the unit can be enabled/disabled (`WPRn.en`)
- a *TAG* specifies the 15 most significant bits of the segment address (`WPRn.tag`)
- a *MASK* specifies which bits within the *TAG* are relevant (`WPRn.mask`)
- protection can be performed inside/outside the segment (`WPRn.bp`)

Figure 20. Tag/Mask Address Scheme Protection Overview



The protection is based on a segmentation of the RAM address space to define a segment in the range of 32 KB up to 1 GB:

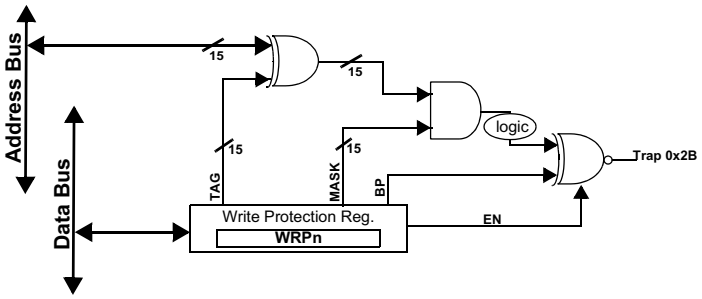
Table 12. Write Address Segmentation

bit num	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	01		Most significant write address bits															32 KB minimum area														

The most significant bits of the write address are `XOR`ed with the *TAG*, and the result is then `AND`ed with the *MASK*. If the final result is zero, the write address is in the defined segment, otherwise the write address is in the block (outside of this segment).

If the write protection unit is enabled and a block or segment protection error is detected, the write access is cancelled and a `write_error` trap (`0x2B`) is generated.

Figure 21. Segment/Block Protection Unit



Mixed Protection Schemes

It is possible to simultaneously use the Start/End Address and the Address/Mask write protection schemes. In that case (**at least one unit is enabled in each scheme**), the following rules applies:

- When **all enabled units are configured in block mode**, a `write_error` trap (`0x2B`) is generated **if at least one unit** signals a protection error.
- When **at least one enabled unit operates in segment mode**, a `write_error` trap (`0x2B`) is generated **only if all units configured in segment mode** signal a protection error.

BRDY*-Controlled Access

The **BRDY*** signal can be used to further stretch a read or write access and is enabled separately for the PROM area (`MCFG1.pbrdy`), the SRAM area decoded by **RAMS*[4]** (`MCFG2.rambrdy`) and the I/O area (`MCFG1.iobrdy`).

An access is always performed with at least the pre-programmed number of wait-states specified in the appropriate memory configuration register (`MCFG1` & `MCFG2`), but is further stretched until **BRDY*** is asserted.

Termination of a **BRDY***-controlled access can be performed in two different modes:

- synchronous mode (`MCFG1.abrdy` = `0`): **BRDY*** is sampled on the rising edge of the clock and shall meet the setup (t19) and hold (t20) timing constraints (see "AC Characterictics").
- asynchronous mode (`MCFG1.abrdy` = `1`): **BRDY*** shall be kept asserted for 1.5 clock cycle so it is guaranteed to meet at least one rising edge of the clock (setup/hold timing constraints do not apply anymore). Data in a read access shall be kept stable until de-assertion of the device select (`ROMS*[0]/ROMS*[1]` or **RAMS*[4]** or **IOS***, as appropriate) and output-enable (**OE*** or **RAMOE*[4]**, as appropriate) signals.

The access is terminated on the rising edge of the clock that immediately follows the detection of the asserted **BRDY***.

Figure 22. Synchronous **BRDY***-Controlled PROM Read Access (`MCFG1.abrdy`=0)

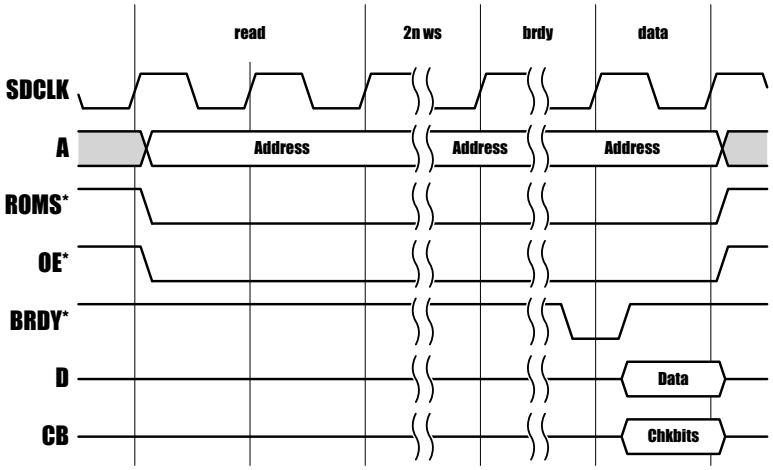


Figure 23. Asynchronous **BRDY***-Controlled PROM Read Access (`MCFG1.abrdy`=1)

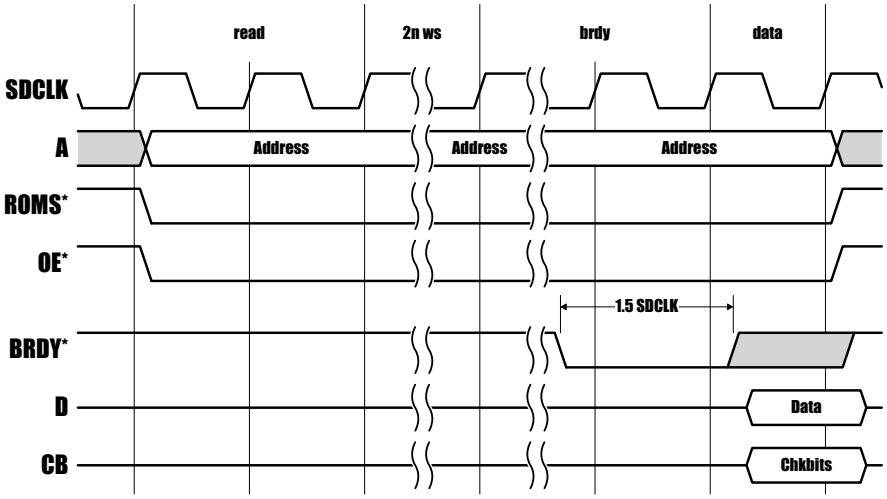


Figure 24. Synchronous **BRDY***-Controlled IO Read Access(`MCFG1.abrdy`=0)

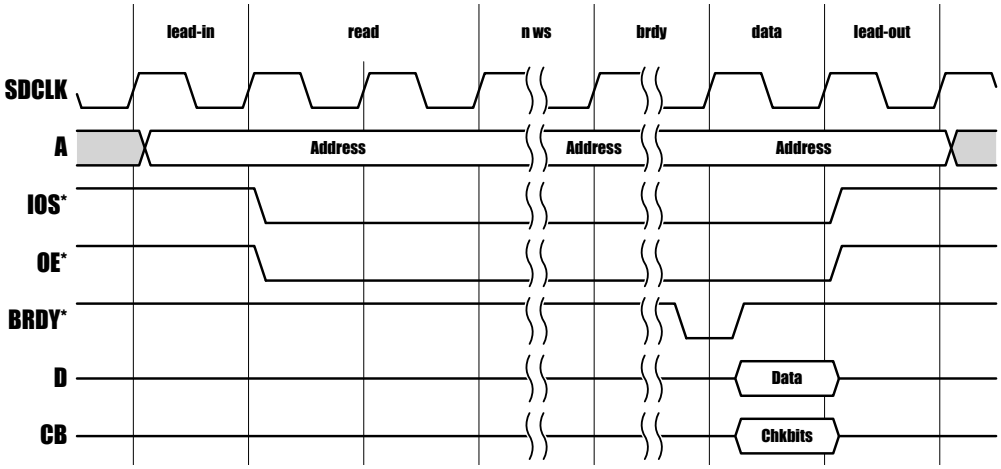


Figure 25. Asynchronous **BRDY***-Controlled IO Read Access (`MCFG1.abrdy`=1)

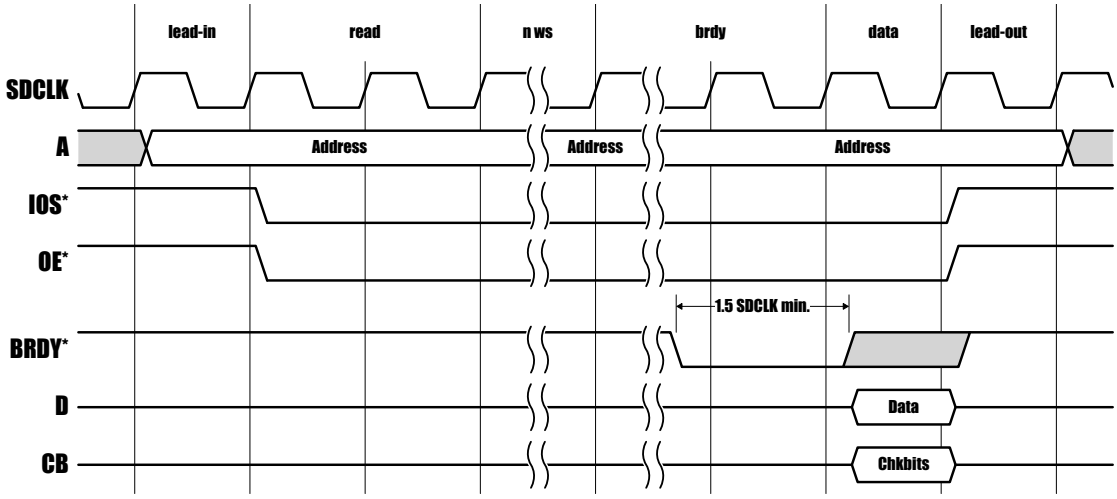


Figure 26. Synchronous **BRDY***-Controlled SRAM4 Read Access (`MCFG1.abrdy`=0)

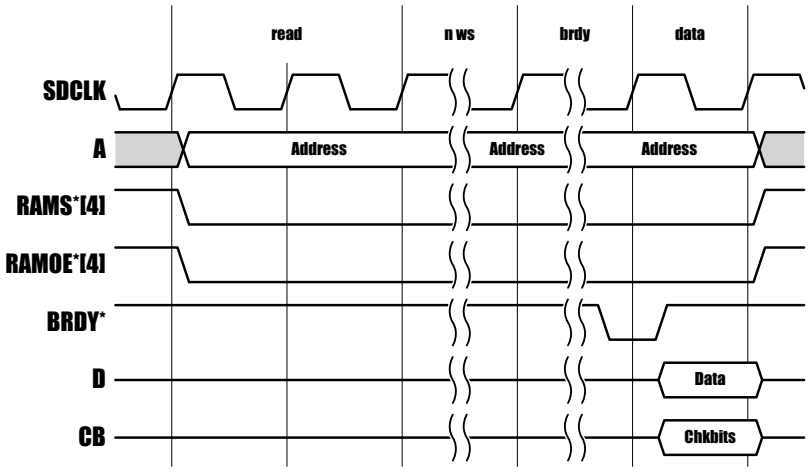
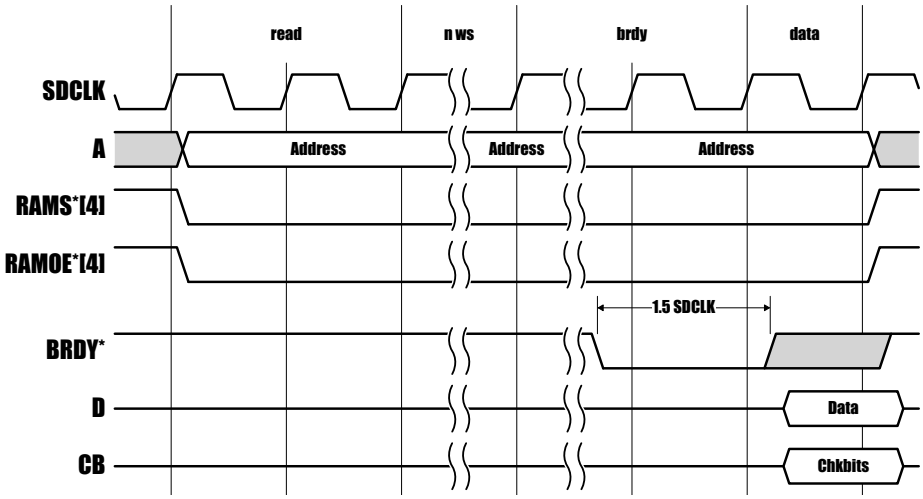


Figure 27. Asynchronous **BRDY***-Controlled SRAM4 Read Access (`MCFG1.abrdy`=1)



Bus Exception

A PROM, SRAM or I/O read/write access error can be signalled to the processor by asserting the **BEXC*** signal which is sampled together with the read/written data, if enabled in the memory controller (`MCFG1.bexc` = `1`):

- an `instruction_access_exception` trap (`0x01`) is generated if an instruction fetch is in progress
- a `data_access_exception` trap (`0x09`) is generated if a data load is in progress
- a `write_error` trap (`0x2B`) is generated if a data store is in progress

EDAC Management

Overview

The AT697F implements on-chip error detection and correction (EDAC). It can correct any single-bit error and detect double-bit errors in a 32-bit word.

EDAC Capability Mapping

EDAC is available on PROM, SRAM and SDRAM memory areas.

Table 13. External Memory EDAC Capability

Address Range	Area		EDAC Protected
0x00000000 - 0x1FFFFFFF	PROM	8 bits	yes
		32 bits	yes
0x20000000 - 0x3FFFFFFF	I/O	All	no
0x40000000 - 0x7FFFFFFF	SRAM	8 bits	yes
		32 bits	yes
	SDRAM	32 bits	yes

PROM Protection

When EDAC is activated on the PROM area[1] (`MCFG3.pe` = `1`), error detection and correction is performed on every instruction fetch and data load in that area.

Note: 1. Upon reset, EDAC on the PROM area is automatically configured from the value read on the **PIO[2]** pin. By driving **PIO[2]** high during reset, it is possible to enable EDAC on PROM on boot.

RAM Protection

When EDAC is activated on the RAM area[1][2] (`MCFG3.re` = `1`), error detection and correction is performed on every instruction fetch and data load in that area.

Notes: 1. When EDAC is enabled on the RAM area, read-modify-write on the SRAM (`MCFG2.ramrmw`) shall be enabled as well so the integrity of the EDAC checkbits is preserved on sub-word writes.
2. Activating EDAC on the RAM area automatically enables read-modify-write on sub-word writes to the SDRAM.

Caution: *The RAM area shall always be initialized with 32-bit word writes prior to EDAC activation so further sub-word writes (performed as 32-bit read-modify-write atomic operations) always successfully pass the initial read & check step (see "Read Access").*

Operation

When enabled, the EDAC operates on every access to the external memory.

Hamming Code

For each word, a 7-bit checksum is generated according to the following equations:

$$CB0 = D0 \wedge D4 \wedge D6 \wedge D7 \wedge D8 \wedge D9 \wedge D11 \wedge D14 \wedge D17 \wedge D18 \wedge D19 \wedge D21 \wedge D26 \wedge D28 \wedge D29 \wedge D31$$
$$CB1 = D0 \wedge D1 \wedge D2 \wedge D4 \wedge D6 \wedge D8 \wedge D10 \wedge D12 \wedge D16 \wedge D17 \wedge D18 \wedge D20 \wedge D22 \wedge D24 \wedge D26 \wedge D28$$
$$CB2 = \overline{D0 \wedge D3 \wedge D4 \wedge D7 \wedge D9 \wedge D10 \wedge D13 \wedge D15 \wedge D16 \wedge D19 \wedge D20 \wedge D23 \wedge D25 \wedge D26 \wedge D29 \wedge D31}$$
$$CB3 = \overline{D0 \wedge D1 \wedge D5 \wedge D6 \wedge D7 \wedge D11 \wedge D12 \wedge D13 \wedge D16 \wedge D17 \wedge D21 \wedge D22 \wedge D23 \wedge D27 \wedge D28 \wedge D29}$$
$$CB4 = D2 \wedge D3 \wedge D4 \wedge D5 \wedge D6 \wedge D7 \wedge D14 \wedge D15 \wedge D18 \wedge D19 \wedge D20 \wedge D21 \wedge D22 \wedge D23 \wedge D30 \wedge D31$$
$$CB5 = D8 \wedge D9 \wedge D10 \wedge D11 \wedge D12 \wedge D13 \wedge D14 \wedge D15 \wedge D24 \wedge D25 \wedge D26 \wedge D27 \wedge D28 \wedge D29 \wedge D30 \wedge D31$$

CB6 = D0 ^ D1 ^ D2 ^ D3 ^ D4 ^ D5 ^ D6 ^ D7 ^ D24 ^ D25 ^ D26 ^ D27 ^ D28 ^ D29 ^ D30 ^ D31

Write Access

When the processor performs a write access to an EDAC protected memory, it also outputs the 7-bit EDAC checkbits on the **CB[6:0]** pins (**CB[7]** always driven low unless EDAC testing is enabled).

Read Access

When the processor performs a read access to an EDAC protected memory, the checkbits read together with the data are compared against checkbits generated by the EDAC from the same read data. Any discrepancy yields an error and a syndrome is computed to further qualify the error as correctable (single-bit error) or uncorrectable (double-bit error).

Correctable Error

A single-bit error qualifies as a correctable error. The correction is performed on-the-fly inside the processor during the current access and no timing penalty is incurred.

The correctable error event is reported in the fail address register (`FAILAR`) and in the fail status register (`FAILSR`). If unmasked, interrupt 1 (trap `0x11`) is generated.

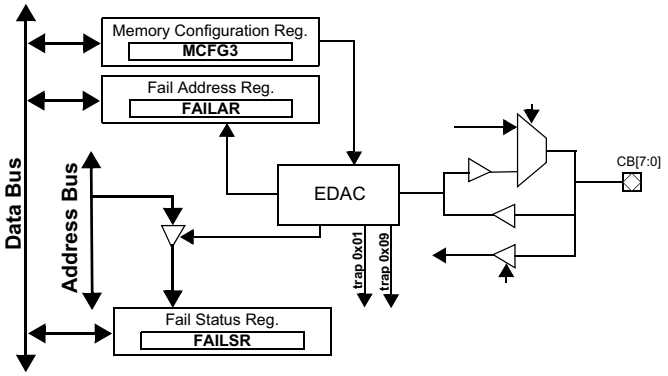
Caution: *The single-bit error remains in memory until a software-initiated rewrite is performed at the faulty memory location.*

Uncorrectable Error

A double-bit error qualifies as an uncorrectable error:

- an `instruction_access_exception` trap (`0x01`) is generated if an instruction fetch is in progress
- a `data_access_exception` trap (`0x09`) is generated if a data load is in progress

Figure 28. EDAC overview



EDAC on 8-bit Memories

EDAC protection on 8-bit memories can be performed as well. CB[7:0] is not used and the EDAC checkbits are stored in the upper part of the 8-bit memory bank where the protected data reside.

When EDAC is enabled:

- an instruction fetch or a data load is performed as a burst of 4 read access to retrieve the 4 bytes and a 5th read access to retrieve the checkbits
- any word and sub-word store can be performed in RAM
- only byte store shall be performed in PROM

The protected memory bank is partitioned as follows:

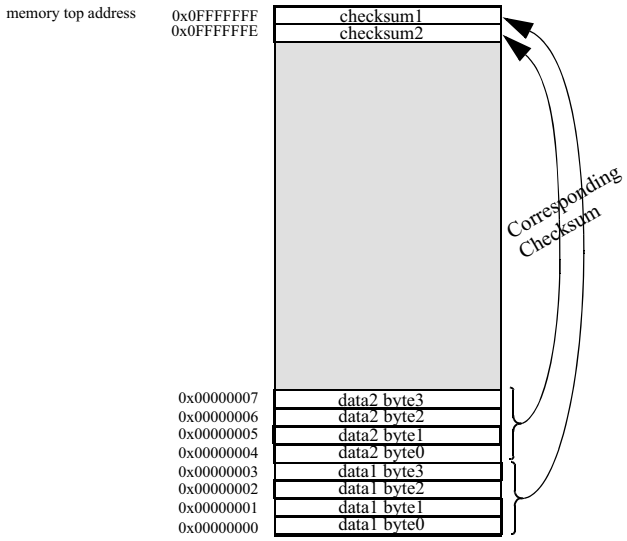
- lower 80% of the memory bank available as program or data memory
- upper 20% of the memory bank allocated to the EDAC checkbits (a maximum of 4 unusable bytes before the checkbit area)

Accessing the EDAC checkbits is performed as follows:

- start address from the topmost byte in the same memory bank (no bank size information needed)

  $\texttt{addr}_{checkbits} = \texttt{addr}_{bank\text{-}top} - ((\texttt{addr}_{data} - \texttt{addr}_{bank\text{-}start}) / 4)$
- checkbits bytes allocated downwards (address bits inversion technique used)

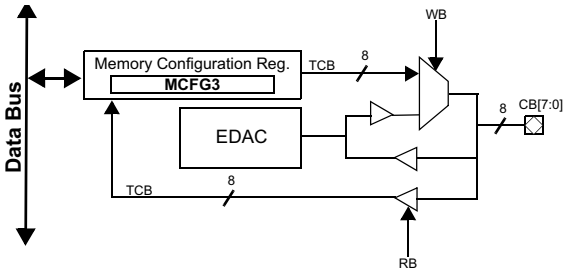
Figure 29. 8-bit EDAC-Protected Memory Organization



EDAC Testing

Operation of the EDAC can be bypassed for testing purpose and is controlled in a memory configuration register (`MCFG3`).

Figure 30. EDAC testing overview



Write Test

When EDAC write bypass is enabled (`MCFG3.wb` = `1`), the test checkbits (`MCFG3.tcb`) replace the EDAC generated checkbits during a data store.

Read Test

When EDAC read diagnostic is enabled (`MCFG3.rb` = `1`), the test checkbits (`MCFG3.tcb`) are updated[1] with the read checkbits during a data load or an instruction fetch.

Note: 1. The EDAC test routine shall be executed entirely from the instruction cache (when activated) or from an area without EDAC activated and different from the one being

accessed. Otherwise the checkbits read during instruction fetch will overwrite those from the area to be tested.

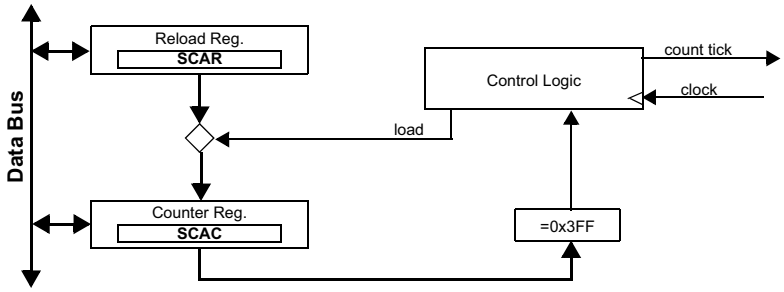
Timer Unit

The timer unit implements two 32-bit timers, one 32-bit watchdog and one 10-bit shared prescaler.

Prescaler

The prescaler is an internal device shared by the two timers and the watchdog.

Figure 31. Prescaler Block Diagram



The prescaler operation is controlled by two registers (`SCAC` and `SCAR`):

- the prescaler is always enabled
- the counter (`SCAC.cnt`) is clocked by the system clock and decremented on each clock cycle
- the counter is reloaded from the prescaler reload register (`SCAR.rl`) after it underflows and a tick pulse is generated for the two timers and the watchdog
- after reset, the prescaler counter & reload registers are initialized to the minimum division rate

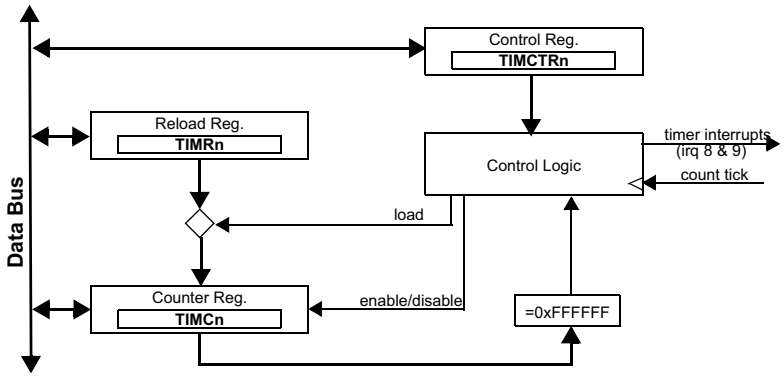
The effective division rate is therefore equal to the prescaler reload register value + `1`.

Caution: The two timers and the watchdog share the same decrementer, so the minimum possible prescaler division rate is 4 to allow processing of the two timers and the watchdog.

Timer 1 & Timer 2

Timer1 and Timer2 are two general purpose 32-bit timers. They share the same decrementer with the watchdog.

Figure 32. Timer 1/2 Block Diagram



Each timer *n* operation is controlled by a dedicated set of timer control registers (`TIMCTRn`, `TIMRn` and `TIMCn`):

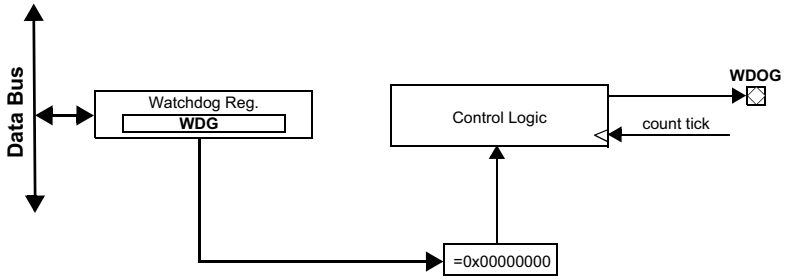
- a timer can be enabled/disabled (`TIMCTRn.en`)
- the counter (`TIMCn.cnt`) is decremented each time the prescaler generates a tick pulse
- the counter can be manually loaded (`TIMCTRn.ld`) from the reload register (`TIMCTRn.rv`)
- the counter can be configured to stop or to automatically reload (`TIMRn.rl`) after it underflows

Each time a timer underflows, a timer-dedicated interrupt is generated (`ITP.ipend[]`) if unmasked in the interrupt mask and priority register (`ITMP.mask[]`).

Watchdog

The watchdog is a specific 32-bit timer (decrementer shared with Timer1 and Timer2).

Figure 33. Watchdog Block Diagram



The watchdog is accessible through a single watchdog register (`WDG`):

- the watchdog is always enabled
- the counter (`WDG.cnt`) is decremented each time the prescaler generates a tick pulse unless it has reached zero
- the **`WDOG*`** signal is asserted when the counter expires at zero (no other internal event generated)
- the counter never underflows and shall be refreshed by directly reloading a value into the counter
- after reset, the watchdog counter is initialized to the maximum possible value[1]

Note: 1. Considering the prescaler is initialized to the minimum value after a reset (a division rate of 4), the watchdog will expire after $(2^{32} - 1)\times 4$ cycles, unless later programmed otherwise.

The watchdog can be used to generate a system reset on expiration by directly connecting the **`WDOG*`** open-drain output pin to the **`RESET*`** pin.

UART Interface

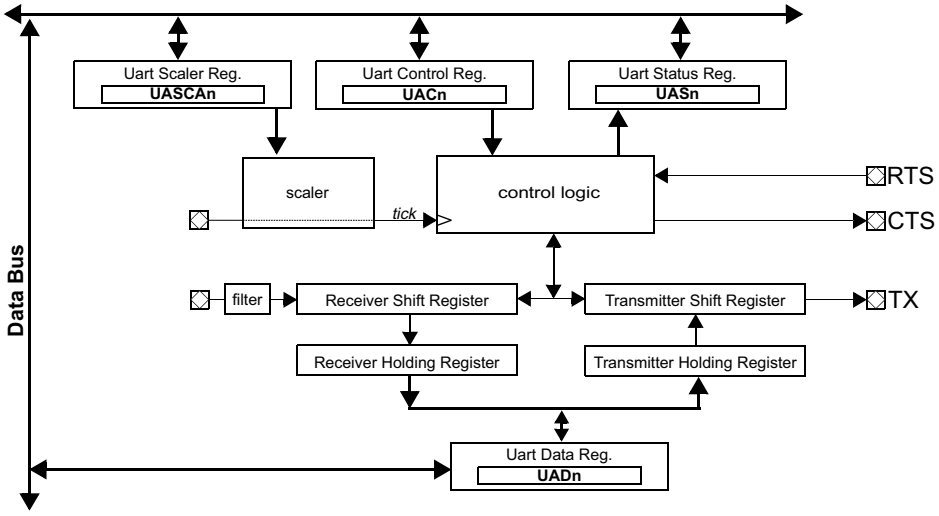
The Universal Asynchronous Receiver and Transmitter (UART) is a highly flexible serial communication module. The AT697F implements two uarts: UART1 and UART2. UARTs on the processor are defined as alternate functions of the general purpose interface (GPI).

Overview

Each UART *n* operates independently and is fully controlled by a set of 4 registers:

- a control register (`UACn`)
- a status register (`UASn`)
- a scaler register (`UASCAn`)
- a data register (`UADn`)

Figure 34. UART Block Diagram

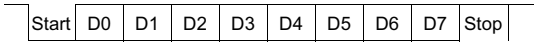


Data Frame

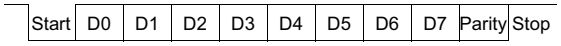
A data frame consists in a start bit, 8 data bits, an optional parity bit and a stop bit.

Figure 35. Data Frames

Data frame, no parity:



Data frame with parity:



Parity in a data frame is controlled as follows:

- the parity can be enabled or disabled (`UACn.pe`)
- when enabled (`UACn.pe` = `1`), the parity can be even or odd (`UACn.ps`)

When even (`UACn.ps` = `1`), the parity bit is generated such as the number of `1`s in the data and the parity is even. When odd (`UACn.ps` = `0`), the parity bit is generated such as the number of `1`s is odd.

Baud-Rate

The internal baud-rate generator requires a clock source to operate, which can either be internal or external (`UACn.ec`).

Internal Clock

When configured for internal clock (`UACn.ec` = `0`), the UART baud-rate comes from a programmable 12-bits scaler controlled by a configuration register (`UASCAn`):

- the scaler is enabled only when the UART transmitter (`UACn.te` = `0`) and/or the UART receiver (`UACn.re` = `0`) are enabled
- when enabled (`UACn.te` = `1` and/or `UACn.re` = `1`), the scaler counter is clocked by the system clock and decremented on each clock cycle
- the scaler counter is reloaded from the scaler reload register (`UASCAn.rv`) after it underflows and a UART tick is generated for the transmitter and the receiver (tick frequency is 8 times the desired baud-rate)

The following equations shall be used to calculate the scaler value or the baudrate value based on the clock frequency:

$$scaler_{rv} = \frac{sdclk_{freq}}{baudrate \times 8} - 1$$

$$baudrate = \frac{sdclk_{freq}}{8 \times (scaler_{rv} + 1)}$$

variable description:

- ***sdclk*$_{freq}$**: internal clock frequency
- ***baudrate***: targeted/resulting baud rate
- ***scaler*$_{rv}$**: resulting/targeted scaler reload value

External Clock

When configured for an external clock (`UACn.ec` = `1`), the UART scaler is bypassed and **`PIO[3]`** directly provides the UART tick to the transmitter and the receiver (tick frequency is 8 times the desired baud-rate).

The external clock frequency shall be 8 times the desired baud-rate.

Caution: *When configured for an external clock source, the clock high and low time on **`PIO[3]`** shall each be longer than the period of the internal system clock (so proper sampling is achieved).*

Double Buffering

Each UART performs double-buffering (a holding register and a shift register) on the transmitter and the receiver to optimize the data transfer in both directions (no transmitter stopped waiting for reload, no data loss on receiver overrun).

Hardware Flow-Control

Each UART *n* can perform hardware flow-control to further optimize and secure data transfer in both directions:

- hardware flow-control can be enabled or disabled (`UACn.fl`)
- when enabled (`UACn.fl` = `1`) together with the transmitter (`UACn.te` = `1`), no new data transmit is initiated until the *clear-to-send* input pin (`CTSn`) is asserted (data transmission is not interrupted is deasserted in the middle of the transmission)
- when enabled (`UACn.fl` = `1`) together with the receiver (`UACn.re` = `1`), the *request-to-send* output pin (`RTSn`) is asserted as long as new data can be received

Noise Filtering

The serial input is shifted through an 8-bit filter which changes output only when all bits in the filter have the same value, effectively forming a low-pass filter with a cut-off frequency of 1/8 system clock.

Operation

Transmitter Operation

UART *n* transmitter is first configured as follows:

- the transmitter shall be enabled (`UACn.te` = `1`)
- the transmitter *serial-output* pin (`TXn`) shall be enabled on the general-purpose interface by configuring the appropriate pin to output mode (**`PIO[15]`** for UART1 and **`PIO[11]`** for UART2)
- if flow-control is enabled (`UACn.fl` = `1`), the transmitter *clear-to-send* pin (`CTSn`) shall be enabled on the general-purpose interface by configuring the appropriate pin to input mode (**`PIO[12]`** for UART1 and **`PIO[8]`** for UART2)

When ready to transmit, data written to the transmitter holding register (`UADn.rtd`) is transferred into the transmitter shift register and converted to a serial data frame on the transmitter serial output pin (`TXn`).

Following the transmission of the stop bit, the transmitter serial data output remains high and the transmitter shift register empty flag is asserted (`UASn.ts` = `1`) if no new data is available in the transmitter holding register.

Transmission resumes and the transmitter shift register empty flag is deasserted (`UASn.ts` = `0`) when new data is loaded in the transmitter holding register (`UADn.rtd`).

Receiver Operation

UART *n* receiver is first configured as follows:

- the receiver shall be enabled (`UACn.re` = `1`)
- the receiver *serial-input* pin (`RXn`) shall be enabled on the general-purpose interface by configuring the appropriate pin (**`PIO[14]`** for UART1 and **`PIO[10]`** for UART2) to input mode
- if flow-control is enabled (`UACn.fl` = `1`), the receiver *request-to-send* pin (`RTSn`) shall be enabled on the general-purpose interface by configuring the appropriate pin (**`PIO[13]`** for UART1 and **`PIO[9]`** for UART2) to output mode

The receiver constantly looks for the high to low transition of a start bit on the receiver serial data input pin (`RXn`). If a transition is detected, the state of the serial input is sampled a half-bit later for confirmation and a valid start bit is assumed if the serial input is still low, otherwise the search for a valid start bit continues.

Then the receiver continues to sample the serial input at one bit time intervals (at the theoretical centre of the bit) until the proper number of data bits and optionally the parity bit have been assembled and one stop bit has been detected.

The data is transferred to the receiver holding register and the *data ready* flag is asserted (`UASn.dr` = `1`) by the end of the reception if no error was detected. Otherwise, no *data ready* flag is asserted and the error is reported in the appropriate flag:

- a *parity error* (`UASn.pe` = `1`) occurs when parity is enabled (`UACn.pe` = `1`) and the received parity does not match the selected parity configuration (`UACn.ps`)
- a *framing error* (`UASn.fe` = `1`) occurs when the received stop bit is a `0` rather than a `1`
- a *break received* (`UASn.br` = `1`) occurs when the received data and the stop bit are all `0`s
- an *overrun error* (`UASn.ov` = `1`) occurs when the holding register already contains an un-read data

Caution: *The errors bits are never cleared by the receiver and shall be cleared in software so new errors can later be detected.*

Reading the UART data register (`UADn.rtd`) empties the receiver holding register and deasserts the *data ready* flag (`UASn.dr` = `0`).

Interrupt Generation

Each UART *n* can be configured to generate an interrupt each time a byte has been received and/or is about to be sent:

- transmitter interrupt can be enabled/disabled (`UACn.ti`)
- receiver interrupt can be enabled/disabled (`UACn.ri`)

When enabled (`UACn.ti` = `1`), the transmitter issues an interrupt when the transmitter holding register is emptied (transfer into the transmitter shift register for sending).

When enabled (`UACn.ri` = `1`), the receiver issues an interrupt after serial data has been received (data made ready into the receiver holding register or errors reported).

Note: *The interrupt is made effective (`ITP.ipend[3]` for UART1 and `ITP.ipend[2]` for UART2) only if unmasked in the interrupt mask and priority register (`ITP.imask[3]` for UART1 and `ITP.imask[2]` for UART2).*

Loop-Back Mode

Each UART *n* can be configured in *loop-back* mode (`UACn.lb`) for testing purpose.

When enabled(1) (`UACn.lb` = `1`), the transmitter *serial-output*(2) is internally connected to the receiver *serial-input*(3) and the receiver *request-to send* output(4) is internally connected to the transmitter *clear-to-send* input(5).

Notes:
1. In *loop-back* mode, the corresponding general-purpose I/O pins need not be configured since all the connections are directly performed internally.
2. If the transmitter is enabled and the corresponding general purpose I/O pin (`TXn`) is configured as an output, a constant `1` is output instead of the programmed I/O data.
3. No *parity error* or *framing error* or *break received* can be generated since the transmitter and the receiver both share the same parity and baud-rate configuration.
4. If flow-control is enabled and the corresponding general purpose I/O pin (`RTSn`) is configured as an output, a constant `1` is output instead of the programmed I/O data.
5. No *overrun error* can be generated if flow-control is enabled.

General Purpose Interface

The general purpose interface (GPI) consists in a partially bit-wise programmable 32-bit I/O port with alternate facilities.

GPI as a 32-bit I/O Port

The port is split in two parts - the lower 16-bits are accessible via the **PIO[15:0]** pins while the upper 16-bits are accessible via **D[15:0]** and can only be used when all the external memory areas (ROM, SRAM and I/O) are in 8-bit mode (see “8-bit PROM and SRAM Access”). If the SDRAM controller is enabled, the upper 16-bits cannot be used .

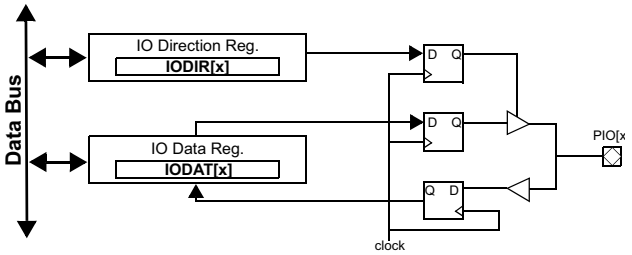
Lower 16-bits Operation

The lower 16 bits of the I/O port can be individually programmed as output or input, they are accessible through **PIO[15:0]**.

Each pin *n* in **PIO[15:0]** is controlled by two registers (IODIR and IODAT):

- the pin can be configured as an input or an output (IODIR.piodir[*n*])
- when configured as an input (IODIR.piodir[*n*] = 0), the bit value in the data register (IODAT.piodat[*n*]) continuously reflects the pin value
- when configured as an output (IODIR.piodir[*n*] = 1), the bit value in the data register (IODAT.piodat[*n*]) is continuously output on the pin

Figure 36. I/O Port Block Diagram - **PIO[15:0]**



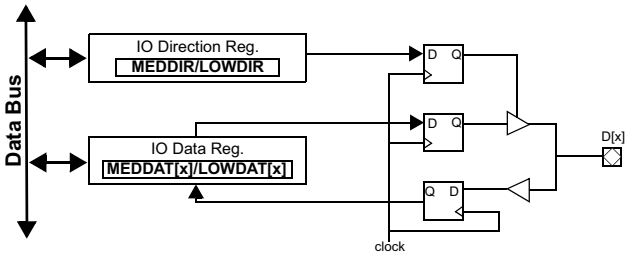
Upper 16-bits Operation

The upper 16 bits of the I/O port can only be configured as outputs or inputs on a byte basis. **D[15:8]** is referenced as the *medium* byte while **D[7:0]** is referenced as the *lower* byte.

Each byte in **D[15:0]** is controlled by 2 registers (IODIR and IODAT):

- the whole byte can be configured as an input or an output (IODIR.meddir and IODIR.lowdir)
- when configured as an input (IODIR.meddir = 0 and/or IODIR.lowdir = 0), the byte value in the data register (IODAT.meddat and IODAT.lowdat) continuously reflects the corresponding pins byte value
- when configured as an output (IODIR.meddir = 1 and/or IODIR.lowdir = 1), the byte value in the data register (IODAT.meddat and IODAT.lowdat) is continuously output on the corresponding byte pins

Figure 37. I/O Port Block Diagram - **D[15:0]**



GPI Alternate Functions

Most GPI pins have alternate functions in addition to being general I/O. Facilities like serial communication link, interrupt input and configuration are made available through these functions. The following table summarizes the assignement of the alternate functions.

Table 14. GPI alternate functions

GPI port pin	Alternate function
PIO[15]	TXD1 - UART1 transmitter data(1)(3)
PIO[14]	RXD1 - UART1 receiver data(2)(4)
PIO[13]	RTS1 - UART1 request-to-send(1)(4)(5)
PIO[12]	CTS1 - UART1 clear-to-send(2)(3)(5)
PIO[11]	TXD2 - UART2 transmitter data(1)(3)
PIO[10]	RXD2 - UART2 receiver data(2)(4)
PIO[9]	RTS2 - UART2 request-to-send(1)(4)(5)
PIO[8]	CTS2 - UART2 clear-to-send(2)(3)(5)
PIO[3]	EXTCLK - Use as alternative UART clock(2)
PIO[2]	PROM EDAC enable - Enable EDAC protection on boot(6)
PIO[1:0]	PROM width - Defines PROM data bus width on boot(6)

Notes:
1. The corresponding GPI port pin shall be configured in output mode so the UART output signal is effective on that pin.
2. The corresponding GPI port pin shall be configured in input mode so the UART input signal is effective on that pin.
3. The corresponding UART transmitter shall be enabled
4. The corresponding UART receiver shall be enabled
5. Flow-control shall be enabled on the corresponding UART
6. Pin is sampled during reset and can be used as a general purpose I/O pin after reset

In addition to these alternate functions, each GPI interface pin can be configured as an interrupt input to catch interrupts from external devices. Up to eight interrupts can be configured on the GPI interface by programming the I/O interrupt registers (IOIT1 and IOIT2).

For a detailed description about the external interrupts configuration, please refer to the “Traps and Interrupts” section.

PCI Arbiter

The embedded PCI arbiter enables the arbitration of up to 4 PCI agents (numbered from 0 to 3). A round-robin algorithm is implemented as arbitration policy.

Since the PCI interface arbitration logic is not connected internally to the PCI arbiter, the **REQ*/GNT*** signals shall be connected externally to one of the **AREQ*[]/AGNT*[]** pairs of the arbiter so the PCI interface is arbitered amongst the other agents on the bus.

The PCI interface can also be operated with an external PCI arbiter, thus not using the internal arbiter (the **AREQ*[3:0]** input signals shall then be tied to a high level).

Operation

An agent on the PCI bus requests the bus by driving low one of the **AREQ*[]** signal. When the arbiter determines the bus can be granted to an agent, it drives low the corresponding **AGNT*[]** signal.

The agent is only granted the PCI bus for one transaction. An agent willing further access to the bus shall continue to assert its **AREQ*[]** line and wait to be granted the bus again.

Policy

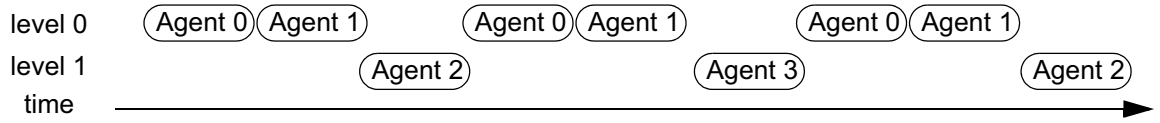
The arbitration policy is based on a round-robin algorithm with two nested priority loops. A high priority loop is defined as level 0, a low priority loop is defined as level 1.

Agents 0,1 and 2 can be individually configured to operate either on level 0 or on level 1 in the PCI Arbiter register (PCIA), whereas agent 3 operates on the fixed level 1 (low priority).

Operation

The arbitration is performed by checking the **AREQ*[3:0]** signals one after the other. In the first place, only agents with level 0 (high priority) are considered. If an agent asserts its **AREQ*[]** signal and the bus is not already granted, the corresponding **AGNT*[]** signal is driven low to grant the agent the bus. After a complete round-turn in level 0, a complete turn is done in level 1. The following figure illustrates the operation of the arbiter:

Figure 38. Arbiter Operation



With : agents 0 and 1 at level 0
agents 2 and 3 at level 1

Considering only agents submitting a request at the same time, the odds for being granted the bus can be summarized as follows:

- All agents in the same level have equal probability of grant
- All agents in level 1 have the same cumulated probability of grant as a single agent in level 0

Re-arbitration

Re-arbitration occurs as soon as a transfer is finished and a new request is made (the PCI arbiter has internal knowledge of the **FRAME*** signal) or when no agent is requesting the bus anymore (leading to bus parking).

Caution: No re-arbitration occurs during a transfer. Long bursts of one agent, even if assigned a low priority, can therefore significantly deteriorate the bandwidth available for other agents, especially the ones assigned a high priority.

In time critical systems, splitting long bursts into smaller chunks shall be considered as a way to favor re-arbitration more often.

Bus Parking

As long as no bus request is active, the bus always remains granted (parked) to the last owner until another agent requests the bus.

After reset, the bus is automatically granted (parked) to agent 0.

PCI Interface

Overview

The PCI interface implementation is compliant with the PCI specification revision 2.2. It is a high performance 32-bit bus interface with multiplexed address and data lines. It is intended for use as an interconnect mechanism between processor/memory systems and peripheral controller components.

The PCI interface has initiator (master) and target capability, and data transfer can be in transmit (from AT697 to PCI bus) or in receive (from PCI to AT697) direction.

The PCI bus can be operated at a frequency up to 33 MHz independently of the processor clock. The PCI clock domain and the processor clock domain are fully decoupled, allowing the processor clock to be faster, equal or slower than the PCI clock. Data transfer is through 4 synchronizing data FIFOs of 8 words each:

- MXMT: master/initiator-transmit-FIFO (from AT697 to PCI bus, for store instructions)
- MRCV: master-receive-FIFO (from PCI bus to AT697, for load instructions)
- TXMT: target-transmit-FIFO (from AT697 memory to PCI bus, PCI-read)
- TRCV: target-receive-FIFO (from PCI bus to AT697 memory, PCI-write)

Depending on the configuration mode, the lower part of the PCI configuration registers can be accessed either locally in the register address space (address `0x80000100` to `0x80000144`) or by another PCI device via the PCI bus with PCI configuration cycles and the **`IDSEL`** signal (the AT697 can never access its own configuration registers via the PCI bus). The upper part of the PCI configuration registers (`0x80000148` to `0x80000178`) and the PCI arbiter register `PCIA` (`0x80000280`) can only be accessed locally through the register address space. The configuration mode is selected by a hardware bootstrap on the **`SYSEN*`** pin. The following two modes are available:

- **Host-Bridge (`SYSEN*` = `0`)**

  In host-bridge mode, the PCI registers at address `0x80000100` to `0x80000144` are only accessible locally by the AT697 processor, but not through the PCI bus. The host-bridge is sometimes also called *System Controller*, it controls other satellite devices through PCI configuration commands.

- **Satellite (`SYSEN*` = `1`)**

  In satellite mode, the lower part of the AT697 PCI registers can be written and read by another PCI device (the host-bridge) using PCI configuration cycles, whereas the local registers addresses `0x80000100` to `0x80000144` are read-only.

The state of the **`SYSEN*`** pin is available internally (`PCIIS.sys`) to enable a boot software to load the appropriate driver(s).

Some other features are supported by this interface like

- Target lock
- Target zero-latency fast back-to-back transfers
- Zero wait-state burst mode transfers
- Memory read line/multiple
- Memory write and invalidate
- Delayed read
- Flexible error reporting by polling

PCI Initiator (Master)

PCI initiator transactions are issued by the processor either as memory-mapped load/store instructions or as DMA-assisted data transfers between the PCI bus and the local memory.

Load/store instructions to a memory address in the PCI area (`0xA0000000` - `0xFFFFFFFF`) are automatically translated by the interface into the appropriate PCI transaction. Any PCI address outside of this range can only be accessed via a DMA-assisted data transfer.

The PCI initiator (memory-mapped or DMA-assisted) is enabled by setting bits `PCISC.com2` and `PCIIC.mod`.

Memory-Mapped Access

Instructions of different width (byte, half-word, word or double-word) can be performed for each address of the PCI address range. The three least-significant bits of the address (**`A/D[2:0]`**) are used to determine which PCI byte-enable signals (**`C/BE*[3:0]`**) should be active during the transaction.

According to the SPARC architecture, big-endian mapping is implemented where the most significant byte standing at the lower address (`0x..00`) and the least significant byte standing to the upper address (`0x..03`).

Writing a byte to a PCI word-aligned address (**`A/D[1:0]`** = `00`) results in the byte-enable pattern (**`C/BE*[3:0]`** = `0111`) indicating the most significant byte lane (**`A/D[31:24]`**) of the PCI data bus is selected.

For all sub-word load instructions using a PCI memory command, the byte enables are all-0s, assuming reading more bytes than necessary has no side effects on a prefetchable target. Non-prefetchable targets where exact read byte-enables are required should be accessed with PCI I/O commands.

Byte, half-word and word size load/store instructions are translated into a single word PCI transaction with the appropriate byte-enable pattern, while a double-word load/store instruction are translated into a 2-word burst PCI transaction.

The following table presents the mapping between instructions and PCI byte enables generated for memory write and I/O read/write commands:

Table 15. Byte-Enable(1) vs Instruction

Bit Width	8	16	32	64
Instruction	LDSB, LDUB, STB	LDSH, LDUH, STH	LD, ST	LDD, STD
A_i[2:0]=000(4)	0111	0011	0000	0000(2)
A_i[2:0]=100(4)				*n/a*(3)
A_i[1:0]=01(4)	1011	*n/a*(3)	*n/a*(3)	*n/a*(3)
A_i[1:0]=10(4)	1101	1100	*n/a*(3)	*n/a*(3)
A_i[1:0]=11(4)	1110	*n/a*(3)	*n/a*(3)	*n/a*(3)

Notes:
1. PCI byte-enables signals are active low (**`C/BE*[3:0]`**)
2. Operation is performed as a single data burst transaction
3. Improperly aligned access is cancelled and causes a `mem_address_not_aligned` trap (`0x07`)
4. A_i is the source/destination memory address referenced by the load/store instruction

Command Type

The PCI command type to be used for memory-mapped transactions is set in `PCIIC.cmd` to one of IO-read/write, memory-read/write (default after reset), configuration-read/write(*caution*) or memory-read-line/write-invalidate.

Memory commands are issued on the PCI bus with the 2 least significant bits of the address cleared (**`A/D[1:0]`** = `00`) to indicate the linear incrementing mode is being used.

Configuration(*caution*) and I/O commands are issued on the PCI bus with the address unchanged.

Caution: *Configuration transactions shall only be generated in host-bridge mode (**`SYSEN*`** pin tied to a low level).*

Operation

To engage memory-mapped transactions on the PCI interface:

1. Enable PCI initiator mode (`PCISC.com2` = `1`) and memory-mapped transactions (`PCIIC.mod` = `1`).
2. Clear the PCI interrupt pending register (`PCIITP` = `0xF0`).
3. In interrupt-assisted operation, enable any of the 4 possible PCI interrupt sources: **`SERR*`** asserted (`PCIITE.serr` = `1`), initiator parity error (`PCIITE.iper` = `1`), initiator fatal error (`PCIITE.ife` = `1`) and/or initiator internal error (`PCIITE.iier` = `1`). Interrupts shall be enabled as well in the processor (`PSR.et` = `1` and `PSR.pil` < 14) and the interrupt controller (`ITMP.imask[14]` = `1`).

   The interrupt service routine (ISR) shall check the PCI interface status: **`SERR*`** asserted (`PCIITP.serr` = `1`), initiator parity error (`PCIITP.iper` = `1`), initiator fatal error (`PCIITP.ife` = `1`) or initiator internal error (`PCIITP.iier` = `1`) and clear each bit in software by rewriting a `1` as appropriate so further events can be detected.
4. Select the appropriate PCI command type (`PCIIC.cmd`).
5. Execute a load/store instruction on a local memory address mapped in the PCI address range (`0xA0000000` to `0xFFFFFFFF`).
6. If not using interrupts, check the PCI interface status: **`SERR*`** asserted (`PCIITP.serr` = `1`), initiator parity error (`PCIITP.iper` = `1`), initiator fatal error (`PCIITP.ife` = `1`) and/or initiator internal error (`PCIITP.iier` = `1`) then clear each bit as appropriate (in software by rewriting a `1`) so further events can be detected.
7. Repeat steps 5 to 6 as many times as needed.
8. Repeat steps 4 to 7 as many times as needed with a new PCI command type.

Limitations

The following PCI features are not supported:

- PCI interrupt acknowledge, special cycles and memory read-multiple
- 64-bit addressing and Dual Address Cycles (DAC)
- Cacheline wrap mode with memory commands
- PCI power management.
- Master fast back-to-back transactions

Direct Memory Access

A DMA controller is available to perform data transfers between the local memory and a remote target on the PCI bus.

The processor needs only initiate the transfer by programming the DMA controller. Once programmed, the DMA controller is fully autonomous and performs data transfers in the background while the processor is running. Interrupts are provided for synchronization.

The DMA controller only performs word transfers with all 4 PCI byte-lanes enabled (**C/BE*[3:0]** = 0000).

Operation

To engage DMA-assisted transactions on the PCI interface:

1. Enable PCI initiator mode (PCISC.com2 = 1) and DMA-assisted transactions (PCIIC.mod = 1).
2. Clear the PCI interrupt pending register (PCIITP = 0xF0).
3. In interrupt-assisted operation, enable any of the 5 possible PCI interrupts sources: DMA transfer finished (PCIITE.dmaf = 1), **SERR*** asserted (PCIITE.serr = 1), initiator parity error (PCIITE.iper = 1), initiator fatal error (PCIITE.ife = 1) and/or initiator internal error (PCIITE.iier = 1). Interrupts shall be enabled as well in the processor (PSR.et = 1 and PSR.pil < 14) and the interrupt controller (ITMP.imask[14] = 1).

   The interrupt service routine (ISR) shall check the PCI interface status: DMA transfer finished (PCIITP.dmaf = 1), **SERR*** asserted (PCIITP.serr = 1), initiator parity error (PCIITP.iper = 1), initiator fatal error (PCIITP.ife = 1) and/or initiator internal error (PCIITP.iier = 1) then clear each bit as appropriate (in software by rewriting a 1) so further events can be detected.
4. Define the start address in the PCI address space (PCISA).
5. Define in a single write operation the PCI command and the number of words to be transferred (PCIDMA.cmd and PCIDMA.wcnt, 1 to 255 words). At this point for PCI read-based transactions, the PCI interface starts pre-fetching data from the PCI remote target.
6. Define the start address in local memory (PCIDMAA). At this point, data transfer starts in local memory.
7. Wait (interrupt or poll) for the transfer to finish (PCIITP.dmaf = 1).
8. If not using interrupts, check the PCI interface status: **SERR*** asserted (PCIITP.serr = 1), initiator parity error (PCIITP.iper = 1), initiator fatal error (PCIITP.ife = 1) and/or initiator internal error (PCIITP.iier = 1) then clear each bit as appropriate (in software by rewriting a 1) so further events can be detected.
9. Repeat steps 4 to 8 as many times as needed.

Limitations

The following limitations shall be considered when using the DMA controller:

- Memory-mapped access and DMA are mutually exclusive: any load/store instruction to the PCI area in local memory (0xA0000000 - 0xFFFFFFFF) during a DMA transfer will stall the processor until the DMA transfer is completed.

  Moreover, a PCI memory-mapped access (like in an interrupt service routine) which occurs during the initiate procedure of the DMA transfer (between steps 3 to 5) will cause a deadlock requiring a reset of the processor. The application shall ensure the atomicity of steps 4 to 6.
- A wrong DMA initialization sequence may cause the DMA state machine to lock and report an error (PCIITP.iier = 1). The PCI interface shall then be reset (PCIIC = 0xFFFFFFFF).
- A DMA transfer cannot cross a 256 words aligned segment boundary in local memory. If the combination of the start address (PCISA) and the number of words to be transferred (PCIDMA.wcnt) is to cross that boundary, the DMA controller will

terminate the transfer by the end of the segment (PCIITP.dmaf = 1), flush the FIFOs and report an error (PCIITP.iier = 1).

- A DMA transfer cannot operate within the PCI memory-mapped address range in local memory. If the local address of a DMA transfer lies in the PCI memory-mapped address range (0xA0000000 - 0xFFFFFFFF), the DMA controller will cancel the transfer and report an error (PCIITP.iier = 1).

Configuration

Access to a PCI target configuration address space requires the target device to be selected at its **IDSEL** pin. In many systems, the **IDSEL** pins of the satellite devices are directly connected to one of the **A/D[31:11]** signals.

Memory-Mapped

Because of the local memory address range limitation (0xA0000000 to 0xFFFFFFFF), the remote target **IDSEL** signal shall only be connected to lines from **A/D[29:11]**. This allows up to 19 PCI targets to be configured: the target connected to **A/D[29]** is selected with address 0xE0000*xxx*, **A/D[28]** with address 0xD0000*xxx*, **A/D[27]** with address 0xC8000xxx, **A/D[26]** with address 0xC4000*xxx* and so on.

DMA-Assisted

Any target connected to **A/D[31:11]** can be configured with the DMA controller.

PCI Target

PCI target transactions originate from remote PCI initiators (masters) to the PCI interface.

The processor needs only configure the interface by programming the target controller. Once programmed, the target controller is fully autonomous and performs data transfers in the background while the processor is running. Interrupts are provided for synchronization.

Interface Setup

The target interface is programmed as follows(1):

- Enable/Disable(1) parity error checks on the PCI interface (PCISC.com6).
- Enable/Disable(1)(2) remote access to target Memory Space (PCISC.com1).

  If enabled, set(1) the base address to each 16 MB target memory area in the PCI address space (MBAR1.badr & MBAR2.badr) and in local memory (PCITPA.tpa1 & PCITPA.tpa2).
- Enable/Disable(1)(2) remote access to target I/O Space (PCISC.com0).

  If enabled, set(1) the base address to the 1024 bytes target I/O area in the PCI address space (IOBAR3.badr). Base address in local memory is not programmable (PCITPA.tpa3) and is mapped to the AT697 configuration registers (0x80000000 - 0x80000400).
- Enable/Disable the storage in local memory of remote data received with PCI parity error (PCITSC.rfpe). If disabled, data received with a parity error will be discarded.

Notes: 1. The PCI configuration registers with writable bits (PCISC, PCIBHLC, MBAR1, MBAR2, IOBAR3, PCILI, PCIRT & PCICW) can only be programmed by the processor when

in Host-Bridge mode (**SYSEN*** = 0), while they can only be programmed by the remote host-bridge when in Satellite mode (**SYSEN*** = 1).
The other PCI target registers (PCITSC & PCITPA) are always and only accessible by the processor.

2. At least one of target Memory Space (PCISC.com1) and target I/O Space (PCISC.com0) shall be enabled for target operation

Caution: If the PCI target is to share any data with the processor while the data cache is active (CCR.dcs = 01 or 11), care shall be taken to first enable the data cache snooper (CCR.ds = 1) or flush the data cache (CCR.fd = 1) when appropriate.

If the PCI target is to provide any instruction to the processor to execute while the instruction cache is active (CCR.ics = 01 or 11), care shall be taken to flush the instruction cache (CCR.fi = 1) when appropriate.

Limitations

The following limitations shall be considered when using the PCI target:

- The PCI target cannot operate within the PCI memory-mapped address range in local memory. If the programmed target local address (PCITPA) lies in the PCI memory-mapped address range (0xA0000000 - 0xFFFFFFFF), the target controller will cancel the transfer and report an error (PCIITP.tier = 1).
- Target read transactions assume the target space to be prefetchable (reading from an address does not alter the data) and target Memory Read and I/O Read commands are generally prefetched.

  The target controller prefetches up to 8 words into the transmit FIFO once the target read address is available. After the last required data word is transferred to the PCI interface, the FIFO is automatically flushed to discard any unused prefetched data.

  This behavior shall be considered if a non-prefetchable device (like a UART) is to be read through the PCI target interface.
- The interface supports the following PCI write byte-enable patterns (**C/BE*[3:0]**): *single-byte* (0111, 1011, 1101 & 1110), *half-word* (0011 & 1100), *word* (0000) and *ignore-data* (1111, frequently used as a dummy write cycle). A data received with any other byte-enable pattern is discarded and an error is reported (PCIITP.tber = 1).

Delayed-Read

As specified in the PCI standard, delayed-read functionality is implemented as follows:

- When a read request was retried (because data from local memory is not available yet), the interface remains locked for any other target read (targeting different addresses). The initiator of the original read is expected to later repeat the request to the same address.
- A delayed-read can however be interrupted by one or more PCI write accesses. The PCI standard requires each write command to be processed first so to prevent a system lock-up.
- Meanwhile, the interface prefetches read-data from local memory into the target-transmit FIFO (TXMT). When the read request is repeated (after the interfering write, if any), the requested data is available in the FIFO and the delayed-transfer completes normally.

PCI Error Reporting

Parity check, parity error signal (**PERR***) and system error signal (**SERR***) are implemented as foreseen by the PCI standard. They can be controlled in the combined PCI Command & Status register (PCISC).

In addition, PCI initiator and PCI target error conditions and status information are always reported(1) in PCIITP (PCI Interrupt Pending).

If also enabled in PCIITE (PCI Interrupt Enable), each error condition or status information set by the PCI core in PCIITP will trigger the PCI interrupt (ITP.ipend[14] = 1) if enabled in the interrupt controller (ITMP.imask[14] = 1). Interrupts shall then be enabled as well in the processor (PSR.et = 1 and PSR.pil < 14) so the event can be handled.

For testing purpose, the error condition and status information can also be forced in PCIITP by setting the corresponding bit in PCIITF (PCI Interrupt Force).

Note: 1. These bits are never cleared in hardware and shall be cleared in software (by rewriting a 1) so new events can later be registered.

Status information of the various data FIFOs and state machines is available in PCIIS (PCI Initiator Status) and PCITSC (PCI Target Status). It is recommended to check these registers for idle state when configuring the PCI interface and before performing any transaction. Non-nominal values may indicate a previous transaction was not properly completed and spurious data possibly remains in the FIFOs. In such a case, the PCI initiator interface shall be reset (PCIIC = 0xFFFFFFFF) and/or the PCI target interface shall be reset (PCITSC.cs = 0xF), its FIFOs flushed (PCITSC.xfe = 1 and/or PCITSC.rfe = 1) and the transaction aborted (PCITSC.xff = 1).

PCI Data Rate

During PCI initiator and target transfers, the interface tries PCI burst transactions whenever possible to approach the theoretical PCI data rate (~1 Gbit/s at 33 MHz).

However, the exact scheduling of PCI transactions depends on so many factors (clock ratio between PCI and processor, PCI bus traffic, PCI arbitration, processor internal bus activity, wait-states on external memory & I/O peripherals...) there is no guarantee for a sustained burst. The effective data rate may even be far below the theoretical performance in some specific situations.

For a reasonable performance:

- the processor clock frequency should be at least 3 times the PCI clock frequency
- processor accesses to slow devices (IO or memory with high wait-states) should be minimized

Disabling the PCI

In applications where the PCI function is not used, the **PCI_CLK** and the **PCI_RST*** pin shall be tied to a low level. As a consequence, all bidirectional PCI pins including the bus request pin **REQ*** are tri-stated so they shall be driven to a valid high/low level with a pull-up/down to prevent them from floating. When the PCI arbiter is not used, the **AREQ*[3:0]** input signals shall be tied to a high level.

Debug Support Unit

Caution: *This chapter is for information purpose only.*

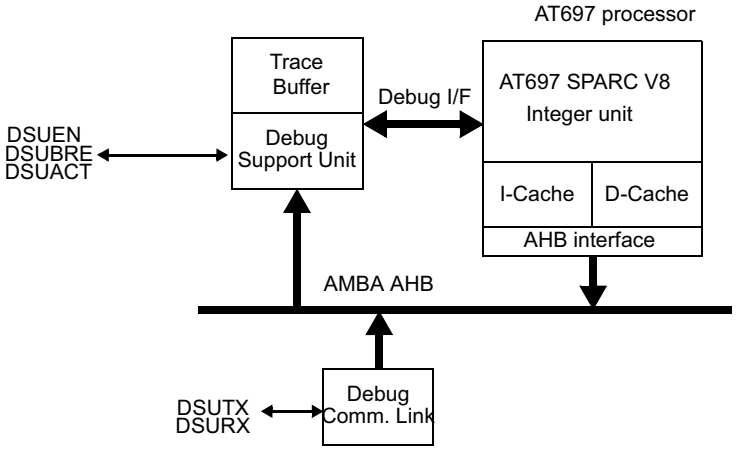
*As its name clearly states, the Debug Support Unit is exclusively meant for debugging purpose. None of the DSU features shall ever be used in the final application where the DSU shall be turned into an inactive state (**DSUEN**, **DSURX** and **DSUBRE** tied to a permanent low level).*

Overview

The AT697F includes a hardware debug support unit to aid in software debugging in the final application. The support is provided through two modules: a debug support unit (DSU) and a debug communication link (DCL).

The DSU can put the processor in debug mode, allowing read/write access to all processor registers and cache memories. The DSU also contains a trace buffer which stores executed instructions or data transfers on the internal bus. The debug communications link implements a simple read/write protocol and uses standard asynchronous UART communications.

Figure 39. Debug Support Unit and Communication Link



It is possible to debug the processor through any master on the internal bus. The PCI interface is build in as a master on the internal bus. All debug features are available from any PCI master.

Debug Support Unit

The debug support unit is used to control the trace buffer and the processor debug mode. The DSU master occupies a 2 MB address space on the internal bus. Through this address space, any other masters like PCI can access the processor registers and the contents of the trace buffer.

The DSU control registers can be accessed at any time, while the processor registers and caches can only be accessed when the processor has entered debug mode. The trace buffer can be accessed only when tracing is disabled or completed. In debug mode, the processor pipeline is held and the processor is controlled by the DSU.

Debug mode can only be entered when the debug support unit is enabled through an external pin (**DSUEN**). Driving the **DSUEN** pin high enables the debug support unit. Enter-

ing debug mode occurs on the following events (provided the appropriate setup was performed, see notes):

- executing a breakpoint instruction (`ta 1`)[1]
- integer unit hardware breakpoint/watchpoint hit (trap `0x0B`)[2]
- rising edge of the external break signal (**DSUBRE**) [2]
- setting the *break-now* bit (`DSUC.bn` = `1`)[2]
- a trap that would cause the processor to enter error mode[3]
- occurrence of any, or a selection of traps as defined in the DSU control register[4]
- after a single-step operation[5]
- DSU breakpoint hit[6]
- instead of entering Error Mode[7]

Notes:
1. Only after the *break-on-S/W-breakpoint* was set (`DSUC.bs` = `1`)
2. Only after the *break-on-IU-watchpoint* was set (`DSUC.bw` = `1`)
3. Only after the *break-on-error-traps* was set (`DSUC.bz` = `1`)
4. Only after the *break-on-trap* was set (`DSUC.bx` = `1`)
5. Only after the *single-step* was set (`DSUC.ss` = `1`)
6. Only after the *break-on-DSU-breakpoint* was set (`DSUC.bd` = `1`)
7. Only after the *break-on-error* was set (`DSUC.be` = `1`)

When debug mode is entered, the following actions are taken:

- `PC` and `nPC` are saved in temporary registers (accessible by the debug unit)
- an output signal (**DSUACT**) is asserted to indicate the debug state
- the timer unit is (optionally) stopped to freeze the AT697F timers and watchdog

The instruction that caused the processor to enter debug mode is not executed, and the processor state is kept unmodified. Execution is resumed by clearing the *break-now* bit (`DSUC.bn` = `0`) or by de-asserting **DSUEN**. The timer unit will be re-enabled and execution will continue from the saved `PC` and `nPC`. Debug mode can also be entered after the processor has entered error mode, for instance when an application has terminated and halted the processor. The error mode can be cleared and the processor restarted at any address.

DSU Breakpoint

The DSU contains two breakpoint registers for matching either internal bus addresses or executed processor instructions. A breakpoint hit is typically used to freeze the trace buffer, but can also put the processor in debug mode.

Freeze operation can be delayed by programming the trace buffer delay counter (`DSUC.dcnt`) to a non-zero value. In this case, the trace buffer delay counter value (`DSUC.dcnt`) is decremented for each additional trace until it reaches zero, after which the trace buffer is frozen. If the *break on trace freeze* bit is set (`DSUC.bt` = `1`), the DSU forces the processor into debug mode when the trace buffer is frozen.

Note: Due to pipeline delays, up to 4 additional instruction can be executed before the processor is placed in debug mode.

A mask register is associated with each breakpoint, allowing breaking on a block of addresses. Only address bits with the corresponding mask bit set to ‘1’ are compared during breakpoint detection.

Time Tag

The DSU implements a time tag counter. The time tag counter is incremented each clock as long as the processor is running. The counter is stopped when the processor enters debug mode, and restarted when execution is resumed.

The time tag counter is stored in the trace as an execution time reference.

Trace Buffer

The trace buffer consists in a circular buffer that stores the executed instructions and/or the internal bus data transfers. The size of the trace buffer is 512 lines of 16 bytes. The trace buffer operation is controlled through the DSU control register (`DSUC`) and the trace buffer control register (`TBCTL`). When the processor enters debug mode, tracing is suspended.

The trace buffer can contain the executed instructions, the transfers on the internal bus or both (mixed-mode). The trace buffer control register (`TBCTL`) contains an instruction trace index counter (`TBCTL.icnt`) and an internal bus trace index counter (`TBCTL.bcnt`) that store the address of the trace buffer location that will be written on next trace. Since the buffer is circular, they actually point to the oldest entry in the buffer. The index counters are automatically incremented after each stored trace entry.

The trace buffer operation is controlled as follows:

- Tracing can be globally enabled/disabled (`DSUC.te`)
- Instruction tracing can be enabled/disabled (`TBCTL.ti`)
- Internal bus tracing can be enabled/disabled (`TBCTL.ta`)
- Internal bus trace freeze on entry into debug mode can be enabled/disabled (`TBCTL.af`)

Instruction trace

When instruction tracing is enabled (`TBCTL.ti` = `1`), one instruction is stored per line in the trace buffer with the exception of multi-cycle instructions. Multi-cycle instructions can be entered two or three times in the trace buffer:

- For store instructions, bits [95:64] correspond to the store address on the first entry and to the stored data on the second entry (and third in case of `STD`). Bit 126 is set logical one on the second and third entry to indicate this.
- A double load (`LDD`) is entered twice in the trace buffer, with bits [95:64] containing the loaded data.
- Multiply and divide instructions are entered twice, but only the last entry contains the result. Bit 126 is set for the second entry.
- For FPU operation producing a double-precision result, the first entry contains the most-significant 32 bits of the results in bits [63:32] while the second entry contains the least-significant 32 bits in bits [63:32].

Table 16. Instruction Trace Buffer Line Allocation

Bits	Name	Definition
127	Instruction breakpoint hit	Set to ‘1’ if a DSU instruction breakpoint hit occurred.
126	Multi-cycle instruction	Set to ‘1’ on the second and third instance of a multi-cycle instruction (`LDD`, `ST` or `FP`*op*)
125:96	Time tag counter	The value of the DSU time tag counter
95:64	Load/Store parameters	Instruction result, store address or store data
63:34	Program counter	Program counter (2 lsb bits removed since they are always zero)
33	Instruction trap	Set to ‘1’ if traced instruction trapped
32	Processor error mode	Set to ‘1’ if the traced instruction caused processor error mode
31:0	Opcode	Instruction opcode

Note: When a trace is frozen, a `watchpoint_detected` trap (`0x0B`) is generated.

Bus Trace

When internal bus tracing is enabled (`TBCTL.ta` = `1`), one internal bus operation is stored per line in the trace buffer.

Table 17. Internal Bus Trace Buffer Line Allocation

Bits	Name	Definition
127	AHB breakpoint hit	Set to ‘1’ if a DSU AHB breakpoint hit occurred.
126	-	Unused
125:96	DSU counter	The value of the DSU counter
95:92	IRL	Processor interrupt request input
91:88	PIL	Processor interrupt level (`PSR.pil`)
87:80	Trap type	Processor trap type (`PSR.tt`)
79	Hwrite	AHB HWRITE
78:77	Htrans	AHB HTRANS
76:74	Hsize	AHB HSIZE
73:71	Hburst	AHB HBURST
70:67	Hmaster	AHB HMASTER
66	Hmastlock	AHB HMASTLOCK
65:64	Hresp	AHB HRESP
63:32	Load/Store data	AHB HRDATA or HWDATA
31:0	Load/Store address	AHB HADDR

Mixed Trace

In mixed mode, the buffer is divided in two halves, with instructions stored in the lower half and bus transfers in the upper half. The most-significant bit of the internal bus trace index counter is then automatically kept high, while the most-significant bit of the instruction trace index counter is kept low.

DSU Memory Map

Table 18. DSU Map

Address	Register
0x90000000	DSU control register
0x90000004	Trace buffer control register
0x90000008	Time tag counter
0x90000010	AHB break address 1
0x90000014	AHB mask 1
0x90000018	AHB break address 2
0x9000001C	AHB mask 2
0x90010000 - 0x9001FFFC	Trace buffer
0x9001...0	Trace bits 127 - 96
0x9001...4	Trace bits 95 - 64
0x9001...8	Trace bits 63 - 32
0x9001...C	Trace bits 31 - 0
0x90020000 - 0x9003FFFC	IU/FPU register file
0x90080000 - 0x900FFFFC	IU special purpose registers
0x90080000	Y register
0x90080004	PSR register
0x90080008	WIM register
0x9008000C	TBR register
0x90080010	PC register
0x90080014	nPC register
0x90080018	FSR register
0x9008001C	DSU trap register
0x90080040	ASR16
0x90080060 - 0x9008007C	ASR24 - ASR31
0x90100000 - 0x9013FFFC	Instruction cache tags
0x90140000 - 0x9017FFFC	Instruction cache data
0x90180000 - 0x901BFFFC	Data cache tags
0x901C0000 - 0x901FFFFC	Data cache data

The IU/FPU registers address depends on the number of register windows implemented. The registers are accessed at the following addresses (*WINDOWS* = total number of implemented SPARC register windows = 8, 0 ≤ *window* < *WINDOWS*):

- %o*n*: 0x90020000 + (((*window* × 64) + 32 + 4×*n*) mod (*WINDOWS* × 64))
- %l*n*: 0x90020000 + (((*window* × 64) + 64 + 4×*n*) mod (*WINDOWS* × 64))
- %i*n*: 0x90020000 + (((*window* × 64) + 96 + 4×*n*) mod (*WINDOWS* × 64))
- %g*n*: 0x90020000 + (*WINDOWS* × 64) + 128 + 4×*n*
- %f*n*: 0x90020000 + (*WINDOWS* × 64) + 4×*n*

Debug Operations

Instruction Breakpoints

To insert instruction breakpoints, the breakpoint instruction (`ta 1`) should be used. This will leave the four IU hardware breakpoints free to be used as data watchpoints. Since cache snooping is only perfomed on the data cache, the instruction cache must be flushed after the insertion or removal of breakpoints. To minimize the influence on execution, it is enough to clear the corresponding instruction cache tag (which is accessible through the DSU).

The DSU hardware breakpoints should only be used to freeze the trace buffer, and not for software debugging since there is a 4-cycle delay from the breakpoint hit before the processor enters the debug mode.

Single Stepping

When single-stepping is enabled (`TBCTL.ss` = `1`), clearing the *break-now* bit (`TBCTL.bn` = `0`) resumes processor execution for one instruction and then automatically re-enters debug mode.

DSU Trap

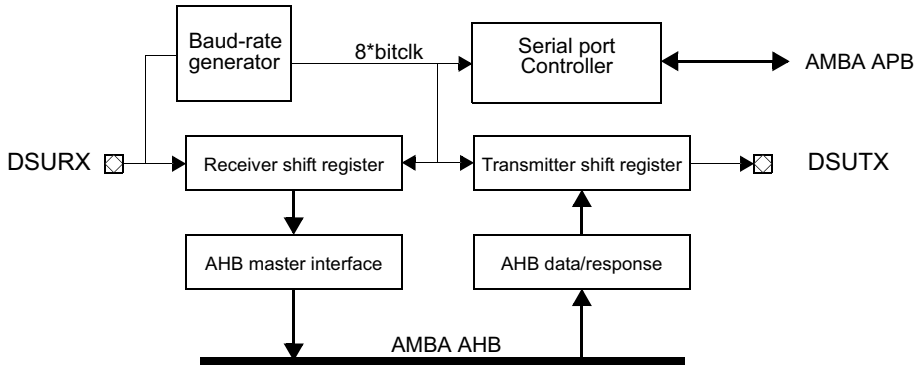
The DSU trap register (`DTR`) consists in a read-only register that indicates which SPARC trap type caused the processor to enter debug mode.

When debug mode is forced by asserting the *break-now* bit (`TBCTL.bn` = `1`), a `watchpoint_detected` trap (`0x0B`) is generated.

DSU Communication Link

DSU communication link consists of a UART connected to the internal bus as a master.

Figure 40. DSU Communication Link Block Diagram

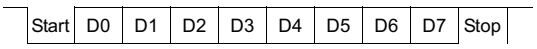


A simple communication protocol is supported to transmit access parameters and data. A link command consist of a control byte, followed by a 32-bit address, followed by optional write data. If the DSU link response is enabled (`DSUC.lr` = `1`), a response byte is sent after each read/write access. If disabled, a write access does not return any response, while a read access only returns the read data.

Data Frame

A DSU data frame consists in a start bit, 8 data bits and a stop bit..

Figure 41. DSU UART Data Frame

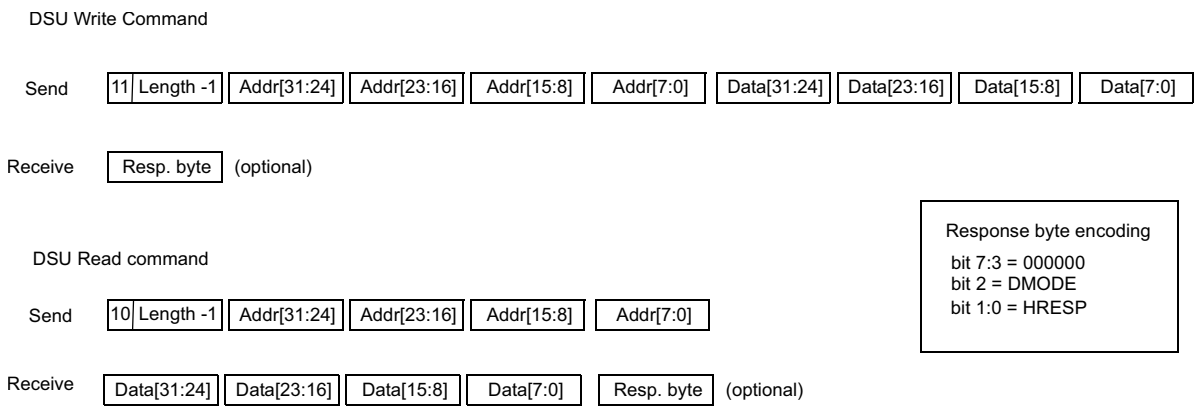


Commands

Through the communication link, a read or write transfer can be generated to any address on the internal bus. A response byte is can optionally be sent when the processor goes from execution mode to debug mode. Block transfers can be performed be setting the length field to *n*-1, where *n* denotes the number of transferred words. For write accesses, the control byte and address is sent once, followed by the number of

data words to be written. The address is automatically incremented after each data word. For read accesses, the control byte and address is sent once and the corresponding number of data words is returned.

Figure 42. DSU Commands



Clock Generation

The UART contains an 16-bit down-counting scaler to generate the desired baud-rate. The scaler counter is clocked by the system clock and generates a UART tick each time it underflows. The counter is reloaded with the value of the UART scaler reload register (`DSUUR.rv`) after each underflow. The resulting UART tick frequency is 8 times the desired baud-rate.

If not programmed in software, the baud-rate is automatically discovered. This is done by searching for the shortest period between two falling edges of the received data (corresponding to two bit periods). When three identical two-bit periods has been found, the corresponding scaler reload value is latched into the reload register (`DSUUR.rv`), the *baud-rate locked* bit is set (`DSUUC.bl` = `1`) and the UART is enabled (`DSUUC.uen` = `1`). If the *baud-rate locked* bit is cleared in software (`DSUUC.bl` = `0`), the baud-rate discovery process is restarted. The baud-rate discovery is also restarted when a *break* is detected on the serial line by the receiver, allowing to change the baud-rate from the external transmitter. For proper baud-rate detection, a *break* followed by the value `0x55` should be transmitted to the receiver.

The best scaler value for manually programming the baudrate can be calculated as follows:

$$scaler_{rv} = \frac{sdclk_{freq}}{baudrate \times 8} - 1$$

$$baudrate = \frac{sdclk_{freq}}{8 \times (scaler_{rv} + 1)}$$

Booting from DSU

By asserting **DSUEN** and **DSUBRE** at reset time, the processor will directly enter debug mode without executing any instructions. The system can then be initialized from the communication link, and applications can be downloaded and debugged. Additionally, external (flash) PROMs for standalone booting can be re-programmed.

JTAG Interface

Overview

The AT697 implements a standard interface compliant with the IEEE 1149.1 JTAG specification. This interface can be used for PCB testing using the JTAG boundary-scan capability.

The JTAG interface is accessed through five dedicated pins. In JTAG terminology, these pins constitute the Test Access Port (TAP).

The following table summarizes the TAP pins and there function at JTAG level.

Table 19. TAP Pins

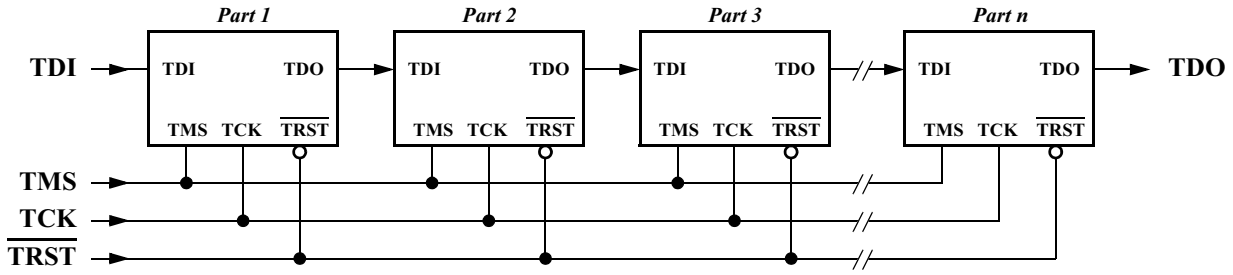
Pin	Name	Type	Description
TCK	Test Clock	Input	Used to clock serial data boundary into scan latches and control sequence of the test state machine. TCK can be asynchronous with CLK
TMS	Test Mode select	Input	Primary control signal for the state machine. Synchronous with TCK. A sequence of values on TMS adjusts the current state of the TAP.
TDI	Test Data Input	Input	Serial input data to the boundary scan latches. Synchronous with TCK
TDO	Test Data Output	Output	Serial output data from the boundary scan latches. Synchronous with TCK
TRST*	Test Reset	Input	Resets the test state machine. can be asynchronous with TCK

For more details, please refer to the 'IEEE Standard Test Access Port and Boundary Scan' specification.

Any AT697F based system will contain several JTAG compatible chips. These are connected using the minimum (single TMS signal) configuration. This configuration contains three broadcast signals (TMS, TCK, and TRST*,) which are fed from the JTAG master to all JTAG slaves in parallel, and a serial path formed by a daisy-chain connection of the serial test data pins (TDI and TDO) of all slaves.

The TAP supports a BYPASS instruction which places a minimum shift path (1 bit) between the chip's TDI and TDO pins. This allows efficient access to any single chip in the daisy-chain without board-level multiplexing.

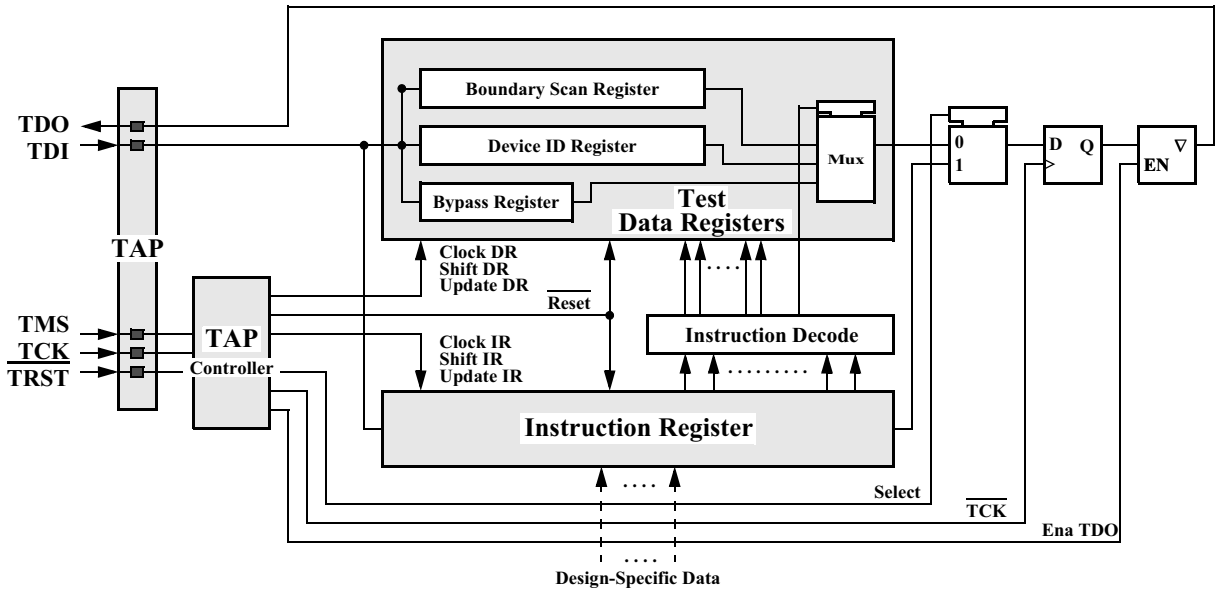
Figure 43. JTAG Serial connection using 1 TMS Signal



TAP Architecture

The TAP implemented in the AT697F consists in a TAP interface, a TAP controller and a number of shift registers including an instruction register (IR) and some other registers.

Figure 44. AT697 TAP Architecture



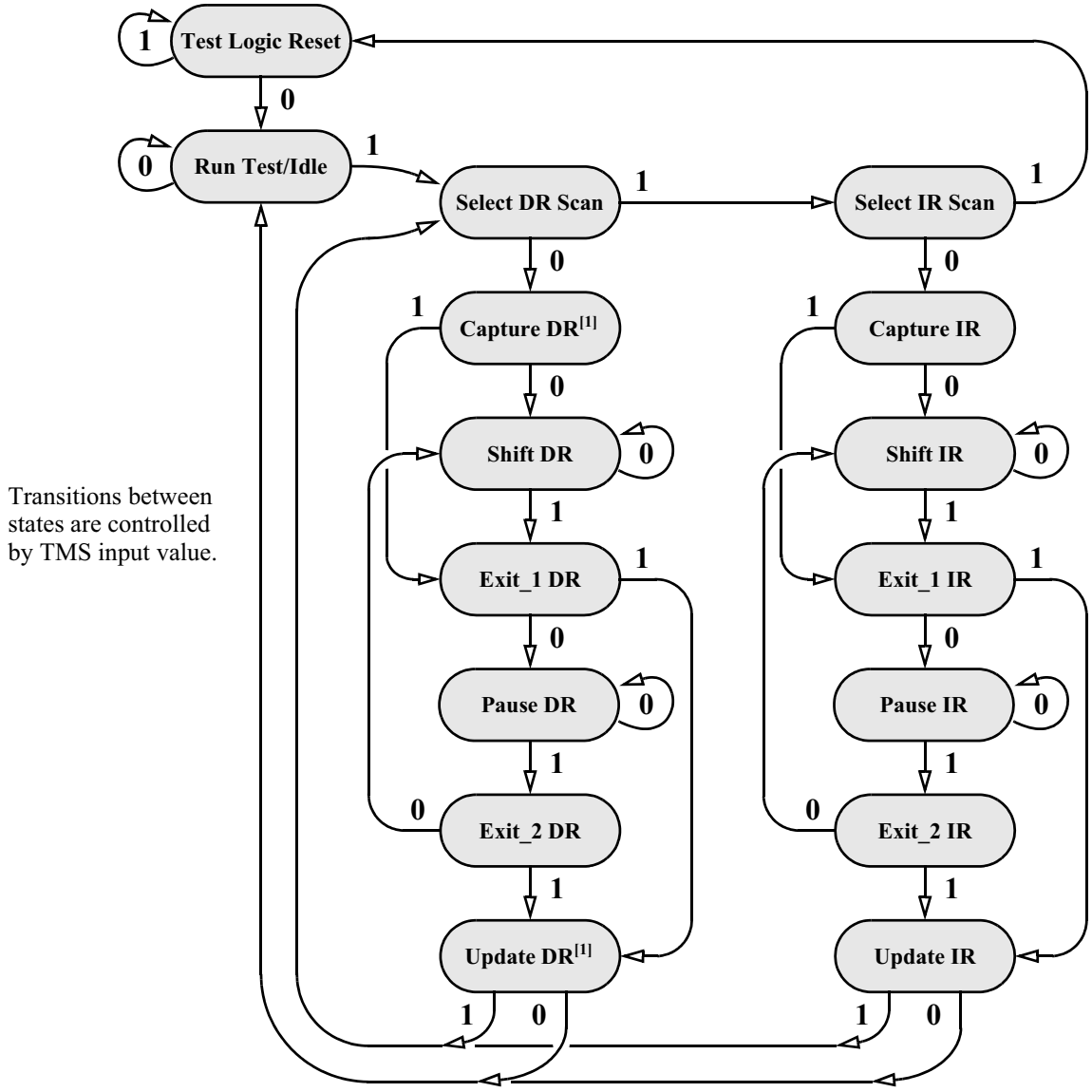
TAP Controller

The TAP controller is a synchronous finite state machine (FSM) which controls the sequence of operations of the JTAG test circuitry, in response to changes at the JTAG bus. (Specifically, in response to changes at the TMS input with respect to the **TCK** input.)

The TAP controller FSM implements the state (16 states) diagram as detailed in the following diagram. The IR is a 3-bit register which allows a test instruction to be shifted into the AT697F. The instruction selects the test to be performed and the test data register to be accessed. Although any number of loops may be supported by the TAP, the finite

state machine in the TAP controller only distinguishes between the IR and a DR. The specific DR can be decoded from the instruction in the IR.

Figure 45. TAP - State Machine



Due to the scan cell layout, "Capture DR" and "Update DR" are states without associated action during the scanning of internal chains.

TAP Instructions

The following instruction are supported by the TAP.

Table 20. TAP instruction set

Binary Value	Instruction Name	Data Register	Scan Chain Accessed
000	EXTEST	Boundary scan register	Boundary scan chain
001	SAMPLE/PRELOAD	Boundary scan register	Boundary scan chain

Binary Value	Instruction Name	Data Register	Scan Chain Accessed
010	BYPASS	Bypass register	Bypass scan chain
111	IDCODE	Device id register	ID register scan chain

BYPASS

This instruction is binary coded "010"

It is used to speed up shifting at board level through components that are not to be activated.

EXTEST

This instruction is binary coded "000"

It is used to test connections between components at board level. Components output pins are controlled by boundary scan register during Capture DR on the rising edge of TCK.

SAMPLE/PRELOAD

This instruction is binary coded "001"

It is used to get a snapshot of the normal operation by sampling I/O states during Capture DR on the rising edge of TCK. It allows also to preload a value on the output latches during Update DR on falling edge of TCK. It do not modify system behaviour.

IDCODE

This instruction is binary coded "111"

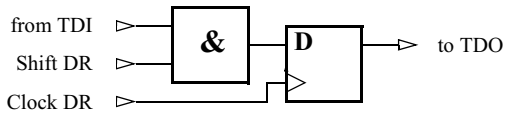
Value of the IDCODE is loaded during Capture DR.

TAP Data Registers

Bypass Register

Bypass register containing a single shift register stage is connected between **TDI** and **TDO**.

Figure 46. Bypass Register Cell



Device ID register

Device ID register is a read only 32-bit register. It is connected between **TDI** and **TDO**.

Figure 47. Device ID Register

31 28	27 12	11 1	0
Vers.	**Part ID**	**Manufacturer's ID**	**Const.**
0001	1011 . 0110 . 0100 . 0101	000 . 0101 . 1000	1

ID. register value: 0x 1b64 50b1

Field Definitions:

[31:28]: Vers - Version number - 0x1

[27:12]: Part ID - Represent part number as assigned by Vendor- 0x b645

[11:01]: Manufacturer's ID - Represent manufacturer's ID as per JEDEC - 0x 058

[0]: Const - Constant tied to logic '1'.

ID. register value: 0x 1b64 50b1

Field Definitions:

- [31:28]: Vers - Version number - 0x1
- [27:12]: Part ID - Represent part number as assigned by Vendor- 0x b645
- [11:01]: Manufacturer's ID - Represent manufacturer's ID as per JEDEC - 0x 058
- [0]: Const - Constant tied to logic '1'.

Boundary Scan Register

A single scan chain consisting of all of the boundary scan cells (input, output and in/out cells).

The purpose of the boundary scan is the support of scan-based board testing. The Boundary Scan register is connected between **TDI** and **TDO**.

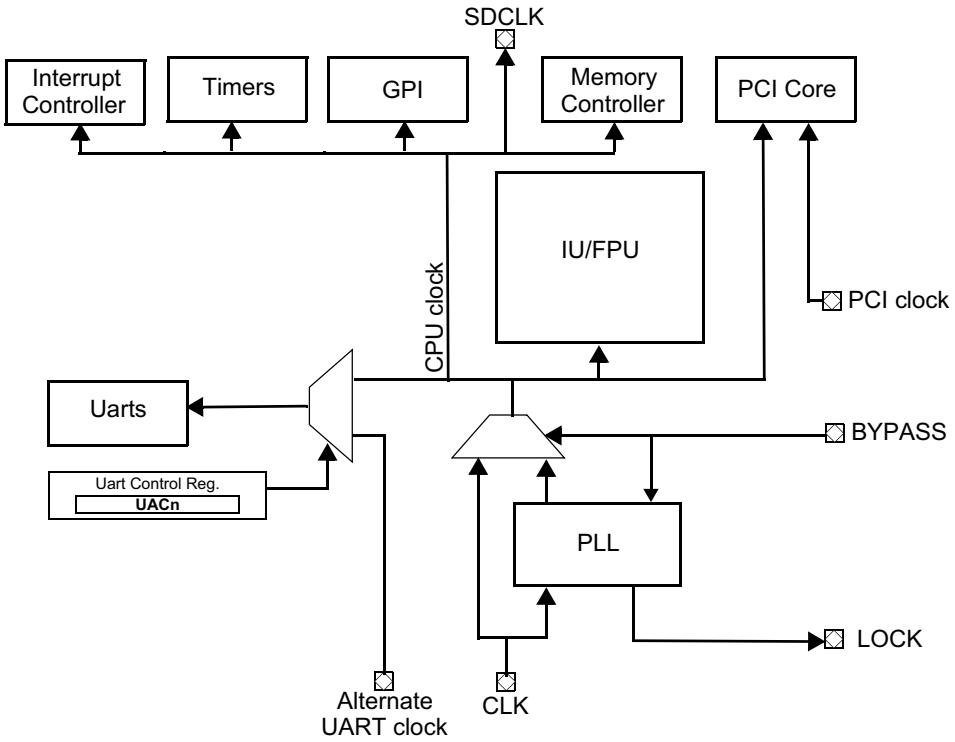
Note: To use the boundary scan feature, the PLL shall be bypassed (**BYPASS** signal asserted).

Clock System

Overview

The AT697F operates two clocks domains: the CPU clock and the PCI clock. The following figure presents the clock system of the processor and its distribution.

Figure 48. Clock Distribution



Note: The PLL is powered-down when the **BYPASS** signal is asserted.

PCI Clock

The PCI clock is dedicated to the PCI Interface. It is used in particular by the PCI wrapper that shares its activity between the two clock domains.

External Clock

The PCI interface and its associated wrapper can only be driven from an external clock. The PCI clock shall be connected to the **PCI_CLK** pin of the PCI interface. This input shall be driven at a frequency in the range of 0 up to 33 MHz.

CPU Clock

The CPU clock is routed to the parts of the system concerned with operation of the SPARC core. Examples of such modules are the CPU core itself, the register files... The CPU clock is also used by the majority of the I/O modules like Timers, Memory controller, Interrupt Controller, with the exception of the PCI Interface.

The CPU clock is driven either directly by an external oscillator or by the internal PLL.

External Clock

To drive the device directly from an external clock source, the CLK input shall be driven by an external clock generator while the **BYPASS** pin is driven high. In that way, the CPU clock is the direct representation of the clock applied to **CLK**.

When the external CPU clock source is selected, the clock input can be driven at a frequency in the range of 0 MHz up to 100 MHz.

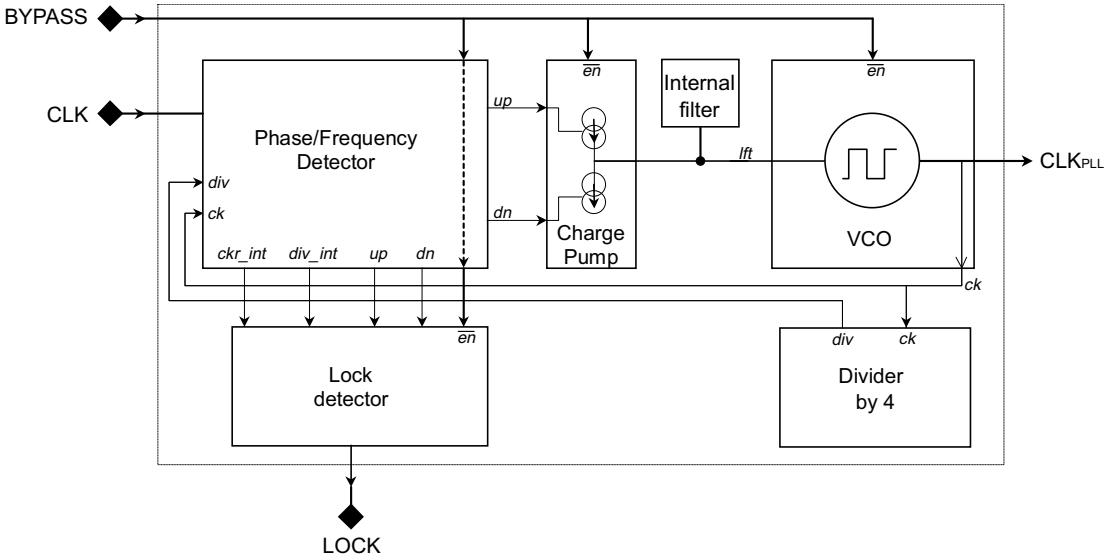
PLL

The CPU clock can be issued from the internal PLL.

Overview

The PLL contains a phase/frequency detector, charge pump, voltage control oscillator, low-pass filter, lock detector and divider.

Figure 49. PLL Block Diagram



The PLL implemented is configured in hardware to provide an internal clock frequency of four times the frequency of the input clock.

PLL control

The PLL control is performed in hardware through dedicated pins.

The following table presents the assignement and functions of the PLL control pins.

Table 21. PLL Ports Description

Pin name	Function
LOCK	The PLL is locked and delivers the expected internal clock
CLK	External clock input
BYPASS	Bypass the internal PLL and directly drive the internal clock from **CLK**

Operation

To drive the device from the internal PLL, the **CLK** input shall be driven by an external clock generator while the **BYPASS** pin is driven low. That way, the CPU clock frequency is four times the frequency of the clock applied to **CLK**.

When the PLL is enabled, the **CLK** clock input shall be driven at a frequency in the range of 18 MHz up to 25 MHz.

Fault-Tolerance & Clock

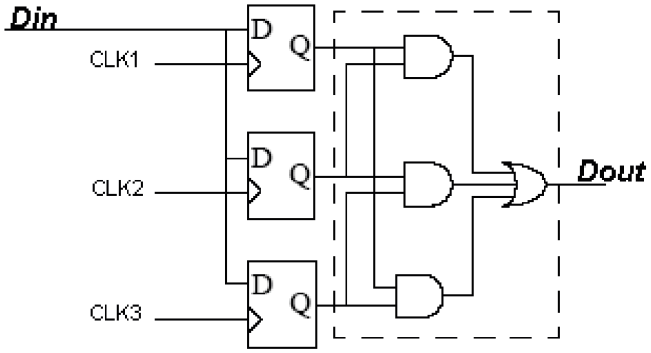
To protect against SEU errors (Single Event Upset), each on-chip register is implemented using triple modular redundancy (TMR).

Moreover, an independent clock tree is used for each of the three registers making up one TMR module. This feature protects against SET errors (Single Event Transient) in the clock tree, to the expense of increased routing.

The CPU clock and the PCI clock are built as three-clock trees.

To prevent erroneous operations from single event upset (SEU) errors and single event upset (SEU), the AT697F is based on full triple modular redundancy (TMR) architecture.

Figure 50. TMR structure



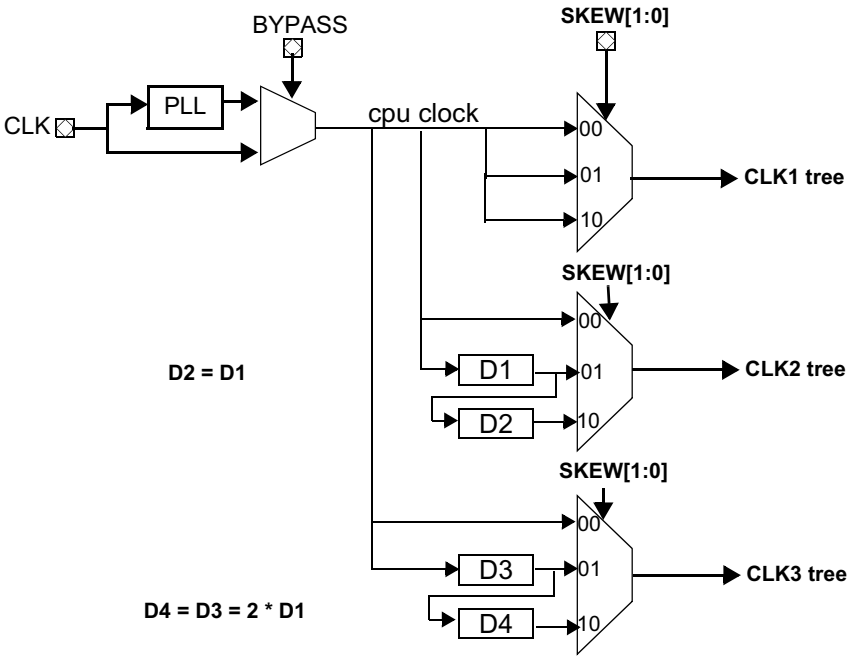
Such architecture is based on a fully triplicated clock distribution (CLK1, CLK2 and CLK3). In that way, each one of the PCI clock and the cpu clock are build as three-clock trees.

Skew

To prevent the processor from corruption by SET errors (Single Event Transient), skew can be programmed on the clock trees. The two dedicated pins **`SKEW[1:0]`** are used to control the skew on the clock trees.

Here is a short description of the skew implementation:

Figure 51. CPU clock tree overview



Three configuration are available:

- natural skew (**SKEW[1:0]** = 00), this is the standard clock-tree as routed internally
- medium skew (**SKEW[1:0]** = 01), the 3 clock-trees are *shifted away* in time one from each other
- maximum skew (**SKEW[1:0]** = 10), the 3 clock-trees are further *shifted away* in time

Table 22. SKEW Assignements

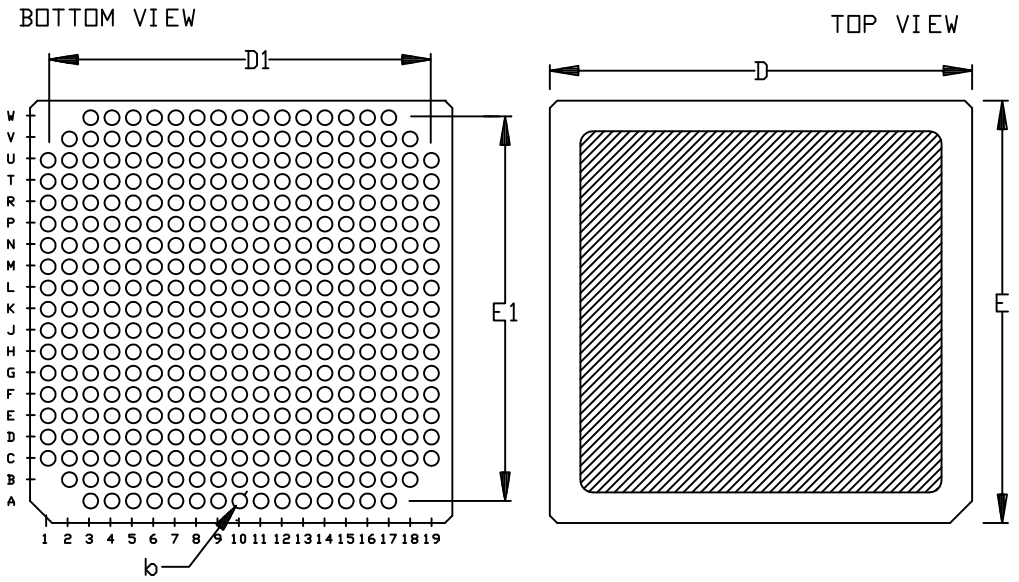
SKEW[1:0]	DELAY		Comments
	CLK1 -> CLK2	CLK1 -> CLK3	
00	natural	natural	natural skew
01	D1	D3	medium skew
10	D1 + D2	D3 + D4	maximum skew
11	*reserved (shall not be used)*		

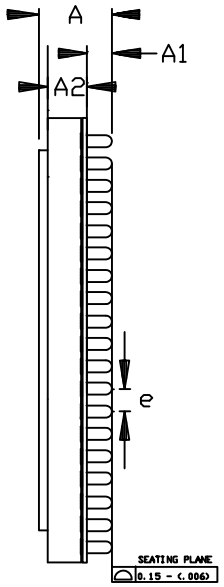
Note: Medium skew and maximum skew configurations improve SET protection but lead to reduced operating performance: maximum clock frequency is reduced and timings are slower than when configured for natural skew (see "Electrical Characteristics").

Packages

MCGA-349

Mechanical Outlines





	MM		INCH	
	Min	Max	Min	Max
D/E	24, 80	25, 20	0. 976	0. 992
D1 /E1	22, 86(1. 27x18)		0. 900(. 05x18)	
A1	1, 40	1, 85	. 055	. 073
A2	2, 40	3. 45	. 094	. 136
A	4. 30	5. 90	. 169	. 232
b	0, 79	0, 99	. 031	. 040
e	1, 27 REF		. 050	

Note: The columns (SCI) are being soldered to the main body (LGA) using an eutectic made of $Sn_{63}Pb_{37}$.

Pin Mapping

Table 23. AT697F MCGA-349 Pinout (1/3)

	A	B	C	D	E	F	G
1			VDD18	VSS18	PIO[6]	PIO[1]	RAMS*[1]
2		VSS18	VDD18	PIO[0]	*N.C.*	PIO[4]	RAMS*[2]

	A	B	C	D	E	F	G
3	VDD18	VDD18	VSS18	VCC33	PIO[2]	*N.C.*	RAMOE*[3]
4	VSS18	VDD18	PIO[9]	*N.C.*	PIO[5]	PIO[3]	RAMS*[4]
5	*N.C.*	*N.C.*	PIO[11]	*N.C.*	*N.C.*	VSS33	RAMOE*[1]
6	PIO[13]	PIO[10]	VCC33	*reserved*	CB[0]	*N.C.*	VSS33
7	CB[1]	VSS33	*N.C.*	PIO[15]	VSS33	PIO[12]	PIO[7]
8	CB[6]	CB[4]	D[2]	VCC33	CB[7]	CB[2]	PIO[8]
9	D[3]	*N.C.*	D[1]	VSS33	D[6]	VCC33	CB[3]
10	D[8]	D[5]	VCC33	VSS33	*reserved*	D[10]	D[4]
11	D[12]	VSS33	VCC33	D[13]	D[7]	D[15]	*N.C.*
12	D[17]	D[18]	D[11]	VSS33	D[14]	D[16]	D[19]
13	D[21]	D[23]	VCC33	VCC33	VSS33	VSS33	A[1]
14	D[25]	*N.C.*	D[22]	D[27]	*N.C.*	VSS33	A[3]
15	D[30]	*N.C.*	D[26]	D[29]	*N.C.*	*N.C.*	A[12]
16	VSS18	VSS18	D[28]	VCC33	*N.C.*	*N.C.*	A[6]
17	VDD18	VDD18	VSS18	D[31]	*N.C.*	A[7]	VSS33
18		VSS18	VDD18	VCC33	A[0]	A[4]	A[8]
19			VDD18	VSS18	A[2]	VSS33	A[9]

Table 24. AT697F MCGA-349 Pinout (2/3)

	H	j	k	l	m	n	p
1	RAMOE*[0]	VSS33	READ	DSUACT	BEXC*	VCC33	SDWE*
2	RAMOE*[2]	ROMS*[1]	TCK	DSURX	SDCLK	VSS33	PCI_CLK
3	VCC33	ROMS*[0]	TDI	DSUTX	DSUBRE	SDDQM[1]	VSS33
4	RAMOE*[4]	RWE*[0]	TDO	DSUEN	SDDQM[2]	*N.C.*	SDCS*[0]
5	RWE*[1]	WRITE*	VSS33	TMS	*N.C.*	SDDQM[3]	SDCAS*
6	RWE*[3]	RWE*[2]	IOS*	VSS33	VSS33	GNT*	A/D[24]
7	RAMS*[0]	*N.C.*	TRST*	SDDQM[0]	VSS33	VCC33	A/D[30]
8	RAMS*[3]	VCC33	OE*	BRDY*	VCC33	A/D[21]	A/D[18]
9	CB[5]	PIO[14]	VSS33	SDRAS*	A/D[22]	A/D[16]	A/D[17]
10	D[9]	D[0]	*N.C.*	A/D[14]	VSS33	PERR*	IRDY*
11	D[20]	A[5]	A[16]	*N.C.*	A/D[12]	A/D[9]	A/D[15]
12	D[24]	A[14]	A[26]	VDD_PLL	AGNT*[3]	A/D[1]	A/D[8]
13	*N.C.*	VCC33	A[21]	*N.C.*	*N.C.*	VSS33	A/D[5]
14	A[10]	VCC33	A[27]	LOCK	SKEW[1]	A/D[0]	AGNT*[1]
15	*N.C.*	VSS33	VCC33	A[24]	*reserved*	BYPASS	CLK
16	A[11]	VSS33	A[23]	RESET*	*N.C.*	AREQ*[2]	VSS33
17	A[19]	A[17]	VSS33	VCC33	WDOG*	*N.C.*	VSS33

	H	j	k	l	m	n	p
18	A[13]	A[18]	A[22]	VSS33	VSS_PLL	AREQ*[3]	*N.C.*
19	A[15]	A[20]	A[25]	ERROR*	SKEW[0]	VCC33	AREQ*[1]

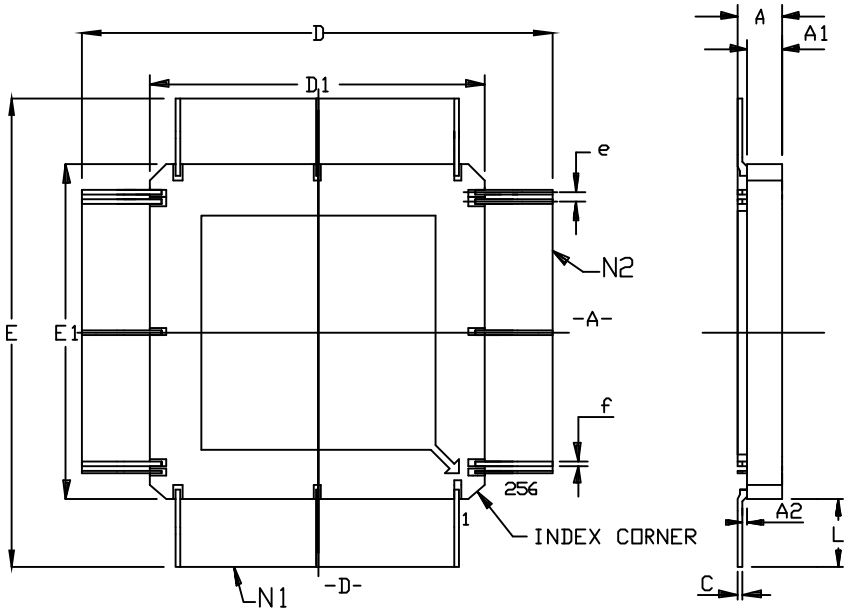
Table 25. AT697F MCGA-349 Pinout (3/3)

	r	t	u	v	w
1	REQ*	VSS18	VDD18		
2	*N.C.*	SDCS*[1]	VDD18	VSS18	
3	PCI_RST*	A/D[31]	VSS18	VDD18	VDD18
4	*N.C.*	A/D[29]	VCC33	VSS18	VSS18
5	*N.C.*	*N.C.*	A/D[26]	*N.C.*	A/D[28]
6	*N.C.*	A/D[27]	IDSEL	VSS33	A/D[25]
7	SYSEN*	VSS33	VCC33	C/BE*[3]	A/D[23]
8	VSS33	VSS33	FRAME*	A/D[20]	A/D[19]
9	TRDY*	VCC33	*N.C.*	C/BE*[2]	VSS33
10	PCI_LOCK*	DEVSEL*	STOP*	VCC33	VCC33
11	VSS33	VCC33	VSS33	C/BE*[1]	SERR*
12	*N.C.*	A/D[11]	PAR	VSS33	A/D[13]
13	VCC33	A/D[7]	A/D[10]	VSS33	VSS33
14	VCC33	VSS33	C/BE*[0]	A/D[4]	A/D[6]
15	*N.C.*	A/D[2]	VCC33	*N.C.*	A/D[3]
16	*N.C.*	VCC33	*N.C.*	VDD18	VSS18
17	VCC33	AGNT*[0]	VSS18	VDD18	VDD18
18	*N.C.*	AGNT*[2]	VDD18	VSS18	
19	AREQ*[0]	VSS18	VDD18		

Notes: 1. *reserved* pins shall not be driven to any voltage
2. *N.C.* refers to unconnected pins

MQFP-256

Mechanical Outlines



	mm		mils	
	Min	Max	Min	Max
A	2.41	3.18	.095	.125
C	0.10	0.20	.004	.008
D	53.23	55.74	2.095	2.195
D1	36.83	37.34	1.450	1.470
E	53.23	55.74	2.095	2.195
E1	36.83	37.34	1.450	1.470
e	0.508 BSC		.020 BSC	
f	0.15	0.25	.006	.010
A1	2.06	2.56	.081	.101
A2	0.05	0.36	.002	.014
L	8.20	9.20	.323	.362
N1	64		64	
N2	64		64	

Pin Mapping

Table 26. AT697F MQFP-256 Pinout

pin number	pin name	pin number	pin name	pin number	pin name	pin number	pin name
1	VCC33	65	PIO[5]	129	D[30]	193	A/D[0]
2	REQ*	66	PIO[6]	130	VCC33	194	VCC33
3	GNT*	67	VCC33	131	D[31]	195	A/D[1]
4	PCI_CLK	68	PIO[7]	132	*N.C.*	196	A/D[2]
5	PCI_RST*	69	PIO[8]	133	A[0]	197	A/D[3]
6	SDCS*[0]	70	PIO[9]	134	A[1]	198	A/D[4]

pin number	pin name	pin number	pin name	pin number	pin name	pin number	pin name
7	VSS	71	VSS	135	VSS	199	VSS
8	VDD18	72	VDD18	136	VDD18	200	VDD18
9	SDCS*[1]	73	PIO[10]	137	A[2]	201	VCC33
10	SDWE*	74	PIO[11]	138	A[3]	202	A/D[5]
11	SDRAS*	75	*reserved*	139	A[4]	203	A/D[6]
12	VSS	76	PIO[12]	140	VCC33	204	A/D[7]
13	VSS	77	PIO[13]	141	A[5]	205	C/BE*[0]
14	SDCAS*	78	PIO[14]	142	A[6]	206	VSS
15	VCC33	79	PIO[15]	143	A[7]	207	VCC33
16	SDDQM[0]	80	VCC33	144	A[8]	208	A/D[8]
17	SDDQM[1]	81	CB[0]	145	A[9]	209	A/D[9]
18	SDDQM[2]	82	CB[1]	146	A[10]	210	A/D[10]
19	SDDQM[3]	83	CB[2]	147	VCC33	211	A/D[11]
20	SDCLK	84	CB[3]	148	A[11]	212	VCC33
21	BRDY*	85	VCC33	149	A[12]	213	A/D[12]
22	BEXC*	86	CB[4]	150	A[13]	214	A/D[13]
23	VSS	87	CB[5]	151	A[14]	215	A/D[14]
24	VSS	88	CB[6]	152	A[15]	216	A/D[15]
25	DSUEN	89	CB[7]	153	A[16]	217	VCC33
26	DSUTX	90	D[0]	154	VCC33	218	C/BE*[1]
27	DSURX	91	VCC33	155	A[17]	219	PAR
28	DSUBRE	92	D[1]	156	A[18]	220	SERR*
29	DSUACT	93	D[2]	157	A[19]	221	PERR*
30	TRST*	94	D[3]	158	A[20]	222	VCC33
31	TCK	95	D[4]	159	A[21]	223	PCI_LOCK*
32	TMS	96	D[5]	160	A[22]	224	STOP*
33	VSS	97	D[6]	161	VSS	225	DEVSEL*
34	TDI	98	*reserved*	162	VCC33	226	TRDY*
35	TDO	99	VCC33	163	A[23]	227	VCC33
36	WRITE*	100	D[7]	164	A[24]	228	IRDY*
37	READ	101	D[8]	165	A[25]	229	FRAME*
38	OE*	102	D[9]	166	A[26]	230	VSS
39	IOS*	103	D[10]	167	A[27]	231	C/BE*[2]
40	VCC33	104	D[11]	168	WDOG*	232	A/D[16]
41	ROMS*[0]	105	D[12]	169	ERROR*	233	VCC33
42	ROMS*[1]	106	VCC33	170	VCC33	234	A/D[17]
43	RWE*[0]	107	D[13]	171	RESET*	235	A/D[18]

pin number	pin name	pin number	pin name	pin number	pin name	pin number	pin name
44	RWE*[1]	108	D[14]	172	*reserved*	236	A/D[19]
45	RWE*[2]	109	D[15]	173	LOCK	237	SYSEN*
46	RWE*[3]	110	D[16]	174	SKEW[1]	238	A/D[20]
47	RAMOE*[0]	111	D[17]	175	SKEW[0]	239	VCC33
48	RAMOE*[1]	112	VSS	176	BYPASS	240	A/D[21]
49	RAMOE*[2]	113	D[18]	177	VSS_PLL	241	A/D[22]
50	RAMOE*[3]	114	VCC33	178	*N.C.*	242	A/D[23]
51	RAMOE*[4]	115	D[19]	179	VDD_PLL	243	IDSEL
52	RAMS*[0]	116	D[20]	180	CLK	244	C/BE*[3]
53	VCC33	117	D[21]	181	VCC33	245	VCC33
54	RAMS*[1]	118	D[22]	182	AREQ*[3]	246	A/D[24]
55	RAMS*[2]	119	D[23]	183	AGNT*[3]	247	A/D[25]
56	RAMS*[3]	120	D[24]	184	AREQ*[2]	248	A/D[26]
57	VSS	121	VSS	185	VSS	249	VSS
58	VDD18	122	VDD18	186	VDD18	250	VDD18
59	RAMS*[4]	123	VCC33	187	AGNT*[2]	251	A/D[27]
60	PIO[0]	124	D[25]	188	AREQ*[1]	252	VCC33
61	PIO[1]	125	D[26]	189	VCC33	253	A/D[28]
62	PIO[2]	126	D[27]	190	AGNT*[1]	254	A/D[29]
63	PIO[3]	127	D[28]	191	AREQ*[0]	255	A/D[30]
64	PIO[4]	128	D[29]	192	AGNT*[0]	256	A/D[31]

Notes:
1. *reserved* pins shall not be driven to any voltage
2. *N.C.* refers to unconnected pins
3. VSS is a common ground pin to VSS33 and VSS18

Register Description

Table 27. Register Legend

Address = `0x01010101`

Bit Number	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**...**	**...**	**...**	**...**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
field name	field					*reserved*									bit								
access type	r = read-only					w = write-only									r/w = read & write								
default value after reset	0	100			1	x = undefined or non affected by reset									p = depends on the value of one or more external pins								

Notes:
- All registers are equally accessible in user and supervisor mode.
- Reserved fields usually are read-only (unless specified otherwise) and writing them usually has no side effects (unless specified otherwise) but should better be done with their default value for compatibility with possible use of the field in future revisions of the product.
- Writing to read-only fields or registers has no effect.

Integer Unit Registers

Table 28. Processor State Register - PSR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
impl				ver				icc				*reserved*						ec	ef	pil				s	ps	et	cwp				
								n	z	v	c																				
r				r				r/w				r						r/w	r/w	r/w				r/w			r		r/w		
0000				0000				x	x	x	x	0000 00						x	x	xxxx				1	x	0	00		xxx		

Bit Number	Mnemonic	Description
31..28	impl	Implementation Implementation or class of implementations of the architecture.
27..24	ver	Version Identify one or more particular implementations or is a readable and writable state field whose properties are implementation-dependent.
23	n	Negative Indicates whether the 32-bit 2's complement ALU result was negative for the last instruction that modified the `icc` field (`1` = negative, `0` = not negative).
22	z	Zero Indicates whether the 32-bit ALU result was zero for the last instruction that modified the icc field (`1` = zero, `0` = nonzero).
21	v	Overflow Indicates whether the ALU result was within the range of (was representable in) 32-bit 2's complement notation for the last instruction that modified the `icc` field (`1` = overflow, `0` = no overflow)
20	c	Carry Indicates whether a 2's complement carry out (or borrow) occurred for the last instruction that modified the `icc` field. Carry is set on addition if there is a carry out of bit 31. Carry is set on subtraction if there is borrow into bit 31 (`1` = carry/borrow, `0` = no carry/borrow).
13	ec	Enable Coprocessor Determines whether the implementation-dependent coprocessor is enabled (`1` = enabled, `0` = disabled). If disabled, a coprocessor instruction will trap. *Although this bit is marked as read/write, this implementation has no coprocessor and will always behave as the coprocessor is permanently disabled.*
12	ef	Enable Floating-Point Determines whether the FPU is enabled (`1` = enabled, `0` = disabled). If disabled, a floating-point instruction will trap.
11..8	pil	Processor Interrupt Level Identify the interrupt level above which the processor will accept an interrupt. *Interrupt 15 is not maskable (NMI) in the IU and is always accepted whatever the current processor interrupt level (however, interrupt masking is still possible in `ITMP`).*
7	s	Supervisor Determines whether the processor is in supervisor or user mode (`1` = supervisor mode, `0` = user mode).

Bit Number	Mnemonic	Description
6	ps	Previous Supervisor Contains the value of the `s` bit at the time of the most recent trap.
5	et	Enable Traps Determines whether traps are enabled (`1` = traps enabled, `0` = traps disabled). A trap automatically resets it to `0`. When `0`, an interrupt request is ignored and an exception trap causes the IU to halt execution and enter error-mode.
4..0	cwp	Current Window Pointer A counter that identifies the current window into the *r* registers. The hardware decrements `cwp` on traps & `SAVE` instructions and increments it on `RESTORE` & `RETT` instructions (modulo 8).

Notes:
- This TMR-protected register can be safely read after power-up without prior initialization. However, bits unaffected by the reset operation will have an undetermined value.
- This register is read and written using the priviledged `RDPSR` and `WRPSR` instructions.

Table 29. Window Invalid Mask Register - WIM

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																								w7	w6	w5	w4	w3	w2	w1	w0
r																								r/w							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x

Bit Number	Mnemonic	Description
0 < *n* < 7	w*n*	Window *n* Invalid Mask Determines whether a window overflow or underflow trap is to be generated on an invalid-marked window by a `SAVE`, `RESTORE`, or `RETT` instruction (`1` = invalid, `0` = valid).

Notes:
- This TMR-protected register can be safely read after power-up without prior initialization. However, bits unaffected by the reset operation will have an undetermined value.
- This register is read and written using the priviledged `RDWIM` and `WRWIM` instructions.

Table 30. Multiply/Divide Register - Y

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
y																															
r/w																															
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx																															

Bit Number	Mnemonic	Description
31..0	y	Y Register Contains the most significant word of the double-precision product of an integer multiplication, as a result of either an integer multiply instruction (`SMUL`, `SMULcc`, `UMUL`, `UMULcc`), or of a routine that uses the integer multiply step instruction (`MULScc`). Also holds the most significant word of the double-precision dividend for an integer divide instruction (`SDIV`, `SDIVcc`, `UDIV`, `UDIVcc`).

Notes:
- This TMR-protected register can be safely read after power-up without prior initialization. However, bits unaffected by the reset operation will have an undertermined value.
- This register is read and written using the non-priviledged `RDY` and `WRY` instructions.

Table 31. Trap Base Register - TBR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
tba																				tt								*reserved*			
r/w																				r								r			
xxxx xxxx xxxx xxxx xxxx																				xxxx xxxx								0000			

Bit Number	Mnemonic	Description
31..12	tba	Trap Base Address The most-significant 20 bits of the trap table address.
11..4	tt	Trap Type Written by the hardware when a trap occurs (except for an external reset request), and retains its value until the next trap. It provides an offset into the trap table.

Notes:
- This TMR-protected register can be safely read after power-up without prior initialization. However, bits unaffected by the reset operation will have an undetermined value.
- This register is read and written using the priviledged `RDTBR` and `WRTBR` instructions.

Table 32. Program Counter - PC

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
pc																															
0x00000000																															

Bit Number	Mnemonic	Description
31..0	pc	Program Counter Contains the address of the instruction currently being executed by the IU. When a trap occurs, it is saved into a local register (`l1`). When returning from the trap, the local register is copied back.

Table 33. Next Program Counters - nPC

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
npc																															
0x00000004																															

Bit Number	Mnemonic	Description
31..0	npc	Next Program Counter Holds the address of the next instruction to be executed by the IU (assuming a trap does not occur). When a trap occurs, it is saved into a local register (`l2`). When returning from the trap, the local register is copied back.

Table 34. Watch Point Address Registers - ASR24, ASR26, ASR28 and ASR30

Address = `%asr24`, `%asr26`, `%asr28`, `%asr30`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
waddr																														*reserved*	if
r/w																														r	r/w
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xx																														0	0

Bit Number	Mnemonic	Description
31..2	waddr	Watchpoint Address Defines the address range to be watched.
0	if	Hit on Instruction Fetch If set, enables hit generation on instruction fetch.

Notes:
- These TMR-protected registers can be safely read after power-up without prior initialization. However, bits unaffected by the reset operation will have an undetermined value.
- These non-priviledged registers are read and written using the `RDASR` and `WRASR` instructions.

Table 35. Watch Point Mask Registers - ASR25, ASR27, ASR29 and ASR31
Address = %asr25, %asr27, %asr29, %asr31

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
wmask																														dl	ds
r/w																														r/w	r/w
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xx																														0	0

Bit Number	Mnemonic	Description
31..2	wmask	Watchpoint Address Mask Defines which bits are to be compared to the matching watchpoint address (`0` = comparison disabled, `1` = comparison enabled).
1	dl	Hit on Data Load If set, enables hit generation on data load.
0	ds	Hit on Data Store If set, enables hit generation on data store.

Notes:
- These TMR-protected registers can be safely read after power-up without prior initialization. However, bits unaffected by the reset operation will have an undetermined value.
- These non-priviledged registers are read and written using the `RDASR` and `WRASR` instructions.

Table 36. Register File Protection Control Register - ASR16

Address = %asr16

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																				cnt			tcb							te	di
r																				r/w			r/w							r/w	r/w
xxxx xxxx xxxx xxxx xxxx																				xxx			x xxxx xx							0	0

Bit Number	Mnemonic	Description
11..9	cnt	Error Counter. Incremented each time a register correction is performed (but saturates at `111`).
8..2	tcb	Test Checkbits If the test mode is enabled, the destination register checksum is XORed with this field before being written to the register file.
1	te	Test Enable If set, errors can be inserted in the register file to test the EDAC protection function.
0	di	Disable Checking If set, disables the register-file checking function.

Notes:
- This TMR-protected register can be safely read after power-up without prior initialization. However, bits unaffected by the reset operation will have an undetermined value.
- This non-priviledged register is read and written using the `RDASR` and `WRASR` instructions.

Table 37. Working Registers - r*n* (0 < *n* < 31)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
r*n*																															
r/w*(caution)*																															
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx																															

Caution: These EDAC-protected registers will come uninitialized after power-up so each register in each window shall be first initialized before it can be safely read. Reading an uninitialized register may trigger a single-bit or a double-bit error in an undeterministic manner.

Table 38. Window Registers

Type	Name		Definition
	Window	Absolute	
in	i7	r31	return address
	i6	r30	frame pointer
	i5	r29	incoming parameter register 5
	i4	r28	incoming parameter register 4
	i3	r27	incoming parameter register 3
	i2	r26	incoming parameter register 2
	i1	r25	incoming parameter register 1
	i0	r24	incoming parameter register 0
local	l7	r23	local register 7
	l6	r22	local register 6
	l5	r21	local register 5
	l4	r20	local register 4
	l3	r19	local register 3
	l2	r18	nPC (for RETT)
	l1	r17	PC (for RETT)
	l0	r16	local register 0
out	o7	r15	temp
	o6	r14	stack pointer
	o5	r13	outgoing parameter register 5
	o4	r12	outgoing parameter register 4
	o3	r10	outgoing parameter register 3
	o2	r11	outgoing parameter register 2
	o1	r9	outgoing parameter register 1
	o0	r8	outgoing parameter register 0
global	g7	r7	global register 7
	g6	r6	global register 6
	g5	r5	global register 5
	g4	r4	global register 4
	g3	r3	global register 3
	g2	r2	global register 2
	g1	r1	global register 1
	g0	r0	global register 0 - always 0x00000000

Table 39. FPU Status Register - FSR

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
rd		*reserved*		tem					ns	*reserved*		ver			ftt			*reserved*		fcc		aexc					cexc				
				nvm	ofm	ufm	dzm	nxm														nva	ofa	ufa	dza	nxa	nvc	ofc	ufc	dzc	nxc
r/w		r		r/w					r	r		r			r			r		r		r/w					r/w				
xx		00		x	x	x	x	x	0	00		000			xxx			00		xx		x	x	x	x	x	x	x	x	x	x

Bit Number	Mnemonic	Description
31..30	rd	Rounding Direction Selects the rounding direction for floating-point results according to ANSI/IEEE Standard 754-1985 (`00` = to nearest, `01` = to zero, `10` = to +∞, `11` = to -∞).
27..23	tem	Trap Enable Mask Enable bits for each of the five floating-point exceptions that can be indicated in the `current_exception` field (`cexc`). If a floating-point operate instruction generates one or more exceptions and the corresponding mask bit is `1`, an `fp_exception` trap is caused. A value of `0` prevents that exception type from generating a trap.
22	ns	Non Standard Floating-Point *Not implemented, always reads as `0`.*
19..17	ver	FPU Version Number
16..14	ftt	Floating-Point Trap Type After a floating-point exception occurs, this field encodes the type of floating-point exception until an `STFSR` or another `FPop` is executed.
11..10	fcc	Floating-Point Condition Codes Updated by the floating-point compare instructions (`FCMP` and `FCMPE`). The floating-point conditional branch instruction (`FBfcc`) bases its control transfer on this field.
9..5	aexc	Accrued Floating-Point Exceptions Accumulate IEEE-754 floating-point exceptions while fp_exception traps are disabled using the `tem` field. After an `FPop` completes, the `tem` and `cexc` fields are logically ANDed together. If the result is nonzero, an `fp_exception` trap is generated; otherwise, the new `cexc` field is ORed into this field. Thus, while traps are masked, exceptions are accumulated in this field.
4..0	cexc	Current Floating-Point Exceptions Indicate that one or more IEEE-754 floating-point exceptions were generated by the most recently executed `FPop` instruction. The absence of an exception causes the corresponding bit to be cleared.

Notes:
- This TMR-protected register can be safely read after power-up without prior initialization. However, bits unaffected by the reset operation will have an undetermined value.
- This register is read and written using the non-priviledged `LDFSR` and `STFSR` instructions.

Floating-Point Trap Types - ftt

Table 40. Trap Type Definition

ftt	Name	Description
0	none	No trap.
1	`IEEE_754_exception`	An `IEEE_754_exception` floating-point trap type indicates that a floating-point exception occurred that conforms to the ANSI/IEEE Standard 754-1985. The exception type is encoded in the `cexc` field.
2	*reserved*	*reserved*
3	*reserved*	*reserved*
4	`sequence_error`	A `sequence_error` indicates one of three abnormal error conditions in the FPU, all caused by erroneous supervisor software: • An attempt was made to execute a floating-point instruction when the FPU was not able to accept one. This type of `sequence_error` arises from a logic error in supervisor software that has caused a previous floating-point trap to be incompletely serviced (for example, the floating-point queue was not emptied after a previous floating-point exception). • An attempt was made to execute a `STDFQ` instruction when the floating-point deferred-trap queue (FQ) was empty, that is, when `FSR.qne` = `0`. (Note that generation of `sequence_error` is recommended, but not required in this case)
5	*reserved*	*reserved*
6	*reserved*	*reserved*
7	*reserved*	*reserved*

Floating-Point Condition Code - fcc

Table 41. `fcc` Field Definition

fcc	Description
0	$f_{rs1} = f_{rs2}$
1	$f_{rs1} < f_{rs2}$
2	$f_{rs1} > f_{rs2}$
3	f_{rs1} ? f_{rs2} Indicates an unordered relation, which is true if either f_{rs1} or f_{rs2} is a signaling NaN or quiet NaN

Note: f_{rs1} and f_{rs2} correspond to the single, double, or quad values in the f registers specified by an instruction's rs1 and rs2 fields. Note that `fcc` is unchanged if `FCMP` or `FCMPE` generate an `IEEE_754_exception` trap.

Floating-Point Exception Fields - aexc / cexc

The accrued and current exception fields and the trap enable mask assume the following definitions of the floating-point exception conditions.

Table 42. Exception Fields

Aexc Mnemonic	Cexc Mnemonic	Name	Description
nva	nvc	Invalid	An operand is improper for the operation to be performed (`1` = invalid operand, `0` = valid operand(s)). Examples: 0 ÷ 0, ∞ - ∞ are invalid.

Aexc Mnemonic	Cexc Mnemonic	Name	Description
ofa	ofc	Overflow	The rounded result would be larger in magnitude than the largest normalized number in the specified format (`1` = overflow, `0` = no overflow).
ufa	ufc	Underflow	The rounded result is inexact and would be smaller in magnitude than the smallest normalized number in the indicated format (`1` = underflow, `0` = no underflow). Underflow is never indicated when the correct unrounded result is zero. if `ufm` = `0`: The `ufc` and `ufa` bits will be set if the correct unrounded result of an operation is less in magnitude than the smallest normalized number and the correctly-rounded result is inexact. These bits will be set if the correct unrounded result is less than the smallest normalized number, but the correct rounded result is the smallest normalized number. nxc and nxa are always set as well. if `ufm` = `1`: An `IEEE_754_exception` trap will occur if the correct unrounded result of an operation would be smaller than the smallest normalized number. A trap will occur if the correct unrounded result would be smaller than the smallest normalized number, but the correct rounded result would be the smallest normalized number.
dza	dzc	Div_by_zero	X÷0, where X is subnormal or normalized. Note that 0 ÷ 0 does not set the `dz`c bit. `1` = division-by-zero, `0` = no division-by-zero.
nxa	nxc	Inexact	The rounded result of an operation differs from the infinitely precise correct result. `1` = inexact result, `0` = exact result.

Table 43. Floating-point Registers - f*n* (0 < *n* < 31)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
f*n*																															
r/w*(caution)*																															
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx																															

Caution: These EDAC-protected registers come uninitialized after power-up so each register shall be first initialized before it can be safely read. Reading an uninitialized register may trigger a single-bit or a double-bit error in an undeterministic manner.

Memory Interface Registers

Table 44. Memory Configuration Register 1 - MCFG1

Address = `0x80000000`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved	pbrdy	abrdy	iowdh		iobrdy	bexc	*reserved*	iows				ioen	*reserved*							prwen	*reserved*	prwdh		prwws				prrws			
r	r/w	r/w	r/w		r/w	r/w	r	r/w				r/w	r							r/w	r	r/w		r/w				r/w			
0	0	0	xx		0	0	0	xxxx				0	000 0000							0	0	pp		1111				1111			

Bit Number	Mnemonic	Description
30	pbrdy	PROM area bus-ready enable. If set, a PROM access will be extended until **`BRDY*`** is asserted.
29	abrdy	Asynchronous bus ready If set, the **`BRDY*`** input can be asserted asynchronously to the system clock, provided it is at least 1.5 clock cycles long. Termination of the access after assertion of **`BRDY*`** will be delayed by at least one clock cycle.
28..27	iowdh	I/O bus width Defines the bus width of the I/O area (`00` = 8, `10` = 32).
26	iobrdy	IO area bus ready enable If set, an IO access will be extended until **`BRDY*`** is asserted
25	bexc	Bus error enable for RAM, PROM and IO access If set, the assertion of the **`BEXC*`** will generate an error response on the internal bus and causes a trap (`0x01`, `0x09`, `0x2B`) depending on the access type.
23..20	iows	I/O waitstates Defines the number of waitstates during I/O accesses (`0000` = 0, `0001` = 1, `0010` = 2,..., `1111` = 15).
19	ioen	I/O area enable `0` = read and write access to I/O area is disabled `1` = read and write access to I/O area is enabled.
11	prwen	PROM write enable If set, enables write cycles to the PROM area.
9..8	prwdh	PROM width Defines the bus width of the PROM area (`00` = 8, `1X` = 32). *During reset, the PROM width is set with the value read on **`PIO[1:0]`**.*
7..4	prwws	PROM write waitstates Defines the number of waitstates during PROM write cycles (`0000` = 0, `0001` = 2,... `1111` = 30). *During reset, the PROM write waitstates is set to the maximum to allow booting.*
3..0	prrws	PROM read waitstates Defines the number of waitstates during PROM read cycles (`0000` = 0, `0001` = 2,... `1111` = 30). *During reset, the PROM read waitstates is set to the maximum to allow booting.*

Note: In 8-bit PROM mode, the last 20% of each PROM bank are used to store the EDAC checksums when EDAC is enabled and cannot be used to store instructions or data.

Table 45. Memory Configuration Register 2 - MCFG2

Address = `0x80000004`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
sdrref	trp	trfc			sdrcas	sdrbs			sdrcls		sdrcmd		*reserved*				se	si	rambs				*reserved*	rambrdy	ramrmw	ramwdh		ramwws		ramrws	
r/w	r/w	r/w			r/w	r/w			r/w		r/w		r				r/w	r/w	r/w				r	r/w	r/w	r/w		r/w		r/w	
0	1	111			1	000			10		00		0000				0	0	xxxx				0	x	x	xx		xx		xx	

Bit Number	Mnemonic	Description
31	sdrref	SDRAM refresh If set, the SDRAM refresh is enabled.
30	trp	SDRAM t_{RP} timing t_{RP} is equal to 2 or 3 system clocks (`0` or `1`).
29..27	trfc	SDRAM t_{RFC} timing t_{RFC} is equal to 3 + *field-value* system clocks.
26	sdrcas	SDRAM CAS delay Selects 2 or 3 cycle CAS delay (`0` or `1`). When changed, a `LOAD-COMMAND-REGISTER` command must be issued at the same time. Also sets RAS/CAS delay (t_{RCD}).
25..23	sdrbs	SDRAM banks size Defines the banks size for SDRAM chip selects: `000` = 4 MB, `001` = 8 MB, `010` = 16 MB .... `111` = 512 MB.
22..21	sdrcls	SDRAM column size `00` = 256, `01` = 512, `10` = 1024, `11` = 4096 when `sdrbs` = `111`, 2048 otherwise
20..19	sdrcmd	SDRAM command Writing a non-zero value generates an SDRAM command: `01` = `PRECHARGE`, `10` = `AUTO-REFRESH`, `11` = `LOAD-COMMAND-REGISTER`. *The field is reset after command has been executed.*
14	se	SDRAM enable If set, the SDRAM controller is enabled.
13	si	SRAM disable If set together with `se` (SDRAM enable), the static ram access is disabled.

Bit Number	Mnemonic	Description
12..9	rambs	SRAM bank size Defines the size of each ram bank (`0000` = 8 KB, `0001` = 16 KB... `1111` = 256 MB).
7	rambrdy	SRAM area bus ready enable If set to `1`, a RAM access to RAM bank 4 (**`RAMS*[4]`**) is extended until **`BRDY*`** is asserted.
6	ramrmw	Read-modify-write on the SRAM Enables read-modify-write cycles on sub-word writes to areas with common write strobe and/or EDAC protection.
5..4	ramwdh	SRAM bus width Defines the bus with of the SRAM area (`00` = 8, `1X` = 32).
3..2	ramwws	SRAM write waitstates Defines the number of waitstates during SRAM write cycles (`00` = 0, `01` = 1, `10` = 2, `11` = 3).
1..0	ramrws	SRAM read waitstates Defines the number of waitstates during SRAM read cycles (`00` = 0, `01` = 1, `10` = 2, `11` = 3).

Table 46. Memory Configuration Register 3 - MCFG3

Address = `0x80000008`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
rfc		*reserved*		me	srcrv															wb	rb	re	pe	tcb							
r		r		r	r/w															r/w	r/w	r/w	r/w	r/w							
11		00		1	xxx xxxx xxxx xxxx															0	0	x	p	xxxx xxxx							

Bit Number	Mnemonic	Description
31:30	rfc	Register file check bits Indicates how many checkbits are used for the register file (`11` = 7 bits)
27	me	Memory EDAC Indicates if a memory EDAC is present
26..12	srcrv	SDRAM refresh counter reload value The period between each `AUTO-REFRESH` command is calculated as follows: $t_{REFRESH}$ = ((reload value) + 1) ÷ **`SDCLK`**$_{frequency}$.
11	wb	EDAC diagnostic write bypass When set, replace the EDAC checkbits with `tcb` on a store operation.
10	rb	EDAC diagnostic read When set, update `tcb` with the EDAC checkbits on instruction fetch or data load operation.

Bit Number	Mnemonic	Description
9	re	RAM EDAC enable When set, enables EDAC protection on the RAM area: • SRAM read-modify-write on sub-word operation shall be enabled as well (`MCFG2.ramrmw = 1`)in order to maintain EDAC protection integrity • SDRAM read-modify-write on sub-word operations is simultaneously activated with EDAC on RAM *Memory shall be initialized before EDAC activation.*
8	pe	PROM EDAC enable When set, enables EDAC protection on the PROM area. *During reset, this bit is initialized with the value of `PIO[2]`.*
7..0	tcb	Test checkbits This field replaces the normal checkbits during store operations when `wb` is set. It is also loaded with the memory checkbits during instruction fetch and data load operations when `rb` is set.

Table 47. Write Protection Register 1 - WPR1

Address = `0x8000001C`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
en	bp	tag															mask														
r/w	r/w	r/w															r/w														
0	x	xx xxxx xxxx xxxx x															xxx xxxx xxxx xxxx														

Bit Number	Mnemonic	Description
31	en	Enable. If set, write protection is enabled.
30	bp	Block Protect If set, *block protect* mode is selected rather than *segment allow* mode.
29..15	tag	Address Tag The tag is XORed with the same bits in the write address.
14..0	mask	Address Mask The mask is applied on the result of the tag/address XOR operation.

Table 48. Write Protection Register 2 - WPR2

Address = `0x80000020`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
en	bp	tag															mask														
r/w	r/w	r/w															r/w														
0	x	xx xxxx xxxx xxxx x															xxx xxxx xxxx xxxx														

Bit Number	Mnemonic	Description
31	en	Enable. If set, write protection is enabled.
30	bp	Block Protect If set, *block protect* mode is selected rather than *segment allow* mode.
29..15	tag	Address Tag The tag is XORed with the same bits in the write address.
14..0	mask	Address Mask The mask is applied on the result of the tag/address XOR operation.

Table 49. Write Protection Start Address 1 - WPSTA1

Address = `0x800000D0`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved		start																												bp	*reserved*
r		r/w																												r/w	r
00		xxxx xxxx xxxx xxxx xxxx xxxx xxxx																												x	0

Bit Number	Mnemonic	Description
29..2	start	Start Address Segment starts from this address (included, 2 null least-significants bits omitted).
1	bp	Block protect If set, *block protect* mode is selected rather than *segment allow* mode.

Table 50. Write Protection End Address 1 - WPSTO1

Address = `0x800000D4`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved		end																												us	su
r		r/w																												r/w	r/w
00		xxxx xxxx xxxx xxxx xxxx xxxx xxxx																												0	0

Bit Number	Mnemonic	Description
29..2	end	End Address Segment finishes at this address (included, 2 null least-significants bits omitted).
1	us	User Mode If set, write protection is active in User Mode (`PSR.s` = `0`).
0	su	Supervisor Mode If set, write protection is active in Supervisor Mode (`PSR.s` = `1`).

Table 51. Write Protection Start Address 2 - WPSTA2

Address = `0x800000D8`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved						start																								bp	*reserved*
r						r/w																								r/w	r
00						xxxx xxxx xxxx xxxx xxxx xxxx xxxx																								x	0

Bit Number	Mnemonic	Description
29..2	start	Start Address Segment starts from this address (included, 2 null least-significants bits omitted).
1	bp	Block protect If set, *block protect* mode is selected rather than *segment allow* mode.

Table 52. Write Protection End Address 2 - WPSTO2

Address = `0x800000DC`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved						end																								us	su
r						r/w																								r/w	r/w
00						xxxx xxxx xxxx xxxx xxxx xxxx xxxx																								0	0

Bit Number	Mnemonic	Description
29..2	end	End Address Segment finishes at this address (included, 2 null least-significants bits omitted).
1	us	User Mode If set, write protection is active in User Mode (`PSR.s` = `0`).
0	su	Supervisor Mode If set, write protection is active in Supervisor Mode (`PSR.s` = `1`).

System Registers

Table 53. Product Configuration Register - PCR

Address = `0x80000024`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
mmu	dsu	sdrctrl	wtpnb			imac	nwin					icsz			ilsz		dcsz			dlsz		divinst	mulinst	wdog	memstat	fpu		pci		wprt	
r	r	r	r			r	r					r			r		r			r		r	r	r	r	r		r		r	
0	1	1	100			0	00111					011			11		011			10		1	1	1	1	01		01		01	

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x7077BBD5																															

Bit Number	Mnemonic	Description
31	mmu	Memory Management Unit 0 = not present
30	dsu	Debug Support Unit 1 = present
29	sdrctrl	SDRAM Controller 1 = present
28..26	wtpnt	IU Watchpoints 100 = 4 watchpoints
25	imac	UMAC/SMAC instructions 0 = not implemented
24..20	nwin	IU Register File Windows 00111 = 8 windows
19..17	icsz	Instruction Cache Set Size (2^{icsz} KB) 011 = 8 KB (× 4 ways = 32 KB total)
16..15	ilsz	Instruction Cache Line Size (2^{ilsz} instructions) 11 = 8 instructions
14..12	dcsz	Data Cache Set Size (2^{dcsz} KB) 011 = 8 KB (× 2 ways = 16 KB total)
11..10	dlsz	Data Cache Line Size (2^{dlsz} words) 10 = 4 words
9	divinst	UDIV/SDIV instructions 1 = implemented
8	mulinst	UMUL/SMUL instructions 1 = implemented
7	wdog	Watchdog 1 = implemented
6	memstat	Memory Status and Address Failing Register 1 = implemented
5..4	fpu	FPU Type 01 = MEIKO
3..2	pci	PCI Core Type 01 = InSilicon
1..0	wprt	Write Protection 01 = implemented

Table 54. Fail Address Register - FAILAR

Address = `0x8000000C`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
hea																															
r(1)/w(2)																															
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx																															

Bit Number	Mnemonic	Description
31:0	hea	Hardware Error Address Identifies the address of the failed access.

Notes:
1. Read value is only valid when a hardware error was detected (`FAILSR.hed` = `1`) and is not relevant otherwise (unpredictable value).
2. Written value is always discarded when no hardware error is detected (`FAILSR.hed` = `0`).

Table 55. Fail Status Register - FAILSR

Address = `0x80000010`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																						eed	hed	het	hem				hes		
r																						r(1)/w(2)	r/w(2)	r(3)/w(4)							
0000 0000 0000 0000 0000 00																						x	0	x	xxxx				xxx		

Bit Number	Mnemonic	Description
9	eed	EDAC-correctable Error Detected Set when an EDAC-correctable memory error is detected(1) (register-file EDAC errors are handled separately). *This bit is never cleared in hardware and shall be cleared in software so a new error can be registered(2). This bit shall also be cleared before the EDAC is activated.*
8	hed	Hardware Error Detected Set when a hardware error is detected (bus exception, write protection error, EDAC correctable and uncorrectable external memory error, PCI initiator error or PCI target error). *This bit is never cleared in hardware and shall be cleared in software so a new hardware error can be registered and the hardware error-related fields updated(2).*
7	het	Hardware Error Type(3) Identifies the type of the failed access (`0` = write, `1` = read).
6...3	hem	Hardware Error Module(3) Identifies the module impacted by the failed access (`0000` = IU/FPU, `0001` = PCI Initiator, `0010` = PCI Target, `0011` = DSU Communication Module).
2..0	hes	Hardware Error Size(3) Identifies the size of the failed access (`000` = byte, `001` = half-word, `010` = word, `011` = double-word).

Notes:
1. Bit might be updated even when a hardware error was already detected (`FAILSR.hed` = `1`).
2. These bits shall be cleared as soon as possible after the error was detected so no subsequent hardware error is missed after the initial detection. Moreover, the register read-and-clear operation shall be best performed by mean of a `SWAP` instruction so to minimize even further the time from read to clear.
3. Read value is only valid when a hardware error was detected (`FAILSR.hed` = `1`) and is not relevant otherwise (unpredictable value).
4. Written value is always discarded while no hardware error is detected (`FAILSR.hed` = `0`).

Caches Register

Table 56. Cache Control Register - CCR

Address = `0x80000014`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
drepl		irepl		isets		dsets		ds	fd	fi	cpc		cptb		ib	ip	dp	ite		ide		dte		dde		df	if	dcs		ics	
r		r		r		r		r/w	w	w	r		r/w		r/w	r	r	r/w		r/w		r/w		r/w		r/w	r/w	r/w		r/w	
11		11		11		01		0	0	0	10		xx		0	0	0	xx		xx		xx		xx		x	x	00		00	

Bit Number	Mnemonic	Description
31..30	drepl	Data cache replacement policy `11` = Least Recently Used (LRU)
29..28	irepl	Instruction cache replacement policy `11` = Least Recently Used (LRU)
27..26	isets	Instruction cache associativity Number of ways in the instruction cache. `11` = 4 way associative
25..24	dsets	Data cache associativity Number of ways in the data cache. `01` = 2 way associative
23	ds	Data cache snoop enable If set, will enable data cache snooping.
22	fd	Flush data cache If set, will flush the data cache. *Always reads as zero.*
21	fi	Flush Instruction cache If set, will flush the instruction cache. *Always reads as zero.*
20..19	cpc	Cache parity bits Indicates how many parity bits are used to protect the caches. `10` = 2 parity bits
18..17	cptb	Cache parity test bits These bits are XOR'ed to the data and tag parity bits during diagnostic writes.
16	ib	Instruction burst fetch This bit enables burst fill during instruction fetch.
15	ip	Instruction cache flush pending This bit is set while an instruction cache flush operation is in progress.
14	dp	Data cache flush pending This bit is set while a data cache flush operation is in progress.
13..12	ite	Instruction cache tag error counter This field is incremented(1) every time an instruction cache tag parity error is detected.

Bit Number	Mnemonic	Description
11.10	ide	Instruction cache data error counter This field is incremented(1) each time an instruction cache data sub-block parity error is detected.
9..8	dte	Data cache tag error counter This field is incremented(1) every time a data cache tag parity error is detected.
7..6	dde	Data cache data error counter This field is incremented(1) each time an instruction cache data sub-block parity error is detected
5	df	Data Cache Freeze on Interrupt If set, the data cache will automatically be frozen when an asynchronous interrupt is taken.
4	if	Instruction Cache Freeze on Interrupt If set, the instruction cache will automatically be frozen when an asynchronous interrupt is taken.
3..2	dcs	Data Cache State `X0` = disabled `01` = frozen `11` = enabled
1..0	ics	Instruction Cache State `X0` = disabled `01` = frozen `11` = enabled.

Note: 1. The counter saturates at `11` (3 events) and shall be cleared in software so new events can later be registered.

Idle Register

Table 57. Idle Register - IDLE

Address = `0x80000018`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
idle																															
w																															
n/a																															

Bit Number	Mnemonic	Description
31..0	idle	A write to this register followed by a load access will cause the system to enter idle mode. *This a write-only register (written value is not relevant), the value returned by a read is not relevant.*

Timer Registers

Table 58. Timer 1 Counter Register - TIMC1

Address = `0x80000040`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cnt																															
r/w																															
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx																															

Bit Number	Mnemonic	Description
31..0	cnt	Timer 1 counter value A read access gives the current decounting value of the timer.

Table 59. Timer 1 Reload Register - TIMR1

Address = `0x80000044`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
rv																															
r/w																															
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx																															

Bit Number	Mnemonic	Description
31..0	rv	Timer 1 reload value A write access programs the reload value of `TIMC1`.

Table 60. Timer 1 Control Register - TIMCTR1

Address = `0x80000048`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																													ld	rl	en
r																													r/w		
0000 0000 0000 0000 0000 0000 0000 0																													0	x	0

Bit Number	Mnemonic	Description
2	ld	Load counter If set, the timer counter register is loaded with the reload value. *Always reads as* `0`.
1	rl	Reload counter If set, the counter is automatically reloaded with the reload value after each underflow. If cleared, the timer is single-shot.
0	en	Enable counter Enables the timer when set.

Table 61. Watchdog Register - WDG

Address = `0x8000004C`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cnt																															
r/w																															
1111 1111 1111 1111 1111 1111 1111 1111																															

Bit Number	Mnemonic	Description
31..0	cnt	Watchdog counter value. A write access programs the new value of the watchdog counter. A read access gives the current decounting value of the watchdog counter.

Table 62. Timer 2 Counter Register - TIMC2

Address = `0x80000050`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cnt																															
r/w																															
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx																															

Bit Number	Mnemonic	Description
31..0	cnt	Timer 2 counter value A read access gives the current decounting value of the timer.

Table 63. Timer 2 Reload Register - TIMR2

Address = `0x80000054`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
rv																															
r/w																															
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx																															

Bit Number	Mnemonic	Description
31..0	rv	Timer 2 reload value A write access programs the reload value of `TIMC2`.

Table 64. Timer 2 Control Register - TIMCTR2

Address = `0x80000058`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																													ld	rl	en
r																													r/w		

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0000 0000 0000 0000 0000 0000 0000 0																													0	x	0

Bit Number	Mnemonic	Description
2	ld	Load counter If set, the timer counter register is loaded with the reload value. *Always reads as 0.*
1	rl	Reload counter If set, the counter is automatically reloaded with the reload value after each underflow. If cleared, the timer is single-shot.
0	en	Enable counter Enables the timer when set.

Table 65. Prescaler Counter Register - SCAC

Address = `0x80000060`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																						cnt									
r																						r/w									
0000 0000 0000 0000 00																						00 0000 0000									

Bit Number	Mnemonic	Description
9..0	cnt	Prescaler counter value A read access gives the current decounting value of the prescaler.

Table 66. Prescaler Reload Register - SCAR

Address = `0x80000064`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																						rv									
r																						r/w									
0000 0000 0000 0000 00																						00 0000 0000[1]									

Bit Number	Mnemonic	Description
9..0	rv	Prescaler reload value A write access programs the reload value of the prescaler. A read access gives the reload value of the prescaler. *The effective division rate is (rv + 1).*[1]

Note: 1. As a special case, reload values 0 & 1 yield a division rate of 4, reload value 2 yields a division-rate of 6

UART Registers

Table 67. UART 1 Data Register - UAD1

Address = `0x80000070`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																								rtd							
r																								r/w							
0000 0000 0000 0000 0000 0000																								xxxx xxxx							

Bit Number	Mnemonic	Description
7..0	rtd	Received/Transmit Data on the UART • A read access provides the last received 8-bit data • A write access initiates the transmission of the 8-bit data

Table 68. UART 1 Status Register - UAS1

Address = `0x80000074`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																									fe	pe	ov	br	th	ts	dr
r																									r/w				r		
0000 0000 0000 0000 0000 0000 0																									0	0	0	0	1	1	0

Bit Number	Mnemonic	Description
6	fe	Framing error(1) Indicates that a framing error was detected.
5	pe	Parity error(1) Indicates that a parity error was detected.
4	ov	Overrun(1) Indicates that one or more character have been lost due to overrun.
3	br	Break received(1) Indicates that a `BREAK` has been received.
2	th	Transmitter hold register empty Indicates that the transmitter hold register is empty.
1	ts	Transmitter shift register empty Indicates that the transmitter shift register is empty.
0	dr	Data ready Indicates that new data is available in the receiver holding register.

Note: 1. Once set, these error bits are never cleared by the processor: it is the responsibility of the application to clear them in software so further errors can be detected.

Table 69. UART 1 Control Register - UAC1

Address = `0x80000078`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																							ec	lb	fl	pe	ps	ti	ri	te	re
r																							r/w								
0000 0000 0000 0000 0000 0000																							0	x	0	x	x	x	x	0	0

Bit Number	Mnemonic	Description
8	ec	External Clock If set, the UART will be directly clocked from **`PIO[3]`** (no scaler).
7	lb	Loop back If set, `RX` will be internally connected to `TX` (with no external activity).
6	fl	Flow control If set, enables hardware flow-control using `CTS` and/or `RTS`.
5	pe	Parity enable If set, enables parity generation and checking.
4	ps	Parity select `0` = even parity `1` = odd parity
3	ti	Transmitter interrupt enable If set, enables generation of transmitter interrupt.
2	ri	Receiver interrupt enable If set, enables generation of receiver interrupt.
1	te	Transmitter enable If set, enables the UART transmitter.
0	re	Receiver enable If set, enables the UART receiver.

Table 70. UART 1 Scaler Register - UASCA1

Address = `0x8000007C`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																				rv											
r																				r/w											
0000 0000 0000 0000 0000																				xxxx xxxx xxxx											

Bit Number	Mnemonic	Description
7..0	rv	UART scaler reload value

Table 71. UART 2 Data Register - UAD2

Address = `0x80000080`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																								rtd							
r																								r/w							
0000 0000 0000 0000 0000 0000																								xxxx xxxx							

Bit Number	Mnemonic	Description
7..0	rtd	Received or Transmitted Data on the UART • A read access provides the last received 8-bit data • A write access initiates transmission of the 8-bit data

Table 72. UART 2 Status Register - UAS2

Address = `0x80000084`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																									fe	pe	ov	br	th	ts	dr
r																									r/w				r		
0000 0000 0000 0000 0000 0000 0																									0	0	0	0	1	1	0

Bit Number	Mnemonic	Description
6	fe	Framing error(1) Indicates that a framing error was detected.
5	pe	Parity error(1) indicates that a parity error was detected.
4	ov	Overrun(1) Indicates that one or more character have been lost due to overrun.
3	br	Break received(1) Indicates that a `BREAK` has been received.
2	th	Transmitter hold register empty Indicates that the transmitter hold register is empty.
1	ts	Transmitter shift register empty Indicates that the transmitter shift register is empty.
0	dr	Data ready Indicates that new data is available in the receiver holding register.

Note: 1. Once set, these error bits are never cleared by the processor: it is the responsibility of the application to clear them in software so further errors can be detected.

Table 73. UART 2 Control Register - UAC2

Address = `0x80000088`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																							ec	lb	fl	pe	ps	ti	ri	te	re
r																							r/w								
0000 0000 0000 0000 0000 0000																							0	x	0	x	x	x	x	0	0

Bit Number	Mnemonic	Description
8	ec	External Clock If set, the UART will be directly clocked from **`PIO[3]`** (no scaler).
7	lb	Loop back If set, `RX` will be internally connected to `TX` (with no external activity).
6	fl	Flow control If set, enables hardware flow-control using `CTS` and/or `RTS`.
5	pe	Parity enable If set, enables parity generation and checking.
4	ps	Parity select Selects parity polarity `0` = even parity `1` = odd parity
3	ti	Transmitter interrupt enable If set, enables generation of transmitter interrupt.
2	ri	Receiver interrupt enable If set, enables generation of receiver interrupt.
1	te	Transmitter enable If set, enables the UART transmitter.
0	re	Receiver enable If set, enables the UART receiver.

Table 74. UART 2 Scaler Register - UASCA2

Address = `0x8000008C`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																				rv											
r																				r/w											
0000 0000 0000 0000 0000																				xxxx xxxx xxxx											

Bit Number	Mnemonic	Description
7..0	rv	UART scaler reload value

Interrupt Registers

Table 75. Interrupt Mask and Priority Register - ITMP

Address = `0x80000090`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ilevel																imask															
I/O7	PCI	I/O6	I/O5	DSU	I/O4	Timer2	Timer1	I/O3	I/O2	I/O1	I/O0	UART1	UART2	AMBA	*reserved*	I/O7	PCI	I/O6	I/O5	DSU	I/O4	Timer2	Timer1	I/O3	I/O2	I/O1	I/O0	UART1	UART2	AMBA	*reserved*
r/w															r	r/w															r
xxxx xxxx xxxx xxx															0	0000 0000 0000 000															0

Bit Number	Mnemonic	Description
31..16	ilevel	Interrupt Level `0` = low priority interrupt `1` = high priority interrupt *High-priority interrupts are always serviced before low-priority interrupts.*
15..0	imask	Interrupt Mask Indicates whether an interrupt is masked or enabled `0` = interrupt masked `1` = interrupt enabled

Table 76. Interrupt Pending Register - ITP

Address = `0x80000094`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																ipend[1]															
																I/O7	PCI	I/O6	I/O5	DSU	I/O4	Timer2	Timer1	I/O3	I/O2	I/O1	I/O0	UART1	UART2	AMBA	*reserved*
r																r/w															r
0000 0000 0000 0000																x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0

Bit Number	Mnemonic	Description
15..0	ipend	Interrupt Pending Indicates whether an interrupt is pending[1]. `1` = interrupt pending[2] `0` = interrupt not pending

Notes: 1. When the IU acknowledges the interrupt, the corresponding pending bit is automatically cleared unless it was forced (see `ITF`).
2. Forced interrupts never show up as pending.

Table 77. Interrupt Force Register - ITF

Address = `0x80000098`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																iforce[1]															
																I/O7	PCI	I/O6	I/O5	DSU	I/O4	Timer2	Timer1	I/O3	I/O2	I/O1	I/O0	UART1	UART2	AMBA	*reserved*
r																r/w															r
0000 0000 0000 0000																x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0

Bit Number	Mnemonic	Description
15..0	iforce	Interrupt Force Indicates whether an interrupt is being forced.[1] `1` = interrupt forced[2] `0` = interrupt not forced

Notes: 1. When the IU acknowledges the interrupt, only the corresponding force bit is automatically cleared if it was forced.
2. Forcing is effective only if the corresponding interrupt is unmasked.

Table 78. Interrupt Clear Register - ITC

Address = `0x8000009C`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																iclear															
																I/O7	PCI	I/O6	I/O5	DSU	I/O4	Timer2	Timer1	I/O3	I/O2	I/O1	I/O0	UART1	UART2	AMBA	*reserved*
r																w															r
xxxx xxxx xxxx xxxx																x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Bit Number	Mnemonic	Description
15..0	iclear	Interrupt Clear If written with a `1`, clears the corresponding bit(s) in the interrupt pending register. *The value returned by a read is not relevant, this is a write-only register.*

General Purpose Interface Registers

Table 79. I/O Port Data Register - IODAT

Address = `0x800000A0`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
meddat								lowdat								piodat															
r/w								r/w								r/w															
xxxx xxxx								xxxx xxxx								xxxx xxxx xxxx xxxx															

Bit Number	Mnemonic	Description
31..24	meddat(1)	D[15:8] bus value
23..16	lowdat(1)	D[7:8] bus value
15..0	piodat	PIO[15:0] port value

Note: 1. These bits are only accessible as I/O ports when all areas (ROM, RAM and I/O) of the memory bus are in 8-bit mode (see “8-bit PROM and SRAM access”) and the SDRAM controller is not enabled.

Table 80. I/O Port Direction Register - IODIR

Address = `0x800000A4`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved														meddir	lowdir	piodir[15:0]															
r														r/w	r/w	r/w															
0000 0000 0000 00														0	0	0000 0000 0000 0000															

Bit Number	Mnemonic	Description
17	meddir(1)	D[15:8] port direction (see note). • `1` = output • `0` = input
16	lowdir(1)	D[7..0] port direction (see note) • `1` = output • `0` = input
15..0	piodir	PIO[15:0] port direction • `1` = output • `0` = input

Note: 1. These bits are only accessible as I/O ports when all areas (ROM, RAM and I/O) of the memory bus are in 8-bit mode (see “8-bit PROM and SRAM access”) and the SDRAM controller is not enabled.

Table 81. I/O Port Interrupt Register - IOIT1

Address = `0x800000A8`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
en3	le3	pl3	isel3					en2	le2	pl2	isel2					en1	le1	pl1	isel1					en0	le0	pl0	isel0				
r/w	r/w	r/w	r/w					r/w	r/w	r/w	r/w					r/w	r/w	r/w	r/w					r/w	r/w	r/w	r/w				
0	x	x	x xxxx					0	x	x	x xxxx					0	x	x	x xxxx					0	x	x	x xxxx				

Bit Number	Mnemonic	Description
31	en3	Enable. If set, the corresponding interrupt will be enabled, otherwise it will be masked.
30	le3	Level/edge triggered. If set, the interrupt will be edge-triggered, otherwise level sensitive.
29	pl3	Polarity If set, the corresponding interrupt will be active high (or edge-triggered on positive edge). Otherwise, it will be active low (or edge-triggered on negative edge).
28..24	isel3	I/O port select. The value of this field defines which I/O port (0 - 31) should generate parallel I/O port interrupt 3.
23	en2	Enable. If set, the corresponding interrupt will be enabled, otherwise it will be masked.
22	le2	Level/edge triggered. If set, the interrupt will be edge-triggered, otherwise level sensitive.
21	pl2	Polarity If set, the corresponding interrupt will be active high (or edge-triggered on positive edge). Otherwise, it will be active low (or edge-triggered on negative edge).
20..16	isel2	I/O port select. The value of this field defines which I/O port (0 - 31) should generate parallel I/O port interrupt 2.
15	en1	Enable. If set, the corresponding interrupt will be enabled, otherwise it will be masked.
14	le1	Level/edge triggered. If set, the interrupt will be edge-triggered, otherwise level sensitive.
13	pl1	Polarity If set, the corresponding interrupt will be active high (or edge-triggered on positive edge). Otherwise, it will be active low (or edge-triggered on negative edge).
12..8	isel1	I/O port select. The value of this field defines which I/O port (0 - 31) should generate parallel I/O port interrupt 1.
7	en0	Enable. If set, the corresponding interrupt will be enabled, otherwise it will be masked.
6	le0	Level/edge triggered. If set, the interrupt will be edge-triggered, otherwise level sensitive.

Bit Number	Mnemonic	Description
5	pl0	Polarity If set, the corresponding interrupt will be active high (or edge-triggered on positive edge). Otherwise, it will be active low (or edge-triggered on negative edge).
4..0	isel0	I/O port select. The value of this field defines which I/O port (0 - 31) should generate parallel I/O port interrupt 0.

Table 82. I/O Port Interrupt Register - IOIT2

Address = `0x800000AC`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
en7	le7	pl7	isel7					en6	le6	pl6	isel6					en5	le5	pl5	isel5					en4	le4	pl4	isel4				
r/w	r/w	r/w	r/w					r/w	r/w	r/w	r/w					r/w	r/w	r/w	r/w					r/w	r/w	r/w	r/w				
0	x	x	x xxxx					0	x	x	x xxxx					0	x	x	x xxxx					0	x	x	x xxxx				

Bit Number	Mnemonic	Description
31	en7	Enable. If set, the corresponding interrupt will be enabled, otherwise it will be masked.
30	le7	Level/edge triggered. If set, the interrupt will be edge-triggered, otherwise level sensitive.
29	pl7	Polarity If set, the corresponding interrupt will be active high (or edge-triggered on positive edge). Otherwise, it will be active low (or edge-triggered on negative edge).
28..24	isel7	I/O port select. The value of this field defines which I/O port (0 - 31) should generate parallel I/O port interrupt 7.
23	en6	Enable. If set, the corresponding interrupt will be enabled, otherwise it will be masked.
22	le6	Level/edge triggered. If set, the interrupt will be edge-triggered, otherwise level sensitive.
21	pl6	Polarity If set, the corresponding interrupt will be active high (or edge-triggered on positive edge). Otherwise, it will be active low (or edge-triggered on negative edge).
20..16	isel6	I/O port select. The value of this field defines which I/O port (0 - 31) should generate parallel I/O port interrupt 6.
15	en5	Enable. If set, the corresponding interrupt will be enabled, otherwise it will be masked.
14	le5	Level/edge triggered. If set, the interrupt will be edge-triggered, otherwise level sensitive.
13	pl5	Polarity If set, the corresponding interrupt will be active high (or edge-triggered on positive edge). Otherwise, it will be active low (or edge-triggered on negative edge).

Bit Number	Mnemonic	Description
12..8	isel5	I/O port select. The value of this field defines which I/O port (0 - 31) should generate parallel I/O port interrupt 5.
7	en4	Enable. If set, the corresponding interrupt will be enabled, otherwise it will be masked.
6	le4	Level/edge triggered. If set, the interrupt will be edge-triggered, otherwise level sensitive.
5	pl4	Polarity If set, the corresponding interrupt will be active high (or edge-triggered on positive edge). Otherwise, it will be active low (or edge-triggered on negative edge).
4..0	isel4	I/O port select. The value of this field defines which I/O port (0 - 31) should generate parallel I/O port interrupt 4.

PCI Registers

Caution: The PCI registers are located between `0x80000100` and `0x800002FC`. Within this range, any address not shown in the following list shall neither be written nor read.

Table 83. PCI Device Identification Register 1 - PCIID1

Address = `0x80000100`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
device id																vendor id															
r																r															
0x1202																0x1438															

Bit Number	Mnemonic	Description
31..16	device id	This field identifies the particular device.
15..0	vendor id	This field identifies the manufacturer of the device (ATMEL).

Table 84. PCI Status & Command Register - PCISC

Address = `0x80000104`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
stat15	stat14	stat13	stat12	stat11	stat10_9		stat8	stat7	stat6	stat5	stat4	stat3	*reserved*								com10	com9	com8	com7	com6	com5	com4	com3	com2	com1	com0
r/w (1)	r/w (1)	r/w (1)	r/w (1)	r/w (1)	r		r/w (1)	r	r	r	r	r	r								r	r/w (1)	r/w (1)	r	r/w (1)	r	r/w (1)	r	r/w (1)	r/w (1)	r/w (1)
0	0	0	0	0	01		0	1	0	0	0	0	0000 0000								0	0	0	0	0	0	0	0	0	0	0

Bit Number	Mnemonic	Description
31	stat15	PCI Bus Parity Error Status 1 = **PERR*** asserted (set even if parity checking is disabled) 0 = **PERR*** not asserted *This bit shall be cleared by writing a 1 (0 has no effect).*
30	stat14	PCI Interface System Error Status 1 = PCI interface asserted **SERR*** 0 = PCI interface does not assert **SERR*** *This bit shall be cleared by writing a 1 (0 has no effect).*
29	stat13	Initiator Interface Termination Status 1 = initiator transaction terminated with Master Abort 0 = initiator transaction successfully terminated (if any) *This bit shall be cleared by writing a 1 (0 has no effect).*
28	stat12	Remote Target Termination Status 1 = initiator transaction terminated with Target Abort 0 = initiator transaction successfully terminated (if any) *This bit shall be cleared by writing a 1 (0 has no effect).*
27	stat11	Target Interface Termination Status 1 = remote initiator transaction terminated with Target Abort 0 = remote initiator transaction successfully terminated (if any) *This bit shall be cleared by writing a 1 (0 has no effect).*
26..25	stat10_9	Target Interface Selection Timing 01 = **DEVSEL*** is asserted with medium timing
24	stat8	Initiator Interface Parity Error Status 1 = initiator interface asserted **PERR*** on a read transaction or observed **PERR*** on a write transaction and Parity Error Response is enabled (bit 6) 0 = initiator interface has not asserted nor observed **PERR*** on a transaction (if any), or Parity Error Response is disabled (bit 6) *This bit shall be cleared by writing a 1 (0 has no effect).*
23	stat7	Target Interface Fast Back-to-Back Capability 1 = the target is capable of accepting fast back-to-back transactions when the transactions are not to the same agent
22	stat6	User definable features *(reserved)*
21	stat5	66 MHz Capability 0 = not capable

Bit Number	Mnemonic	Description
20	stat4	Power Management Capability 0 = no New Capabilities linked list is available at offset 34h, value at that location is not relevant
19	stat3	PCI Interrupt Status *(reserved, no PCI interrupts)*
10	com10	Interrupt Command *(reserved, no PCI interrupts)*
9	com9	Initiator Interface Fast Back-to-Back Control(2) 1 = initiator is allowed to generate fast back-to-back transactions to different targets 0 = initiator shall only generate fast back-to-back transactions to the same target
8	com8	System Error Pin Control 1 = assert **SERR*** 0 = do no assert **SERR*** *Address parity errors are reported only if this bit and bit 6 are 1.*
7	com7	Address/Data Stepping Control *(reserved, not applicable)*
6	com6	Parity Error Response Control 1 = take the normal action when a parity error is detected 0 = set the PCI Bus Parity Error Status bit (bit 31) when an error is detected but do not assert **PERR*** and continue normal operation
5	com5	VGA Palette Snooping *(reserved, not applicable)*
4	com4	Memory Write-and-Invalidate Control 1 = initiator may generate the Memory Write and Invalidate command 0 = use the Memory Write command instead
3	com3	Special Cycles Control *(reserved, not applicable)*
2	com2	PCI Initiator Control 1 = PCI initiator enabled 0 = PCI initiator disabled(3)
1	com1	Target Memory Command Response Control 1 = target interface is allowed to respond to Memory Space accesses 0 = target interface does not respond to Memory Space accesses
0	com0	Target I/O Command Response Control 1 = target interface is allowed to respond to I/O Space accesses 0 = target interface does not respond to I/O Space accesses

Notes:
1. Read-only bit in PCI Satellite mode (**SYSEN*** = 1), reflects what the remote Host-Bridge sees and controls when driving the PCI interface through PCI configuration transactions.
2. Whatever the value of this bit, the PCI interface does not allow to generate fast back-to-back transactions.
3. **Caution:** a memory-mapped PCI transaction shall not be initiated while the PCI initiator is disabled or the processor will stall.

Table 85. PCI Device Identification 2 - PCIID2

Address = 0x80000108

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
class code																								revision id							

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
r																								r							
0x0B4000																								0x02							

Bit Number	Mnemonic	Description
31..8	class code	The Class Code register is read-only and is used to identify the generic function of the device and, in some cases, a specific register-level programming interface. The register is broken into three byte-size fields. The upper byte (at offset 0Bh) is a base class code which broadly classifies the type of function the device performs. The middle byte (at offset 0Ah) is a sub-class code which identifies more specifically the function of the device. The lower byte (at offset 09h) identifies a specific register-level programming interface (if any) so that device independent software can interact with the device. The value `0x0B4000` commonly stands for a processor device.
7..0	revision id	This register specifies a device specific revision identifier. The value is chosen by the vendor. Zero is an acceptable value. This field should be viewed as a vendor defined extension to the Device ID.

Table 86. PCI Bist & Header Type & Latency & Cacheline Size Register - PCIBHLC

Address = `0x8000010C`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
bist								header type								latency timer								cacheline size							
r								r								r/w(1)						r		r/w(1)							
0000 0000								0000 0000								0000 00						00		0000 0000							

Bit Number	Mnemonic	Description
31..24	bist	Built-In Self Test (BIST) *A value of 0 indicates there is no support for this feature.*
23..16	header type	Header Type *A value of 0 indicates this is a single-function interface which implements type 00h Configuration Space Header.*
15..8	latency timer	Latency Timer Specifies the value of the latency timer for this bus master (in units of PCI bus clocks).
7..0	cacheline size	Cacheline Size Specifies the system cacheline size (in units of 32-bit words). Used by master devices to determine whether to use Read, Read Line or Read Multiple commands for accessing memory. Used by slave devices that want to allow memory bursting using cacheline wrap addressing mode to know when a burst sequence wraps to the beginning of the cacheline.

Note: 1. Read-only bit in PCI Satellite mode (**`SYSEN*`** = `1`), reflects what the remote Host-Bridge sees and controls when driving the PCI interface through PCI configuration transactions.

Table 87. Memory Base Address Register 1 - MBAR1

Address = `0x80000110`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
badr																												pref	type		msi
r/w(1)								r																				r	r		r
0000 0000								0000 0000 0000 0000 0000																				1	00		0

Bit Number	Mnemonic	Description
31..4	badr	Base Address (least-significant null nibble omitted) Pointer to a 16 MB address space.
3	pref	Prefetchable `1` = there are no side effects on reads: the device returns all bytes on reads regardless of the byte enables.
2..1	type	Type `00` = the base register is 32 bits wide and mapping can be done anywhere in the 32-bit Memory Space.
0	msi	Memory Space Indicator `0` = the base address maps into Memory Space.

Note: 1. Read-only bit in PCI Satellite mode (**`SYSEN*`** = `1`), reflects what the remote Host-Bridge sees and controls when driving the PCI interface through PCI configuration transactions.

Table 88. Memory Base Address Register 2 - MBAR2

Address = `0x80000114`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
badr																												pref	type		msi
r/w(1)								r																				r	r		r
0000 0000								0000 0000 0000 0000 0000																				1	00		0

Bit Number	Mnemonic	Description
31..4	badr	Base Address (least-significant null nibble omitted) Pointer to a 16 MB address space.
3	pref	Prefetchable `1` = there are no side effects on reads: the device returns all bytes on reads regardless of the byte enables.
2..1	type	Type `00` = the base register is 32 bits wide and mapping can be done anywhere in the 32-bit Memory Space.
0	msi	Memory Space Indicator `0` = the base address maps into Memory Space.

Note: 1. Read-only bit in PCI Satellite mode (**`SYSEN*`** = `1`), reflects what the remote Host-Bridge sees and controls when driving the PCI interface through PCI configuration transactions.

Table 89. IO Base Address Register 3 - IOBAR3

Address = `0x80000118`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
badr																														*reserved*	iosi
r/w(1)																						r								r	r
0000 0000 0000 0000 0000 00																						0000 0000								0	1

Bit Number	Mnemonic	Description
31..2	badr	Base Address (2 least-significant null bits omitted) Pointer to a 1 KB address space.
0	iosi	I/O Space Indicator `1` = the base address maps into I/O Space.

Note: 1. Read-only bit in PCI Satellite mode (**`SYSEN*`** = `1`), reflects what the remote Host-Bridge sees and controls when driving the PCI interface through PCI configuration transactions.

Table 90. Subsystem Identification Register - PCISID

Address = `0x8000012C`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
sid																svi															
r																r															
0x0001																0x1438															

Bit Number	Mnemonic	Description
31..16	sid	Subsystem ID
15..0	svi	Subsystem Vendor ID

Table 91. PCI Latency Interrupt Register - PCILI

Address = `0x8000013C`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
max_lat								min_gnt								int_pin								int_line							
r								r								r								r/w(1)							
0000 0000								0000 0000								0000 0000								0000 0000							

Bit Number	Mnemonic	Description
31..24	max_lat	Maximum Latency Specifies how often the processor needs to gain access to the PCI bus. (in units of 0.25 microseconds assuming a 33 MHz clock). *A value of 0 indicates there are no major requirements for this setting.*
23..16	min_gnt	Minimum Grant Specifies how long a burst period is needed (in units of 0.25 µs assuming a 33 MHz clock). *A value of 0 indicates there are no major requirements for this setting.*
15..8	int_pin	Interrupt Pin Indicates which interrupt pin the processor uses. *Value not relevant (PCI interrupts are not implemented).*
7..0	int_line	Interrupt Line Specifies which input of the system interrupt controller the interrupt pin is connected to. *Value not relevant (PCI interrupts are not implemented).*

Note: 1. Read-only bit in PCI Satellite mode (**`SYSEN*`** = `1`), reflects what the remote Host-Bridge sees and controls when driving the PCI interface through PCI configuration transactions.

Table 92. PCI Initiator Retry & TRDY - PCIIRT

Address = `0x80000140`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																retry								trdy							
r																r/w(1)								r/w(1)							
0																0x80								0x80							

Bit Number	Mnemonic	Description
15..8	retry	Maximum number of retries the PCI initiator attempts.
7..0	trdy	Maximum number of PCI clock cycles the PCI initiator waits for **`TRDY*`**.

Note: 1. Read-only bit in PCI Satellite mode (**`SYSEN*`** = `1`), reflects what the remote Host-Bridge sees and controls when driving the PCI interface through PCI configuration transactions.

Table 93. PCI Configuration Byte-Enable - PCICBE

Address = `0x80000144`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																												ben			
r																												r/w			
0000 0000 0000 0000 0000 0000 0000																												0000			

Bit Number	Mnemonic	Description
3..0	ben	Byte write enables to the PCI configuration registers (`0x80000100` to `0x80000140`): `0` = enabled `1` = disabled *A byte enable pattern, once programmed, applies to all subsequent writes until it is changed.*

Each of the 4 bits is assigned to one 8-bit lane:

- bit ben[3] is applied to Byte 3, the most-significant byte (MSB)
- bit ben[2] is applied to Byte 2
- bit ben[1] is applied to Byte 1
- bit ben[0] is applied to Byte 0, the least-significant byte (LSB)

Table 94. PCI Initiator Start Address - PCISA

Address = `0x80000148`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
stad																															
r/w																															
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx																															

Bit Number	Mnemonic	Description
31..0	stad	PCI start address for PCI initiator transactions in DMA mode.

Table 95. PCI DMA Configuration Register - PCIDMA

Address = `0x80000150`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																			*reserved*	cmd				wcnt							
r																			r/w	r/w				r/w							
0000 0000 0000 0000 000																			0	xxxx				xxxx xxxx							

Bit Number	Mnemonic	Description
11..8	cmd	Command PCI command to use in transaction. *Please refer to section 3.1.1 "Command Definition" of the PCI 2.2 specification for command details.*
7..0	wcnt	Word Count Number of words to transfer during the DMA burst (1 to 255).

Note: Writing to this register effectively initiates the PCI transfer when the PCI core is configured for DMA mode.

Table 96. PCI Initiator Status Register - PCIIS

Address = `0x80000154`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																			sys	dmas				act	xff	xfe	rfe	cs			
r																			r	r				r	r	r	r	r			
0000 0000 0000 0000 000																			p	0000				0	0	1	1	0000			

Bit Number	Mnemonic	Description
12	sys	**SYSEN*** Pin Status `0` = Host mode `1` = Satellite mode
11..8	dmas	DMA State (`0000` = idle)
7	act	PCI Initiator Active `1` = a PCI data transfer is on-going or has been requested `0` = no PCI data transfer on-going or requested
6	xff	MXMT Transmit FIFO Full `1` = transmit FIFO full `0` = transmit FIFO not full

Bit Number	Mnemonic	Description
5	xfe	MXMT Transmit FIFO Empty 1 = transmit FIFO empty 0 = transmit FIFO not empty
4	rfe	MRCV Receive FIFO Full 1 = receive FIFO empty 0 = receive FIFO not empty
3..0	cs	Controller State (0000 = idle)

Table 97. PCI Initiator Configuration - PCIIC

Address = 0x80000158

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
w(1)																															
reserved																								cmd		*reserved*					mod
r																								r/w		r					r/w
0000 0000 0000 0000 0000 0000																								01		00000					0

Bit Number	Mnemonic	Description
7..6	cmd	Most-significant bits of the PCI command used in memory-mapped PCI transactions(2): 00 = I/O read or I/O write 01 = Memory Read or Memory Write 10 = Configuration Read or Configuration Write 11 = Memory Read Line or Memory Write and Invalidate
0	mod	PCI Interface Mode 1 = Memory-Mapped / DMA(3) 0 = PCI initiator disabled

Notes:
1. Writing the whole register with all-1s (0xFFFFFFFF) resets the interface: flush FIFOs, reset byte-enables, reset command to Memory Read/Write and terminate any memory-mapped transaction; any active DMA burst is terminated with an initiator internal error (PCIITP.iier = 1).
2. The least-significant bits depend on the instruction type that initiated the memory-mapped PCI transaction (load = 10, store = 11).
3. **Caution:** a memory-mapped PCI transaction shall not be initiated while the PCI initiator is disabled (PCISC.com2 = 0) or the processor will stall.

Table 98. PCI Target Page Address Register - PCITPA

Address = `0x8000015C`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
tpa1								*reserved*								tpa2								*reserved*							
r/w								r								r/w								r							
0x40								0x00								0x90								0x00							

Bit Number	Mnemonic	Description
31..24	tpa1	Target Page Address for `MBAR1` Specifies the most significant byte of the local memory address(1) where the PCI target Memory Base Address Register 1 is mapped (defaults to the RAM area).
15..8	tpa2	Target Page Address for `MBAR2` Specifies the most significant byte of the local memory address(2) where the PCI target Memory Base Address Register 2 is mapped (defaults to the DSU area).
n/a	*tpa3*(3)	*Target Page Address for `IOBAR3`* *The most significant 22 bits of the local memory address(3) where the PCI target IO Base Address Register 3 is mapped in local memory (defaults to the REGISTER area). This value is not exposed and is not programmable (built-in, `1000000000000000000000`).*

Notes:
1. Assuming `TPA1` is the full 32-bit address: `TPA1` = `tpa1` * 2^{24}. `TPA1` is a pointer to a 16 Mbytes area.
2. Assuming `TPA2` is the full 32-bit address: `TPA2` = `tpa2` * 2^{24}. `TPA2` is a pointer to a 16 Mbytes area.
3. Assuming `TPA3` is the full 32-bit address: `TPA3` = `tpa3` * 2^{10} = `0x80000000`. `TPA3` is a pointer to a 1024 bytes / 256 words area.

Table 99. PCI Target Status and Control Register - PCITSC

Address = `0x80000160`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																							dld	rfpe	xff	xfe	rfe	cs			
r																							r/w (1)	r/w	r/w (1)	r/w (1)	r/w (1)	r/w(1)			
0000 0000 0000 0000 0000 000																							0	0	0	1	1	0000			

Bit Number	Mnemonic	Description
8	dlrd	Delayed Read Automatically asserted by the PCI core during a long delayed read to prevent the on-going read from being overwritten by a subsequent write request (information provided for debug purpose only). *This bit is cleared by writing a 1 (a 0 has no effect).*
7	rfpe	TRCV Receive FIFO parity error `0` = Do not save data with parity error `1` = Ignore any parity error and save data anyway (generation of the perr status bit and assertion of a parity error interrupt is not affected)
6	xff	TXMT Transmit FIFO Full `1` = transmit FIFO full `0` = transmit FIFO not full *Writing this bit with a 1 ends the transaction with a target abort (a 0 has no effect).*
5	xfe	TXMT Transmit FIFO Empty `1` = transmit FIFO empty `0` = transmit FIFO not empty *Writing this bit with a 1 flushes the transmit FIFO (a 0 has no effect).*
4	rfe	TRCV Receive FIFO Empty `1` = receive FIFO empty `0` = receive FIFO not empty *Writing this bit with a 1 flushes the receive FIFO (a 0 has no effect).*
3..0	cs	Controller State (`0000` = idle) *Writing this nibble with all-1s (`0xF`) resets the state machine (any other value has no effect).*

Note: 1. This (group of) bit(s) has a specific action when written with a (group of) 1(s).

Table 100. PCI Interrupt Enable Register - PCIITE

Address = `0x80000164`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																								dmaf	iier	ifer	iper	tier	tber	tper	serr
r																								r/w							
0000 0000 0000 0000 0000 0000																								0	0	0	0	0	0	0	0

Bit Number	Mnemonic	Description
7	dmaf	DMA finished[1] 1 = enable 0 = mask
6	iier	Initiator internal error[1] 1 = enable 0 = mask
5	ifer	Initiator fatal error[1] 1 = enable 0 = mask
4	iper	Initiator parity error[1] 1 = enable 0 = mask
3	tier	Target internal error[1] 1 = enable 0 = mask
2	tber	Target byte-enable error[1] 1 = enable 0 = mask
1	tper	Target parity error[1] 1 = enable 0 = mask
0	serr	**SERR*** signal asserted on the PCI bus[1] 1 = enable 0 = mask

Note: 1. See the corresponding field in `PCIITP` for a complete description.

Table 101. PCI Interrupt Pending Register - PCIITP

Address = `0x80000168`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																								dmaf	iier	ifer	iper	tier	tber	tper	serr
r																								r/w(1)							
0000 0000 0000 0000 0000 0000																								0	0	0	0	0	0	0	0

Bit Number	Mnemonic	Description
7	dmaf	DMA finished 1 = pending 0 = not pending
6	iier	Initiator internal error(2) 1 = pending 0 = not pending *The PCI initiator internally faced a situation that prevented normal completion of the programmed transaction.*
5	ifer	Initiator fatal error 1 = pending 0 = not pending *The PCI initiator reported an address parity error or a transaction abort.*
4	iper	Initiator parity error 1 = pending 0 = not pending *The PCI initiator reported data with parity error on a read or write transaction.*
3	tier	Target internal error(3) 1 = pending 0 = not pending *The PCI target internally faced a situation that prevented normal completion of the programmed transaction.*
2	tber	Target byte-enable error 1 = pending 0 = not pending *The PCI target received data with unsupported byte-enables.*
1	tper	Target parity error 1 = pending 0 = not pending *The PCI target received data with parity error.*
0	serr	**SERR*** signal asserted on the PCI bus 1 = pending 0 = not pending

Notes:
1. Each bit is cleared when written with a 1 (writing a 0 has no effect).
2. An initiator internal error is reported on the following events: initiator busy or not ready (already active DMA transfer), local memory 1 KB address boundary reached

(DMA), attempt to transfer data to/from the PCI mapped-address area (DMA), invalid data read or written (timeout), interface reset during an active DMA transfer...

3. A target internal error is reported on the following events: invalid data read or written (timeout).

Table 102. PCI Interrupt Force Register - PCIITF

Address = `0x8000016C`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																								dmaf	iier	ifer	iper	tier	tber	tper	serr
r(1)																								r(1)/w(2)							r(1)/w
0000 0000 0000 0000 0000 0000																								0	0	0	0	0	0	0	0

Bit Number	Mnemonic	Description
7	dmaf	DMA finished(3) 1 = forced 0 = cleared(4)
6	iier	Initiator internal error(3) 1 = forced 0 = cleared(4)
5	ifer	Initiator fatal error(3) 1 = forced 0 = cleared(4)
4	iper	Initiator parity error(3) 1 = forced 0 = cleared(4)
3	tier	Target internal error(3) 1 = forced 0 = cleared(4)
2	tber	Target byte-enable error(3) 1 = forced 0 = cleared(4)
1	tper	Target parity error(3) 1 = forced 0 = cleared(4)
0	serr	**SERR*** asserted on the PCI bus(3) 1 = forced 0 = not forced

Notes:
1. This is a write-only register, reading this register always yields the contents of `PCIITP`.
2. A PCI interrupt is generated on the write operation itself.
3. See the corresponding field in `PCIITP` for a complete description.
4. Writing a 0 clears the corresponding bit in `PCIITP` rather than not forcing it.

Table 103. PCI DMA Address Register - PCIDMAA

Address = `0x80000178`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
addr																															
r/w																															r
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xx																															00

Bit Number	Mnemonic	Description
31..0	addr	When written, defines the start address of a DMA burst in local memory and initiates the DMA burst. When read, provides the current or last address of the latest DMA burst. *During a DMA burst, this register is automatically incremented (+4) with each word transferred until the programmed burst count or the end of a 1 KB segment is reached, whichever comes first.*

Table 104. PCI Arbiter Register - PCIA

Address = `0x80000280`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																												p3	p2	p1	p0
r																												r	r/w	r/w	r/w
0000 0000 0000 0000 0000 0000 0000																												1	1	1	1

Bit Number	Mnemonic	Description
3	p3	Round robin priority level for agent 3
2	p2	Round robin priority level for agent 2
1	p1	Round robin priority level for agent 1
0	p0	Round robin priority level for agent 0

DSU Registers

Caution: *This section is provided for information purpose only.*

*As its name clearly states, the Debug Support Unit is exclusively meant for debugging purpose. None of the DSU features shall ever be used in the final application where the DSU shall be turned into an inactive state (**DSUEN**, **DSURX** and **DSUBRE** tied to a permanent low level).*

Table 105. DSU Control Register - DSUC

Address = `0x90000000`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved			dcnt									re	dr	lr	ss	pe	ee	eb	dm	de	bz	bx	bd	bn	bs	bw	be	ft	bt	dm	te
r			r/w									w	r/w	r/w	r/w	r/w	r	r	r	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
000			x xxxx xxxx									0	0	0	0	0	p	p	0	0	p	p	0	p	0	p	p	0	0	x	0

Bit Number	Mnemonic	Description
28..20	dcnt	Trace buffer delay counter
19	re	Reset error mode If set, will clear the error mode in the processor. *This is a write-only bit, always reads as a 0.*
18	dr	Debug mode response If set, the DSU communication link will send a response word when the processor enters debug mode
17	lr	Link response If set, the DSU communication link will send a response word after an AHB transfer.
16	ss	Single step If set, the processor will execute one instruction and then return to debug mode.
15	pe	Processor error mode returns `1` on read when processor is in error mode else return `0`.
14	ee	Value of the DSUEN signal (read-only)
13	eb	Value of the DSUBRE signal (read-only)
12	dm	Debug mode If set, indicates the processor has entered debug mode (read-only).
11	de	Delay counter enable If set, the trace buffer delay counter will decrement for each stored trace. This bit is set automatically when an DSU breakpoint is hit and the delay counter is not equal to zero.
10	bz	Break on error traps If set, will force the processor into debug mode on all but the following traps: priviledged_instruction, fpu_disabled, window_overflow, window_underflow, asynchronous_interrupt, ticc_trap. *During reset, this bit is initialized with the value of the DSUBRE signal.*
9	bx	Break on trap If set, will force the processor into debug mode when any trap occurs.
8	bd	Break on DSU breakpoint If set, will force the processor into debug mode when an DSU breakpoint is hit. *During reset, this bit is initialized with the value of the DSUBRE signal.*
7	bn	Break now If set, will force the processor into debug mode provided bit 5 (bw) is also set. If cleared, the processor will resume execution. *During reset, this bit is initialized with the value of the DSUBRE signal.*
6	bs	Break on S/W breakpoint If set, will force the processor into debug mode when a breakpoint instruction (ta 1) is executed.
5	bw	Break on IU watchpoint If set, debug mode will be forced on a IU watchpoint (trap 0xb). *During reset, this bit is initialized with the value of the DSUBRE signal.*

Bit Number	Mnemonic	Description
4	be	Break on error If set, will force the processor into debug mode when the processor would have entered error mode. *During reset, this bit is initialized with the value of the DSUBRE signal.*
3	ft	Freeze timers If set, the scaler in the timer unit will be stopped during debug mode to preserve the time for the software application.
2	bt	Break on trace freeze If set, will generate a DSU break condition on trace freeze.
1	dm	Delay counter mode In mixed tracing mode, setting this bit will cause the delay counter to decrement on AHB traces. If reset, the delay counter will decrement on instruction traces
0	te	Trace enable. If set, the trace buffer is enabled.

Table 106. Trace Buffer Control Register - TBCTL

Address = `0x90000004`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved					af	ta	ti	*reserved*			bcnt									*reserved*			icnt								
r					r/w	r/w	r/w	r			r/w									r			r/w								
000000					x	x	x	000			x xxxx xxxx									000			x xxxx xxxx								

Bit Number	Mnemonic	Description
26	af	AHB trace buffer freeze If set, the trace buffer will be frozen when the processor enters debug mode.
25	ta	Trace AHB enable
24	ti	Trace instruction enable
20..12	bcnt	AHB trace index counter
8..0	icnt	Instruction trace index counter

Table 107. Time Tag Counter - TTC

Address = `0x90000008`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved		cnt																													
r		r/w																													
00		00 0000 0000 0000 0000 0000 0000 0000																													

Bit Number	Mnemonic	Description
29..0	cnt	Counter value

Table 108. Break Address Register 1 - BAD1

Address = `0x90000010`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
adr																														*reserved*	ex
r/w																														r	w
xx xxxx xxxx xxxx xxxx xxxx xxxx xxxx																														0	0

Bit Number	Mnemonic	Description
31..2	adr	Breakpoint address *(32-bit aligned address, hence the 2 omitted LSB)*
0	ex	Enables break on executed instruction *This is a write-only bit, always reads as a 0.*

Table 109. Break Mask Register 1 - BMA1

Address = `0x90000014`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
msk																														ld	st
r/w																														r/w	r/w
xx xxxx xxxx xxxx xxxx xxxx xxxx xxxx																														0	0

Bit Number	Mnemonic	Description
31..2	msk	Breakpoint Address Mask *(32-bit aligned address, hence the 2 omitted LSB)*
1	ld	Enables break on AHB load
0	st	Enables break on AHB write

Table 110. Break Address Register 2 - BAD2

Address = `0x90000018`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
adr																														*reserved*	ex
r/w																														r	w
xx xxxx xxxx xxxx xxxx xxxx xxxx xxxx																														0	0

Bit Number	Mnemonic	Description
31..2	adr	Breakpoint address *(32-bit aligned address, hence the 2 omitted LSB)*
0	ex	Enables break on executed instruction *This is a write-only bit, always reads as a 0.*

Table 111. Break Mask Register - BMA2

Address = `0x9000001C`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
msk																														ld	st
r/w																														r/w	r/w
xx xxxx xxxx xxxx xxxx xxxx xxxx xxxx																														0	0

Bit Number	Mnemonic	Description
31..2	msk	Breakpoint Address Mask *(32-bit aligned address, hence the 2 omitted LSB)*
1	ld	Enables break on AHB load
0	st	Enables break on AHB write

Table 112. DSU UART Status Register - DSUUS

Address = `0x800000C4`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																									fe	*reserved*	ov	br	th	ts	dr
r																									r/w	r	r/w	r/w	r	r	r
0000 0000 0000 0000 0000 0000 0																									0	0	0	0	1	1	0

Bit Number	Mnemonic	Description
6	fe	Framing error Indicates that a framing error was detected.
4	ov	Overrun Indicates that one or more character have been lost due to overrun.
2	th	Transmitter hold register empty Indicates that the transmitter hold register is empty.
1	ts	Transmitter shift register empty Indicates that the transmitter shift register is empty.
0	dr	Data ready Indicates that new data is available in the receiver holding register.

Table 113. DSU UART Control Register - DSUUC

Address = `0x800000C8`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																														bl	ue n
r																														r/w	r/w
0000 0000 0000 0000 0000 0000 0000 00																														0	0

Bit Number	Mnemonic	Description
1	bl	Baud-rate locked Automatically set when the baud rate is locked.
0	uen	UART enable If set, enables both the receiver and the transmitter.

Table 114. DSU UART Scaler Reload Register - DSUUR

Address = `0x800000CC`

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved														ab		rv															
r														r/w		r/w															
0000 0000 0000 00														11 1111 1111 1111 1111																	

Bit Number	Mnemonic	Description
17..16	ab	Scaler reload value
15..0	rv	Scaler reload value

Notes: The following equations shall be used to calculate the scaler value or the baudrate value based on the clock frequency:

$$scaler_{rv} = \frac{sdclk_{freq}}{baudrate \times 8} - 1$$

$$baudrate = \frac{sdclk_{freq}}{8 \times (scaler_{rv} + 1)}$$

Electrical Characteristics

Absolute Maximum Ratings

Operating Temperature ..... -55 °C to +125 °C

Storage Temperature ..... -65 °C to +150 °C

Maximum junction temperature (T_J) ..... 175°C

Thermal resistance junction to case (R_{jc}) ..... 3°C/W

Voltage on VDD18 with respect to VSS18 ..... -0.3 V to + 2.0 V

Voltage on VCC33 with respect to VSS33 ..... -0.3 V to + 4.0 V

DC current VCC33 (VDD18) and VSS33 (VSS18) Pins ..... 200 mA

Input Voltage on I/O pins with respect to Ground ..... -0.5 V to +4 V

DC current per I/O pins ..... 40 mA

ESD ..... 1000 V

Note: Stresses at or above those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

DC Characteristics

Table 115. DC characteristics

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
VCC33	I/O Power Supply Voltage	3.0	3.3	3.6	V	
VDD18	Core Power Supply Voltage	1.65	1.8	1.95	V	
IIL	Low Level Input Leakage Current	-1		1	uA	Vin = VSS33
IILpu	Low Level Input Pull-up Current	-500		-100	uA	Vin = VSS33
IILpd	Low Level Input Pull-down Current	-5		5	uA	Vin = VSS33
IIH	High Level Input Leakage Current	-1		1	uA	Vin = VCC33 (max)
IIHpu	High Level Input Pull-up Current	-5		5	uA	Vin = VCC33 (max)
IIHpd	High Level Input Pull-down Current	100		600	uA	Vin = VCC33 (max)
IOZL	Output leakage current tri-state (low level applied)	-1		1	uA	Vin = VSS33
IOZH	Output leakage current tri-state (high level applied)	-1		1	uA	Vin = VCC33 (max)
VIL CMOS	Low Level Input Voltage			0.8	V	
VIL PCI				0.3*VCC33	V	
VIH CMOS	High Level Input Voltage	2			V	
VIH PCI		0.5*VCC33			V	
VOL	Low Level Output Voltage			0.4	V	VCC33 = VCC33(min) IOL = 2, 4, 8, 16mA
VOL PCI	Low Level Output Voltage for PCI buffers			0.1*VCC33	V	VCC33 = VCC33(min) IOL = 1.5mA
VOH	High Level Output Voltage	VCC33 - 0.4			V	VCC33 = VCC33(min) IOH = 2, 4, 8, 16mA
VOH PCI	High Level Output Voltage for PCI buffers	0.9*VCC33			V	VCC33 = VCC33(min) IOH = 0.5mA
Vcsth(1)	Cold-Sparing Supply Voltage Threshold for CMOS & PCI buffers			0.5	V	Ileakage < 4 µA
ICCSb	Standby Current			5	mA	VCC33 = VCC33(max) no clock active

Note: 1. This value is not tested and is for information only.

Cold-Sparing

Cold-sparing allows a redundant spare to be electrically connected but unpowered until needed.

All CMOS pins are *cold-sparing*: they present a high input impedance when unpowered (VCC33 = 0V) and exhibit a negligible leakage current if exposed to a non-null input voltage at that time.

All PCI pins are required to be clamped to both the ground and power rails to comply with the PCI Specification. However, they are *cold-sparing* as the clamp to VCC33 is removed when unpowered (and clamped to VCC33 when powered). The clamp to VSS33 is always present whatever the power condition.

Power Sequencing

The AT697F is based on Atmel ATC18RHA 0.18 µm CMOS process. When the AT697F needs to be powered "on/off" while other circuits in the application are still powered, the recommended power “on/off” sequence is:

- power-up: first power VCC33 (I/O), and then power VDD18 (Core).
- power-down: first unpower VDD18 (Core), and then unpower VCC33 (I/O).

It is also recommended to stop all activity during these phases as a bi-directional could be in an undetermined state (input or output mode) and create bus contention.

Power Consumption

The power dissipation is the sum of three basic contributions: $P = P_{Core} + P_{I/O} + P_{PCI}$

- P_{Core} represents the contribution of the internal activity.
- $P_{I/O}$ represents the contribution of the IO pads (except the PCI bus) and associated output load current .
- P_{PCI} represents the contribution of the PCI pads and associated output load current.

Table 116. Power Dissipation

Conditions	Typical(1)			Worst-Case(2)		
Power (in W)	P_{Core}	$P_{I/O}$	P_{PCI}	P_{Core}	$P_{I/O}$	P_{PCI}
	0.5	0.2	0.1	0.7	0.3	0.2

Notes: 1. Typical conditions: 25°C, 1.8V core, 3.3V I/O, 100 MHz, high I/O and core activity.
2. Worst-case conditions: -55°C, 1.95 V core, 3.6V I/O, 100 MHz, high I/O and core activity.

Decoupling Capacitance

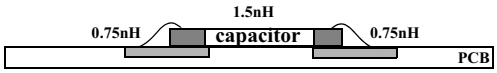
Two main frequencies are involved in the AT697F processor:

- up to 33 MHz for the PCI interface
- up to 100 MHz for the processor clock (when not using the internal PLL)

The following hypothesis is taken for the calculation of the decoupling capacitance:

- 1.5 nH is issued from the connection of the capacitor to the PCB
- 1.5 nH is issued from the capacitor intrinsic inductance

Figure 52. Capacitor description



This hypothesis corresponds to a capacitor connected to two micro-vias on a PCB.

The filter defined by the inductance and the decoupling capacitor shall be able to filter the characteristic frequencies of the application. Each frequency to filter is defined by:

$$fc = \frac{1}{2\pi\sqrt{LC}}$$

- L: the inductance equivalent to the global inductance on the VSS18/VDD18 and VSS33/VCC33 lines.
- C: the decoupling capacitance.

For a processor running at 100 MHz with a PCI interface at a characteristic frequency of 33 MHz and considering that power supply pins are grouped by multiple of four, the decoupling capacitance to set are:

- 33 nF for 33 MHz decoupling
- 3 nF for 100 MHz decoupling

Pin Capacitance

Parameter	Description	MAX
CIN	Standard Input Capacitance	5 pF
CIO	Standard Input/Output Capacitance	5 pF
CINp	PCI Input Capacitance	7 pF
CIOp	PCI Input/Output Capacitance	7 pF

AC Characteristics

The AT697F implements a single event transient (SET) protection mechanism. The influence of this protection is reflected by the timing figures presented in the following tables.

The following tables show the timing figures for the natural and maximum skew conditions.

Natural Skew

Test Conditions

- Natural Skew
- Temperature range: -55°C to 125°C
- Voltage range:
  - I/O: 3.3V ± 0.30V
  - Core: 1.8V ± 0.15V
- Output load: 50 pF
- Voltage threshold Test condition: Vcc/2

Table 117. AC Characteristics - Natural Skew

Parameter	Min (ns)	Max (ns)	Comment	Reference edge ('+' for rising edge)
t1	10		CLK period with PLL disabled	
t1_p	40	50	CLK period with PLL enabled	
t2	4.5		CLK low and high pulse width - PLL disabled	
t2_p	18		CLK low and high pulse width - PLL enabled	
t3	10		SDCLK period	
t4	3	7	SDCLK output delay - PLL disabled	CLK
t5		1.10^7	PLL setup time	
t6	1*t3		RESET* low pulse width(1)	
t10	1.5	7	A[27:0] output delay	SDCLK +
t11	2	8	D[31:0] and CB[7:0] output delay	SDCLK +
t12	4		D[31:0] and CB[7:0] setup time	SDCLK +
t13	0		D[31:0] and CB[7:0] hold time during load/fetch	SDCLK +
t14	0	9	D[31:0] and CB[7:0] hold time during write(2)	SDCLK +
t15	2	7	OE*, READ and WRITE* output delay	SDCLK +
t16	2	5.5	ROMS*[1:0] output delay	SDCLK +
t17	2	6	RAMS*[4:0], RAMOE*[4:0] and RWE*[3:0] output delay	SDCLK +
t18	2	5.5	IOS* output delay	SDCLK +
t19	5		BRDY* setup time	SDCLK +
t20	0		BRDY* hold time	SDCLK +
t21	3	8	SDCAS* output delay	SDCLK +
t22	2	8.5	SDCS*[1:0], SDRAS*, SDWE* and SDDQM*[3:0] output delay	SDCLK +
t23	4		BEXC* setup time	SDCLK +
t24	0		BEXC* hold time	SDCLK +
t25	2.5	9	PIO[15:0] output delay	SDCLK +
t26	4.5		PIO[15:0] setup time	SDCLK +
t27	0		PIO[15:0] hold time during load	SDCLK +
t28	2.5		PIO[15:0] hold time during write(2)	SDCLK +
t101	30		PCI_CLK period	
t102	14.5		PCI_CLK low and high pulse width	
t110	4	12	A/D[31:0] and C/BE[3:0] output delay	PCI_CLK +
t111	6		A/D[31:0] and C/BE[3:0] setup time	PCI_CLK +
t112	0		A/D[31:0] and C/BE[3:0] hold time	PCI_CLK +
t113	4	11	FRAME*, PAR, PERR*, SERR*, STOP* and DEVSEL* output delay	PCI_CLK +

Parameter	Min (ns)	Max (ns)	Comment	Reference edge ('+' for rising edge)
t114	4	11	IRDY* and TRDY* output delay	PCI_CLK +
t115	4	12	REQ* output delay	PCI_CLK +
t116	7		FRAME*, LOCK*, PAR, PERR*, SERR*, IDSEL, STOP* and DEVSEL* setup time	PCI_CLK +
t117	7		IRDY* and TRDY* setup time	PCI_CLK +
t118	9		GNT* setup time	PCI_CLK +
t119	0		FRAME*, LOCK*, PAR, PERR*, SERR*, IDSEL, STOP* and DEVSEL* hold time	PCI_CLK +
t120	0		IRDY* and TRDY* hold time	PCI_CLK +
t121	0		GNT* hold time	PCI_CLK +

Notes:
1. Although the processor is being reset asynchronously, this timing is a minimum requirement to guarantee a proper reset of the processor: a glitch of any shorter duration may lead to an unpredictable behavior.
2. The given timing indicates when the buffer is not driving any level on the bus. This timing is independent of the capacitive load.

Maximum Skew

Test Conditions

- Maximum Skew Programmed
- Temperature range: -55°C to 125°C
- Voltage range:
  - I/O: 3.3V ± 0.30V
  - Core: 1.8V ± 0.15V
- Output load: 50pF
- Voltage threshold Test condition: Vcc/2

Table 118. AC Characteristics - Maximum Skew

Parameter	Min (ns)	Max (ns)	Comment	Reference edge ('+' for rising edge)
t1	12		CLK period with PLL disabled	
t1_p	48	50	CLK period with PLL enabled	
t2	5.4		CLK low and high pulse width - PLL disabled	
t2_p	21		CLK low and high pulse width - PLL enabled	
t3	12		SDCLK period	
t4	3	7	SDCLK output delay - PLL disabled	CLK
t5		1.10^7	PLL setup time	
t6	1*t3		RESET* low pulse width(1)	
t10	1.5	8	A[27:0] output delay	SDCLK +
t11	2	9	D[31:0] and CB[7:0] output delay	SDCLK +
t12	4		D[31:0] and CB[7:0] setup time	SDCLK +
t13	0		D[31:0] and CB[7:0] hold time	SDCLK +
t14	1	11	D[31:0] and CB[7:0] hold time during write(2)	SDCLK +
t15	2	7.5	OE*, READ and WRITE* output delay	SDCLK +
t16	2	8	ROMS*[1:0] output delay	SDCLK +
t17	2	7	RAMS*[4:0], RAMOE*[4:0] and RWE*[3:0] output delay	SDCLK +
t18	2	7	IOS* output delay	SDCLK +
t19	5		BRDY* setup time	SDCLK +
t20	0		BRDY* hold time	SDCLK +
t21	3	10	SDCAS* output delay	SDCLK +
t22	2	9.5	SDCS*[1:0], SDRAS*, SDWE* and SDDQM[3:0] output delay	SDCLK +
t23	4		BEXC* setup time	SDCLK +
t24	0		BEXC* hold time	SDCLK +
t25	2.5	11	PIO[15:0] output delay	SDCLK +
t26	4.5		PIO[15:0] setup time	SDCLK +
t27	0		PIO[15:0] hold time	SDCLK +
t28	2.5		PIO[15:0] hold time during write(2)	SDCLK +
t101	30		PCI_CLK period	
t102	14.5		PCI_CLK low and high pulse width	
t110	4	13	A/D[31:0] and C/BE[3:0] output delay	PCI_CLK +
t111	6		A/D[31:0] and C/BE[3:0] setup time	PCI_CLK +
t112	0		A/D[31:0] and C/BE[3:0] hold time	PCI_CLK +
t113	4	12	FRAME*, PAR, PERR*, SERR*, STOP* and DEVSEL* output delay	PCI_CLK +
t114	4	12.5	IRDY* and TRDY* output delay	PCI_CLK +

Parameter	Min (ns)	Max (ns)	Comment	Reference edge ('+' for rising edge)
t115	4	13	**REQ*** output delay	**PCI_CLK** +
t116	7.5		**FRAME***, LOCK*, **PAR**, **PERR***, **SERR***, **IDSEL**, **STOP*** and **DEVSEL*** setup time	**PCI_CLK** +
t117	7.5		**IRDY*** and **TRDY*** setup time	**PCI_CLK** +
t118	9.5		**GNT*** setup time	**PCI_CLK** +
t119	0.5		**FRAME***, **PCI_LOCK***, **PAR**, **PERR***, **SERR***, **IDSEL**, **STOP*** and **DEVSEL*** hold time	**PCI_CLK** +
t120	0.5		**IRDY*** and **TRDY*** hold time	**PCI_CLK** +
t121	0.5		**GNT*** hold time	**PCI_CLK** +

Notes:
1. Although the processor is being reset asynchronously, this timing is a minimum requirement to guarantee a proper reset of the processor: a glitch of any shorter duration may lead to an unpredictable behavior.
2. The given timing applies when the buffer is not driving any level on the bus. This timing is independent of the capacitive load.

Timing Derating

The timing figures change with the capacitance load on each pin, . The following table summarizes the timing derating versus the capacitance load in the whole process / voltage / temperature range.

Table 119. Timing Derating (ns/pF above 50pF)

Signal	Min.	Max.
A[27:0]	0.019	0.053
A/D[31:0]	0.013	0.035
AGNT*[3:0]	0.013	0.036
CB[7:0]	0.023	0.075
C/BE*[3:0]	0.013	0.035
D[31:0]	0.023	0.075
DEVSEL*	0.013	0.035
DSUACT	0.078	0.215
DSUTX	0.078	0.215
ERROR*	0.019	0.053
FRAME*	0.013	0.035
IOS*	0.019	0.053
IRDY*	0.013	0.035
LOCK	0.039	0.107
OE*	0.019	0.053
PAR	0.013	0.035
PCI_LOCK*	0.013	0.035
PERR*	0.013	0.035
PIO[15:0]	0.046	0.151
RAMOE*[4:0]	0.019	0.053
RAMS*[4:0]	0.019	0.053
READ	0.019	0.053
REQ*	0.013	0.035
ROMS*[1:0]	0.019	0.053
RWE*[3:0]	0.019	0.053
SDCAS*	0.039	0.107
SDCLK	0.019	0.053
SDCS*[1:0]	0.039	0.107
SDDQM[3:0]	0.039	0.107
SDRAS*	0.039	0.107
SDWE*	0.039	0.107
SERR*	0.013	0.035
STOP*	0.013	0.035
TDO	0.019	0.053
TRDY*	0.013	0.035

Signal	Min.	Max.
WDOG*	0.019	0.053
WRITE*	0.039	0.107

Note: The values provided in this table are not tested, they are for information only.

Timing Diagrams

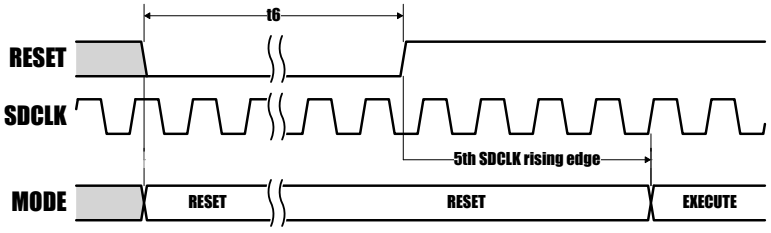
Diagram List

- Reset Sequence
- Clock Input without PLL
- Clock Input with PLL
- Fetch, Read and Write from/to 32-bit PROM - 0 wait-states
- Fetch, Read and Write from/to 32-bit PROM - 2n wait-states
- Fetch, Read and Write from/to 32-bit PROM - 2n wait-states + sync. **BRDY***
- Fetch from 8-bit PROM with EDAC disabled - 2n wait-states
- Word Write to 8-bit PROM with EDAC disabled - 2n wait-states
- Byte and Half-Word Write to 8-bit PROM with EDAC disabled - 2n wait-states
- Fetch from 8-bit PROM with EDAC enabled - 2n wait-states
- Fetch, Read and Write from/to 32-bit SRAM - 0 wait-states
- Fetch, Read and Write from/to 32-bit SRAM - n wait-states
- Fetch, Read and Write from/to 32-bit SRAM with Instruction Burst - 0 wait-states
- Fetch, Read and Write from/to 32-bit SRAM with Instruction Burst - n wait-states
- Burst of SRAM Fetches with Instruction Cache and Burst enabled - 0 wait-states
- Burst of SRAM Fetches with Instruction Cache and Burst enabled - n wait-states
- SDRAM Read (or Fetch) with Precharge - Burst Length = 1; CL = 3
- SDRAM Write with Precharge - Burst Length = 1; CL = 3
- Fetch from ROM, Read and Write from/to 32-bit I/O - 0 wait-states
- Fetch from ROM, Read and Write from/to 32-bit I/O - n wait-states
- Fetch from ROM, Read and Write from/to 32-bit I/O - n wait-states + sync. **BRDY***

Caution: The timing diagrams with fetch, read and/or write operations were generated using specific instruction sequences. Considering the complex nature of the interactions within the processor (IU pipeline, memory-controller, instruction cache...), the signals waveforms found in a final application may possibly exhibit slight functional cycle variations over the proposed timing diagrams. Source code for the timing diagrams is available on request.

Reset

Figure 53. Reset Sequence



Clock

Figure 54. Clock Input without PLL

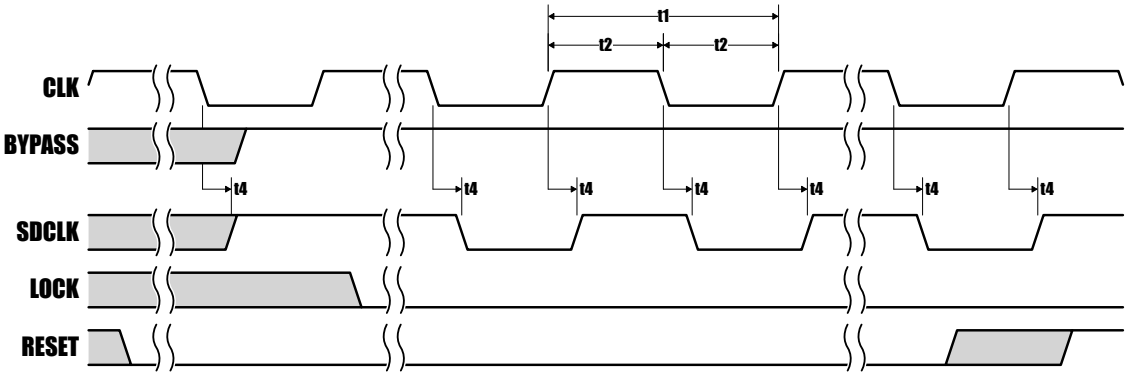


Figure 55. Clock Input with PLL

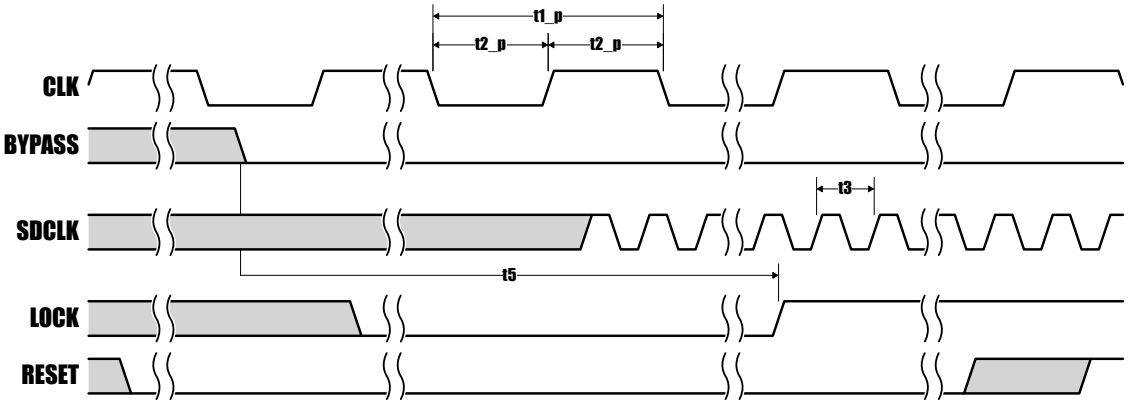


Figure 56. Fetch, Read and Write from 32-bit PROM - 0 wait-states

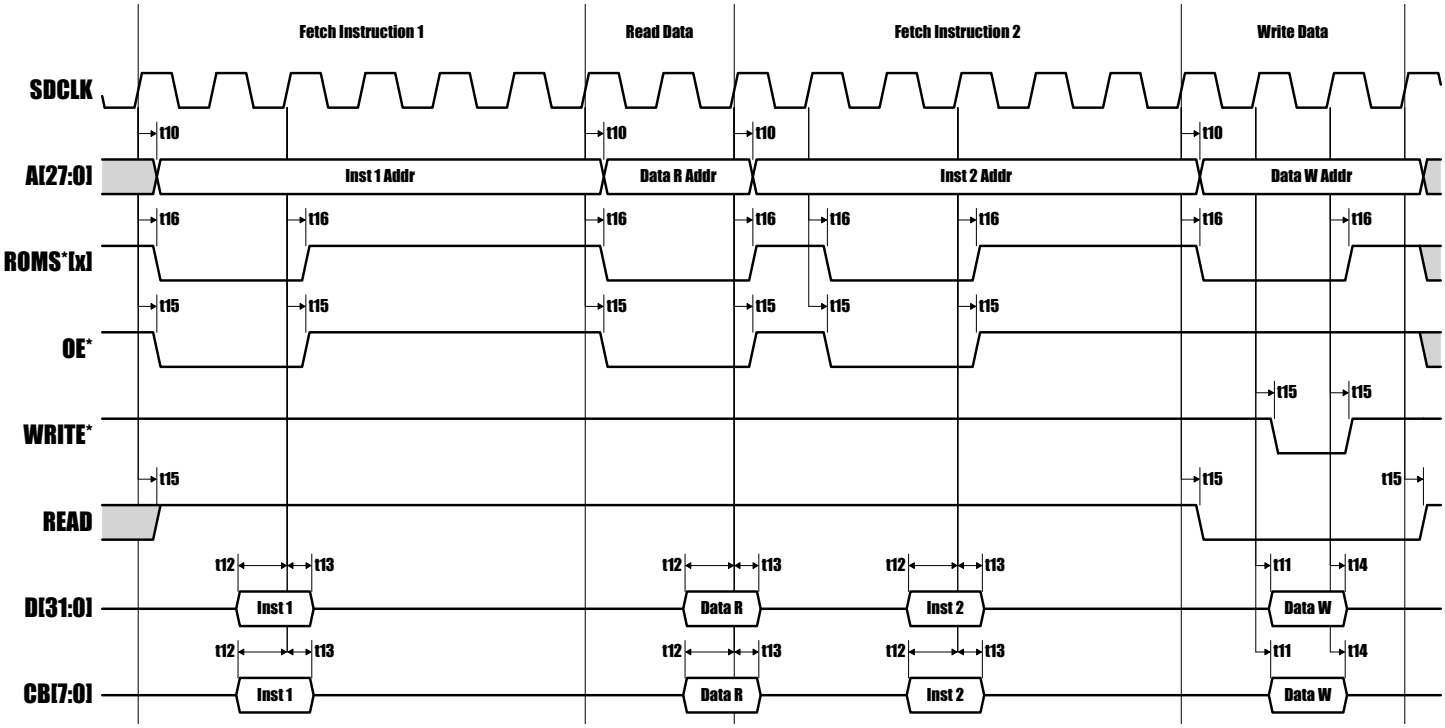


Figure 57. Fetch, Read and Write from 32-bit PROM - 2n wait-states

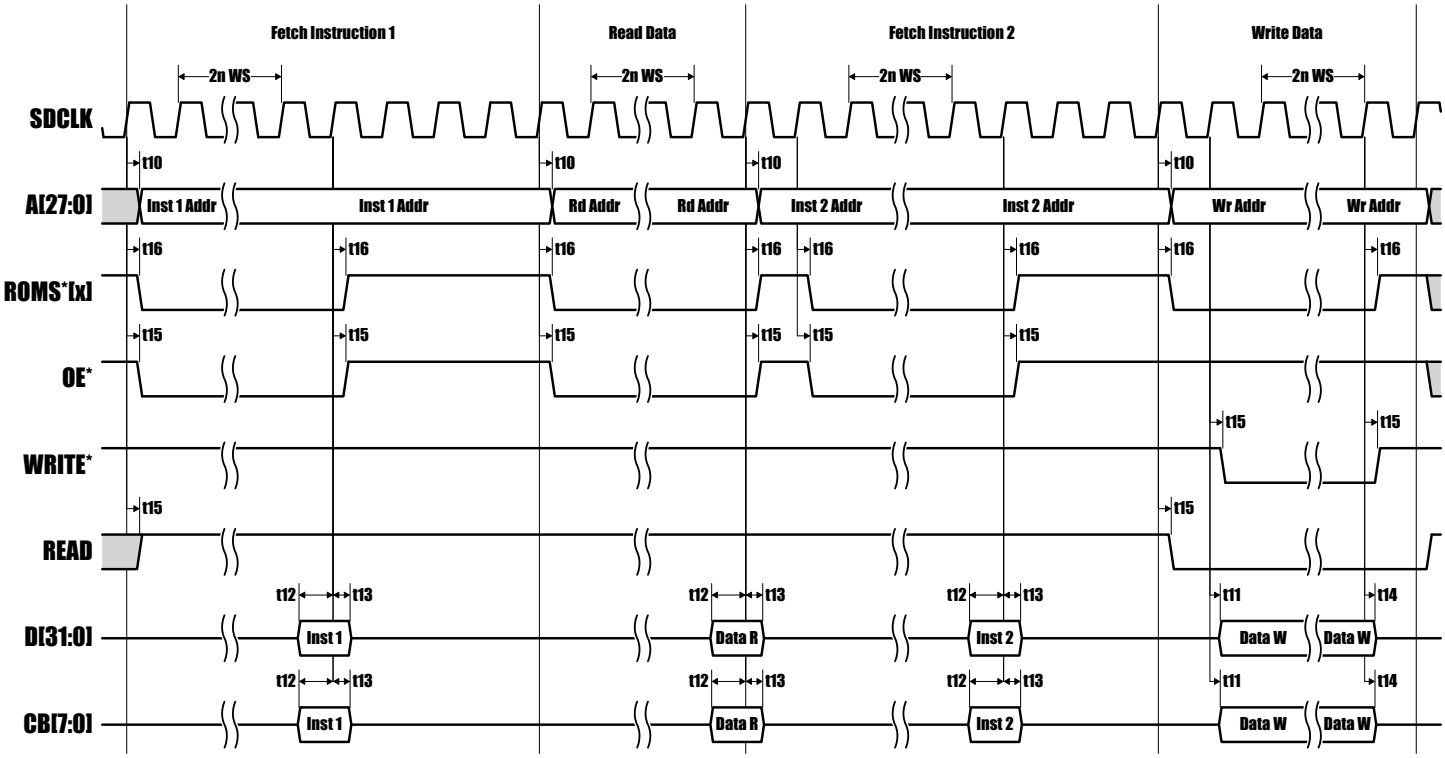


Figure 58. Fetch, Read and Write from 32-bit PROM - 2n wait-states + **BRDY***

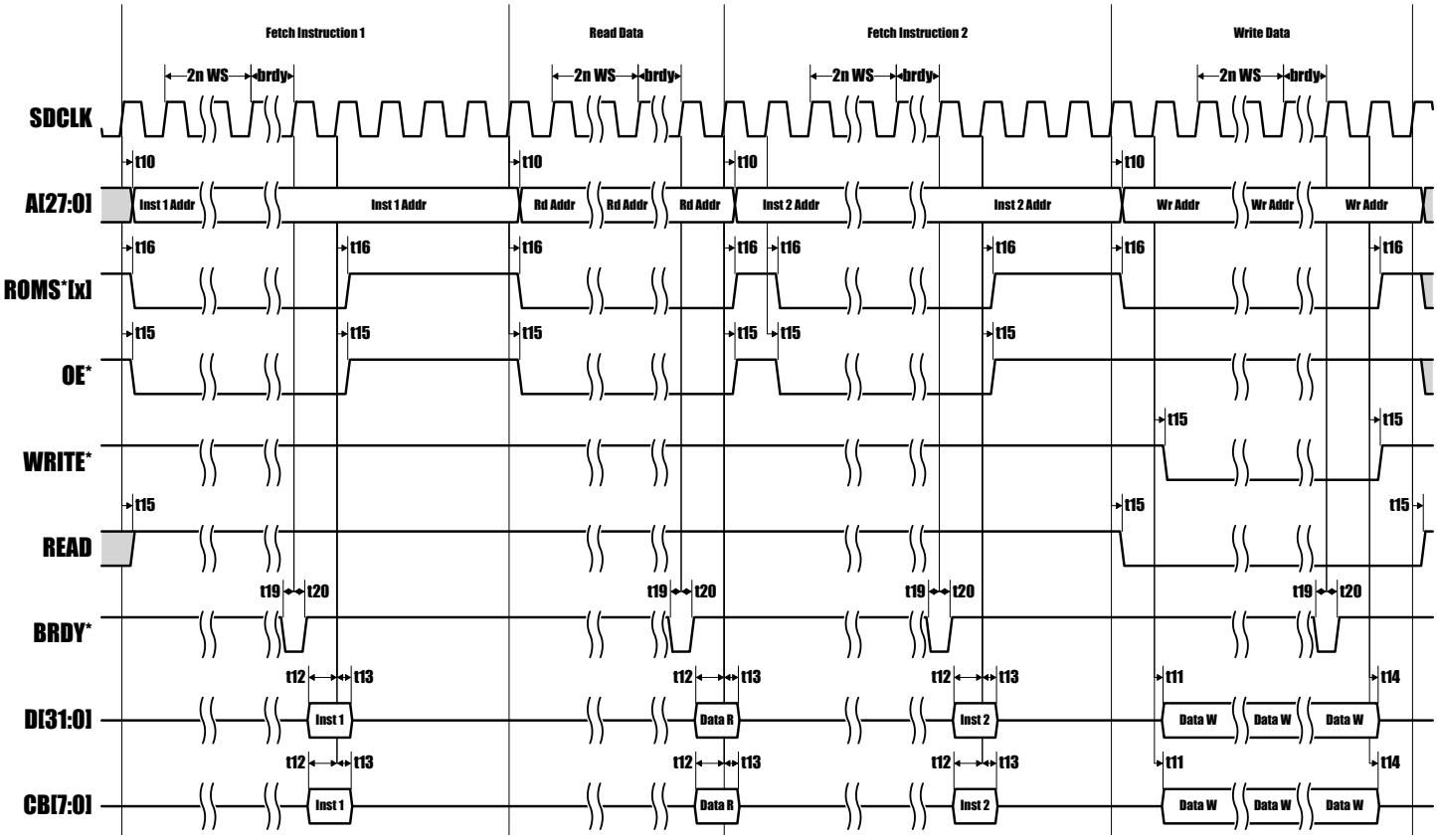


Figure 59. Fetch from 8-bit PROM with EDAC disabled - 2n wait-states

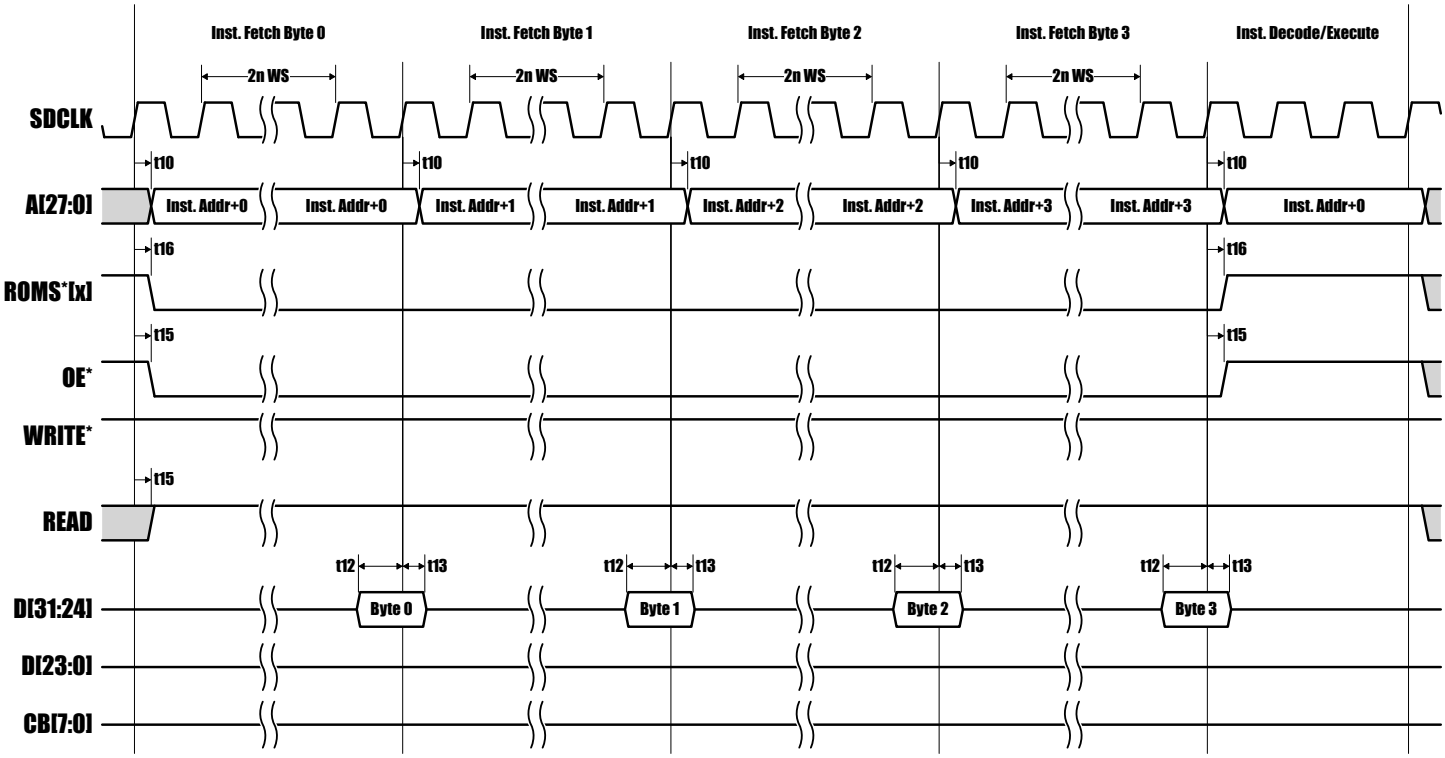


Figure 60. Word Write to 8-bit PROM with EDAC disabled - 2n wait-states

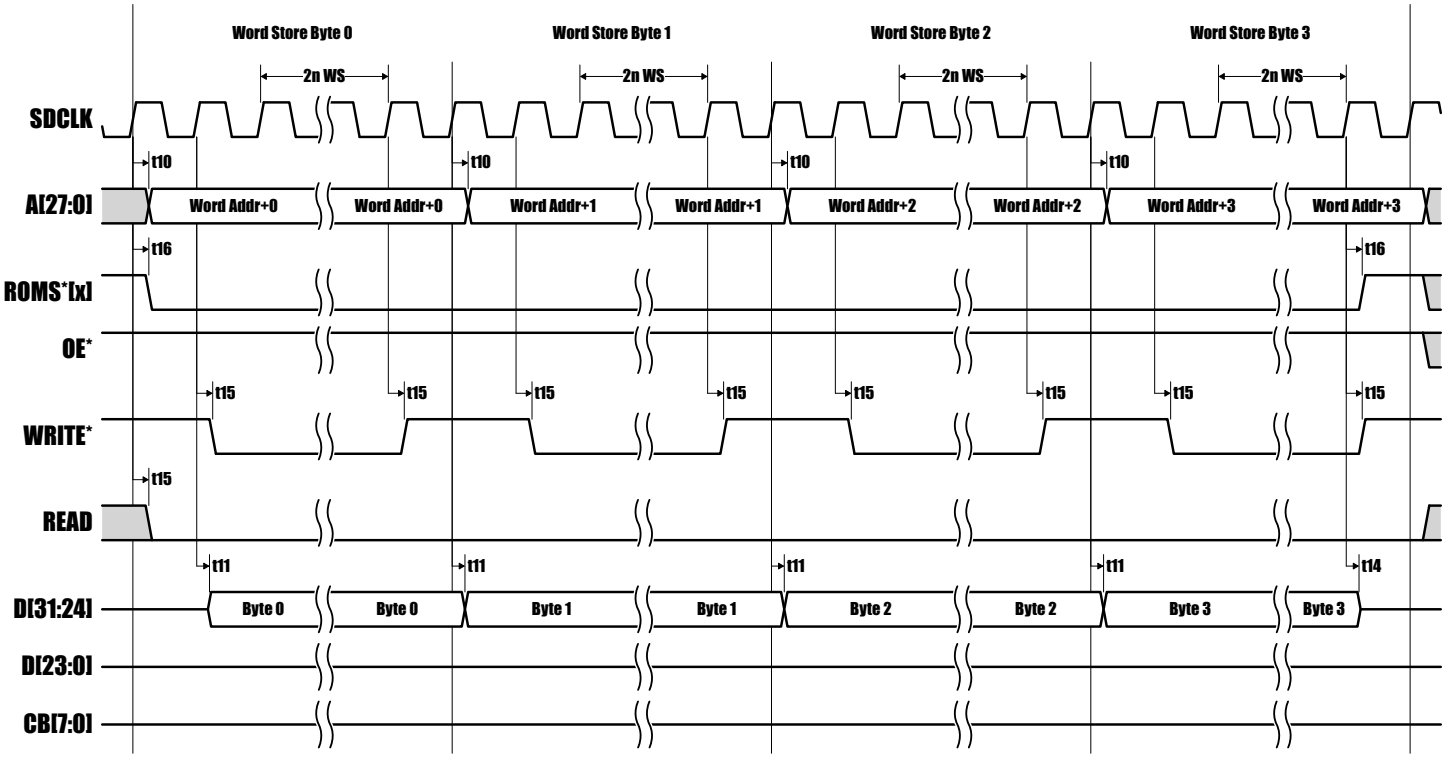


Figure 61. Byte and Half-Word Write to 8-bit PROM with EDAC disabled - 2n wait-states

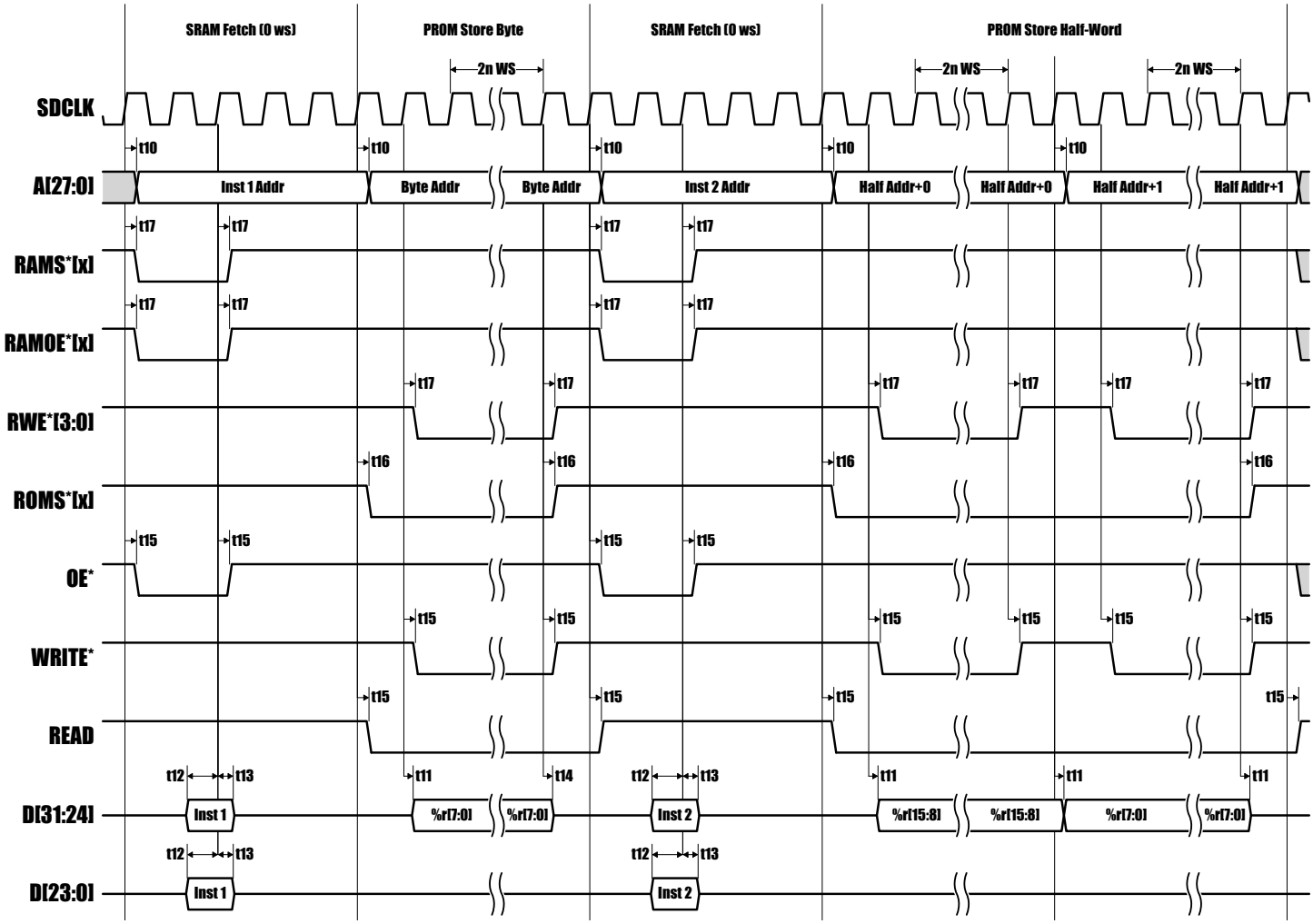


Figure 62. Fetch from 8-bit PROM with EDAC enabled - 2n wait-states

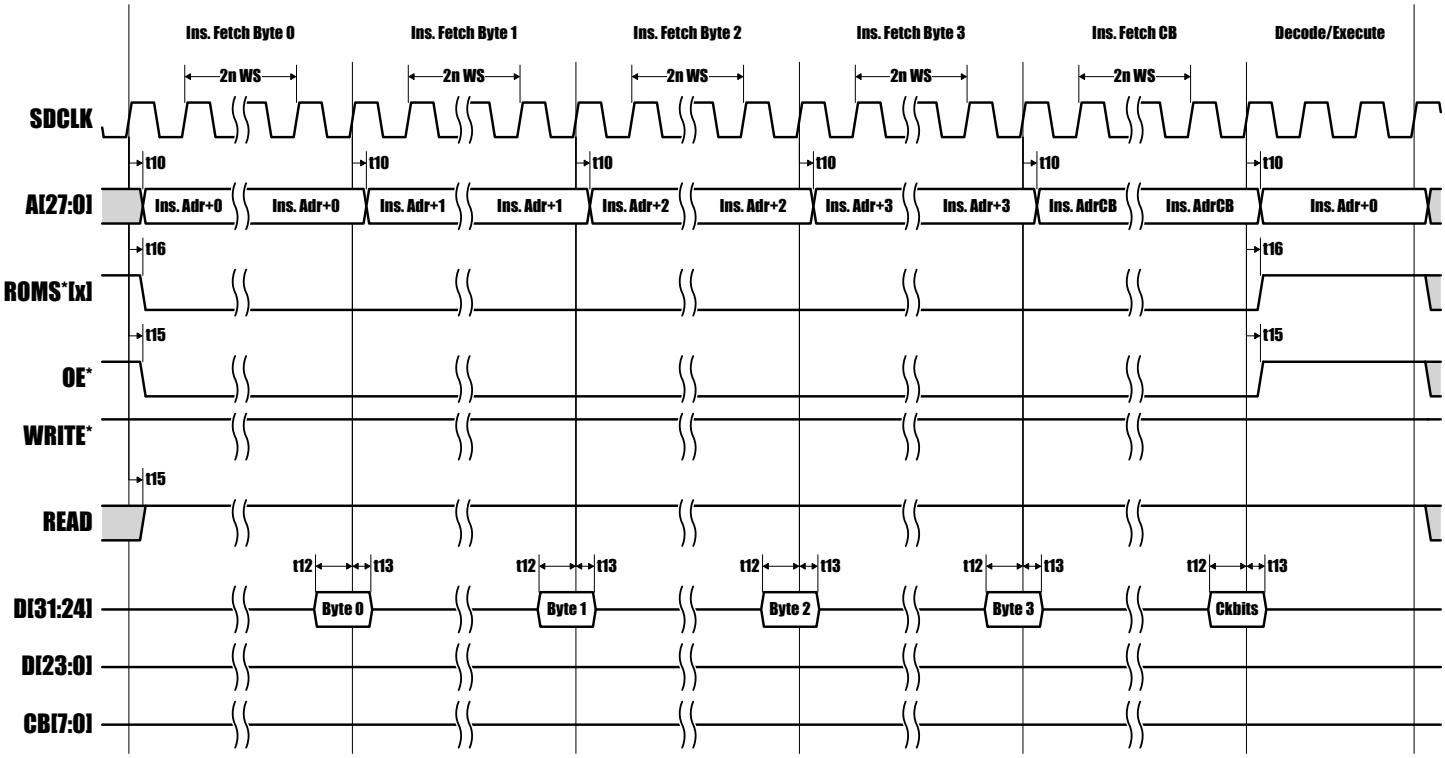


Figure 63. Fetch, Read and Write from/to 32-bit SRAM - 0 wait-states

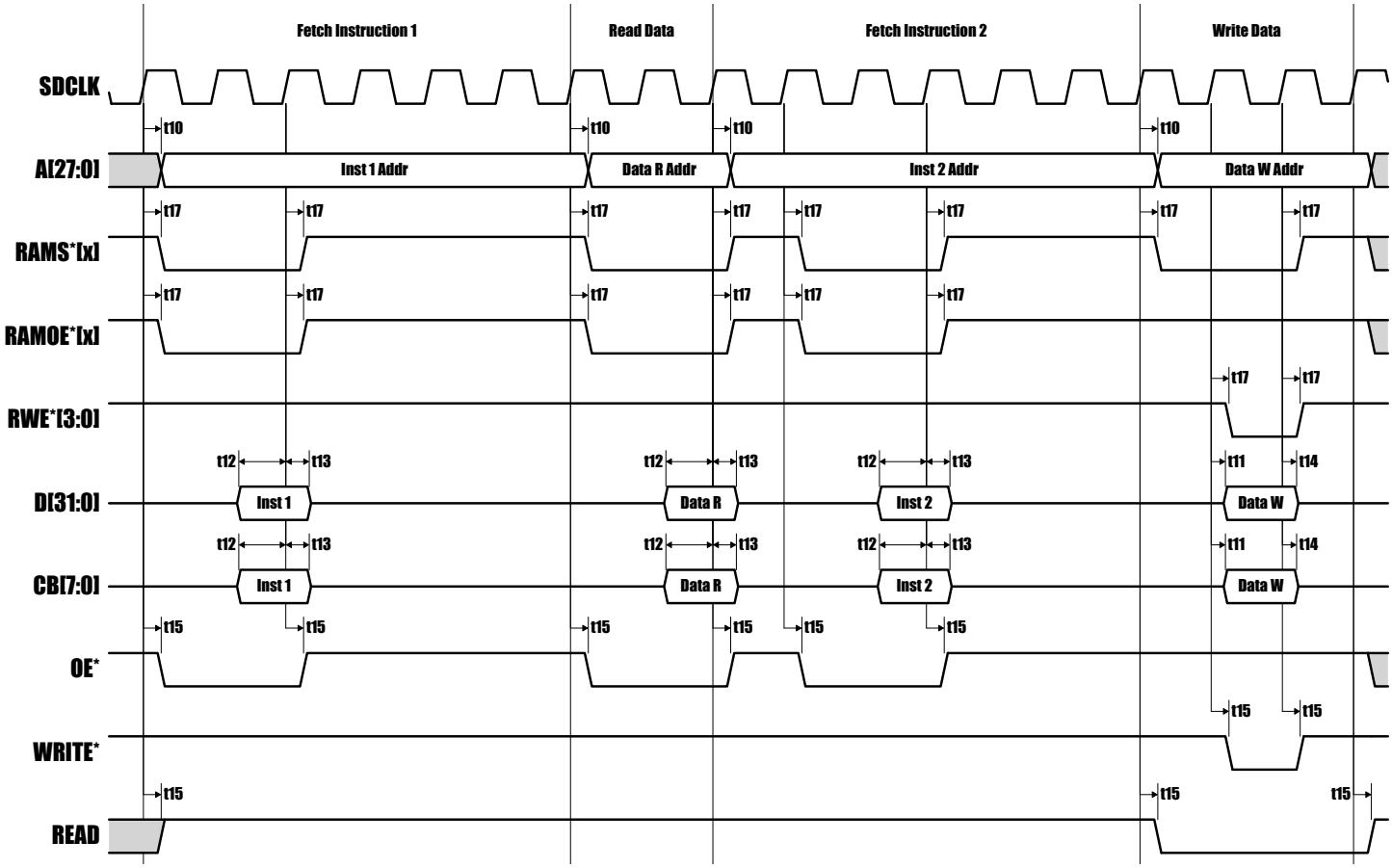


Figure 64. Fetch, Read and Write from/to 32-bit SRAM - n wait-states

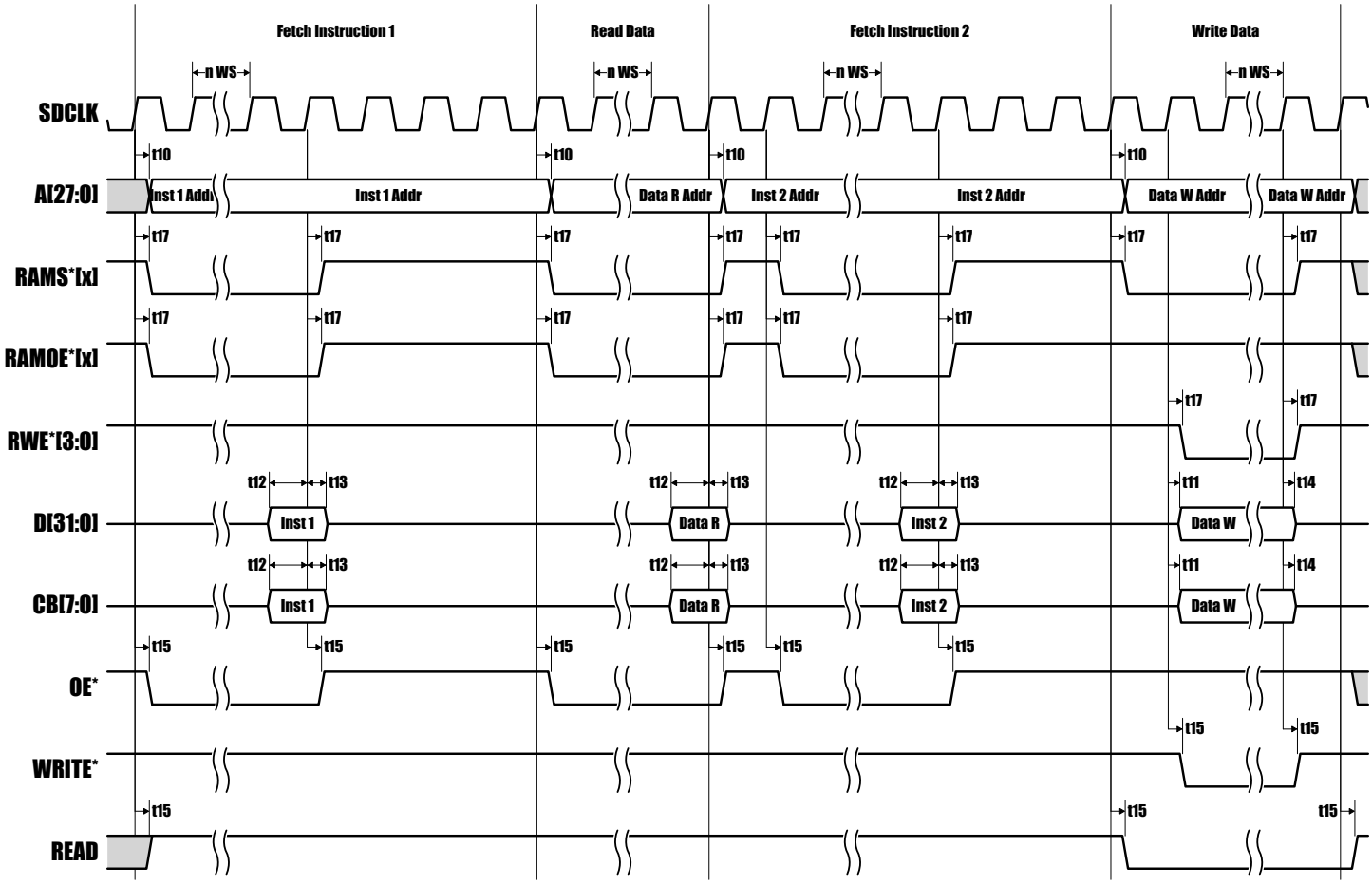


Figure 65. Fetch, Read and Write from/to 32-bit SRAM with Instruction Burst - 0 wait-states

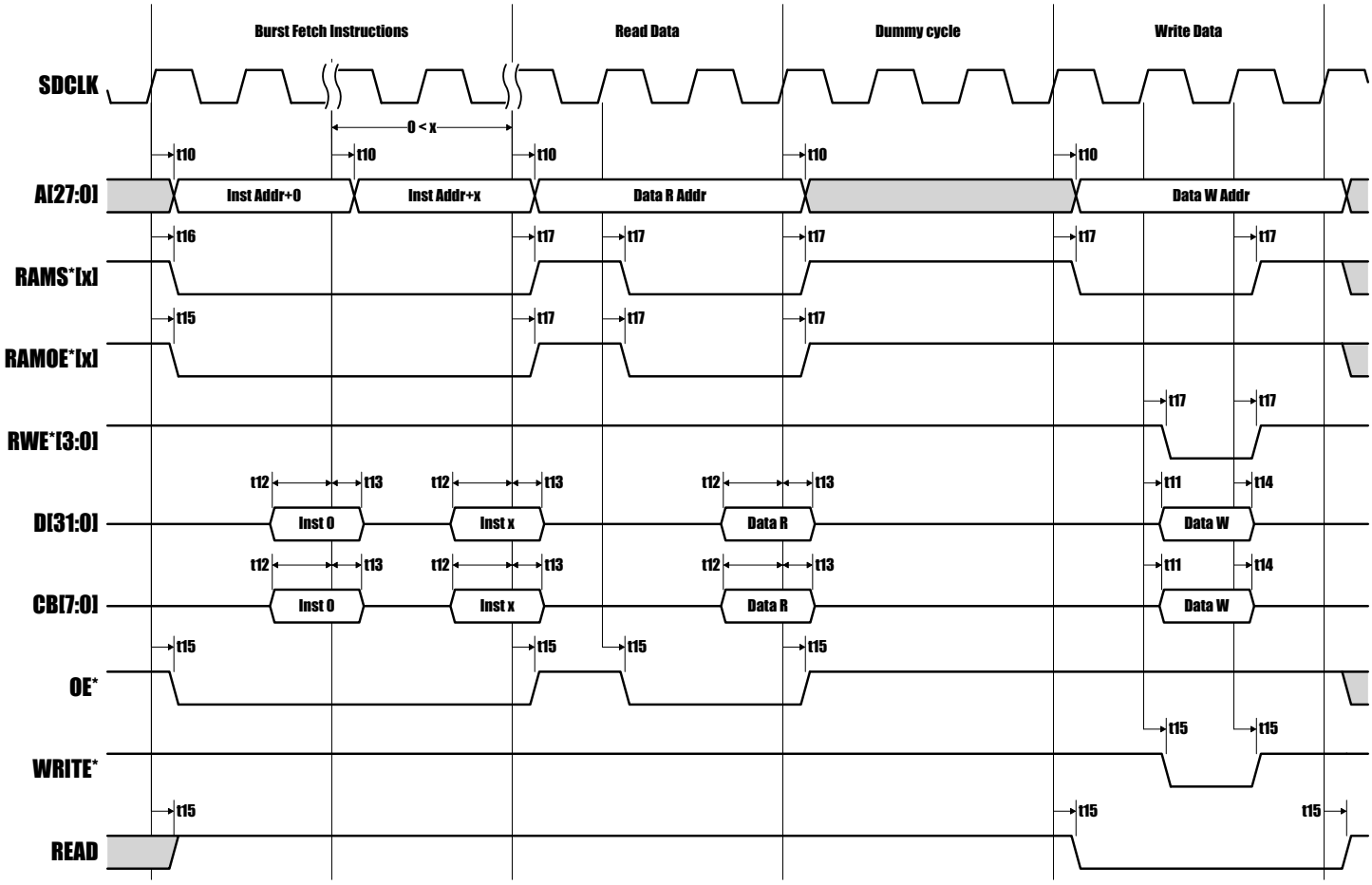


Figure 66. Fetch, Read and Write from/to 32-bit SRAM with Instruction Burst - n wait-states

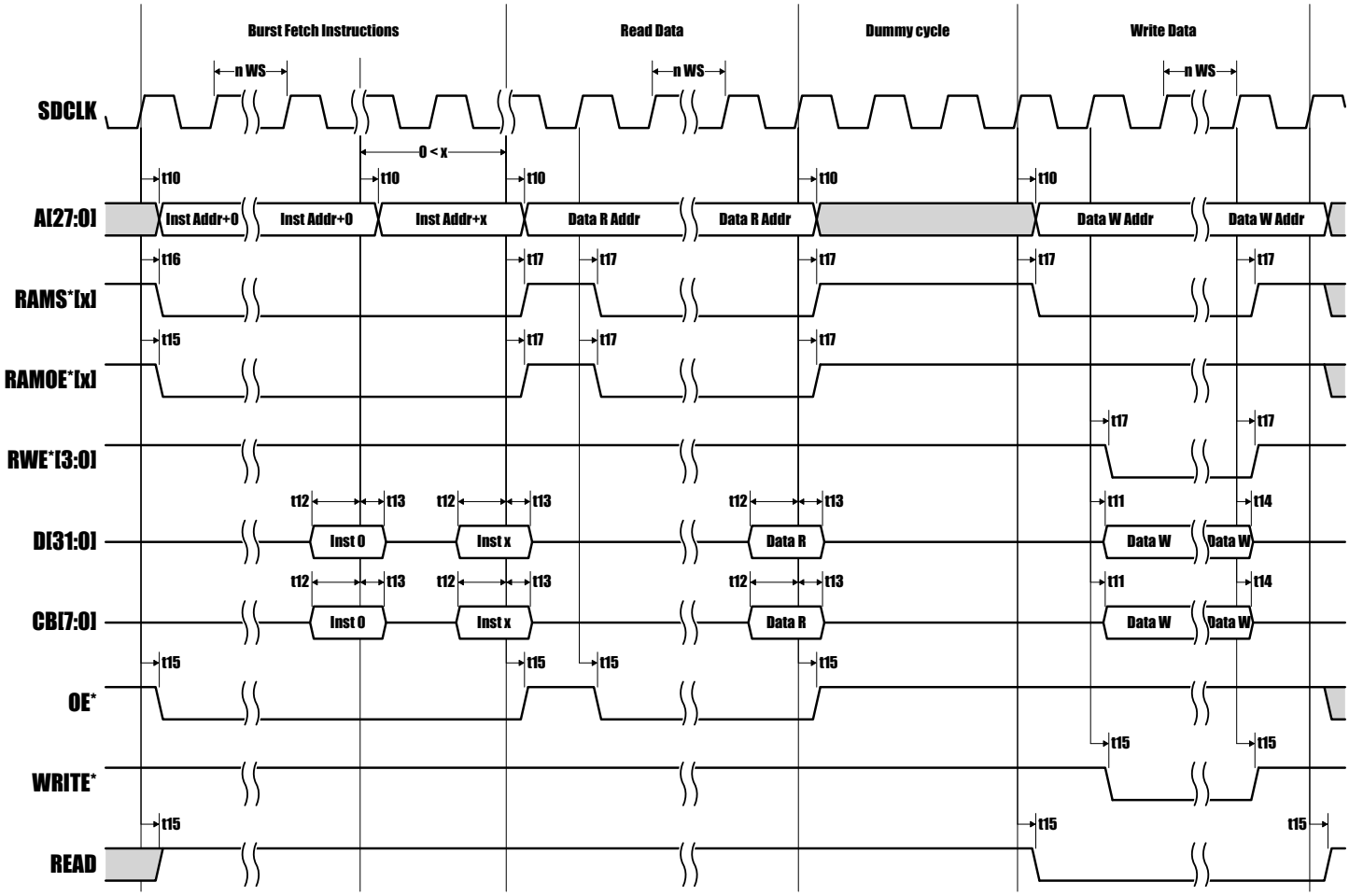


Figure 67. Burst of SRAM Fetches with Instruction Cache and Burst enabled - 0 wait-states

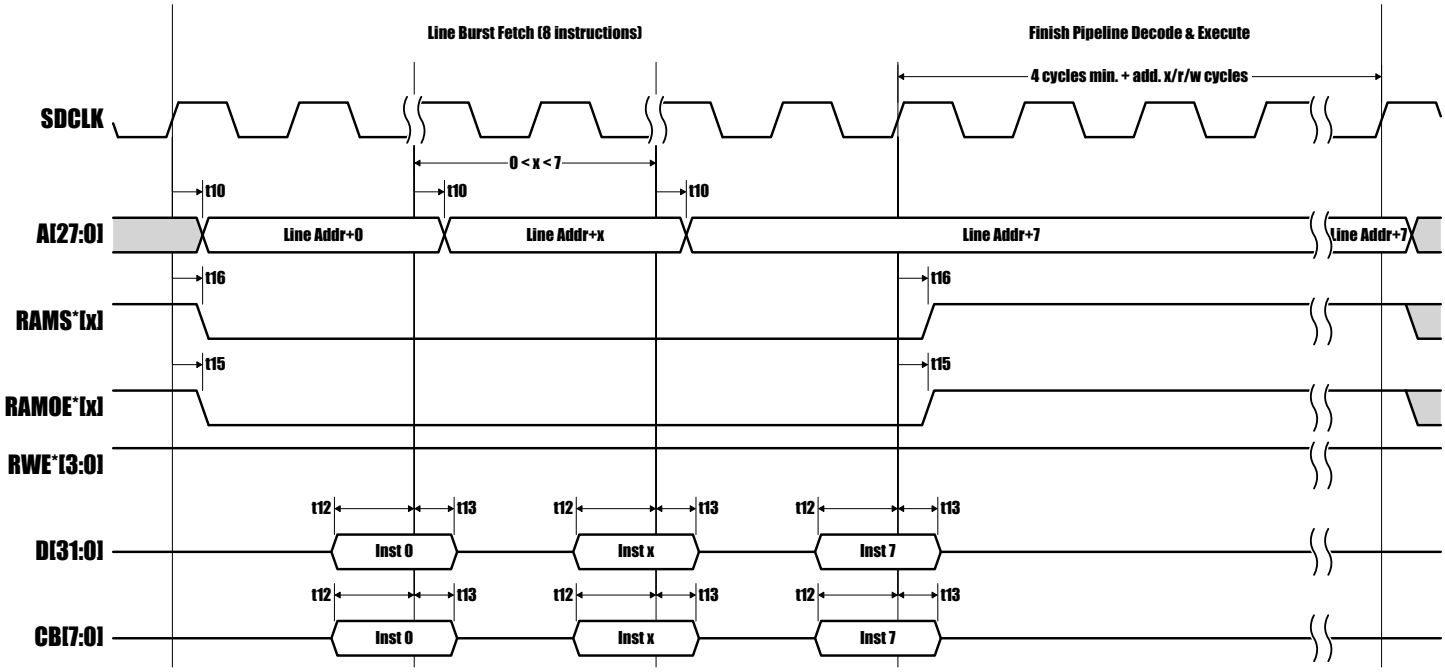
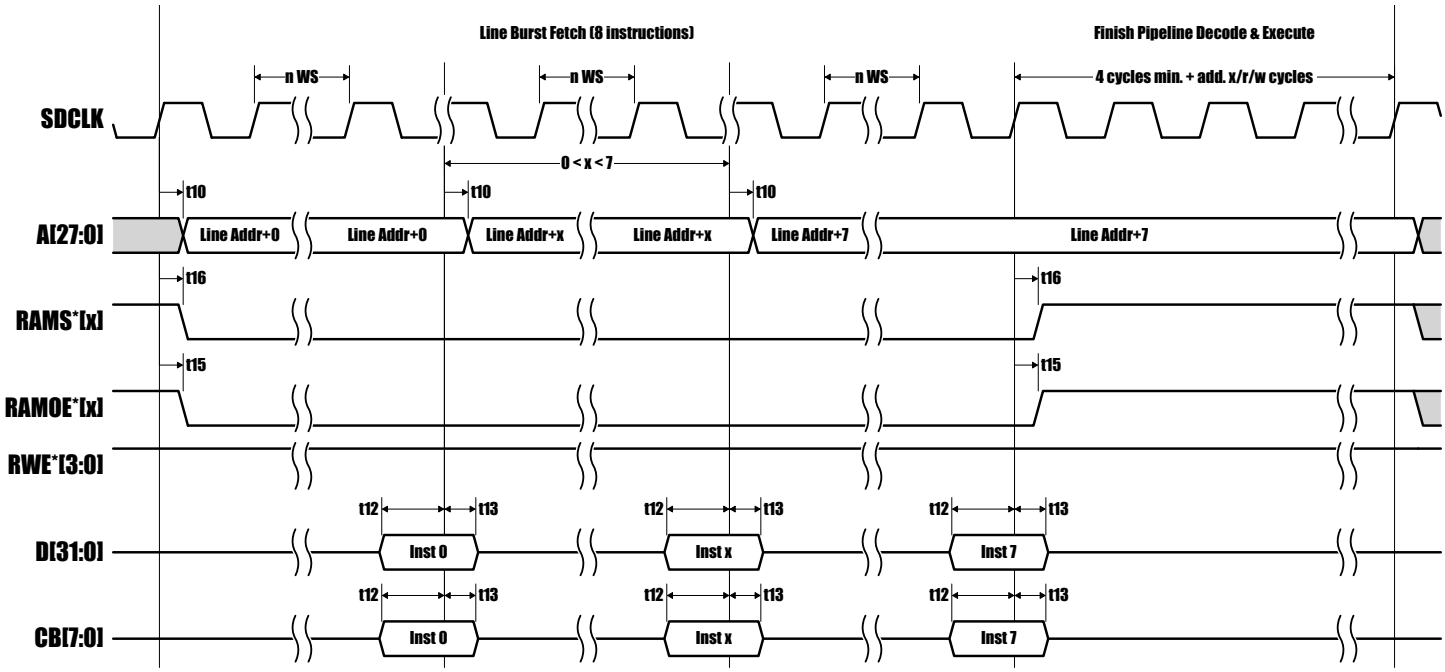


Figure 68. Burst of SRAM Fetches with Instruction Cache and Burst enabled - n wait-states



SDRAM

Figure 69. SDRAM Read (or Fetch) with Precharge - Burst length = 1; CL = 3

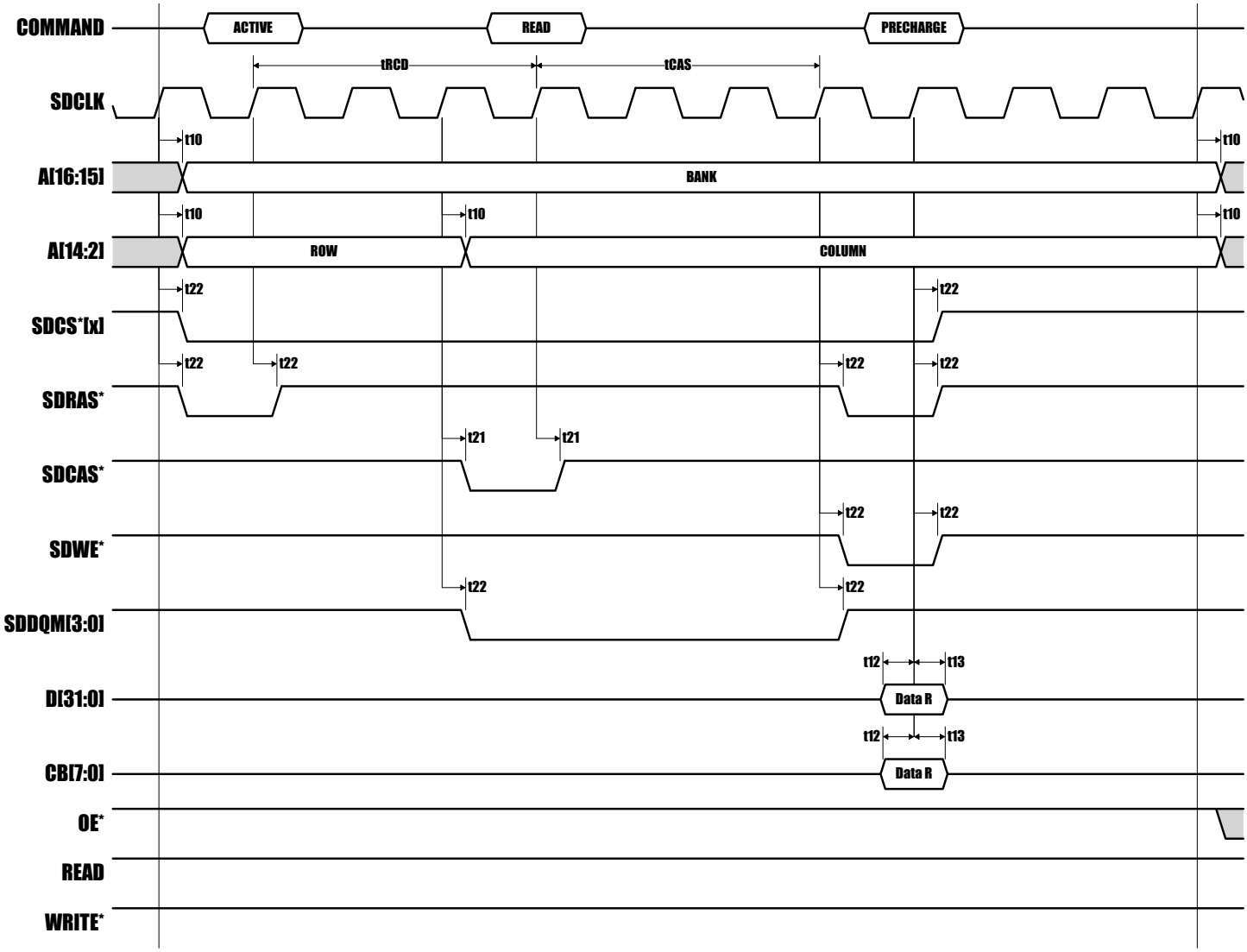
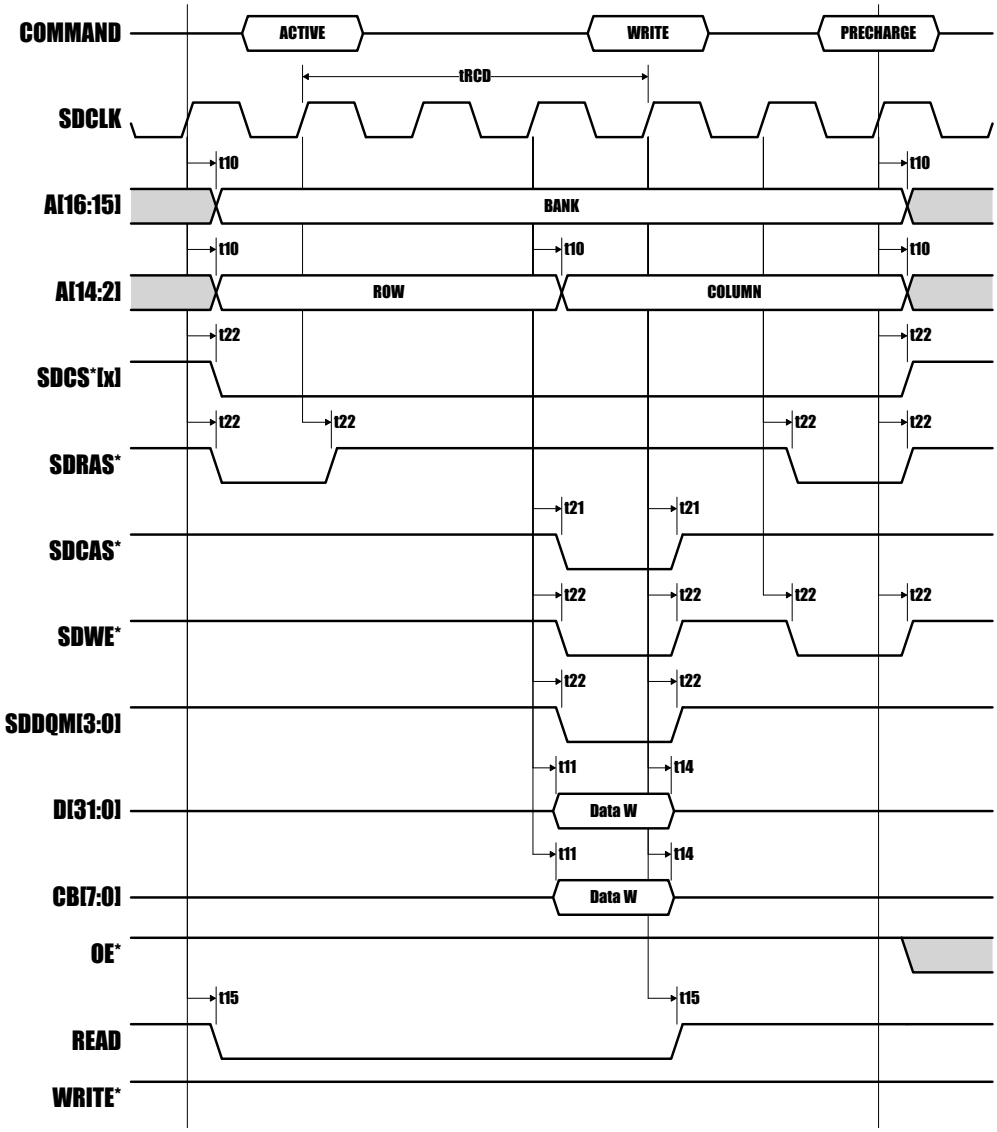


Figure 70. SDRAM Write with Precharge - Burst length = 1; CL = 3



I/O

Figure 71. Fetch from ROM, Read and Write from/to 32-bit I/O - 0 wait-states

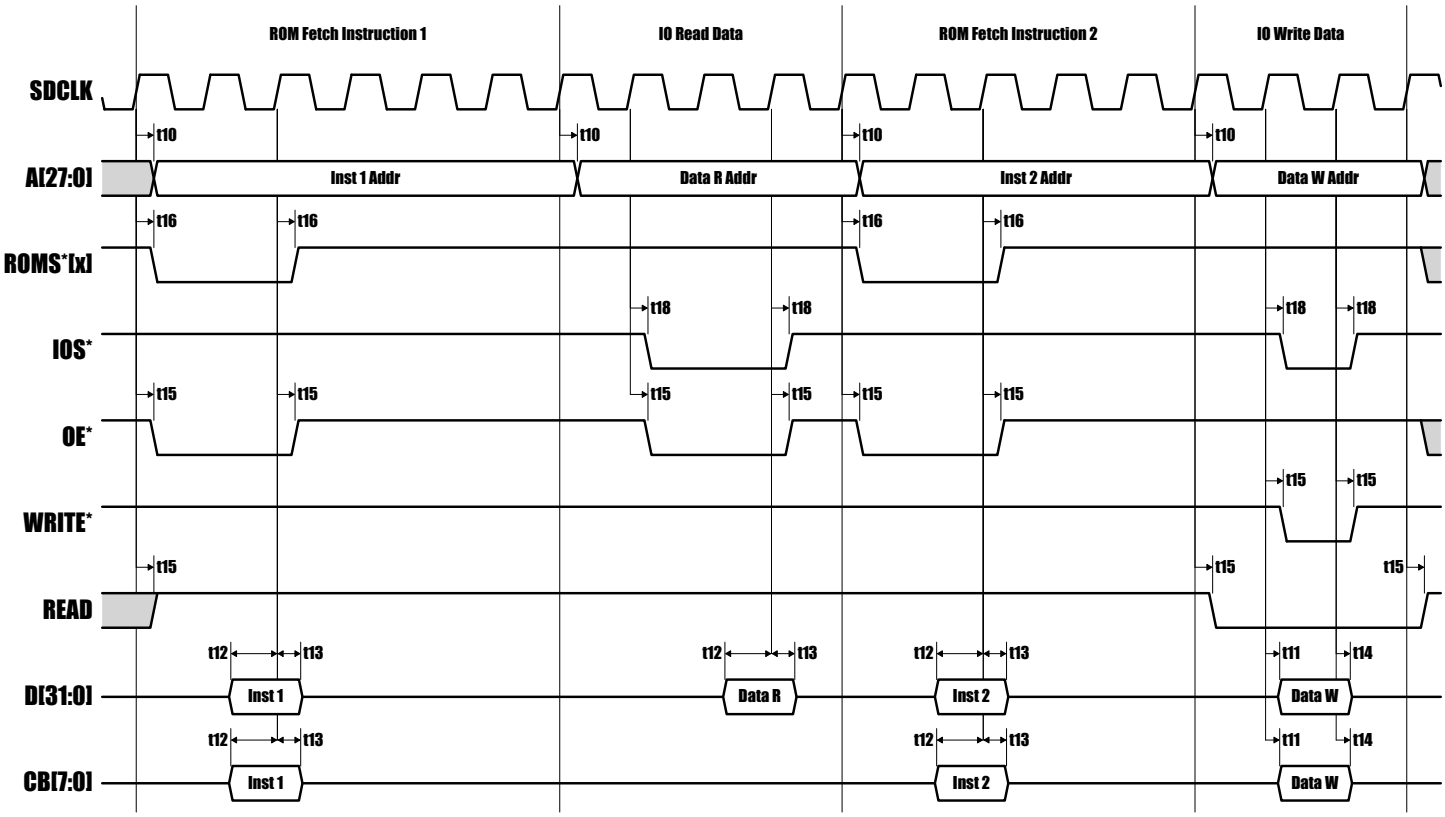


Figure 72. Fetch from ROM, Read and Write from/to 32-bit I/O - n wait-states

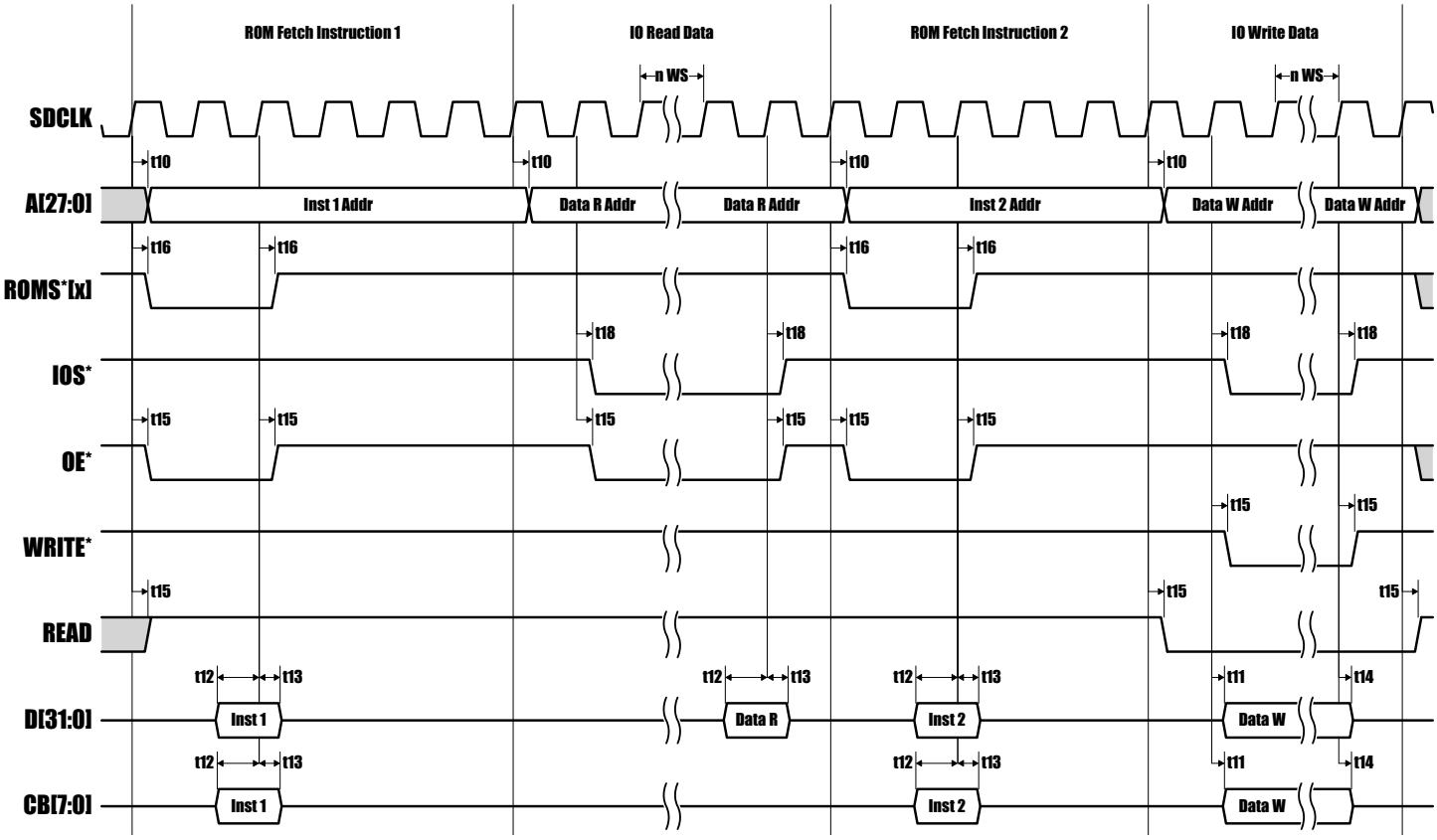
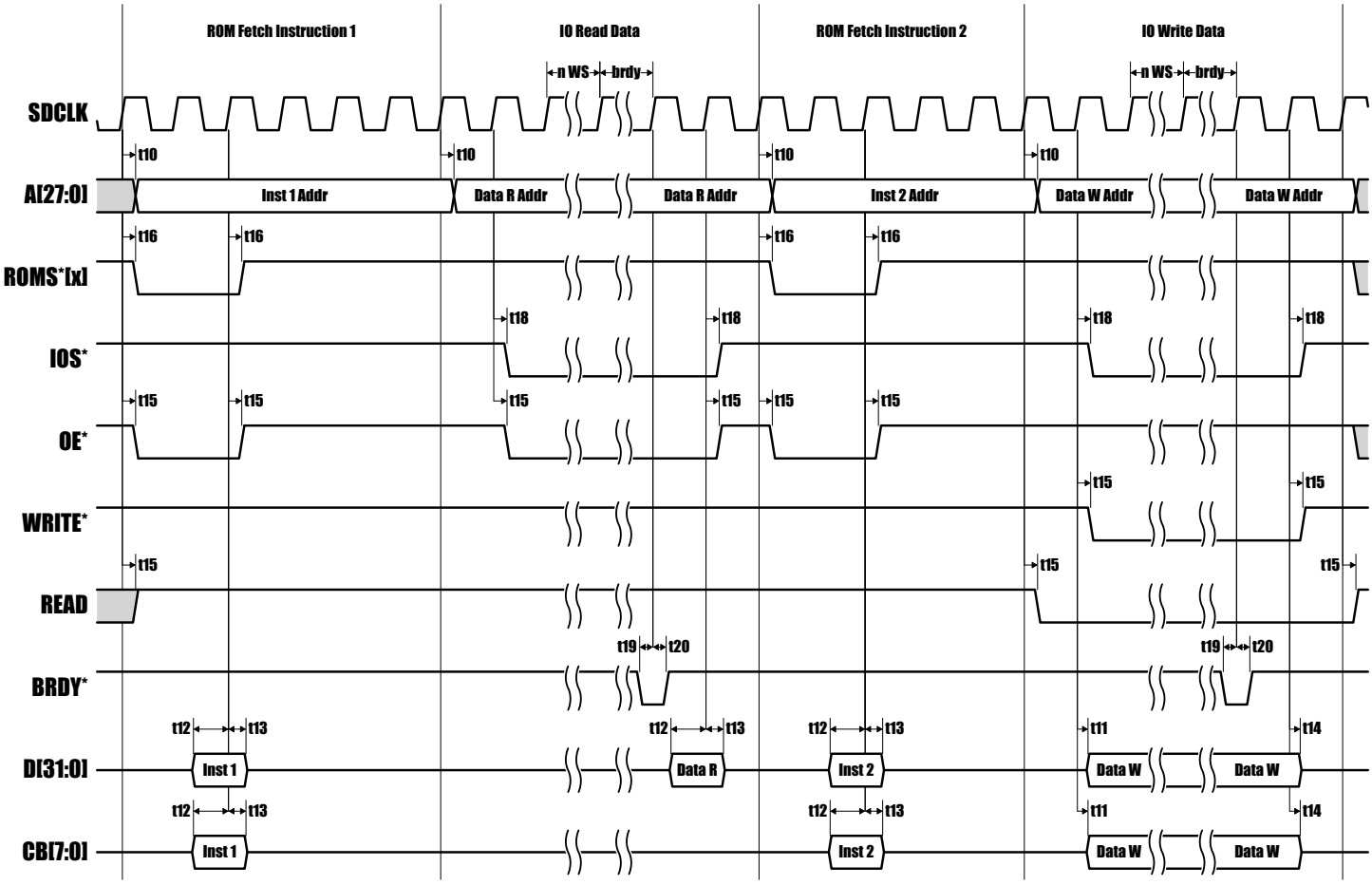


Figure 73. Fetch from ROM, Read and Write from/to 32-bit I/O - n wait-states + sync. **BRDY***



Differences between AT697F and AT697E

This section summarizes the modifications, changes and improvements performed on the AT697F with regards to the AT697E.

New/Modified Features

Table 120. Summary of the new/modified features

Feature	AT697F	AT697E
Write protection scheme	Start/End Address and Tag/Mask based	Tag/Mask Address based
BRDY* capability over PROM area	Implemented	Not Implemented
Asynchronous**BRDY*** capability	Implemented	Not Implemented
16-bit wide memory bus support	Not Implemented	Implemented
32-bit timers and watchdog	Implemented	24-bit only
8 external interrupts support	Implemented	limited to 4
PCI **SYSEN*** state visible in a register	Implemented	Not Implemented
PCI configuration registers local read capability in satellite mode	Implemented	Not Implemented
PCI double word transaction as two single transactions support	Not Implemented	Implemented
AHB trace buffer freeze on debug mode entry	Implemented	Not Implemented

In addition to the new/modified features presented in the above table, most of the functional bugs known from the AT697E model are corrected. Please refer to the AT697 errata sheet - 4409C-AERO-07/07 available at www.atmel.com for detailed information on the functional bugs status.

Register Modifications

Table 121. Registers Changes Summary

Register	Address	AT697F Description	AT697E Description
MCFG1	0x80000000	bit 30 - PROM bus ready enable bit 29 - Asynchronous bus ready enable bit 17:14 - Reserved	bit 30 - reserved bit 29 - reserved bit 17:14 - PROM bank size
WPSTA1	0x800000D0	Write Protection Start Address 1 register bit 29:2 - Start address bit 1 - Block protect mode enable	not available
WPSTO1	0x800000D4	Write Protection Stop Address 1 register bit 29:2 - Stop address bit 1 - User write protection enable bit 0 - Supervisor write protection enable	not available
WPSTA2	0x800000D8	Write Protection Start Address 2 register bit 29:2 - Start address bit 1 - Block protect mode enable	not available

Register	Address	AT697F Description	AT697E Description
WPSTO2	0x800000DC	Write Protection Stop Address 2 register bit 29:2 - Stop address bit 1 - User write protection enable bit 0 - Supervisor write protection enable	not available
TIMC1	0x80000040	bit 31:0 - timer counter	bit 24:0 - timer counter
TIMR1	0x80000044	bit 31:0 - timer counter	bit 24:0 - reload counter
TIMC2	0x80000050	bit 31:0 - timer counter	bit 24:0 - timer counter
TIMR2	0x80000054	bit 31:0 - reload counter	bit 24:0 - reload counter
WDG	0x8000004C	bit 31:0 - counter	bit 24:0 - counter
ITMP	0x80000090	bit 31 - IO interrupt 7 priority level bit 29- IO interrupt 6 priority level bit 28 - IO interrupt 5 priority level bit 26 - IO interrupt 4 priority level bit 15 - IO interrupt 7 mask bit 13 - IO interrupt 6 mask bit 12 - IO interrupt 5mask bit 10 - IO interrupt 4mask	bit 31 - reserved bit 29 - reserved bit 28 - reserved bit 26 - reserved bit 15 - reserved bit 13 - reserved bit 12 - reserved bit 10 - reserved
ITP	0x80000094	bit 15 - IO interrupt 7 pending bit 13 - IO interrupt 6 pending bit 12 - IO interrupt 5 pending bit 10 - IO interrupt 4 pending	bit 15 - reserved bit 13 - reserved bit 12 - reserved bit 10 - reserved
ITF	0x80000098	bit 15 - IO interrupt 7 force bit 13 - IO interrupt 6 force bit 12 - IO interrupt 5 force bit 10 - IO interrupt 4 force	bit 15 - reserved bit 13 - reserved bit 12 - reserved bit 10 - reserved
ITC	0x8000009C	bit 15 - IO interrupt 7 clear bit 13 - IO interrupt 6 clear bit 12 - IO interrupt 5 clear bit 10 - IO interrupt 4 clear	bit 15 - reserved bit 13 - reserved bit 12 - reserved bit 10 - reserved
IOIT1	0x800000A8	IO Port Interrupt Register 1	IOIT
IOIT2	0x800000AC	IO Port Interrupt Register 2 Configuration of IO interrupt for interrupt 4, 5, 6 and 7	not available
PCIID2[1]	0x80000108	class code: 0x0B4000 revision id: 0x02	class code: 0x00000B revision id: 0x01
PCIIS	0x80000154	bit 12 - **SYSEN*** state	bit 12 - reserved
PCIIC	0x80000158	bit 2 - reserved bit 1 - reserved	bit 2 - Double-word write bit 1 - Double-word read
PCITSC	0x80000160	bit 8 - delayed read	bit 8 - reserved
TBCTL	0x90000004	bit 26 - AHB trace buffer freeze	bit 26 - reserved

Note: 1. The values in this register are hardcoded and can be used to detect in software whether an AT697E or AT697F is running, even if the PCI interface is not used.

Pin Modifications

Table 122. MCGA-349 Pin Changes

Pin Name	Pin Number	AT697F Description	AT697E Description
LFT	M16	Not Connected - The PLL filter is internal	PLL filter

Table 123. MQFP-256 Pin Changes

Pin Name	Pin Number	AT697F Description	AT697E Description
LFT	178	Not Connected - The PLL filter is internal	PLL filter

Ordering Information

Table 124. Possible Order Entries

Part-Number	Supply Voltage (core / IOs)	Temperature Range	Maximum Speed (MHz)	Packaging	Quality Flow
AT697F-2H-E	1.8V / 3.3V	+25°C	100	MCGA349	Engineering Samples
5962-0722402QXB(1)	1.8V / 3.3V	-55°C ; +125°C	100	MCGA349	QMLQ
AT697F-KG-E	1.8V / 3.3V	+25°C	100	MQFP256	Engineering Samples
5962-0722402QYC(1)	1.8V / 3.3V	-55°C; +125°C	100	MQFP256	QMLQ
5962-0722402VYC(1)	1.8V / 3.3V	-55°C; +125°C	100	MQFP256	QMLV
5962R0722402VYC(1)	1.8V / 3.3V	-55°C; +125°C	100	MQFP256	QMLV-RHA

Note: 1. SMD number depending on DSCC agreement.

Datasheet Revision History

7703A - 05/2008

1. Document creation.

7703B - 12/2008

1. ADVANCE INFORMATION DATASHEET Document.

7703C - 06/2009

1. `MCFG1.abrdy` bit description change.
2. Suffix N change to *.
3. modify <xxx> bit in <yyy> in register by <yyy>-><xxx>.
4. Replace **`SYSCLK`** by **`SDCLK`**.
5. text and wording modifications.

7703D - 12/2009

1. Text and wording modifications.
2. Timing Charaterization final update.

Rev. E 08/2011

1. Overall rework of the datasheet.

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong
Tel: (852) 2245-6100
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
sparc-applab.hotline@nto.atmel.com

Litterature Requests
www.atmel.com/litterature

Sales Contact
www.atmel.com/contacts

Disclaimer: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. **EXCEPT AS SET FORTH IN ATMEL'S TERMS AND CONDITIONS OF SALE LOCATED ON ATMEL'S WEB SITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.** Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel's products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

© 2011 Atmel Corporation. All rights reserved. Atmel®, logo and combinations thereof, AVR® and others are registered trademarks or trademarks of Atmel Corporation or its subsidiaries. Other terms and product names may be trademarks of others.

7703E–AERO–08/11