Features

- SPARC V8 High Performance Low-power 32-bit Architecture
  - 8 Register Windows
- Advanced Architecture:
  - On-chip Amba Bus
  - 5 Stage Pipeline
  - 16 kbyte Multi-sets Data Cache
  - 32 kbyte Multi-sets Instruction Cache
- On-chip Peripherals:
  - Memory Interface
    - PROM Controller
    - SRAM Controller
    - SDRAM Controller
  - Timers
    - Two32-bit Timers
    - Watchdog 32-bitTimer
  - 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 40-bit boot-PROM Interface Possibilities
- Operating range
  - Voltages
    - 3.3V +/- 0.30V for I/O
    - 1.8V +/- 0.15V for Core
  - Temperature
    - -55°C to 125°C
- Clock: 0MHz up to 100MHz
- Power consumption: 1W at 100MHz
- Performance:
  - 86MIPS (Dhrystone 2.1)
  - 23MFLOPS (Whetstone)
- Radiation Performance
  - Tested up to a total dose of 300Krads (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.cm²/mg
- Package MCGA349 and MQFPF256
- Mass: 9g
- Development Kit Including
  - AT697F Evaluation Board
  - AT697F Sample

Rad-Hard 32 bit SPARC V8 Processor

AT697F

Advance Information

7703C–AERO–6/09

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 on-chip Integer Unit (IU), a Floating Point Unit (FPU), separate instruction and data caches, hardware multiplier and divider, interrupt controller, debug support unit with trace buffer, two 32-bit timers, Parallel and Serial interfaces, a Watchdog, a PCI Interface and a flexible Memory Controller. The design is highly testable with the support of a Debug Support Unit (DSU) and a boundary scan through JTAG interface.

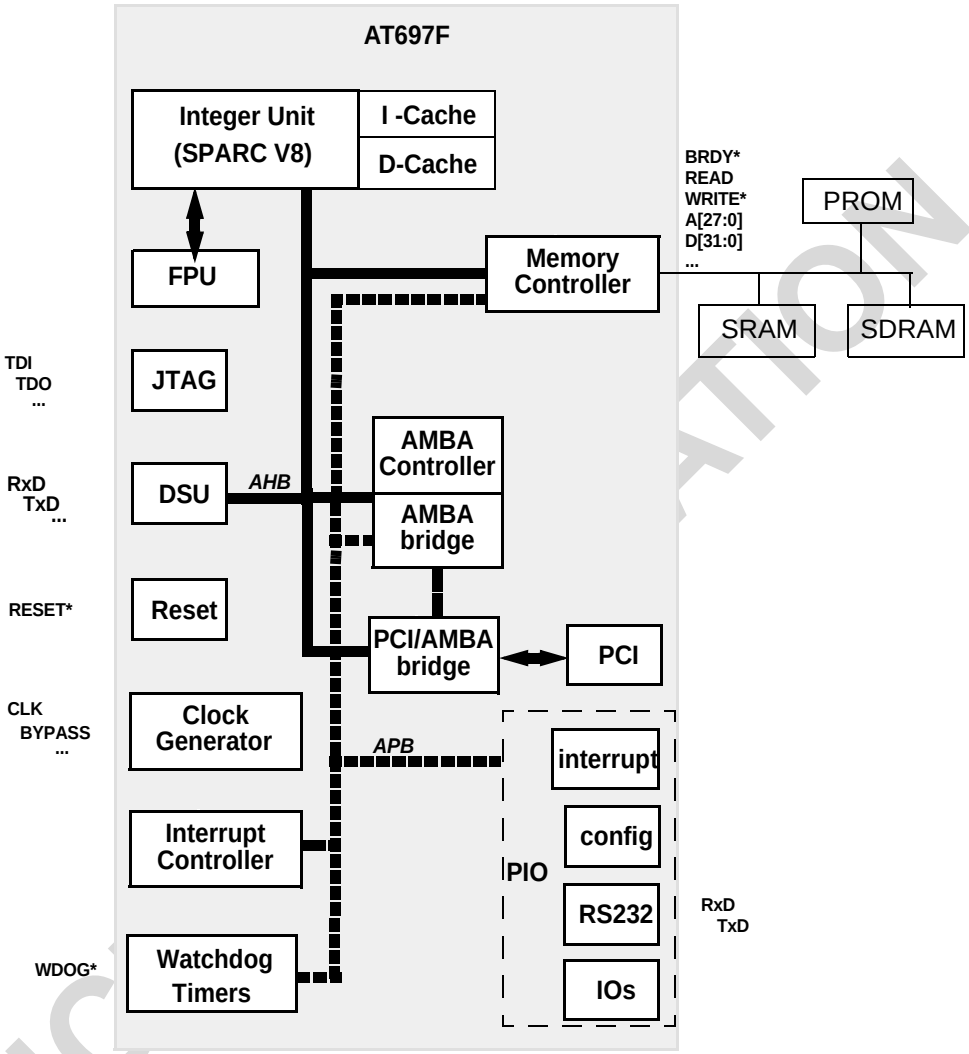
An Idle mode holds the processor pipeline and allows Timer/Counter, Serial ports and Interrupt system to continue functioning.

The processor is manufactured using the Atmel 0.18 µm 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”, page 146“ for detailed description of the differences between AT697F and AT697FE.

Figure 1. AT697F Block Diagram



Pin Configuration

MCGA349 package

Table 1. AT697F MCGA349 pinout - Advanced Information

	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]
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 2. AT697F MCGA349 pinout - Advanced Information

	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
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 3. AT697F MCGA349 pinout - Advanced Information

	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

QFP256 Package

Table 4. AT697F QFP256 pinout

pin number	pin name	pin number	pin name	pin number	pin name
1	VCC33	31	TCK	61	PIO[1]
2	PCI_REQ*	32	TMS	62	PIO[2]
3	PCI_GNT*	33	VSS	63	PIO[3]
4	PCI_CLK	34	TDI	64	PIO[4]
5	PCI_RST*	35	TDO	65	PIO[5]
6	SDCS*[0]	36	WRITE*	66	PIO[6]
7	VSS	37	READ	67	VCC33
8	VDD18	38	OE*	68	PIO[7]
9	SDCS*[1]	39	IOS*	69	PIO[8]
10	SDWE*	40	VCC33	70	PIO[9]
11	SDRAS*	41	ROMS*[0]	71	VSS
12	VSS	42	ROMS*[1]	72	VDD18
13	VSS	43	RWE*[0]	73	PIO[10]
14	SDCAS*	44	RWE*[1]	74	PIO[11]
15	VCC33	45	RWE*[2]	75	Reserved
16	SDDQM[0]	46	RWE*[3]	76	PIO[12]
17	SDDQM[1]	47	RAMOE*[0]	77	PIO[13]
18	SDDQM[2]	48	RAMOE*[1]	78	PIO[14]
19	SDDQM[3]	49	RAMOE*[2]	79	PIO[15]
20	SDCLK	50	RAMOE*[3]	80	VCC33
21	BRDY*	51	RAMOE*[4]	81	CB[0]
22	BEXC*	52	RAMS*[0]	82	CB[1]
23	VSS	53	VCC33	83	CB[2]
24	VSS	54	RAMS*[1]	84	CB[3]
25	DSUEN	55	RAMS*[2]	85	VCC33
26	DSUTX	56	RAMS*[3]	86	CB[4]
27	DSURX	57	VSS	87	CB[5]
28	DSUBRE	58	VDD18	88	CB[6]
29	DSUACT	59	RAMS*[4]	89	CB[7]
30	TRST	60	PIO[0]	90	D[0]

Table 5. AT697F QFP256 pinout

pin number	pin name	pin number	pin name	pin number	pin name
91	VCC33	**124**	D[25]	157	A[19]
92	D[1]	**125**	D[26]	158	A[20]
93	D[2]	**126**	D[27]	159	A[21]
94	D[3]	**127**	D[28]	160	A[22]
95	D[4]	**128**	D[29]	161	VSS
96	D[5]	**129**	D[30]	162	VCC33
97	D[6]	130	VCC33	163	A[23]
98	Reserved	131	D[31]	164	A[24]
99	VCC33	132	N.C.	165	A[25]
100	D[7]	133	A[0]	166	A[26]
101	D[8]	134	A[1]	167	A[27]
102	D[9]	135	VSS	168	WDOG*
103	D[10]	136	VDD18	169	ERROR*
104	D[11]	137	A[2]	170	VCC33
105	D[12]	138	A[3]	171	RESET*
106	VCC33	139	A[4]	172	Reserved
107	D[13]	140	VCC33	173	LOCK
108	D[14]	141	A[5]	174	SKEW[1]
109	D[15]	142	A[6]	175	SKEW[0]
110	D[16]	143	A[7]	176	BYPASS
111	D[17]	144	A[8]	177	VSS_PLL
112	VSS	145	A[9]	178	N.C.
113	D[18]	146	A[10]	179	VDD_PLL
114	VCC33	147	VCC33	180	CLK
115	D[19]	148	A[11]	181	VCC33
116	D[20]	149	A[12]	182	PCI_AREQ*[3]
117	D[21]	150	A[13]	183	PCI_AGNT*[3]
118	D[22]	151	A[14]	184	PCI_AREQ*[2]
119	D[23]	152	A[15]	185	VSS
120	D[24]	153	A[16]	186	VDD18
121	VSS	154	VCC33	187	PCI_AGNT*[2]
122	VDD18	155	A[17]	188	PCI_AREQ*[1]
123	VCC33	156	A[18]	189	VCC33

Table 6. AT697E MQFP256 pinout

pin number	pin name	pin number	pin name	pin number	pin name
190	PCI_AGNT*[1]	**213**	A/D[12]	236	A/D[19]
191	PCI_AREQ*[0]	**214**	A/D[13]	237	SYSEN*
192	PCI_AGNT*[0]	**215**	A/D[14]	238	A/D[20]
193	A/D[0]	**216**	A/D[15]	239	VCC33
194	VCC33	**217**	VCC33	240	A/D[21]
195	A/D[1]	**218**	C/BE*[1]	241	A/D[22]
196	A/D[2]	**219**	PAR	242	A/D[23]
197	A/D[3]	220	SERR*	243	IDSEL
198	A/D[4]	221	PERR*	244	C/BE*[3]
199	VSS	222	VCC33	245	VCC33
200	VDD18	223	PCI_LOCK*	246	A/D[24]
201	VCC33	224	STOP*	247	A/D[25]
202	A/D[5]	225	DEVSEL*	248	A/D[26]
203	A/D[6]	226	TRDY*	249	VSS
204	A/D[7]	227	VCC33	250	VDD18
205	C/BE*[0]	228	IRDY*	251	A/D[27]
206	VSS	229	FRAME*	252	VCC33
207	VCC33	230	VSS	253	A/D[28]
208	A/D[8]	231	C/BE*[2]	254	A/D[29]
209	A/D[9]	232	A/D[16]	255	A/D[30]
210	A/D[10]	233	VCC33	256	A/D[31]
211	A/D[11]	234	A/D[17]		
212	VCC33	235	A/D[18]		

Notes: 1. ‘Reserved’ pins shall not be driven to any voltage
2. N.C. refers to unconnected pins

Pin Description

ATMEL Convention

'*' attached to a signal (e.g OE*) designate an active-low signal.

When a bit of a register is writen in C-like style (e.g MCFG2→RAMWWS) it must be read as the RAMWWS bit in the register MCFG2.

IU and FPU Signals

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

A[27:0] bus carries the addresses during accesses to external memory. When access to cache memory is performed, the address of the last external memory access remains driven on the address bus.

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

D[31:0] bus carries the data during accesses to memory. The processor automatically configures the bus as output and drive the lines during write transactions.
During accesses to 8-bit areas, only D[31:24] are used.

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

CB[6:0] bus carries the EDAC checkbits during memory accesses. CB[7][(1)] takes the value of tcb[7] in the error control register. Processor only drives CB[7:0] during write transactions to areas programmed to be EDAC protected.

Note: 1. CB[7] is implemented to enable programming of flash memories. When only 7 bits are useful for EDAC protection, 8 are needed for programming.

Memory Interface Signals

General management

OE* - Output enable (output)

This active low output is asserted during read transactions on the memory bus.

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.

READ - Read transaction (output)

This active high output is asserted during read transactions on the memory bus.

WRITE* - Write enable (output)

This active low output provides a write strobe during write transactions on the memory bus.

PROM

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

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

SRAM

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

These active low signals provide an individual output enable for each RAM bank.

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

These active low outputs provide the chip-select signals for each RAM bank.

RWE* [3:0] - RAM write enable (output)

These active low outputs provide individual write strobes for each byte. RWEN[0] controls D[31:24], RWEN[1] controls D[23:16], etc.

I/O

IOS* - I/O select (output)

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

SDRAM Interface

SDCLK - SDRAM clock (output)

SDRAM clock provides the SDRAM interface clock reference.

SDCAS* - SDRAM column address strobe (output)

This active low signal provides a common CAS for all SDRAM devices.

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

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

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

These active low outputs provide the DQM signals for both SDRAM banks.

SDRAS*- SDRAM row address strobe (output)

This active low signal provides a common RAS for all SDRAM devices.

SDWE* - SDRAM write strobe (output)

This active low signal provides a common write strobe for all SDRAM devices.

System Signals

CLK - Processor clock (input)

The CLK input provides the main processor clock reference.

RESET* - Processor reset (input)

When asserted, this active low input will reset the processor and all on-chip peripherals.

WDOG* - Watchdog time-out (open-drain output)

This active low output is asserted when the watchdog expires.

BEXC* - Bus exception (input)

This active low input is sampled simultaneously with the data during accesses on the memory bus. If asserted, a memory error will be generated.

ERROR* - Processor error (open-drain output)

This active low output is asserted when the processor has entered error state and is halted. This happens when traps are disabled and a synchronous (un-maskable) trap occurs.

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

These bi-directional signals can be used as inputs or outputs to control external devices.

BYPASS - PLL bypass (input)

When driven to VCC, this active high input set the PLL in bypass mode. The device is then directly clocked by the external clock. When grounded, the device is clocked through the PLL.

SKEW[1:0] - Clock tree skew (input)

These input signals configurate the programmable skew on the triplicated clock trees.

LOCK - PLL lock (output)

This active high output is asserted when the PLL output (internal node) is locked at the frequency corresponding to four times the input command.

DSU Signals

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 active high input will generate break condition and put the processor in debug mode.

DSUEN - DSU enable (input)

The 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.

DSURX - DSU receiver (input)

This active high input provides the data to the DSU communication link receiver

DSUTX - DSU transmitter (output)

This active high input provides the output from the DSU communication link transmitter.

JTAG

TCK - Test Clock (input)

Used to clock serial data into boundary 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. Shall be grounded for end application.

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 (Output)

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

PCI interface

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::00] contain a physical address (32 bits). For I/O, this is a byte address; for configuration and memory, it is a DWORD address. During data phases, A/D[07::00] 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)

The number of "1"s on A/D[31::00], C/BE[3::0]*, and PAR equals an even number

FRAME* - Cycle Frame (bi-directional)

It 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)

IRDY* 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::00]. During a read, it indicates the master is prepared to accept data.

TRDY* - Target Ready (bi-directional)

TRDY* 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 AD[31::00]. During a write, it indicates the target is prepared to accept data.

STOP* - Stop (bi-directional)

STOP* indicates the current target is requesting the master to stop the current transaction.

PCI_LOCK* - Lock (bi-directional)

PCI_LOCK* 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 driving 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.

REQ* - PCI bus request (output)

REQ* 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 must be tri-stated while RST* is asserted.

GNT* - PCI Bus Grant (input)

GNT* 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 must be ignored while RST* is asserted.

PCI_CLK - PCI clock (input)

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

RST* - PCI Reset (input)

Reset is used to bring PCI-specific registers, sequencers, and signals to a consistent state.

PERR* - Parity Error (bi-directional)

Parity Error 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 (bi-directional)

System Error 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. If an agent does not want a non-maskable interrupt (NMI) to be generated, a different reporting mechanism is required.

SYSEN* - PCI Host (input)

This active low input specifies the configuration of the device. At boot-up time, if SYSEN* is sampled at a low level, the device is configured as the host of the PCI bus. If SYSEN* is sampled at a high level, the device is configured as a satellite.

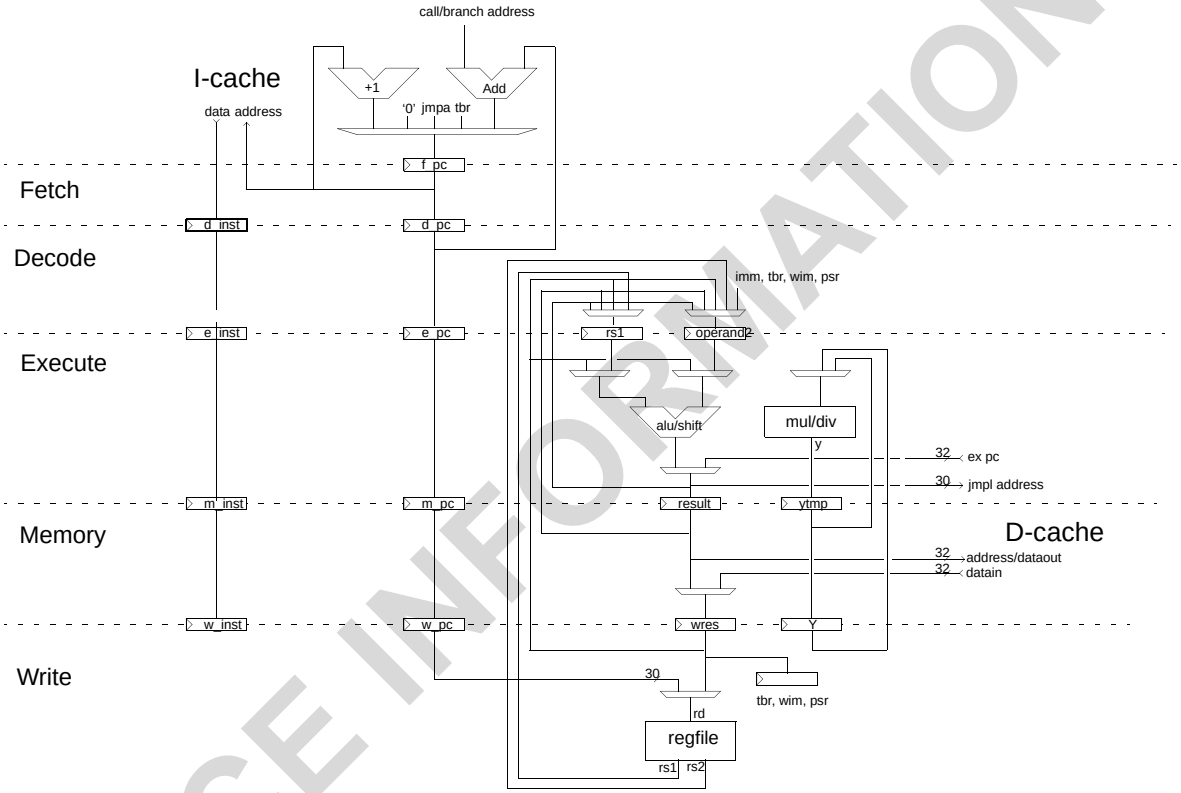
AT697F CPU Core

This section discusses the SPARC core architecture in general. The main function of the CPU core is to ensure correct program execution. The CPU must therefore be able to access memories, perform calculations, control peripherals, and handle interrupts.

SPARC Architecture Overview

The AT697F CPU core is based on the LEON2 architecture.

Figure 2. Block diagram of the AT697F Integer Unit architecture



The AT697F integer unit (IU) implements SPARC integer instructions as defined in SPARC Architecture Manual version 8. The IU is designed for highly dependable space and military applications by including fault tolerance features.

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.

- Instruction Fetch: If the instruction cache is enabled, the instruction is fetched from the instruction cache. Otherwise, the fetch is forwarded to the memory controller. The instruction is valid at the end of this stage and is latched inside the IU.
- Decode: The instruction is decoded and the operands are read. Operands may come from the register file or from internal data bypasses. CALL and Branch target addresses are generated in this stage.
- Execute: ALU, logical, and shift operations are performed. For memory operations and for JMPL/RETT, the address is generated.
- Memory: Data cache is accessed. For cache reads, the data will be valid by the end of this stage, at which point it is aligned as appropriate. Store data read out in the Execute stage is written to the data cache at this time.
- Write: The result of any ALU, logical, shift, or cache read operations re 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 driving 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.

In order to maximize performance and parallelism, the AT697F SPARC implementation uses powerful AMBA bus. Instructions in the program memory are executed with a five level pipelining. While one instruction is being executed, the next instruction is pre-fetched from the program memory. This concept enables instructions to be executed in every clock cycle.

Program Counters

Two 32-bit program counters (PC and nPC) are provided. The 32-bit PC contains the address of the instruction currently being executed by the IU. The nPC holds the address of the next instruction to be executed (assuming a trap does not occur).

When a trap occurs, the PC address is saved in the local register (l1) while the nPC address is saved in the local register (l2). When returning from trap, l1 value is copied back to PC and l2 value is copied back to nPC.

ALU - Arithmetic Logic Unit

The high-performance ALU operates in direct connection with all the 32 general purpose working registers. Within a single clock cycle, arithmetic operations between general purpose registers or between a register and an immediate memory address are executed. The implementation of the architecture also provide a powerful multiplier/divider supporting both signed and unsigned multiplication/division.

Support for high performance 64-bit operation is also provided.The 32-bit 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, or of a routine that uses the integer multiply step instruction. The Y register also holds the most significant word of the double-precision dividend for an integer divide instruction.

Register File - Windows

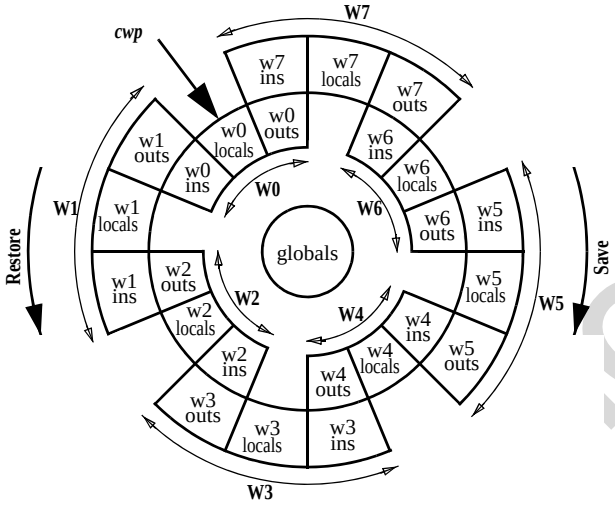
The fast access register file contains 8 SPARC register windows. Each window consists in a 32-register set. When a program is running, it has access to 32 32-bit processor registers which include 8 global registers plus 24 registers that belong to the current register window.

- The first 8 registers in the window are called the in registers' (*i0-i7*). When a function is called, these registers may contain arguments that can be used.
- The next 8 are the 'local registers' (*l0-l7*) which are scratch registers that can be used for anything while the function executes.
- The last 8 registers are the 'out registers' (*o0-o7*) which the function uses to pass arguments to functions that it calls.

AT697F register file implementation is based on two dual-port rams. The first dual-port ram corresponds to %rs1 operand of a SPARC instruction while the second corresponds to %rs2 operand. The two dual-port rams contents are always equal.

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 RETURN instruction acts in the opposite way

Figure 3. Overlapping Windows



The Window Invalid Mask register (WIM) is controlled by supervisor software and is used by 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.

To prevent erroneous operations from SEU errors in the main register file, each word is protected with a 7-bit EDAC checksum. The EDAC checksums are checked when the register is used as operand in an instruction. Any single-bit error is corrected and written back to the register file before the instruction is executed. If an un-correctable error is detected, a register hardware error trap (trap 0x20) is generated.

The protection can be enabled/disabled by programming the asr16→di bit from register file protection control register. By setting the asr16→te bit, errors can be inserted in the register file to test the protection function. When the asr16→te bit is set, the register checksum is combined with the asr16→tcb field before being written to the register file.

Due to the presence of the two dual-port rams for register file implementation, the following rules apply to the error injection test process.

- Test checkbits TCB[2:0] is Xored with checkbit[6:4] corresponding to the %rs1 operand.
- Test checkbits TCB[5:3] is Xored with checkbit[6:4] corresponding to the %rs2 operand.

Here is a simple example for the test of a single error in register file %rs1

```
! 0x32 =
!   register file test enable
!   tcb[2:0] = 0x4
!   tcb[5:3] = 0x1
    mov 0x32, %l1
    mov %l1, %asr16
! clear %l3
! =>  write 0x0 to %l3
!       forces 0x08 as checkbit for %l3 (error insertion in %rs1 dual-port ram)
    mov %g0, %l3
! disable EDAC test mode
    mov %g0, %asr16
! access to %l3 as %rs1 operand
! => single error detection and correction
    add %l3,%l2,%l1
```

A correction counter asr16→cnt is provided for error management. The asr16→cnt field is incremented each time a register correction is performed. It saturates at “111”.

State Register

The State Register (PSR) contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations. Note that the Status Register is updated after all ALU operations, as specified in the SPARC architecture specification. This will in many cases remove the need for using the dedicated compare instructions, resulting in faster and more compact code.

The state also provides some global information on the current window used, the authorized interrupts and peripheral (FPU and coprocessor) presence. A global interrupt management is provided through the processor state register. Trap and Interrupts can be individually enabled/disables from within this register.

Instruction Set

AT697F instructions fall into six functional categories: load/store, arithmetic/logical/ shift, control transfer, read/write control register, floating-point, and miscellaneous. Please refer to SPARC V8 Architecture manual that presents all the implemented instructions.

Floating Point Unit

The FPU is designed to provide execution of single and double-precision floating-point instructions. During the execution of floating-point instructions the processor pipeline is held.

The FPU is designed for highly dependable space and military applications, by including fault tolerance features like error detection and correction and triple modular redundancy.

The FPU depends upon the IU to access all addresses and control signals for memory access. Floating-point loads and stores are executed in conjunction with the IU, which provides addresses and control signals while the FPU supplies or stores the data. Instruction fetch for integer and floating-point instructions is provided by the IU.

The FPU contains 32 32-bit floating-point ***f*** registers, which are numbered from f[0] to f[31]. Unlike the windowed *r* registers, at a given time an instruction has access to any of the 32 *f* registers. The *f* registers can be read and written by FPop (FPop1/FPop2 format) instructions, and by load/store single/double floating-point instructions (LDF, LDDF, STF, STDF).

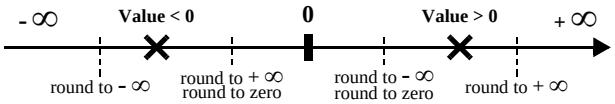
Rounding Direction

Rounding direction for floating point results is built according to the ANSI/IEEE Standard 754-1985.

In this way,

- 0 = round to nearest
- 1 = round to zero
- 2 = round to +infinity
- 3 = round to -infinity

Figure 4. Rounding Direction Schematic



Fault Tolerance

The processor has been especially designed for space application. 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

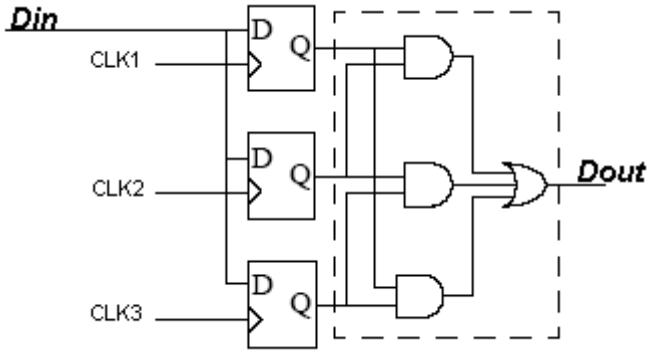
  The TMR architecture is based on a fully triplicated clock distribution (CLK1, CLK2 and CLK3). The PCI clock and the CPU clock are built as three-clock trees. The same triplication is applied to the PCI reset and to the CPU reset. See figure 5 for an overview of the TMR architecture.

Programmable skews on the clock trees are also provided to prevent the processor from arbitrary single-event transient errors.

Refer to the ‘clock’ section for detailed information on TMR implementation and skew implementation.

- EDAC protection on Regfile
- EDAC protection on external memory interface
- Parity protection on instruction and data caches

Figure 5. TMR structure - Clock triplication principle



Watch Points

The integer unit contains four hardware watch-points allowing generation of a trap on an arbitrary memory address range. Any binary aligned address range can be watched (the two less significant bits are ignored)

Each watch-point consists in a pair of application-specific registers

- break address register
  The break address defines a reference address for testing.
- mask register
  The mask indicates which bits of the break address register are to be effectively taken in account during address test

Configuration

A watchpoint is enabled setting logical one at least one of the three bits IF, DI or DS in the watchpoint address and mask registers. When all three bits are set logical zero, the watchpoint is disabled.

If the instruction fetch bit (IF) from the watchpoint address register is set logical one, any attempt to fetch an instruction from one of the address defined by ADDR and MASK results in a trap generation.

If the data store bit (DS) from the watchpoint address register is set logical one, any attempt to store data to one of the address defined by ADDR and MASK results in a trap generation.

If the data load bit (DL) from the watchpoint mask register is set logical one, any attempt to load a data from one of the address defined by ADDR and MASK results in a trap generation.

Operation

To detect if an address is part of the memory address range that traps, address bit 31 down to bit 2 are Xored with the BADx→BADDx.

This operation is based on the following segmentation of an address.

Table 7. Address Segmentation

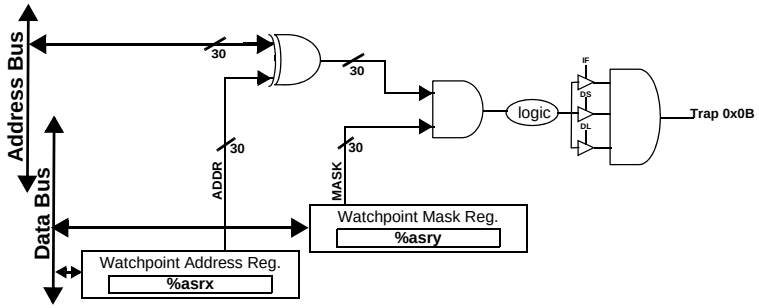
bit num.	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	Address																														ignored	

With such segmentation, it is possible to define trap segment from 4bytes up to 1Gbyte.

The result of the Xor is then Anded with the BMAx→BMAx.

If the result is zero, this indicates that address specified is in the watched range. Then, a watchpoint hit error is generated. Trap 0x0B is generated. If result is different from zero, address is out of the watched address range.

Figure 6. Watchpoint Hit Principle



Traps and Interrupts

Overview

The AT697F supports two types of traps:

- synchronous traps
- asynchronous traps also called interrupts.

Synchronous traps are caused by 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 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 8. Trap Overview

Trap	TT (trap type)	Priority	Description
reset	0x00	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	6	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	8	RESTORE into invalid window
register_hadrware_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
divide_exception	0x2A	15	Divide by zero
trap_instruction	0x80 -0xFF	16	Software trap instruction (TA)

Traps Description

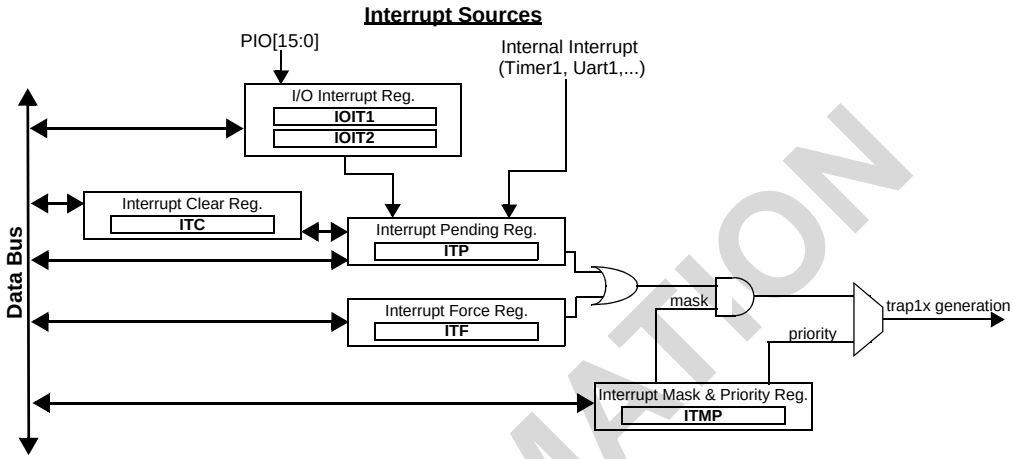
- reset - A reset trap is caused by an external reset request. It causes the processor to begin executing at virtual address 0. After a Reset Trap, no special memory states are defined exept the bits PSR→ET' and PSR→S that are initialized respectively '0' and '1'.
- write error - An error exception occurred 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 supervisor bit PSR→S is '0' (not in supervisor mode).
- fp_disabled - An attempt was made to execute an FPU instruction while FPU is not enabled or not present.
- cp_disabled - An attempt was made to execute a co-processor instruction while co-processor is not enabled or not present.
- watchpoint_detected - An instruction fetch memory address or load/store data memory address matched the contents of a pre-loaded implementation-dependent "watchpoint" register.
- window_overflow - A SAVE instruction attempted to cause the current window pointer (CWP) to point to an invalid window in the WIM.
- window_underflow - A RESTORE or RETT instruction attempted to cause the current window pointer (CWP) to point to an invalid window in the 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 (TEM) bit was 1, 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 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 or external devices to the integer unit.

Figure 7. Interrupt Controller Block Diagram



Operation

When an interrupt is generated, the corresponding bit is set in the interrupt pending register (ITP). The pending bits are ANDed with the interrupt mask register and then forwarded to the priority selector. The highest interrupt from priority level 1 will be forwarded to the IU - if no unmasked pending interrupt exists on priority level 1, then the highest unmasked interrupt from priority level 0 is forwarded.

When the IU acknowledges the interrupt, the corresponding pending bit will automatically be cleared.

Interrupt can also be forced by setting a bit in the interrupt force register. In this case, the IU acknowledgement will clear the force bit rather than the pending bit.

After reset, the interrupt mask register is set to all zeros while the remaining control registers are undefined.

Interrupt List

The following table presents the assignement of the interrupts.

Table 9. Interrupt Overview

Interrupt	TT (Trap Type)	Source
15	0x1F	I/O interrupt [7]
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 [2]
5	0x15	I/O interrupt [1]
4	0x14	I/O interrupt [0]
3	0x13	UART 1

Interrupt	TT (Trap Type)	Source
2	0x12	UART 2
1	0x11	Internal bus error

Non Maskable Interrupt (NMI)

The AT697F handles interrupt 15 (trap type TT = 0x1F). This interrupt can not be masked by the integer unit of the processor. It shall be used with care as the NMI of the processor.

I/O interrupts

As an alternate function of the general purpose interface, the AT697F allows to input interrupt from external devices. Up to eight 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.

Two registers are defined for configuration of the IO interrupts :

- IOIT1 register is used for control of IO interrupt 0, 1, 2 and 3
- IOIT2 register is used for control of IO interrupt 4, 5, 6 and 7

Each I/O interrupt is controlled through four fields in one of the above register (IOITx) : ENx, LEx, PLx and ISELx.

An I/O interrupt is enabled setting logical one to IOITx→ENx . Setting this bit logical zero disables the interrupt. The IOITx→ISELx defines which port of the general purpose interface should generate I/O interrupt x. The port can be selected from within PIO[15:0] and D[15:0]*.

Each I/O interrupt can have its trigger mode and its polarity individually configured. When bit IOITx→LEx is set logical one, the corresponding I/O interrupt is edge triggered. If the polarity bit IOITx→PLx is driven logical one the interrupt triggers when a rising edge is applied on the pin. If the polarity bit is driven logical zero the interrupt triggers when a falling edge is applied on the pin.

When the bit IOITx→LEx is set logical zero, the corresponding I/O interrupt is level sensitive. If the polarity bit IOITx→PLx is driven logical one the interrupt triggers when a high level is applied on the pin. If the polarity bit is driven logical zero the interrupt triggers when a low level is applied on the pin.

The following table summarizes the I/O interrupt configurations.

Table 10. I/O Interrupt Configuration

LEx	PLx	Trigger
0	0	low level
0	1	high level
1	0	falling edge
1	1	rising edge

Interrupt Priority

The 15 interrupts handled by the AT697F are prioritised, with interrupt 15 (TT = 0x1F) having the highest priority and interrupt 1 (TT = 0x11) the lowest.

It is possible to change the priority level of an interrupt using the two priority levels from the interrupt mask and priority register (ITMP). Each interrupt can be assigned to one of two levels as programmed in the Interrupt mask and priority register. Level 1 has higher priority than level 0. Within each level the interrupts are prioritised.

Memory Interface

Overview

The AT697F provides a 32-bit bus capable to interface PROM, memories mapped I/O devices, asynchronous static rams (SRAM) and synchronous dynamic rams (SDRAM). The memory bus can be configured either for 8-bit, 32-bit or 40-bit accesses. The memory controller manages up to 2 Gbytes of external memory. The following table presents the memory controller address map.

Table 11. Memory Controller address map

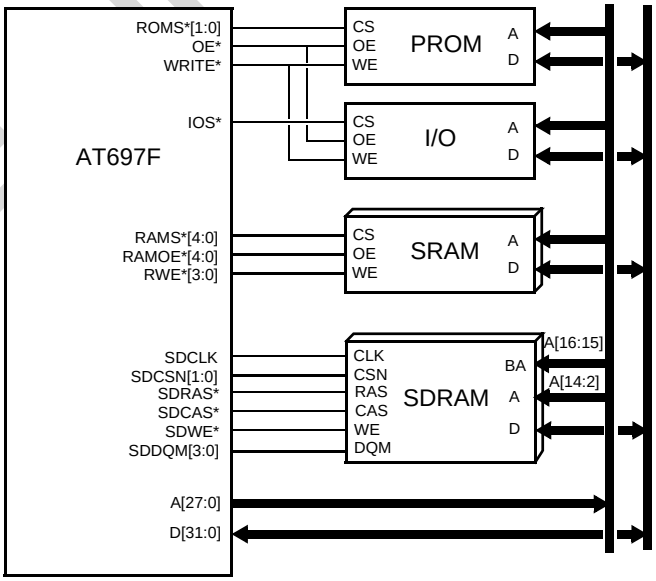
Address Range	Size	Mapping
0x00000000 - 0x1FFFFFFF	512M	PROM
0x20000000 - 0x2FFFFFFF	256M	I/O
0x40000000 - 0x7FFFFFFF	1G	SRAM/SDRAM

For applications that require smaller memory areas and/or smaller performances, it is possible to configure some memory spaces as 8-bit wide data bus.

All the configuration of the memory interface is done through the three memory controller registers : MCFG1, MCFG2 and MCFG3. MCFG1 is the register dedicated to PROM and IO configuration. SRAM and SDRAM are configured through MCFG2 and MCFG3.

Here is an overview of the 32-bit interconnection between the AT697F and external memories.

Figure 8. Memory Interface Overview



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 loads and double stores. The timing of a burst cycle is identical to the programmed basic cycle with the exception that during read transactions, the lead-out cycle will only occurs after the last transfer.

RAM Interface

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

SRAM interface

Overview

The SRAM interface can manage up to five SRAM banks. The control of the SRAM memory accesses uses a standard set of pin, including chip selects (RAMS*x), output enable (RAMOE*x) and write enable (RWE*x) lines.

The bank size of the four first banks of the SRAM area can be configured by setting the value of MCFG2→RAMBS. The bank size can be programmed in binary step from 8 Kbytes to 256 Mbytes. Whatever is the size of the four first banks, they are always contiguous. These memory banks are selected with RAMS*[3] down to RAMS*[0].

The fifth SRAM bank controlled by RAMS*[4] has a fix dimension. This bank always resides at the upper address 0x60000000. This bank is always 256 Mbytes large.

Figure 9. SRAM bank organisation

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]**

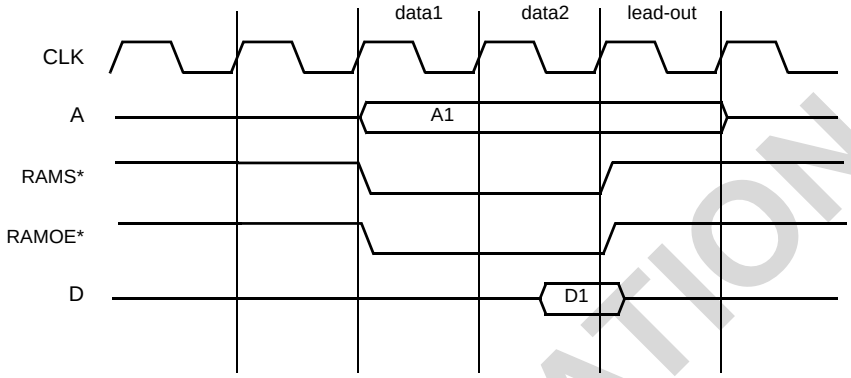
Notes:
1. If the SRAM bank size is set to 256Mbytes, SRAM bank 2 & bank 3 are in overlay with SRAM bank 4. In this case, bank 2 and bank 3 control signals are never asserted. Bank 4 has the priority.
2. When SDRAM is enabled, priority is given to the SDRAM. Any access to addresses higher than 0x60000000 is driven to SDRAM. No SRAM control is activated.

SRAM Read Access

A read access to SRAM consists in two data cycles and between zero and three waitstates. On non-consecutive accesses, a lead-out cycle is added after a read cycle to prevent bus contention due to slow turn-off time of memories or I/O devices. On consecutive accesses, no lead-out cycle is performed between the acesses but only one is performed at the end of the operations (RAMSN and RAMOE are not deasserted).

When a read access to SRAM is performed, a separate output enable signal is provided for each SRAM bank and it is only asserted when that bank is selected.

Figure 10. SRAM read transaction (0-waitstate)

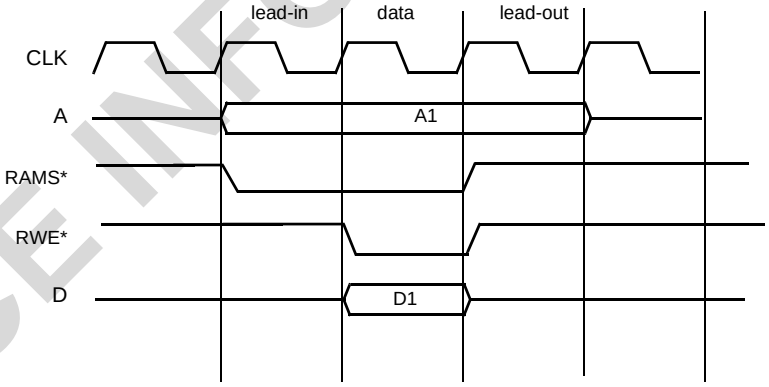


SRAM Write Access

Each byte lane has an individual write strobe (RAMWE*) to allow efficient byte and half-word writes.

Each write access to SRAM consists of three states and between zero and three waitstates. The three mandatory states are divided in one write setup cycle, one data cycle and one lead-out cycle.

Figure 11. SRAM write transaction (0-waitstate)



If the external memory use a common write strobe for the full 32-bit data, set the MCFG2→RMW. This will enable read-modify-write transactions for sub-word writes.

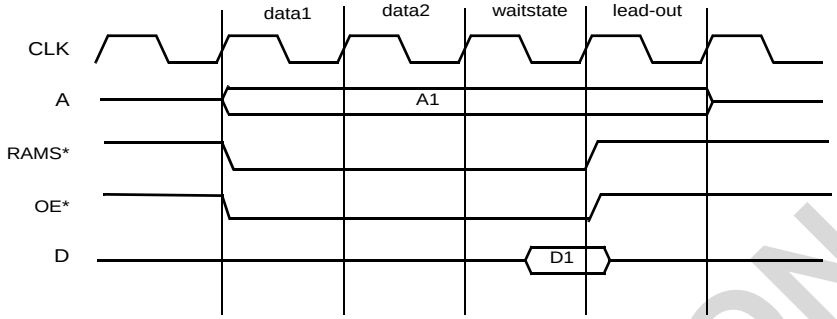
Waitstates

For application using slow SRAM memories, the SRAM controller provides the capability to insert wait-states during the SRAM accesses. Two types of wait-states can be inserted :

- Programmed delay, available for bank 0 up to bank 4
- ‘Hardware’ bus delay, available for bank 4 only

Up to three waitstates can be programmed for SRAM accesses. Read and write waitstates can be individually programmed. Setting the MCFG2→RAMRWS value defines the number of wait-states to insert during an SRAM read. Setting the MCFG2→RAMWWS value defines the number of waitstates to insert during an SRAM write.

Figure 12. RAM read access with one programmed waitstate



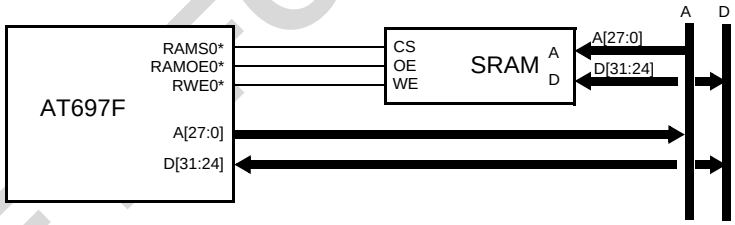
For and **only** for RAM bank 4, If the application needs more delay during the SRAM transfer, it is possible to introduce more delay by activating the hardware bus ready (BRDY*) detection in MCFG2. Refer to paragraph “BRDY Wait states”, page 38.

Bus width

To support applications with low memory performance requirements, the SRAM area can be configured for 8-bit operations. The configuration of SRAM in 8-bit mode is done programming MCFG2→RAMWDH, SRAM bus width field.

When the SRAM bus is configured as an 8-bit wide bus, data 31 downto 24 shall be used as interface.

Figure 13. SRAM 8-bit bus width connection



Since access to memory is always done on 32-bit word basis, read access to 8-bit memory will be transformed in a burst of four read transactions. If EDAC protection is active, 5 read cycles are necessary to complete the access (please refer to “Error Management - EDAC”, page 40 for more details). During write operation, only the necessary bytes are writen.

Write Protection

Two write protection schemes are provided to prevent accidental over-writing to the RAM area, the “*Start/End address Scheme*” and the “*Mask Scheme*”. These two schemes are explained in the following two sub-chapter

Start/End address Scheme

Two memory areas are defined by using a start-address and an end-address register. The first address of the protected memory area is calculated as 0x40000000 + START*4. The last address of the protected memory area is calculated as 0x40000000 + END*4.

Table 12. Start Address Register (WPSTAx)

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
0	0	START																												BP	0

Table 13. End Address Register (WPSTOx)

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
0	0	STOP																												US	SU

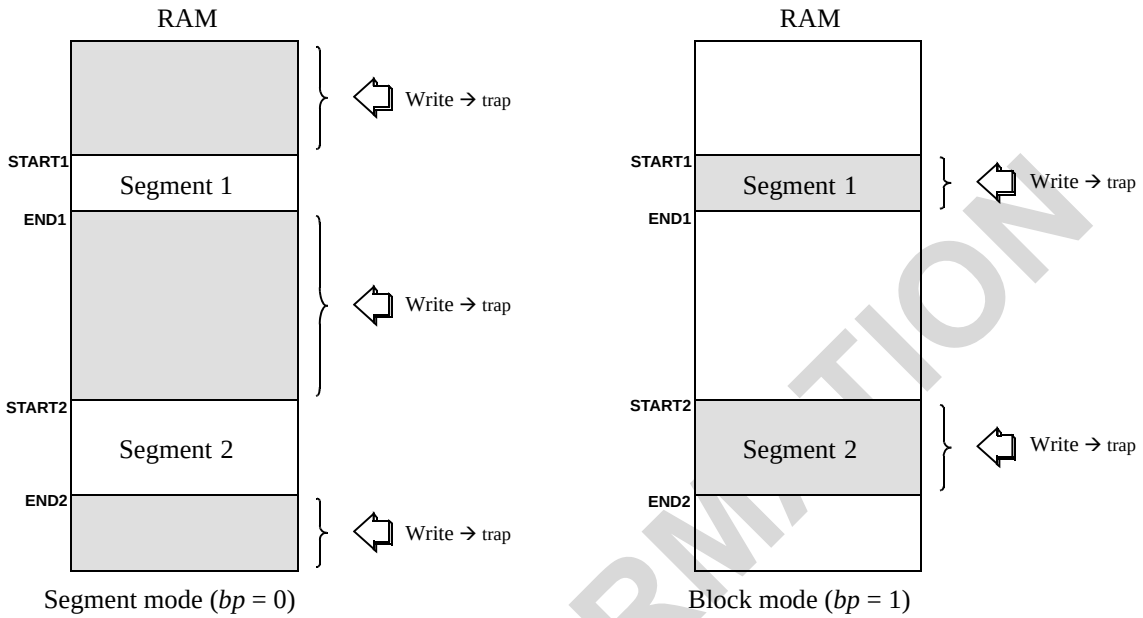
Setting WPSTAx→BPx to logical one, any access inside the two areas defined by the start/end registers will cause a memory exception (trap 0x2B). The first address of the protected against write operation is calculated as 0x40000000 + START*4. The first address outside the protected memory area is calculated as 0x40000000 + END*4 + 4.

Setting WPSTAx→BPx to logical zero the area between the start address and the end address defines the memory where write access is permitted, and a write access outside both areas will cause a memory exception (trap 0x2B). The first address where write operation is permitted is calculated as 0x40000000 + START*4. The first address outside the protected allowed area is calculated as 0x40000000 + END*4 + 4.

The start/end address protection scheme is enabled when at least one of the user mode protection and the supervisor mode protection is valid. The write protection can be configured to prevent the application from user and/or supervisor write access.

- Memory is protected against User write when WPSTOx→USx bit is set logical 1
- Memory is protected against Superviser write when WPSTOx→SUx bit is set logical 1

Figure 14. RAM Protection Mode Overview



Mask Scheme

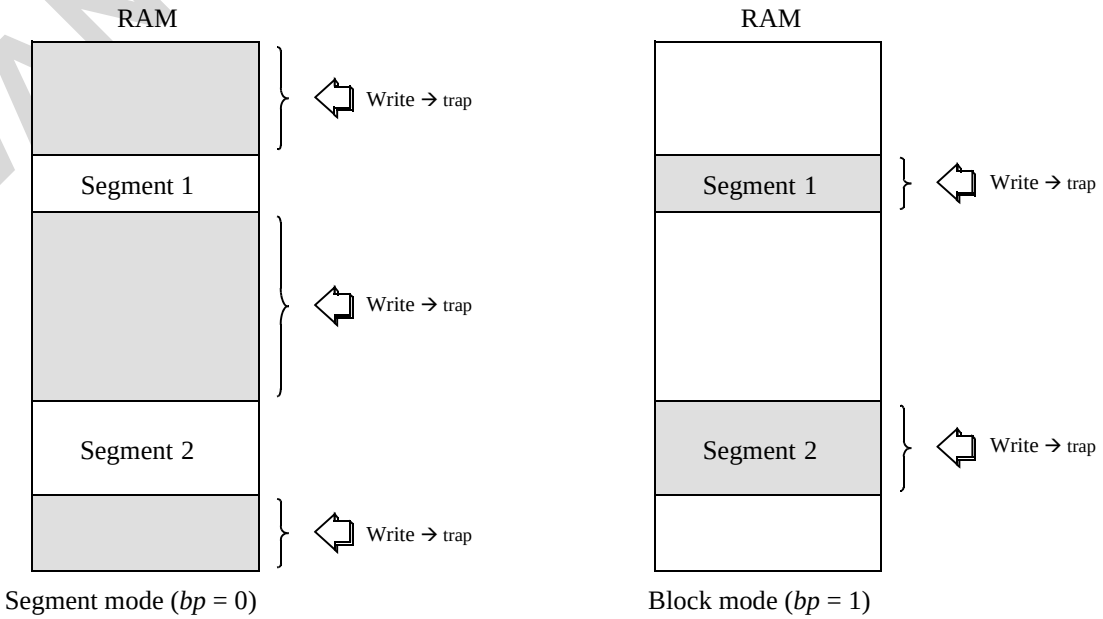
Two block protection units are available for RAM area. Each one is controlled through a write protection register (WPRn). Two major fields are defined : a TAG and a MASK.

- The TAG defines the 15 most significant bits of the address of the block to be write protected.
- The Mask specifies which bits of the TAG are really relevant for the protection.

The write protection on the RAM area is enabled setting logical one in WPRx→EN. If this bit is set logical zero, no protection is activated.

Two protection modes can be programmed. If the WPRx→BP is set logical one the protection is active within the segment. If this bit is set logical zero, the exterior of the segment is protected.

Figure 15. RAM Protection Mode Overview



To detect if the written address is part of a protected segment (or block), address bit 29 down to bit 2 are Xored with the WPRx→TAG. This operation is based on the following segmentation of an address.

Table 14. Address Segmentation

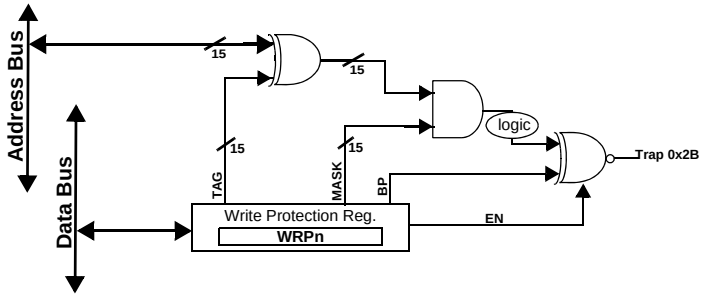
bit num.	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
field	area		most significant byte															32Kbyte protected block														

With such segmentation, memory block in the range of 32Kbyte up to 1Gbyte can be protected.

The result of the Xor is then Anded WPRx→MASK.

If the result is zero, this indicates that address specified is in the protected range. If result is different from zero, address is out of the protected address range. If a write protection error is detected, the write transaction is stopped. Then, a memory access error is generated. Trap 0x2B is generated

Figure 16. RAM Write Protection Overview



Protection Priorities

As a result of the write protection implementation for the RAM area,the two RAM write protection schemes can be used simultaneously. Combining the write protection schemes leads to the following behaviors :

- If all the enable protection units are configured in block protect mode (BP = 0), then a write protect error is generated when any of the units signal a write protection hit. In this mode, if at least one protection error is triggered, the write protection trap is raised.
- If at least one of the protection units operates in segment mode (BP=1), then a write protect error is generated **only** if all units configured in segment mode signal a protection error.

SDRAM

The synchronous dynamic RAM interface can manage up to two SDRAM banks. The control of the SDRAM memory accesses uses a standard set of pin, including chip selects (SDCS*x), write enable (SDWE*), data masks (SDDQM*x) and clock lines.

The bank size of the two SDRAM banks can be configured by setting the value of the MCFG2→SDRBS. The bank size can be programmed in binary step from 4 Mbytes to 512 Mbytes.

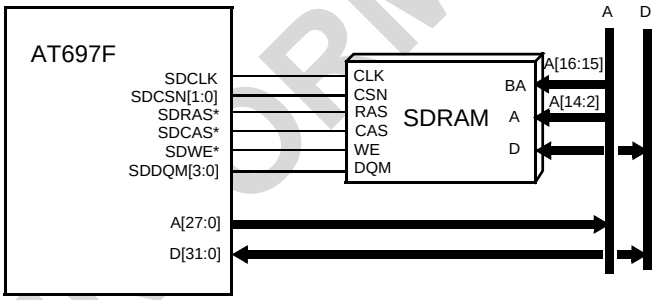
The controller supports 64M, 256M and 512M devices with 8 to 12 column-address bits, up to 13 row-address bits, and 4 banks. Only 32-bit data bus width is supported for SDRAM banks.

Address Mapping

The start address for the SDRAM banks depends upon the SRAM use in the application. If the the SRAM disable bit MCFG2→SI and the SDRAM enable bit MCFG2→SE are set logical one, the SDRAM start address is 0x40000000. If the the SRAM disable bit MCFG2→SI is set logical zero and the SDRAM enable bit MCFG2→SE is set logical one, the SDRAM start address is 0x60000000. If MCFG2→SE if set logical zero, no SDRAM can be used.

The address bus of the SDRAMs shall be connected to A[14:2], 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].

Figure 17. SDRAM connection overview



SDRAM Timing Parameters

To provide optimum access cycles for different SDRAM devices some SDRAM parameters can be programmed through MCFG2 register. The programmable SDRAM parameters are the following :

Table 15. SDRAM Programmable Timing Parameters

Function	Parameter	Range	Unit
CAS latency		2 - 3	clocks
Precharge to activate	t_{RP}	2 - 3	clocks
Auto-refresh command period	t_{RFC}	3 - 11	clocks
Auto-refresh interval		10 - 32768	clocks

SDRAM Commands

The SDRAM controller can issue three SDRAM commands. Commands to be executed are programmed through the MCFG2→SDRCMD. When this field is writen with a non zero value, a SDRAM command is issued :

- if set to ‘01’, Precharge command is sent,
- if set to ‘10’, Auto-Refresh command is sent,
- if set to ‘11’, Load Mode Reg (LMR) is sent.

When the LMR command is issued, the MCFG2→SDRCAS delay programmed is used. MCFG2→SDRCMD is cleared after a command is executed. When changing the value of the CAS delay, a LOAD-MODE-REGISTER command should be generated at the same time.

The SDRAM controller also provides a refresh command. It can be enabled by setting a logical one into MCFG2→SDRREF.

The Auto-Refresh command enables a periodical refresh for both SDRAM banks. The period between two Auto-Refresh command is programmed in MCFG3→SRCRV.

Depending on SDRAM type, required period is typically 7.8 or 15.6µs. This corresponds to 780 or 1560 clock cycle at 100MHz.

Refresh period is calculated as $\text{Refresh Period} = \frac{\text{Reload value} + 1}{\text{sdclk frequency}}$

SDRAM Initialisation

After reset, the SDRAM controller automatically performs the SDRAM initialisation sequence. It consists in PRECHARGE, two AUTO-REFRESH cycles and LOAD-MODE-REG on both banks simultaneously.
The controller programs the SDRAM to use page burst on read and single location access on write. A CAS latency of 3 is programmed by default. This value can be updated later by software.

SDRAM Read Access

A read transaction consists in three main operation. First, an ACTIVATE command to the desired bank and row is performed. Then, after the programmed CAS delay, a READ command is sent. The read transaction is terminated with a PRE-CHARGE command. No bank is left open between two accesses.

A burst read is performed if a burst access is requested on the internal bus.

SDRAM Write Access

A write transactions consists in three main operations. First, an ACTIVATE command to the desired bank and row is performed. Then, a WRITE command is sent. The write transaction is terminated with the PRE-CHARGE command.

A burst write on internal bus generates a burst of write commands without idle cycles in-between.

Access Error

An access error can be indicated to the processor asserting the BEXC* signal. If enabled by setting logical one to MCFG1→BEXC , the BEXC* signal is sampled with the data.

If the BEXC* signal is driven low by the external device during the access, an error response is generated on the internal bus.

- Trap 0x01 is taken if an instruction fetch is in progress
- Trap 0x09 is taken if a data space access is in progress
- Trap 0x2B is taken if a data store is in progress

PROM Interface

Overview

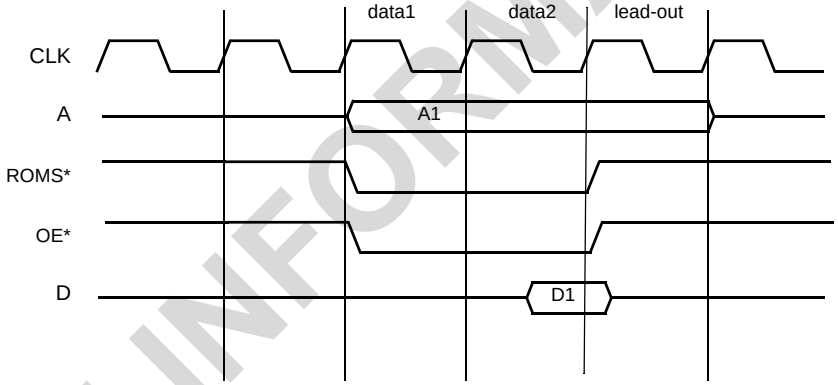
The memory controller give the capability to control up to 512Mbyte of PROM. The PROM interface can manage up to two PROM banks. The control of the PROM memory accesses uses a standard set of pin, including chip selects (ROMS*x), output enable (OE*), read (READ) and write (WRITE*) lines.

The bank size of the PROM banks is not programmable. The lower half part of the PROM area (0x00000000 up to 0x0FFFFFFF) is controlled by the ROMS0* PROM select signal. The upper half part of the PROM area (0x10000000 up to 0x1FFFFFFF) is controlled by the ROMS1* PROM select signal.

PROM Read Access

A read access to PROM consists in two data cycles and waitstates if any programmed. On non-consecutive accesses, a lead-out cycle is added after a read transaction to prevent bus contention due to slow turn-off time of memories or I/O devices. On consecutive accesses, no lead-out cycle is performed between the acesses but only one is performed at the end of the operations.

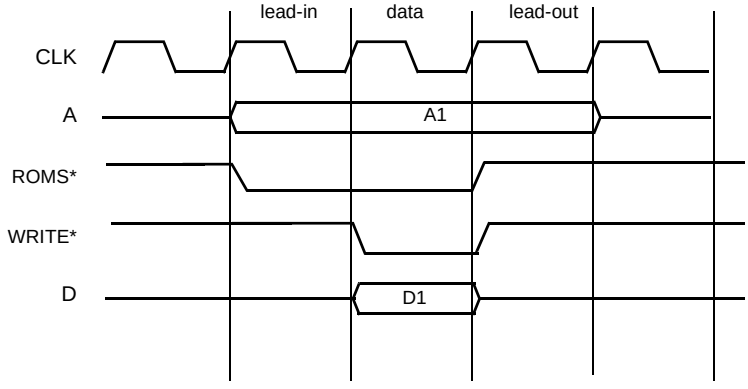
Figure 18. PROM Read transaction (0 Waitstate)



PROM Write Access

Each write access to PROM consists of three states and of waitstates if any programmed. The three mandatory states are divided in one write setup cycle, one data cycle and one lead-out cycle. The write operation is strobed by the WRITE* signal.

Figure 19. PROM Write transaction (0 waitstate)



Waitstates

For application using slow ROM memories, the ROM controller provides the capability to insert wait-states during the accesses. Two types of wait-states can be inserted :

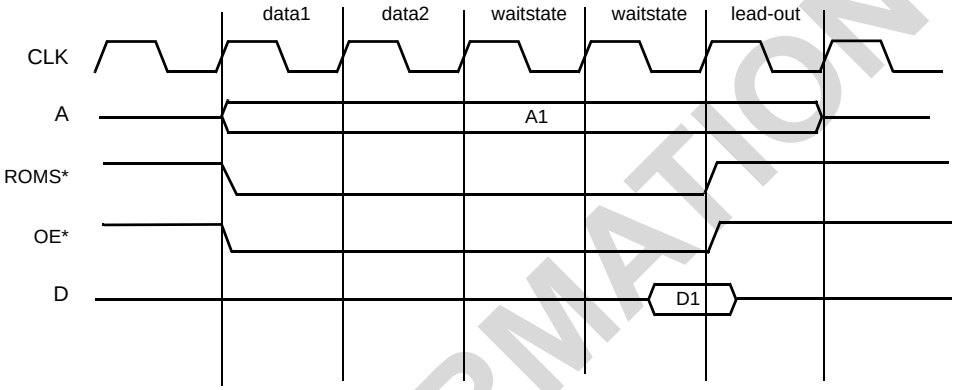
- Programmed delay
- ‘Hardware’ bus ready delay

Up to 30 waitstates can be programmed for PROM accesses. Read and write waitstates can be individually programmed. Setting MCFG1→PRRWS defines the number of waitstates to insert

during a PROM read access. Setting MCFG1→PRWWS defines the number of waitstates to insert during a PROM write.

MCFG1→PRRWS and MCFG1→PRWWS can be programmed to take values from 0 up to 15. The effective number of waitstates applied during an access is then twice the programmed value. In that way, programming two waitstates results in the insertion of four wait cycles during the access.

Figure 20. ROM read access with PRRWS=1 (two programmed waitstates)



If the application needs more time for ROM transfer, it is possible to introduce more delay by activating the hardware bus ready MCFG1→PBRDY. Refer to paragraph “BRDY Wait states”, page 38.

After a reset operation of the processor (or at power up), the MCFG1→PRRWS and MCFG1→PRWWS waitstates for the PROM area are set default to 15, resulting in 30 effective waitstates and the MCFG1→PBRDY is set to 0.

Write Protection

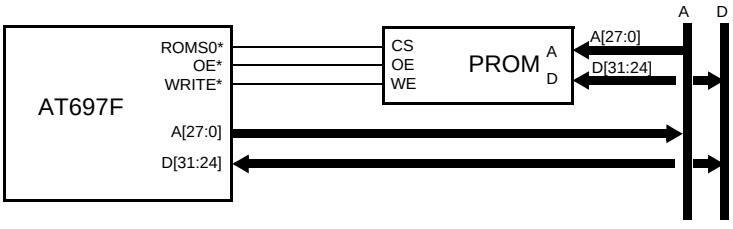
Write protection is provided to prevent accidental over-writing to PROM area. It is controlled through the PROM write enable bit MCFG1→PRWE. When set 1, this bit enables write to PROM. When set 0, no PROM write transaction is available.

Bus width

To support applications with low memory and performance requirements, the PROM area can be configured for 8-bit operations. The configuration of PROM in 8-bit mode is done programming MCFG1→PRWDH.

When the PROM bus is configured as an 8-bit wide bus, data 31 downto 24 shall be used as interface.

Figure 21. PROM 8-bit bus width connection



Since access to memory is always done on 32-bit word basis, read access to 8-bit memory will be transformed in a burst of four read transactions. If EDAC protection is active, 5 read cycles are necessary to complete the access (please refer to protection section for more details). During write operation, only the necessary bytes are writen.

Access Error

An access error can be indicated to the processor asserting the BEXC* signal. If enabled by setting logical one to MCFG1→BEXC , the BEXC* signal is sampled with the data.

- Trap 0x01 is taken if an instruction fetch is in progress
- Trap 0x09 is taken if a data space is in progress

- Trap 0x2B is taken if a data store is in progress

Memory Mapped I/O

Overview

The memory controller give the capability to control up to 256Mbyte of I/O. The I/O area consists in a single large bank. The control of the I/O area accesses uses a standard set of pin, including chip selects (IOS*x), output enable (OE*), read (READ) and write (WRITE*) lines.

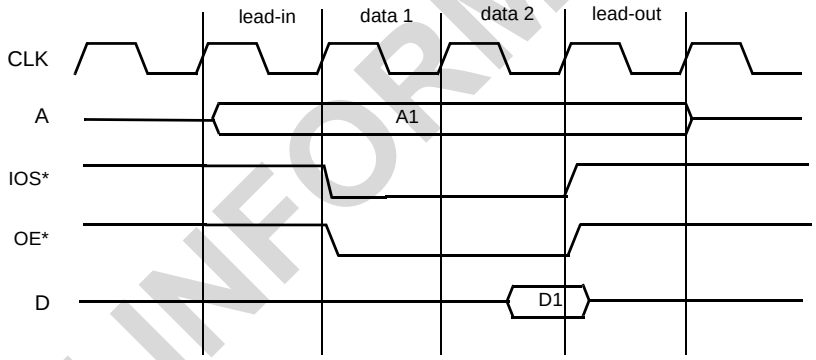
The size of the I/O bank is not programmable. The entire I/O area (0x20000000 up to 0x2FFFFFFF) is controlled by the IOS* select signal.

I/O Read Access

A read access to I/O consists in a lead-in cycle, two data cycles, waitstates if any programmed and a lead-out cycle. On non-consecutive accesses, the lead-out cycle is used to prevent bus contention due to slow turn-off time of memories or I/O devices.

The I/O select signal (IOSEL*) is delayed one clock to provide stable address.

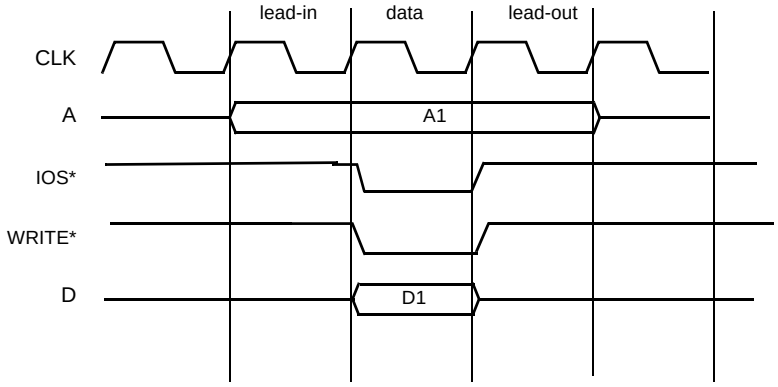
Figure 22. single I/O read transaction with lead-out



I/O Write Access

Each write access to I/O consists of three states and of waitstates if any programmed. The three mandatory states are divided in one write setup cycle, one data cycle and one lead-out cycle. The write operation is strobed by the WRITE* signal.

Figure 23. I/O write transaction



Waitstates

For application using slow I/O devices, the I/O controller provides the capability to insert wait-states during the accesses. Two types of wait-states can be inserted :

- Programmed delay,
- 'Hardware' delay.

Up to 15 waitstates can be programmed for I/O accesses. Read and write waitstates are programmed simultaneously. Setting MCFG1→IOWS defines the number of waitstates to insert during any access to/from I/O areas. MCFG1→IOWS can be programmed to take values from 0 up to 15.

If the application needs more time for IO transfer, it is possible to introduce more delay by activating the hardware bus ready detection bit MCFG1→IOBRDY. Refer to paragraph “BRDY Wait states”, page 38.

Write Protection

Read and write protections are provided to prevent accidental accesses to I/O area. Protection is controlled through the I/O protection bit MCFG1→IOP.

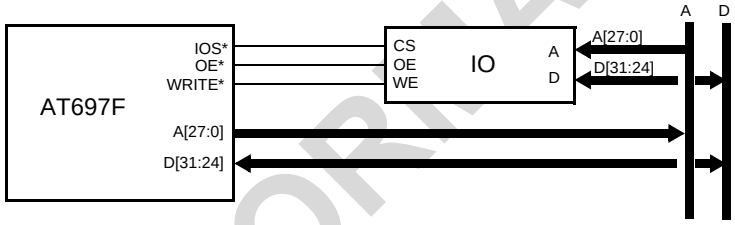
Bus width

To support applications with low memory and performance requirements, I/O area can be configured for 8-bit operations. The configuration of I/O in 8-bit mode is done programming the I/O bus width in MCFG1→IOWDH.

In such configuration, I/O device is not accessed by multiple 8-bit accesses as other memory areas. Only one single access is performed

When the I/O bus is configured as an 8-bit wide bus, data 31 downto 24 shall be used as interface.

Figure 24. I/O 8-bit bus width connection



Access Error

An access error can be indicated to the processor asserting the BEXC* signal. If enabled by setting logical one the MCFG1→BEXC, the BEXC* signal is sampled with the data.

- Trap 0x01 is taken if an instruction fetch is in progress
- Trap 0x09 is taken if a data space is in progress
- Trap 0x2B is taken if a data store is in progress

BRDY Wait states

For PROM accesses, for IO accesses and for RAM bank 4, but not for the other RAM banks, it is possible to introduce additional wait states determined by the peripherals with the BRDY* mechanism. This capability can be enabled separatly by the respective configuration bits MCFG1→PBRDY, MCFG1→IOBRDY and MCFG2→RAMBRDY. If the configuration bit is set to one, the processor waits before ending the transfer, as long as the BRDY* pin is driven high. If the configuration bit is set to zero (reset state), the BRDY* pin is ignored.Termination of the BRDY* induced wait states can be in two different modes:

- If MCFG1→ABRDY is set to zero (reset state), BRDY* needs to be asserted zero synchronously with respect to SDCLK, respecting the setup and hold times t19 and t20 (Refer to section “AC Characteristics”, page 130).The processor will terminate the access at the rising clock edge immediately following the rising edge during which BRDY* was low by de-asserting the OE* and the select signal (RAMS*[4], IOS* or ROMS*), as shown in the figures.
- If MCFG1→ABRDY is set to one, BRDY* is double synchronised in the processor, and it can be asserted asynchronously, without respecting t19 and t20, provided it is asserted low for at least 1.5 clock cycle. Asynchronous BRDY* timing implies an uncertainty, the access terminates at the second or third edge after its assertion, and read data needs to be kept stable until OE* and the select signal (RAMS*[4], IOS* or ROMS*) are de-asserted.

It should be noted that the BRDY* mechanism can be used in addition to the nominal duration of an access (one or two data cycles depending on the access type) and to the fixed wait states programmed in the “WS” fields (MCFG2→RAMWWS, MCFG1→PRWWS, MCFG1→IOWS). Even when BRDY* goes low earlier, the trasaction does not terminate until expiration of the programmed wait states.

Figure 25. Read access with one BRDY* controlled waitstate, MCFG1→AB=0

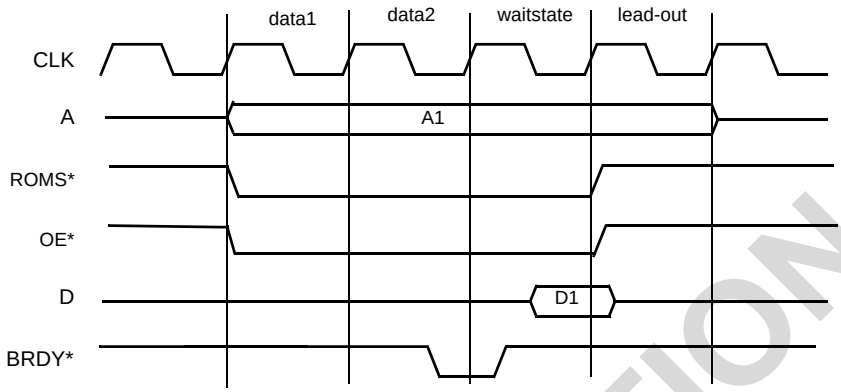
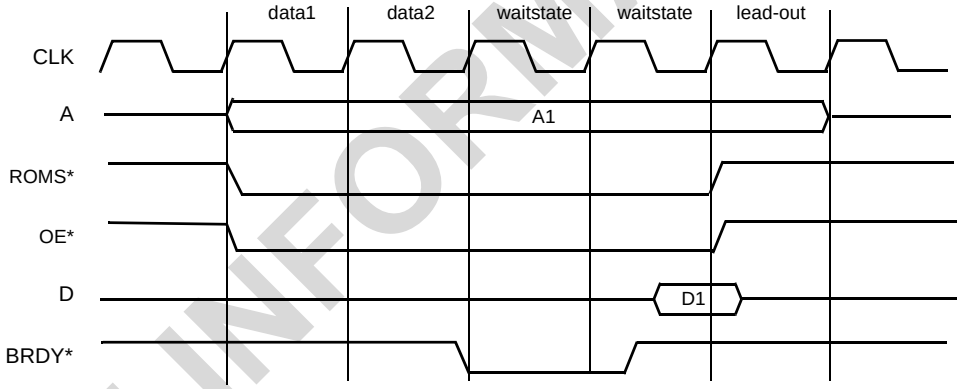


Figure 26. Read access with one BRDY* controlled waitstate, MCFG1→AB=1



Error Management - EDAC

Overview

The AT697F processor implements an on-chip error detector and corrector (EDAC). The on-chip memory EDAC can correct one error in a 32-bit word and detect two errors in a 32-bit word. The processor EDAC implemention enables data correction on-the-fly so that no timing penalty occurs during correction.

EDAC capability mapping

Data error management with the EDAC can be used on both PROM and RAM memory areas. The following table presents the EDAC protection capabilities provided by the processor.

Table 16. EDAC capability on Memories

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

PROM protection

Setting logical one the PROM EDAC enable bit MCFG3→PE, the data protection is enabled. For each read and write transaction to the PROM area the EDAC act as an error detector and an error corrector. When set logical zero, the EDAC is transparent for the PROM access.

At power-on or at reset, the value of the MCFG3→PE is directly copied from the PIO2 pin. In that way, it is possible to start the application with the EDAC enabled by driving high PIO2 during the power-on sequence (or reset sequence).

RAM protection

Setting logical one the RAM EDAC enable bit MCFG3→RE, the data protection is enabled. For each read and write transaction to the RAM area the EDAC act as an error detector and an error corrector. When set logical zero, the EDAC is transparent for the RAM access.

Operation

The processor uses an EDAC based on a seven bit Hamming code that detects any double error on a 32-bit bus and corrects any single error on a 32-bit bus. Note when the EDAC is enabled the read-modify-write bit MCFG2→RMW must be set.

Hamming code

For each 32-bit data, a seven bit a 7-bit checksum is generated. The equations below show how the Hamming checkbits (CBx) are generated:

CB0 = D0 ^ D4 ^ D6 ^ D7 ^ D8 ^ D9 ^ D11 ^ D14 ^ D17 ^ D18 ^ D19 ^ D21 ^ D26 ^ D28 ^ D29 ^ D31

CB1 = D0 ^ D1 ^ D2 ^ D4 ^ D6 ^ D8 ^ D10 ^ D12 ^ D16 ^ D17 ^ D18 ^ D20 ^ D22 ^ D24 ^ D26 ^ D28

$\overline{CB2}$ = D0 ^ D3 ^ D4 ^ D7 ^ D9 ^ D10 ^ D13 ^ D15 ^ D16 ^ D19 ^ D20 ^ D23 ^ D25 ^ D26 ^ D29 ^ D31

$\overline{CB3}$ = D0 ^ D1 ^ D5 ^ D6 ^ D7 ^ D11 ^ D12 ^ D13 ^ D16 ^ D17 ^ D21 ^ D22 ^ D23 ^ D27 ^ D28 ^ D29

CB4 = D2 ^ D3 ^ D4 ^ D5 ^ D6 ^ D7 ^ D14 ^ D15 ^ D18 ^ D19 ^ D20 ^ D21 ^ D22 ^ D23 ^ D30 ^ D31

CB5 = D8 ^ D9 ^ D10 ^ D11 ^ D12 ^ D13 ^ D14 ^ D15 ^ D24 ^ D25 ^ D26 ^ D27 ^ D28 ^ D29 ^ D30 ^ D31

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

Write operation

When the processor performs a write operation to a memory protected by the EDAC, it also outputs the seven bit checksum on the CB[6:0] pins.

Read operation

During a read operation from a protected memory, the seven bit checksum is sampled from the CB[6:0] inputs. Then, the EDAC verify the checksum to check the presence of an error.

According to the checksum equations, the EDAC calculates its own checksum. Then a syndrome generator uses the calculated and the read checksum to qualify if there is no error, one error or two errors in the read word.

Correctable error

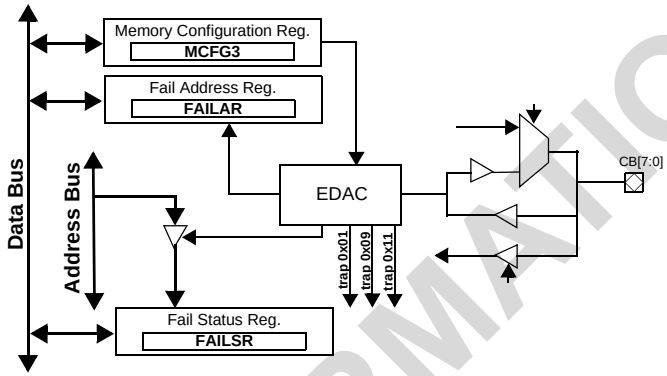
If a single error is detected, this leads to a correctable error. The correction is done on-the-fly during the current access and no timing penalty is induced but the corrected data is not automatically written back to the memory.

The correctable error detection event is reported in the fail address register (FAILAR) and in the fail status register (FAILSR). If unmasked, interrupt 1 (trap 0x11) is generated. The interrupt can then be attached to a low priority interrupt handler that scrubs the failing memory location.

Uncorrectable error

If a double error is detected, this leads to an un-correctable error. An un-correctable error detection during a data access leads to a data access exception (trap 0x09). In case the double error is detected during instruction fetch, it leads to an instruction access error (trap 0x01).

Figure 27. EDAC overview



EDAC on 8-bit areas

The 8-bit mode applies to RAM and PROM while SDRAM always uses 32-bit accesses.

When a memory area is configured in 8-bit mode, the EDAC checkbit bus (CB[7:0]) is not used but it is still possible to use EDAC protection.

The data bus mapped on D31:24 is always accessed in a 32-bit wide word basis (4bytes at a time). The corresponding checkbits are located on top of the selected memory

bank according to the following operation:

- The address A[27:2] of the 32-bit data word is inverted
- The resulting address is then shifted twice right to become a byte address
- The checkbit is written to the derived byte address while the data address chipselect is kept active so that the current memory area is still active.

A word written as four bytes to addresses 0, 1, 2, 3 will have its checkbits at address 0x0FFFFFFF, addresses 4, 5, 6, 7 at 0x0FFFFFFE and so on.

Here is an example of checkbit addressing:

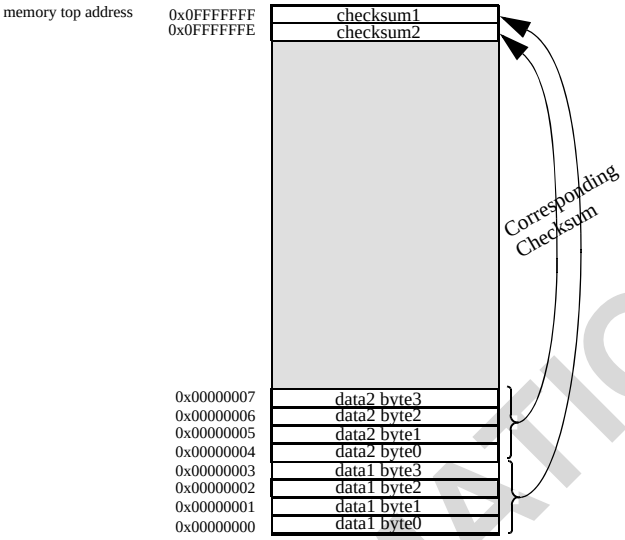
- The data is written at address 0x00000004
- Inversion of this address lids to 0xFFFFFFFB
- Once shifted we have 0xFFFFFFFE
- The checkbit is located at address 0xFFFFFFFE in the same memory bank as the data.

All the bits up to the maximum bank size will be inverted while the same chip-select is always asserted.

This way all the bank size can be supported and no memory will be unused (except for a maximum of 4 Bytes in the gap between the data and checkbit area).

Here is an overview of the memory organization when EDAC is enbled on a 8-bit area.

Figure 28. Memory Organization when EDAC enabled

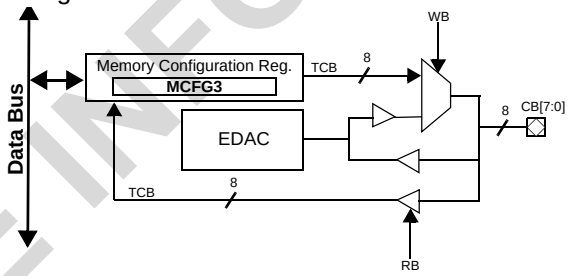


Note In addition, only byte-writes shall be performed to ROM area when the EDAC is enabled. In this case, only the corresponding byte are written.

EDAC testing

The operation of the EDAC can be tested trough the MCFG3 memory configuration register.

Figure 29. EDAC testing overview



Write test

If the write bypass MCFG3→WB is set logical one, the value of the test checksum from the MCFG3→TCB field replaces the normal checkbits during memory write transactions.

Read test

During memory read transactions, if the read bypass MCFG3→RB is set logical one, the memory checkbits of the loaded data is stored to the test checkbit MCFG3→TCB.

Cache Memories

Overview

The AT697F processor implements a Harvard architecture with separate instruction and data buses, connected to two independent cache controllers. In order to improve the speed performance of the cpu core, multi-set-caches are used for both instruction and data caches.

The cache replacement policy used for both instruction and data caches is based on the LRU algorithm. The least recently used (LRU) set of the cache is replaced when new data need to be stored in cache.

Cache mapping

Most of the main memory areas can be cached. The cacheable areas are the PROM and RAM areas. The following table presents the caching capabilities of the processor.

Table 17. Cache Capability List

Address Range	Area	Cache status
0x00000000 - 0x1FFFFFFF	PROM	Cached
0x20000000 - 0x3FFFFFFF	I/O	Non-cacheable
0x40000000 -0x7FFFFFFF	RAM	Cached
0x80000000 -0xFFFFFFFF	Internal	Non-cacheable

Operation

During normal operation, the processor accesses instructions and data using ASI 0x8 - 0xB as defined in the SPARC standard.

Using the LDA/STA instructions, alternative address spaces as caches can be accessed. ASI[3:0] are used for the mapping when ASI[7:4] have no influence on operation.

- Access with ASI 0 - 3 will force a cache miss, update the cache if the data was previously cached or allocate a new line if the data was not in the cache and the address refers to a cacheable location.
- Access to ASI 4 and 7 will force a cache miss and update the cache if the data was previously cached.

The following table shows the ASI implementation on the AT697F.

Table 18. ASI Usage

ASI	Usage
0x0, 0x1, 0x2, 0x3	Forced cache miss (replace if cacheable)
0x4, 0x7	Forced cache miss (update on hit)
0x5	Flush instruction cache
0x6	Flush data cache
0x8, 0x9, 0xA, 0xB	Normal cached access (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 v8 specification for detailed information on ASI usage.

Instruction Cache

Overview

The AT697F instruction cache is a multi-set cache of 32 kbyte divided in 4 memory sets. Multi-set-cache use improves speed performance of the core. The instruction cache is divided into cache lines with 32 bytes of data. Each line has a cache tag associated with it consisting of a tag field and one valid bit per 4-byte sub-block.

Cache Control

The instruction cache operations are controled with the cache control register (CCR).

Operation

On an instruction cache miss to a cachable location, the instruction is fetched and the corresponding tag and data line updated. The instruction cache always works in one of three modes:

- disabled,
- enabled
- or frozen.

The instruction cache current state is reported in the instruction cache state CCR→ICS.

Disabled mode

If disabled, no cache operation is performed and load and store requests are passed directly to the memory controller.

Enabled mode

If enabled, the cache operates as described above. In the frozen state, the cache is accessed and kept in synchronisation with the main memory as if it was enabled, but no new lines are allocated on read misses.

Freeze mode

If CCR→IF is set logical one, the instruction cache is 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 a code segment. The execution of the interrupt handler will not evict any cache lines and when control is returned to the interrupted task, the cache state is identical to what it was before the interrupt.

If a cache has been frozen by an interrupt, it can only be enabled again by enabling the cache in the CCR. This is typically done at the end of the interrupt handler before control is returned to the interrupted task.

Burst fetch

An instruction burst fetch mode can be enabled setting logical one in CCR→IB. If the burst fetch is enabled, 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. If the IU cannot accept the streamed instructions due to internal dependencies or multi-cycle instruction, the IU is halted until the line fill is completed.

If the IU executes a control transfer instruction during the line fill, the line fill will be 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

Instruction cache can be flushed by executing the FLUSH instruction, setting logical one in CCR→FI, or writing any location with ASI=0x5. The flush operation takes one cycle per line during which the IU will is not halted, but during which the cache is disabled. When the flush operation is completed, the cache will resume the state indicated in the cache control register.

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 (tt=0x1) is generated.

Instruction Cache Parity

Error detection of cache tags and data is implemented using two parity bits per tag and per 4-byte data sub-block. The tag parity is generated from the tag value and the valid bits. The data parity is derived from 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 data (tag) bits.

If a tag parity error is detected during a cache access, a cache miss is generated. The tag and the data are automatically updated. All valid bits except the one corresponding to the newly loaded data are cleared. Each error is reported in the instruction cache tag error counter from the CCR. The instruction cache tag error counter CCR→ITE is incremented after each instruction cache tag error detection.

If a data sub-block parity error occurs, a miss is also generated but only the failed sub-block is updated with data from main memory. Each error is reported in the instruction cache data error counter from the CCR. The instruction cache data error counter CCR→IDE is incremented after each instruction cache data error detection.

Data Cache

Overview

The AT697F data cache is a multi-set cache of 16 kbyte divided in 2 memory sets. Multi-set-cache use improves speed performance. The data cache is divided into cache lines with 16 bytes of data. Each line has a cache tag associated with it consisting of a tag field and one valid bit per 4-byte sub-block.

Cache Control

The instruction cache operations are controled with the cache control register (CCR).

Operation

Write

The write policy for stores is write-through with no-allocate on write-miss. The write buffer (WRB) consists of three 32-bit registers used to temporarily hold store data until it is sent to the destination device. For half-word or byte stores, the stored data replicated into proper byte alignment for writing to a word-addressed device, before being loaded into one of the WRB registers.

The WRB is emptied prior to a load-miss cache-fill sequence to avoid any stale data from being read in to the data cache.

Read

On a data cache read-miss to a cachable location, 4 bytes of data are loaded into the cache from main memory.

Cache Flush

Data cache can be flushed by executing the FLUSH instruction, setting logical one in CCR→FD in the cache control register, or writing any location with ASI=0x6. The flush operation takes one cycle per line during which the IU will is not halted, but during which the cache is disabled. When the flush operation is completed, the cache will resume the state indicated in the cache control register.

Error Reporting

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

Note: the 0x2B trap handler should flush the data cache, since a write hit would update the cache while the memory would keep the old value due the write error.

If a memory access error occurs during a data load, the corresponding valid bit in the cache tag will not be set. and a data access error trap (tt=0x09) is generated.

Data Cache Parity

Error detection of cache tags and data is implemented using two parity bits per tag and per 4-byte data sub-block. The tag parity is generated from the tag value and the valid bits. The data parity is derived from 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 data (tag) bits.

If a tag parity error is detected during a cache access, a cache miss is generated. The tag and the data are automatically updated. All valid bits except the one corresponding to the newly loaded data are cleared. Each error is reported in the instruction cache tag error counter from the CCR. CCR→DTE is incremented after each data cache tag error detection.

If a data sub-block parity error occurs, a miss is also generated but only the failed sub-block is updated with data from main memory. Each error is reported in the data cache data error counter from the CCR. CCR→DDE is incremented after each data cache data error detection.

Data Cache Snooper

In addition to the cache controller, a snooper is implemented on the on-chip cache subsystem. The cache snooper is enabled setting logical one in CCR→DS.

This snooper is able to verify if a master on the internal bus accesses and modifies some cached data. If a master accesses a data in memory and this data is cached, the snooper will invalidate the corresponding cache tag. Next time the IU will access the modified data, a cache miss will be generated due to not valid tag.

Diagnostic Cache Access

Tags and data in the instruction and data cache can be accessed through ASI address space 0xC, 0xD, 0xE and 0xF by executing LDA and STA instructions. Address bits making up the cache offset will be used to index the tag to be accessed while the least significant bits of the bits making up the address tag will be used to index the cache set.

Diagnostic read of tags is possible by executing an LDA 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 may be read by executing an LDA instruction with ASI=0xD for instruction cache data and ASI=0xF for data cache data. The sub-block to be read in the indexed cache line and set is selected by A[4:2].

The tags can be directly written by executing a STA instruction with ASI=0xC for the instruction cache tags and ASI=0xE for the data cache tags. The cache line and 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.

D[31:10] is written into the ATAG filed and the valid bits are written with the D[7:0] of the write data. The data sub-blocks can be directly written by executing a STA instruction with ASI=0xD for the instruction cache data and ASI=0xF for the data cache data. The sub-block to be read in the indexed cache line and set is selected by A[4:2].

Note: Diagnostic access to the cache is not possible during a FLUSH operation and will cause a data exception (trap=0x09) if attempted.

Timer Unit

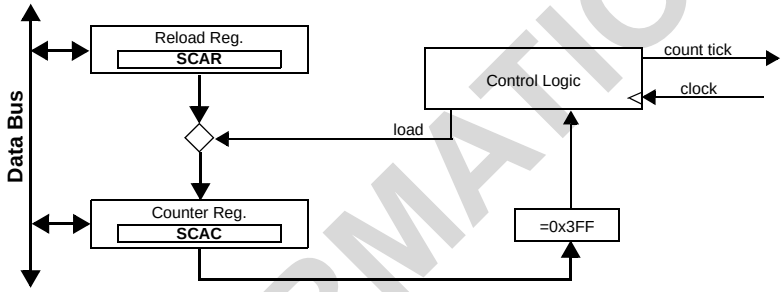
Prescaler

Timer/Counter1, Timer/Counter2 and the watchdog share the same prescaler.

The prescaler consists of a 10-bit down counter clocked by the system clock. The prescaler is decremented on each clock cycle. When the prescaler underflows, it is automatically reloaded with the content of the prescaler reload register. A count tick is generated for the two timers and the watchdog.

The effective division rate is equal to **prescaler reload register value + 1**.

Figure 30. Prescaler Block Diagram



Note: The reset value for SCAR is 0. This is not a legal value, it is however equivalent to a value of 3 and leads to a division rate of 4.

Caution :

The two timers and watchdog share the same decrementer. The minimum allowed prescaler division factor is 4 (reload register = 3).

Timer/Counter 1 & Timer/Counter 2

Timer/Counter1, Timer/Counter2 are two general purpose 32-bit timers. They share the same decrementer. The timer value is then decremented each time the prescaler generates a timer pulse.

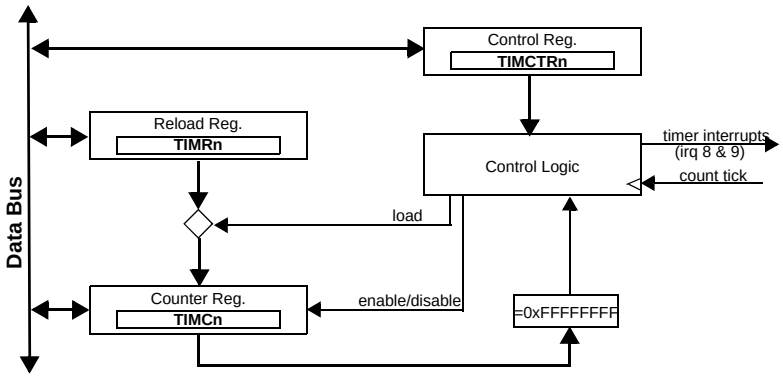
Each timer operation is controlled through a dedicated Timer Control register (TIMCTR). A timer is enabled/disabled by setting TIMCTRx→ENx.

Each time a timer underflows, an interrupt is generated. These interrupts can be masked with the Interrupt Mask and Priority register (ITMP).

Setting TIMCTRx→RLx, the content of the reload register (TIMR) is automatically reloaded in the Timer Counter register (TIMC) after an underflow and the timer continue running. If the reload bit is reset, the timer stops running after its first underflow.

Timer Counter can be forced with the Timer Reload value at any time by asserting the load bit TIMCTRx→LDx in the Timer Control register.

Figure 31. Timer/Counter 1/2 Block Diagram



Watchdog

The watchdog operates the same way as the timers, with the difference that it is always enabled and upon underflow asserts the external signal WDOG. This signal can be used to generate a system reset.

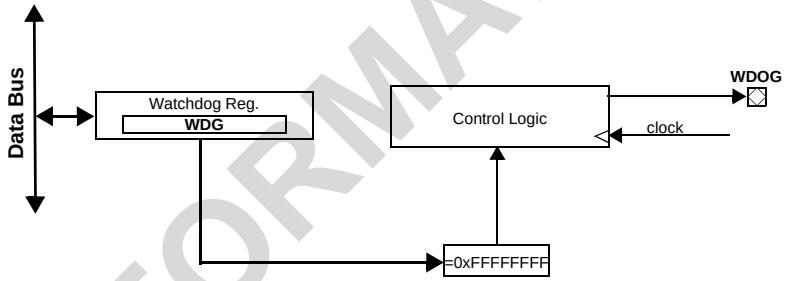
If the watchdog counter is refreshed by writing to WDG register before the counter reaches zero, the counter restarts counting from the new value.

If the counter is not refreshed before the counter reaches zero, WDOG signal is asserted.

After reset, the watchdog is automatically enabled and starts running. The watchdog is reset to a “all ones”. Together with the default prescaler ratio of 4, the time until first expiration of the watchdog after reset is about 2^34 clock cycles.

Note: A read access gives the decounting value of the watchdog, the reload value itself is not stored in the processor.

Figure 32. Watchdog Block Diagram



General Purpose Interface

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

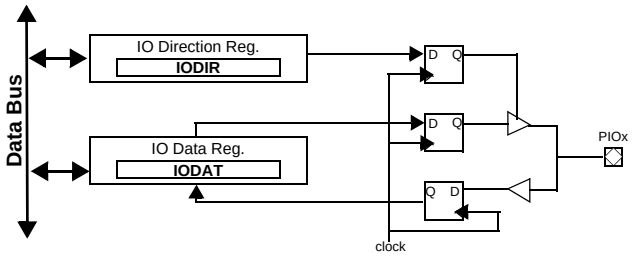
GPI as 32-bit I/O port

The interface is based on bi-directional I/O ports.The port is split in two parts, with the lower 16-bits accessible by the parallel IO pads and the upper 16-bits via the data bus.

lower 16-bits

The lower 16-bits of the general purpose interface are accessible through PIO[15:0]. All I/O ports have true Read-Modify-Write functionality when used as general I/O ports. This means that the direction of one port pin can be changed without unintentionally the direction of any other pin. The same applies when changing the drive value of the port.

Figure 33. I/O port block diagram - PIO[15:0]



configuring the pin

Each pin from PIO[15:0] consists of two register bits : IODIRx and IODATx. As shown in the “Register Description” section, the IODIRx bits are accessed at IODIR address and iodatx at IODAT address.

The IODIR→IODIRx bit selects the direction for port number x. If IODIRx is written logic one, the corresponding pin is configured as output. If written logic zero, the pin is configured as an input.

When the pin is configured as an input, a read of the IODAT→IODATx bit returns the current value of the pin. When the pin is configured as an output, if a logical one is written to IODAT→IODATx bit, the port x is driven high. If a logical zero is written to IODAT→IODATx bit, the port x is driven low.

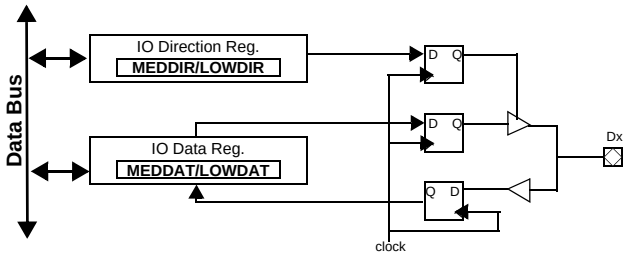
switching between input & output

When the port x is switched from input to output by switching IODIRx, the value of IODATx is immediatly driven on the corresponding pin.When switched from output to input by toggling IODIRx, the value from the pin is immediatly written to IODATx.

upper 16-bits

The upper 16-bits of the general purpose interface are accessible through D[15:0]. They can only be used when all memory areas (ROM, RAM and I/O) are 8-bit wide. If the SDRAM controller is enabled, the upper 16-bits cannot be used.

Figure 34. I/O port block diagram - D[15:0]



configuring the pin

The upper 16 bits of the general purpose interface can only be configured as outputs or inputs on byte basis. D[15:8] is referenced as the medium byte when D[7:0] is referenced as the lower byte.

Each byte from D[15:0] consists of two register fields. As shown in the “Register Description” section, the direction fields are accessed at IODIR address when data fields at IODAT address.

The IODIR→MEDDIR bit and the IODIR→LOWDIR bit select the direction for respectively the medium byte (D[15:8]) and the lower byte (D[7:0]). If MEDDIR (or LOWDIR) is written logic one, the corresponding byte in D[15:0] is configured as output. If written logic zero, the byte is configured as an input.

When configured as an input, a read of the IODAT→MEDDAT fileds returns the current value of D[15:8]. When configured as an output, the logical value from IODAT→MEDDAT field is translated in physical values on D[15:8] bus.

When configured as an input, a read of the IODAT→LOWDAT fileds returns the current value of D[7:0]. When configured as an output, the logical value from IODAT→LOWDAT field is translated in physical values on D[7:0] bus.

switching between input & output

When the medium byte (or the lower) is switched from input to output by switching MEDDIR (or LOWDIR), the value of MEDDAT (or LOWDAT) is immediatly driven on the corresponding pin. When switched from output to input by toggling MEDDIR (or LOWDIR), the value from the pins are immediatly written to MEDDAT (or LOWDAT).

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 summaryses the assignement of the alternate functions.

Table 19. GPI alternate functions

GPI port pin	Alternate function
PIO[15]	TXD1 - UART1 transmitter data
PIO[14]	RXD1 - UART1 receiver data
PIO[13]	RTS1 - UART1 request-to-send
PIO[12]	CTS1 - UART1 clear-to-send
PIO[11]	TXD2 - UART2 transmitter data
PIO[10]	RXD2 - UART2 receiver data
PIO[9]	RTS2 - UART2 request-to-send
PIO[8]	CTS2 - UART2 clear-to-send
PIO[3]	UART clock - Use as alternative UART clock
PIO[2]	EDAC enable - Enable EDAC checking at reset
PIO[1:0]	Prom width - Defines PROM bus width at reset

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

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

PCI Arbiter

A PCI arbiter is embedded on the AT697 chip. The¨PCI arbiter enables the arbitration of 4 PCI agents numbered from 3 downto 0. A round-robin algorithm is implemented as arbitration policy. The PCI arbiter is totally independent from the PCI interface

Operation

An Agent on the PCI bus requests the bus by driving low its REQ* line. When the arbiter determines that the bus can be granted to an agent, it drives low the corresponding GNT* line.

When the bus is granted to a PCI agent, the agent keeps the bus for only one transaction. If the agent desires more accesses, it shall continue to assert its REQ* line and wait to be granted the bus again.

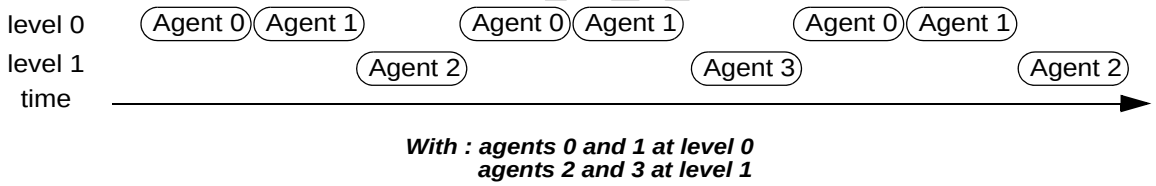
Round Robin

The round robin algorithm used for the arbitration is based on various loops with different priority levels. The implementation in the AT697 is based on two priority loops. A high priority loop is defined as level 0. A low priority loop is defined as level 1.

Operation

The arbitration is done checking the REQ* lines of the PCI agents one after each other. In first place, the loop with level 0 is checked. If a a REQ* is active and no master is currently granted ther bus, the corresponding GNT* line is driven low. Then, the agent is granted the bus. At each complete round-turn in level 0, one step is done in level 1. The following figure illustrates the operation of the arbitre.

Figure 35. Arbitre operation - Agent



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

If all agents have a request at the same time, the following probabilities of access are implemented:

- All agents in one level have equal probability
- All agents in level 1 together have the same probability of access as one agent in level 0.
- If no agent is in level 0, or no agent in level 0 has a request, all agents in level 1 are granted with equal probability

Bus Parking

As long as no bus request is active on the arbiter, the bus is granted to the last owner. It remains granted to the last owner until another agent requests the bus. When another request is asserted, re-arbitration occurs after one turnover cycle.

After reset, the bus is parked to agent 0. Agent 0 is the default owner after a reset operation.

Re-arbitration

When a master is managing a transfer and another one makes a request to the arbiter, re-arbitration occurs. Only one re-arbitration is performed during a transfer. A new arbitration will take place when the master which was granted the bus frees the bus. As long as all the PCI agents have no request pending, the arbitration is performed. A re-arbitration cycle also occurs when living the bus parking state.

Priority definition

Two different priority levels are defined for the PCI arbiter. Level 0 is defined as the high priority level. Level 1 defines the low priority level. Assignment of the PCI agents priority level is programmable through the arbiter configuration register (ACR).

Each PCI agent can be individually configured to operate either on level 0 or on level 1, except agent 3 that is defined by hardware with a low priority (level 1).

Setting logical one in PCIA→Px leads the agents x to a low priority level. Setting this bit logical zero leads to a high priority.

After reset, all the PCI agents are configured in the low priority loop.

PCI Interface

Overview

The PCI interface implementation is compliant with the PCI 2.2 specification. 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 AT697 processor embedds the In-Silicon PCI core. It is interfaced to the processor core through the PCI to AMBA bridge developped by the European Space Agency.

The PCI bus operations can be clocked at a frequency up to 33MHz, independently of the processor clock. Synchronization of the operation between PCI interface and AT697 core implies numerous FIFO usage. This implementation allows to use the device for Initiator (Master) and Target operations. In each mode single word and burst transfer can be executed.

Two different operating modes can be used with the PCI interface :

- Host Bridge
  The host-bridge connects the local bus of a processor to the PCI bus. Its PCI configuration registers are accessible locally by the processor, but not through PCI configuration cycles. Host-bridge initialises other satellite devices through PCI configuration commands.
- Satellite
  The satellite is a PCI device, configurable via PCI configuration cycles and the idsel line, but not locally.

Both, host-bridge and satellites can be initiator and/or target on the bus. The present interface has universal functionality, allowing both operation modes. The mode is configured via a hardware bootstrap on the SYSEN* pin. The state of the SYSEN* pin is copied in PCIIS→SYS. This enables plug and play boot programs loading the appropriate driver depending on the hardware configuration. In the same manner, the configuration registers are made visible as read only when the device is configured as satellite

Some other features are supported by this interface like

- Target lock support
- Zero-latency Fast Back-to-Back transfers
- Zero wait state burst mode transfers
- Support for memory read line/multiple
- Support for memory write and invalidate commands
- Delayed read support
- Flexible error reporting by polling

The PCI bus is a multiplexed one. In this way, address and data through the same medium. That is why PCI communication is based on two phase burst transfer. Each transfer is composed of the following phases :

- An address phase
  During the address phase, the initiator of the communication drives the 32-bit address concerned by the transfer and the command involved through this transfer. The command defines the space area concerned with the transfer and the direction of the transfer.
- A data phase
  During the data phase the initiator of the communication drives the enable bit signal so that only active part of the bus is enabled. When reading, the initiator drives the enable bits and the target set the data on the bus.

PCI Initiator (Master)

The PCI initiator mode of the AT697 gives a direct memory-mapped (initiator) access to the PCI bus. Any access to a memory address in the PCI address range is automatically translated by the interface into the appropriate PCI transaction. In this configuration, the PCI bus is accessed by the same instructions as the main memory. The SPARC instruction set foresees various load/store instruction types. The PCI bus foresees 32 bit wide transactions with byte-enables for each byte lane.

Initiator Mapping

For standard operation, the PCI interface only works in a limited address range. The address range for such initiator transaction is limited to addresses between 0xA0000000 and 0xF0000000.

PCI addresses outside of this predefined range can be accessed only via DMA transactions.

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

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

A byte-writing to A[1:0] = 00 results in the byte enable pattern 0111, indicating that the e most significant byte lane (bits 31:24) of the PCI data bus is selected.

The following table presents the transaction width authorized for PCI transfers.

Table 20. Byte Enable Settings

width	8	16	32	64
Assembler	ld[s/u]b, stb	ld[s/u]h, sth	ld, st	ldd, std
C-datatype	char	short	int	long long
A[2:0]=000	0111	0011	0000	0000 (burst)
A[2:0]=100	0111	0011	0000	not aligned
A[2:0]=x01	1011	not aligned	not aligned	not aligned
A[2:0]=x10	1101	1100	not aligned	not aligned
A[2:0]=x11	1110	not aligned	not aligned	not aligned

Note: PCI byte enables are active low.

For non-aligned accesses, the byte enable pattern (1111) is issued on PCI, to avoid destroying data in the remote PCI target.

Memory cycles

Many memory transactions such as memory-read/write and memory-read-line/write-invalidate can be issued from the processor with common SPARC instruction set. Selection of the command to execute is performed setting the value PCIIC→COMMSB.

Setting logical ‘01’ in PCIIC→COMMSB result in the generation of memory read/write access when PCI address is accessed. A logical value of ‘10’ result in a memory read line or write and invalidate on PCI address access.

For the memory commands the address issued on the PCI bus is a word address with bits (1:0) set to 00. This indicates that the linear incrementing mode is used.

operation

The following procedure shall be used to engage memory transaction on the PCI interface:

1. Select the initiator mode by setting logical one in the PCIIC→MOD.
2. Select the memory load/store command or the memory read-line/write and invalidate command in the PCI initiator configuration register. The PCIIC→COMMSB shall be set logical ‘01’ for simple load/store operation and shall be set logical ’11’ for read-line/write-&-invalidate.
3. Enabling the interrupt signalisation is optionnal. It can be enabled setting logical one in PCIITE→IMIER. Up to four interrupt sources can be defined : Initiator Error, Initiator Parity Error, PCI core error and system error.
4. Engage an access to a memory address mapped in the PCI address range.

IO transaction cycles

operation

The following procedure shall be used to engage I/O transaction on the PCI interface:

1. Select the initiator mode by setting logical one in PCIIC→MOD.
2. Select the I/O load/store command in the PCI initiator configuration register. The PCI-IC→COMMSB shall be set logical ‘00’ for I/O operation.
3. Enabling the interrupt signalisation is optionnal. It can be enabled setting logical one in PCIITE→IMIER. Up to four interrupt sources can be defined : Initiator Error, Initiator Parity Error, PCI core error and system error.
4. Engage an access to an I/O address mapped in the PCI address range.

Configuration cycles

Target selection

Accesses to a configuration address space requires the target device to be selected. Due to the address range limitation, the chip-select (IDSEL) connection necessary for device selection shall be done using only A/D[27:16]. This allows up to 12 PCI devices to be connected on the bus.

Devices with chip-select line connected to A/D[31:28] can’t be configured through standard operations. DMA configuration cycles shall be used to configure the devices connected to A/D[31:28].

The PCI bus configuration cycles can be performed using the same instructions as the main memory. To generate such configuration cycle with the standard instructions,PCIIC→COMMS shall be programmed to ‘01’.

Then, if a load (or store) cycle is performed to an addresss in the PCI address range, a physical configuration cycle is performed on the PCI bus. The full 32-bit address defined on the internal bus is propagated on the PCI bus. Once a target is selected (DEVSEL* asserted).

Operation

The following procedure shall be used to engage configuration cycle on the PCI interface:

1. Select the initiator mode by setting logical one tin PCIIC→MOD
2. Select the configuration load/store command in PCIIC→COMMS shall be set logical ‘10’ for configuration operation.
3. Enabling the interrupt signalisation is optionnal. It can be enabled setting logical one in PCIITE→IMIER. Up to four interrupt sources can be defined : Initiator Error, Initiator Parity Error, PCI core error and system error.
4. Engage an access to an configuration space.

Limitation

Configuration cycles shall only be generated by the PCI host of the bus or by a PCI-to-PCI bridges.

Special cycles

By default, all requests are translated into single cycle PCI transactions, each transaction consisting in an address phase followed by a single data phase.

Linear incrementing store-word

Linear incrementing store-word sequences are translated into undetermined length PCI write bursts with up to a maximum of 255 words. The PCI burst mode is then maintained as long as possible. Read/write direction is unchanged and the address $A_{n+1} = A_n + 4$. When the sequence is discontinued, the PCI burst stops with a last data phase during which byte enables are 1111.

Fast back2back cycles

The PCI implementation only supports fast back2back cycles to the same target. Before using fast back-2-back transfers, fast back-2-back cycles shall be enabled setting logical one the bit COM9 in the status command register (PCISC). PCISC→COM9 shall only be set one if all targets on the bus support fast back-2-back transfers. Issuing a fast back to back transfer is done setting logical one in PCIDMA→B2B.

Note: Fast back-2-back can only be generated by the initiator. It is not accepted by the AT697 PCI target.

Error reporting

Fatal (abort) and address parity errors

On a fatal error (or address parity error), the interface flushes all the current buffer requests and all other buffer requests. Then, the interface reports the fatal error driven logical one the PCIITE→CMFER.

The PCI core is restarted as soon as a new request is engaged.

DMA transfer

A DMA facility is available on the AT697 processor. The DMA transfer are performed through the PCI interface. The DMA controller executes data transfer between the local memory and a remote target on the PCI bus.

The processor core only intervenes for the initiation of the transfer. Once transfer is initiated, DMA controller is fully autonomous. DMA transfers take place in background of the processor core activity. Thus, interrupts are provided to help to synchronise the application with start and end of the transfer.

The DMA interface executes only word-size transactions with all 4 byte lanes enabled.

Operation

The DMA is enabled setting logical one the PCIIC→MOD. To synchronize the application with the start and the end of the transaction, two interrupts can be enabled : PCIITE→DMAER for transfer control and PCIITE→IMIER for error control.

Each DMA sequence shall program the following parameters :

- PCI start address
- PCI command type
- number of words to be transferred
- the start address in the local memory

A DMA transfer is performed assuming the following operations are done in the given order :

1. Write the PCI start address of burst to the PCI start address register (PCISA). The PCISA register shall be re-writen each time a DMA transfer is initiated, even if the address is identical to the address of the previous DMA request.
2. Write together the PCI command and the number of words to be transfered in the PCI DMA configuration register (PCIDMA).
   Writing to the PCIDMA passes the PCI address, the word count and the PCI command to the PCI core and initiates the transaction on the PCI bus.
3. Write the start address in the local memory map to the PCI DMA address register (PCIDMAA).

Once the three operation are executed, data transfer is started in background. Once the specified number of words is transfered, the interface set logical one the PCIITE→DMAER and generate an interrupt if enabled. Then DMA controller goes back to idle state.

Error Reporting

If the PCI core does not accept the DMA cycle request, the DMA state controller remains locked and an error is reported as initiator error with the PCIITE→IMIER bit set logical one. If the request on the PCI core was just delayed, rewriting PCIDMAA may succeed. If the problem persists, reset the interface by writing –1 (0xFFFFFFFF) to PCIIC.

Transfer Limitation

A DMA transaction may never cross a 1 KByte border. The value represented by PCIDMAA(9:2) + PCIDMA(7:0) must be less than 256. If this restriction is not respected, the data transfer stops at the 1 kByte border. Then the PCI core is flushed. Simultaneously, in the PCI interrupt pending register (PCIITP) the dma error bit PCIITE→DMAER and the initiator error bit PCIITE→IMIER are asserted logical one.

If enabled with the PCI interrupr enable register (PCIITE) and unmasked in the general interrupt mask register, the PCI interrupt 14 is generated (TT = 0x1E).

Debug Facilities

Not implemented for application use.

Target Mode Transfer

In the target mode, the PCI interface receives requests originated from remote PCI initiators (masters). Target data transfer is executed in background without AT697 core intervention. AT697 core can only intervenes is the configuration of the target.

- In host bridge mode the target is configured by the AT697 core
- In satellite mode the configuration is done by a remote device using the PCI command set

Target Programming

The target is configured through the following registers :

- PCISC register
  bits 0/1 for memory and I/O command response
  bit 6 for check of data and address parity error
  bit 7 for response to data and address parity error
- base address registers
  memory base address : MEMBAR1, MEMBAR2
  I/O base address : IOBAR
- PCITPA register to indicate the storage location
- PCITSC→FRTY bit to write data in memory

transaction Ordering

As specified in the PCI standard, delayed read functionality is implemented, obeying to the following rules:

- The interface stores one delayed read at a time. When a read request was retried (because local data not yet available), the interface remains locked for any other target read (targeting different addresses). The initiator of the original read has to repeat its request to the same address.
- A retried (delayed) read can be interrupted by one or more PCI write accesses. The PCI standard requires this write command to be processed first, to prevent a system lock-up.
- Meanwhile, the interface will prefetch read-data into the TXMT FIFO. After the (interfering) write, when the read request is repeated, and the requested data is available in the FIFO the delayed transfer completes normally.

All target read accesses are generally prefetching, also reads with I/O command. Once a start address is given, the interface prefetches up to 8 words into the TXMT FIFO. After the last required data word was transferred to PCI, the PCI core automatically flushes the FIFO to discard the unused prefetched data. The interface assumes the complete local address space to be ‘prefetchable’, defined here as the fact, that reading from an address does not alter the data. This behaviour is to be considered if non-prefetchable devices (for example the UART’s) shall be read through the PCI target.

PCI Error Reporting

According to the PCI standard, error and status bits are implemented in the PCI status register.(PCISC). The PCI standard foresees a single parity check, by which bus-errors can be detected, but not corrected. Errors which occur in the PCI interface or on the PCI bus are also saved in status bits in the PCIITP register, and optionally, the PCI interrupt (IRQ14) is asserted.

Different events can be selected to assert the interrupt. By the interrupt enable register (PCIITE) configuration you can select the interrupt events which will assert IRQ14. then an interrupt handler can read the interrupting event in the status register (PCIITP).

Furthermore, interrupts can be forced for test purposes by writing to PCIITF.

In host-bridge configuration, this allows an error detection by polling. Certain events and errors are also reported by the interface in the interrupt status register. For each bit of this register , interrupt generation can be programmed individualy. All PCI interrupt generated are then reported to AT697 core through the PCI interrupt (IT14). The different interrupt causes are distinguished by the interrupt status registers settings.

Please refer to the register description chapter for more details on interrupt status register.

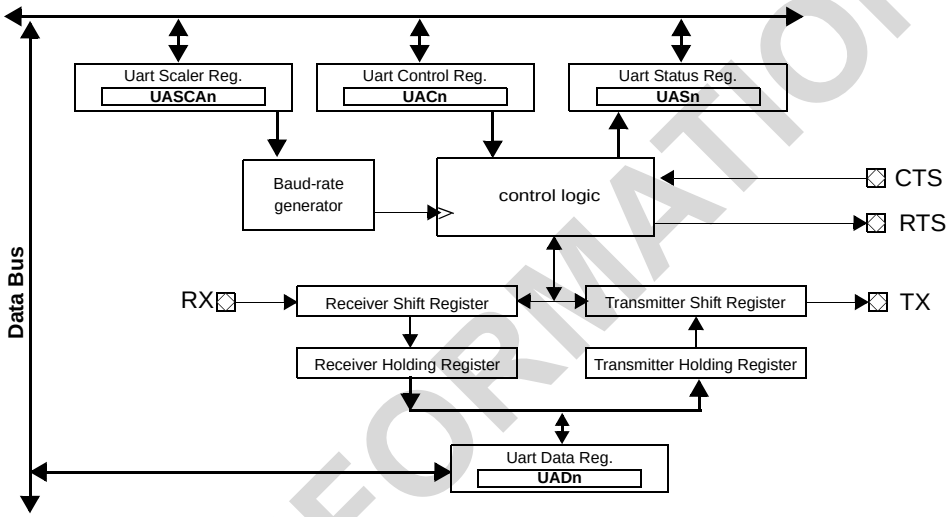
UARTs (UART1 and UART2)

The Universal Asynchronous Receiver and Transmitter (UART) is a highly flexible serial communication module. The AT697 implements two uarts : UART1 and UART2. Uarts on the processor are defined as alternate functions of the general purpose interface (GPI).

Overview

The two UART's provide double buffering. Each UART consists of a transmitter holding register, a receiver holding register, a transmitter shift register, and a receiver shift register. Each of these registers are 8-bit wide.

Figure 36. UART Block Diagram



Each UART is fully controlled by a set of four registers including :

- a control register
- a status register
- a scaler register
- and a data register

Serial Frame

A serial frame is defined to be one character of data bits with synchronisation bits (start and stop bits), and optionnaly a parity bit for error checking.

Frame formats

Two frame formats are accepted by the AT697 UARTs, the only difference being the presence or the absence of the parity bit. All the frames are built on an eight data bits basis.

A frame starts with the synchronization start bit followed by the least significant data bit. Then the next data bits, up to a total of eight, are succeeding, ending with the most significant bit. If enabled by setting the UACx→PEx, the parity bit is inserted after the data bits and before the stop bit.

The following figure illustrates the accepted frame formats.

Figure 37. Data frame format

Data frame, no parity:

Start	D0	D1	D2	D3	D4	D5	D6	D7	Stop

Data frame with parity:

Start	D0	D1	D2	D3	D4	D5	D6	D7	Parity	Stop

Parity bit

The parity bit is calculated by doing an exclusive-or of all the data bits. The odd parity is configured setting logical one the UACx→PSx . In this case, the result of the exclusive or is inverted. An even parity can be selected setting logical zero the UACx→PSx.

If used, the parity bit is located between the last data bit and the stop bit of the serial frame.

The relation between the parity bit and data bits is as follows:

$$P_{even} = d_7 \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0$$
$$P_{odd} = d_7 \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1$$

$\mathbf{P_{even}}$	Parity bit using even parity
$\mathbf{P_{odd}}$	Parity bit using odd parity
$\mathbf{d_n}$	Data bit n of the character

Clock Generation

The clock generation logic generates the base clock for the Transmitters and Receivers. The bit rate of the UART is issued from the clock generator after a combination between the input clock of the clock module and a scaler.

Uart Clock

Two clock inputs can be used by the clock generator :

- An internal clock
- An external clock

Each UART can be configured to use either the internal or the external clock source by programming the UACx→ECx. If set logical zero, the UART is clocked by the internal clock. If UACx→ECx is set logical one, the UART is clocked by the external clock. When using the external configuration, the UART clock shall be provided by PIO[3] from the general purpose interface. This clock input is used as an alternate function for PIO[3].

caution :

When using the external clock source, the frequency of PIO[3] must be less than half the frequency of the system clock.

Baud Rate Generation

To generate the bit-rate, each UART has a programmable 12-bits clock divider (UASCAx). According to the configuration of the UACx→ECx, the scaler is clocked either by the system or by an external clock.

Each time the scaler underflows, a UART tick is generated. The scaler is automatically reloaded with the value of the UART scaler register after each underflow. The resulting UART tick frequency should be 8 times the desired baud-rate.

The following equation shall be used to calculate the scaler value to define, depending on the clock source and the expected baud rate.

$$scaler = \frac{\frac{uartclk \times 10}{baudrate \times 8} - 5}{10}$$

variable description :

- ***uartclk*** : frequency of the uart clock
- ***baudrate*** : expected baud rate
- ***scaler*** : value to set in (UASCAx) to reach the expected baudrate

Communication Operations

UARTS operations are controlled through the uart control registers (UACx) and the Uart status registers (UASx).

Transmitter Operation

The transmitter is enabled setting logical one the UACx→TEx . When ready to transmit, data is transferred from the transmitter holding register to the transmitter shift register and converted to a serial frame on the transmitter serial output pin (TX).

Following the transmission of the stop bit, if a new character is not available in the transmitter holding register, the transmitter serial data output remains high and the transmitter shift register

empty bit UASx→TSx. Transmission resumes and the UASx→TSx is cleared when a new character is loaded in the transmitter holding register.

If the transmitter is disabled, it will continue operating until the character currently being transmitted is completely sent out. The transmitter holding register cannot be loaded when the transmitter is disabled.

If flow control is enabled, the CTS input must be low in order for the character to be transmitted. If it is deasserted in the middle of a transmission, the character in the shift register is transmitted and the transmitter serial output then remains inactive until CTS is asserted again. If the CTS is connected to a receivers RTS, overrun can effectively be prevented.

Receiver Operation

The receiver is enabled for data reception when the receiver enable bit UACx→REx is set logical one. The receiver looks for a high to low transition of a start bit on the receiver serial data input pin. If a transition is detected, the state of the serial input is sampled a half bit clocks later. If the serial input is sampled high the start bit is invalid and the search for a valid start bit continues. If the serial input is still low, a valid start bit is assumed and the receiver continues to sample the serial input at one bit time intervals until the proper number of data bits and the parity bit have been assembled and one stop bit has been detected. During this process the least significant bit is received first.

The serial input is sampled three times for each bit and averaged to filter out noise.

The data is then transferred to the receiver holding register and the data ready bit UASx→DRx is set logical one. The parity, framing and overrun error bits are set at the received byte boundary, at the same time as the receiver ready bit is set.

If both receiver holding and shift registers contain an un-read character when a new start bit is detected, then the character held in the receiver shift register will be lost and the overrun bit UASx→OVx is set logical one.

If flow control is enabled, then the RTS will be negated (high) when a valid start bit is detected and the receiver holding register contains an un-read character. When the holding register is read, the RTS will automatically be reasserted again.

A correctly received byte is indicated by the data ready bit UASx→DRx. In case of error (framing error, stop bit error,...), the respective bits UASx→FEx, UASx→PEx, ... are set logical one when the data ready bit remains logical zero.

Interrupt Generation

The two UARTs can be configured to generate interrupt each time a byte is received or a byte is sent.

If the UACx→TIx is set logical one, an interrupt is issued after each character sending. If set logical zero, no interrupt is issued on character sending.

If the UACx→RIx is set logical one, an interrupt is issued after each character reception. If set logical zero, no interrupt is issued after a character reception.

If the receiver interrupt is enabled, when error is detected during the reception of a character,an interrupt is generated. To identify the origin of the transaction failure, refer to the uart status register bits (UASx→OVx, UASx→PEx, UASx→TEx) that indicate either it is a parity, a framing or an overrun error.

Loop back mode

If the UACx→LBx is set, the UART will be in loop back mode. In this mode, the transmitter output is internally connected to the receiver input and the RTS is connected to the CTS. It is then possible to perform loop back tests to verify operation of receiver, transmitter and associated software routines. In this mode, the outputs remain in the inactive state, in order to avoid sending out data.

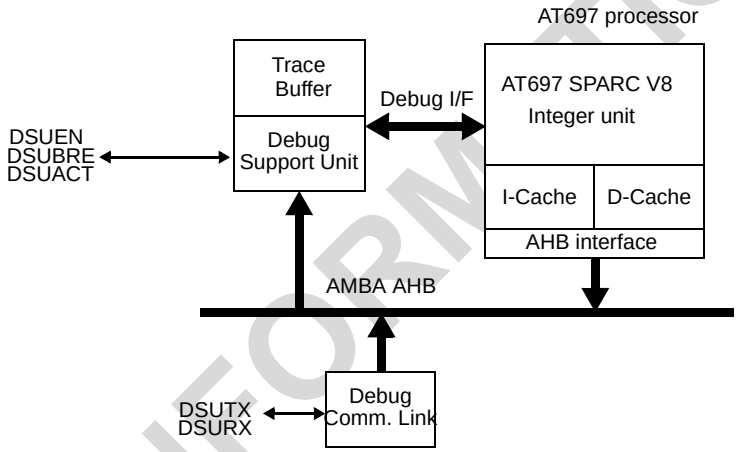
Debug Support Unit - DSU

Overview

The AT697 processor includes an hardware debug support unit to aid software debugging on target hardware. 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 38. 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 Mbyte 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. Entering the debug mode can occur on the following events:

- executing a breakpoint instruction (ta 1)
- integer unit hardware breakpoint/watchpoint hit (trap 0x0B)
- rising edge of the external break signal (DSUBRE)
- setting the break-now DSUC→BN
- a trap that would cause the processor to enter error mode
- occurrence of any, or a selection of traps as defined in the DSU control register
- after a single-step operation
- DSU breakpoint hit

The 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 mode. When the 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 AT697 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 DSUC→BN 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 reset 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 DSUC→DCNT to a non-zero value. In this case, the DSUC→DCNT value will be decremented for each additional trace until it reaches zero, after which the trace buffer is frozen. If the brake on trace freeze bit DSUC→BT is set logical one, 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. This counter is decremented each clock as long as the processor is running. The counter is stopped when the processor enters debug mode. It is restarted when execution is resumed.

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

Trace Buffer

The trace buffer consists of a circular buffer that stores the executed instructions 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 (TBC). 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 (TBC) contains two counters TBC→BCNT and TBC→ICNT that store the address of the trace buffer location that will be written on next trace. Since the buffer is circular, it actually points to the oldest entry in the buffer. The indexes are automatically incremented after each stored trace entry.

Instruction trace

The instruction trace mode is enabled setting logical one the trace instruction enable bit TBC→TI.

During instruction tracing, 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 [63:32] 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 [63:32] 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 puts the MSB 32 bits of the results in bit [63:32] while the second entry puts the LSB 32 bits in this field.

Table 21. Trace buffer data allocation, Instruction tracing mode

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 FPOP)
125:96	DSU counter	The value of the DSU 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

When a trace is frozen, interrupt 11 is generated.

Bus Trace

The bus trace mode is enabled setting logical one the trace instruction enable bit TBC→TA.

During bus tracing, one operation of the internal bus is stored per line in the trace buffer.

Table 22. Trace Buffer Data Allocation, Internal bus Tracing Mode

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)
95:80	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 on two halves, with instructions stored in the lower half and bus transfers in the upper half. The MSB bit of the AHB index counter is then automatically kept high, while the MSB of the instruction index counter is kept low.

DSU Memory Map

Table 23. DSU Map

Address	Register
0x800000c4	DSU UART status register
0x800000c8	DSU UART control register
0x800000cc	DSU UART scaler 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 - 0x90020000	Trace buffer
..0	Trace bits 127 - 96
...4	Trace bits 95 - 64
...8	Trace bits 63 - 32
...C	Trace bits 31 - 0
0x90020000 - 0x90040000	IU/FPU register file
0x90080000 - 0x90100000	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 - 0x9008007C	ASR16 - ASR31 (when implemented)
0x90100000 - 0x90140000	Instruction cache tags
0x90140000 - 0x90180000	Instruction cache data
0x90180000 - 0x901C0000	Data cache tags
0x901C0000 - 0x90200000	Data cache data

The addresses of the IU/FPU registersis defined according to how many register windows has been implemented. The registers can be accessed at the following addresses (NWINDOWS = number of SPARC register windows = 8):

- %o*n*: 0x90020000 + (((psr.cwp * 64) + 32 + *n*) mod (NWINDOWS*64))
- %l*n*: 0x90020000 + (((psr.cwp * 64) + 64 + *n*) mod (NWINDOWS*64))
- %i*n*: 0x90020000 + (((psr.cwp * 64) + 96 + *n*) mod (NWINDOWS*64))
- %g*n*: 0x90020000 + (NWINDOWS*64) + 128
- %f*n*: 0x90020000 + (NWINDOWS*64)

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 done 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 accesible 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

By writing the TBC→SS and reseting the TBC→BN bit, the processor will resume execution for one instruction and then automatically enter debug mode.

DSU Trap

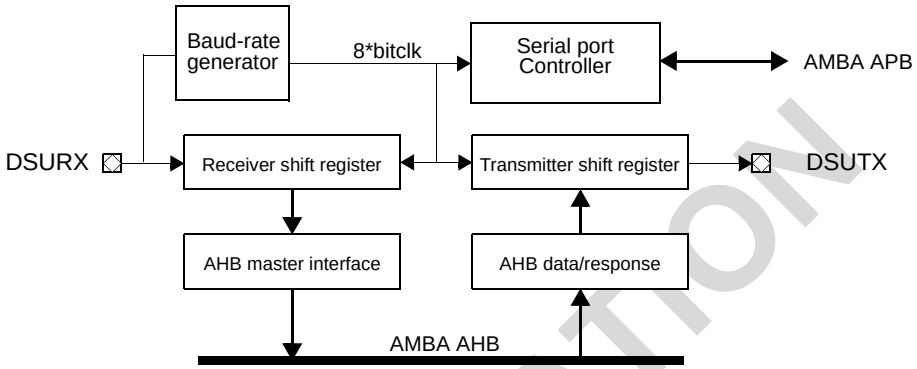
The DSU trap register (DTR) consists in a read-only register that indicates which SPARC trap type caused the processor to enter debug mode.

When debug mode is forced by setting the TBC→BN, the trap type is 0x0B.

DSU Communication Link

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

Figure 39. DSU Communication Link Block Diagram

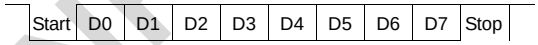


A simple communication protocol is supported to transmit access parameters and data. A link command consist of a control byte, followed by a 32-bit address, followed by optional write data. If the TBC→LR is set, a response byte will be sent after each AHB transfer. If the TBC→LR is not set, a write access does not return any response, while a read access only returns the read data.

Data Frame

Data is sent on 8-bit basis.

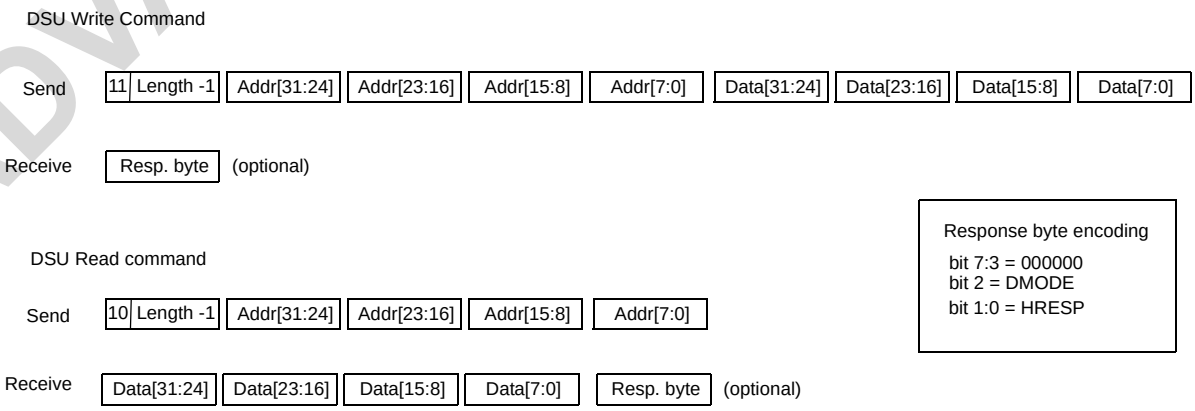
Figure 40. DSU UART Data Frame



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 41. DSU Commands



Clock Generation

The UART contains a 14-bit down-counting scaler to generate the desired baud-rate. The scaler is clocked by the system clock and generates a UART tick each time it underflows. The scaler is reloaded with the value of the UART scaler reload register after each underflow. The resulting UART tick frequency should be 8 times the desired baud-rate.

If not programmed by software, the baud rate will be automatically be 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, and the DSUUC→BL bit is set . If the DSUUC→BL is reset by software, the baud rate discovery process is restarted. The baud-rate discovey is also restarted when a 'break' is received by the receiver, allowing to change to baudrate from the external transmitter. For proper baudrate detection, the value 0x55 should be transmitted to the receiver after reset or after sending break.

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

$$scaler = \frac{\frac{sdclk\ frequency \times 10}{baudrate \times 8} - 5}{10}$$

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 initialised 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 24. TAP Pins

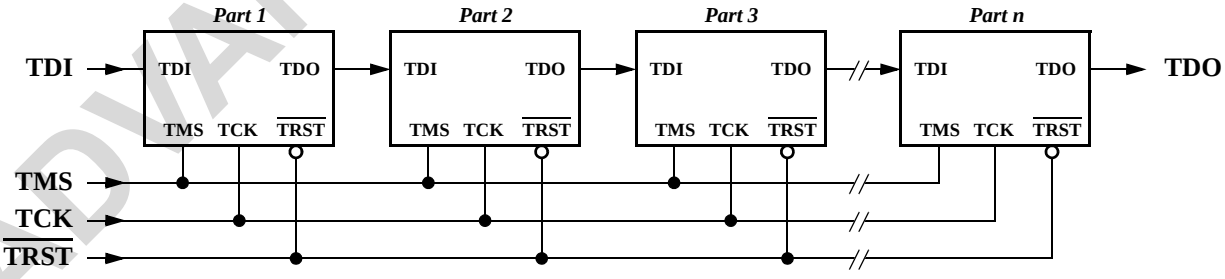
Pin	Name	Type	Description
TCK	Test Clock	Input	Used to clock serial data boundary into scan latches and control sequence of the test state machine. TCK can be asynchronous with CLK
TMS	Test Mode select	Input	Primary control signal for the state machine. Synchronous with TCK. A sequence of values on TMS 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 AT697 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.

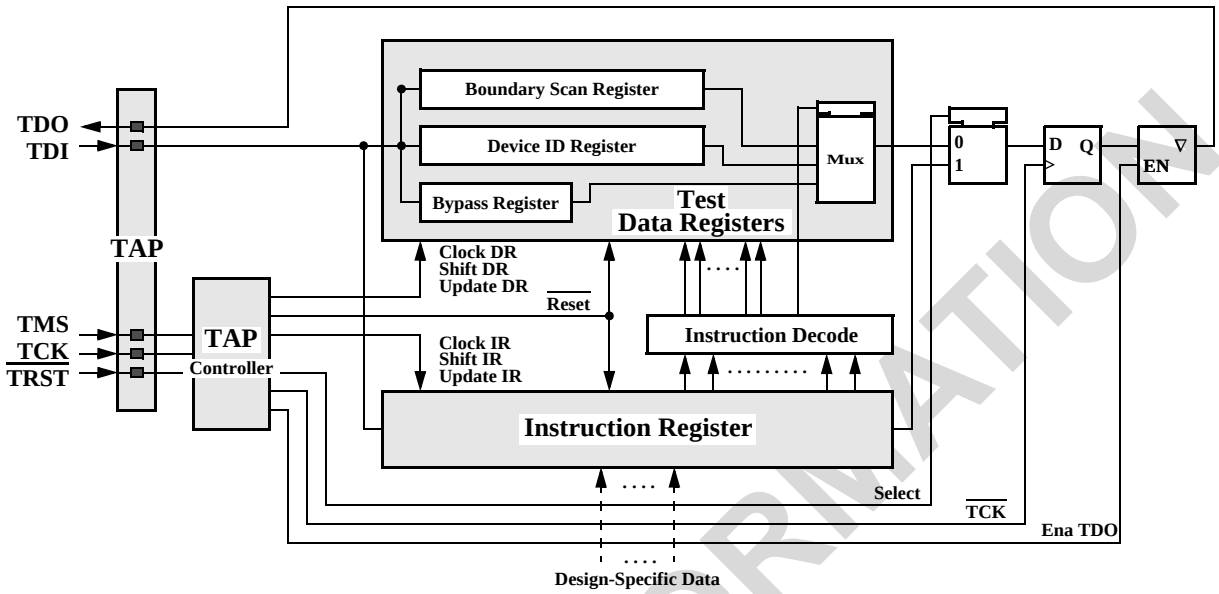
Figure 42. JTAG Serial connection using 1 TMS Signal



TAP Architecture

The TAP implemented in the AT697 consists of a TAP interface, a TAP controller, plus a number of shift registers including an instruction register (IR) and some registers .

Figure 43. AT697 TAP Architecture

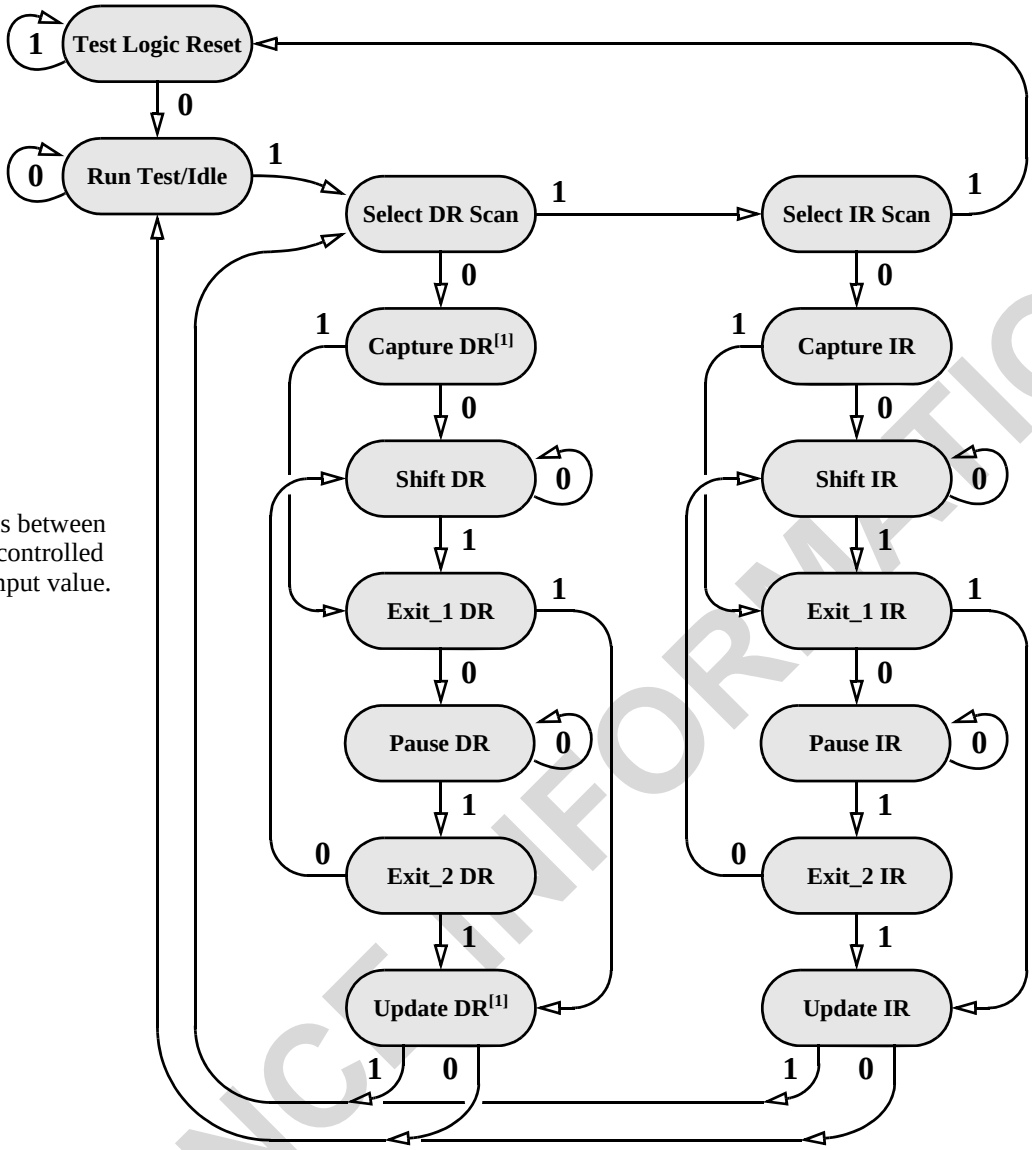


TAP Controller

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 AT697. 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 44. TAP - State Machine



Transitions between states are controlled by TMS input value.

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 AT697 TAP.

Table 25. 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
010	BYPASS	Bypass register	Bypassscan 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"

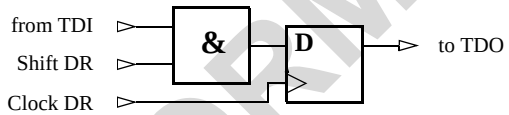
Value of the IDCODE is loaded during Capture DR.

Test Data Registers

The following data registers are supported in the AT697 TAP:

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

Figure 45. Bypass Register Cell



Device ID register

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

Figure 46. 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'.

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.

Boundary Scan register is connected between TDI and TDO.

To use the boundary scan feature, the PLL will be in bypass mode, i.e. BYPASS signal direction to VCC.

Checker Scan Register A single scan chain consisting of all of the scan cells of IU parity checkers. The checkers scan is only used for factory test. Checkers scan register is connected between TDI and TDO.

Execution Mode

Reset Mode

When the RESET input is asserted for at least two cycles, the processor enters reset mode. Under this mode, the CPU and all the peripherals are halted. Only the following registers are affected by the reset. All other registers maintain their value or are undefined.

Table 26. Reset Operation

Register	Description	Reset Value
PC	program counter	0x0000 0000
nPC	new program counter	0x0000 0004
PSR	processor status register	et = 0 s = 1
CCR	cache control register	0x0000 0000
MCFG1→PRWDH	PROM bus width	PIO[1:0]
MCFG3→PE	PROM EDAC enable	PIO[2]

When RESET is deasserted, execution restarts from address 0.

Debug Mode

Debug mode can be entered when the DSU is enabled through the external DSUEN pin. This allows read/write access to all processor registers and caches memories. In debug mode, the processor pipeline is held and the processor is controlled by the DSU.

Power-down/Idle Mode

AT697 can be idled by writing any value to the power-down register. During power-down mode, only the integer unit is halted. All other functions and peripherals operate as nominal.

When a single write to the idle register is performed, idle mode is entered on the next load instruction. Idle mode is terminated when an unmasked interrupt with higher level than the current processor interrupt level is pending. Then, the integer unit is re-enabled.

Here is a simple example allowing Idle mode entry :

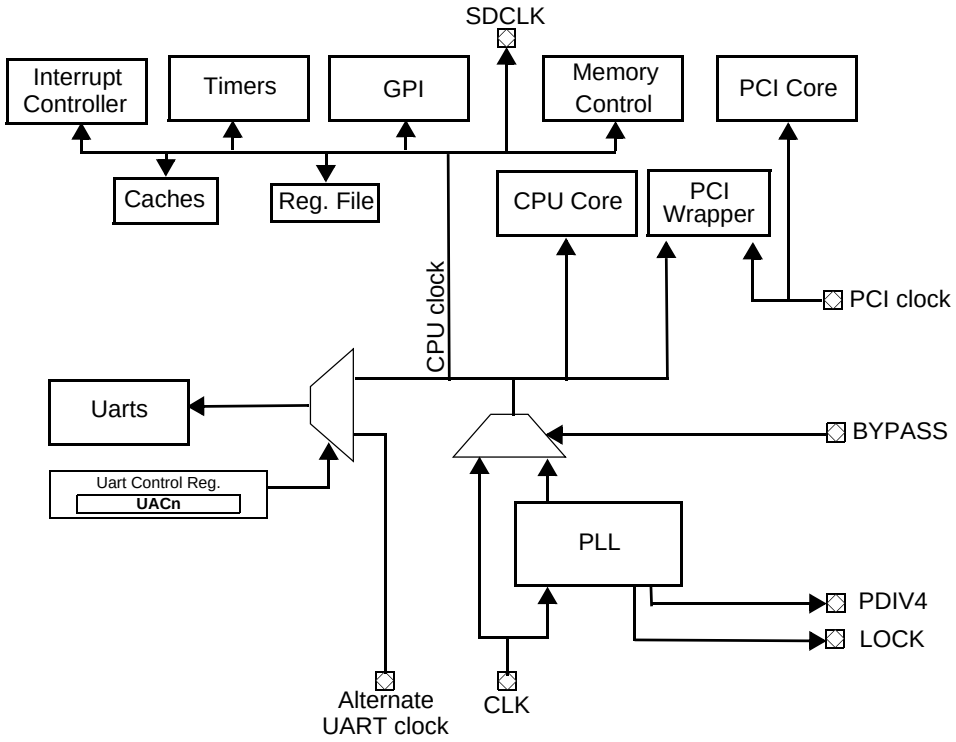
```
! write any value to Idle register
st %g2,[%g1 + 0x18]
! enter Idle mode
ld [%o1 + 0x08],%g3
```

System Clock

Overview

The AT697F clock system is mainly based on two main clock trees: the PCI clock and the CPU clock. The following figure presents the clock system of the processor and its distribution.

Figure 47. Clock Distribution



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 33MHz.

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 0MHz up to 100MHz.

PLL

Overview

The CPU clock can be issued from the internal PLL. This PLL contains a phase/frequency detector, charge pump, voltage control oscillator, low pass filter, lock detector and divider.

The PLL implemented is configured by hardware to provide a cpu clock frequency four times the frequency of the input clock.

PLL control

The PLL control is done by hardware through dedicated ports, including a bypass, a clock input and a filter input.

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

Table 27. PLL ports description

Pin name	Function
LOCK	Lock
CLK	Board clock input
BYPASS	Bypass

Operation

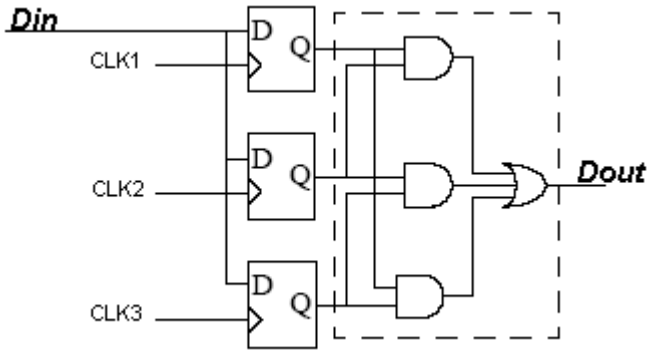
To drive the device from the internal PLL, the CLK input shall be driven by an external clock generator while the BYPASS pin is driven low. In that way, the CPU clock frequency is four time the frequency of the clock applied to CLK.

When the PLL based CPU clock source is selected, the clock input shall be driven at a frequency in the range of 18MHz up to 25MHz.

Fault Tolerance & Clock

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

Figure 48. TMR structure



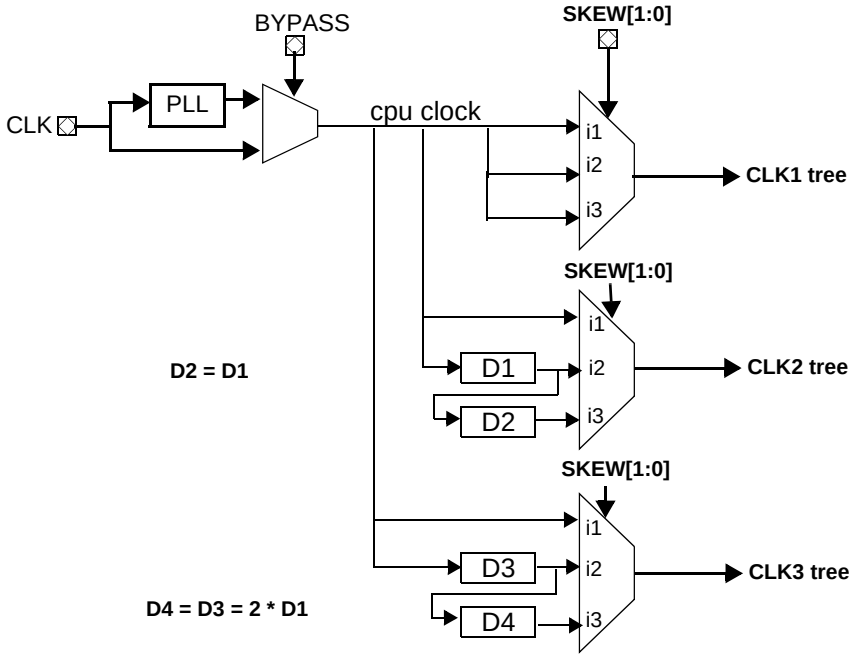
Such architecture is based on a fully triplicated clock distribution (CLK1, CLK2 and CLK3). In that way, each one of the PCI clock and the cpu clock are build as three-clock trees.

Skew

To prevent the processor from corruption by single event transient (SET) phenomenon, additional skew can be programmed on the clock trees. The two dedicated pins SKEW1 and SKEW0 are used to program the delay induced by the skew.

Here is a short description of the skew implementation :

Figure 49. CPU clock tree overview



Three configuration of skew are available :

- SKEW[1:0] = ’00’ : natural skew corresponding to the intrinsec routage of the chip
- SKEW[1:0] = ’01’ : medium skew ‘artificially’ injected
- SKEW[1:0] = ’10’ : maximum skew ‘artificially’ injected

The remaining configuration (SKEW[1:0] = ’11’) is reserved and must not be used at application level.

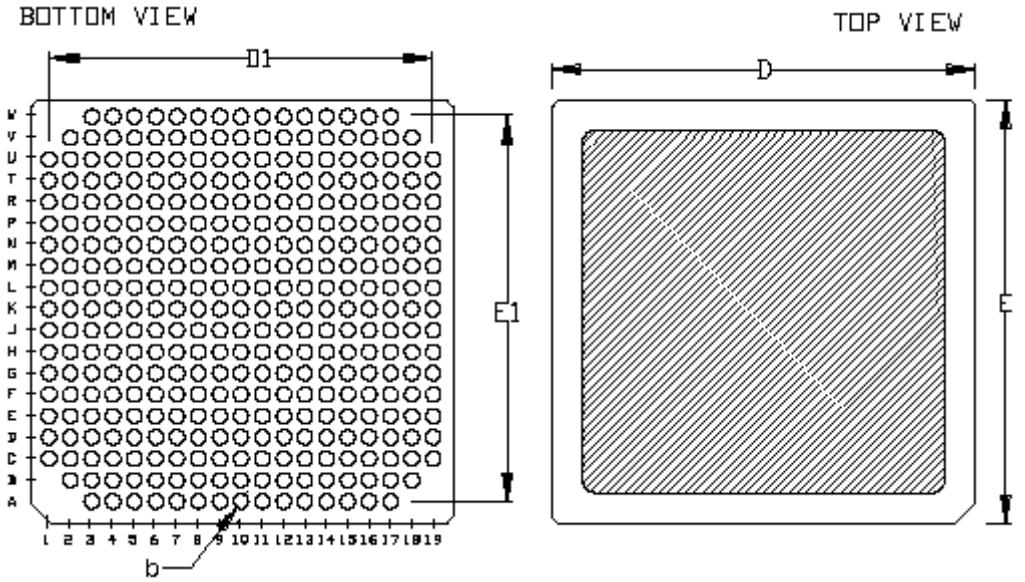
Table 28. SKEW assignements

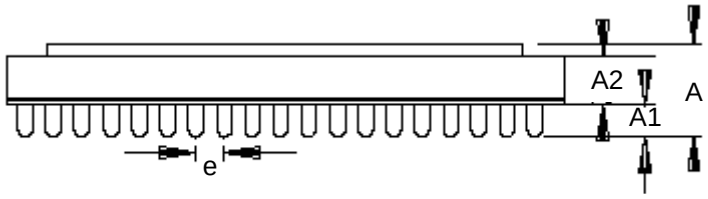
SKEW[1:0]	DELAY		Comments
	CLK1 -> CLK2	CLK1 -> CLK3	
‘00’	natural	natural	natural skew
‘01’	D1	D3	medium skew
‘10’	D1 + D2	D3 + D4	maximum skew
‘11’	Reserved		

Use of a high level of skew improves the efficiency of SET prevention but leads to an operating loss performance. Maximum speed is decreased and timings on the interfaces are slower than with natural skew. Refer to the ’Electrical Characteristics’ section for detailed timings at each skew.

Package MCGA 349

Mechanical Outlines

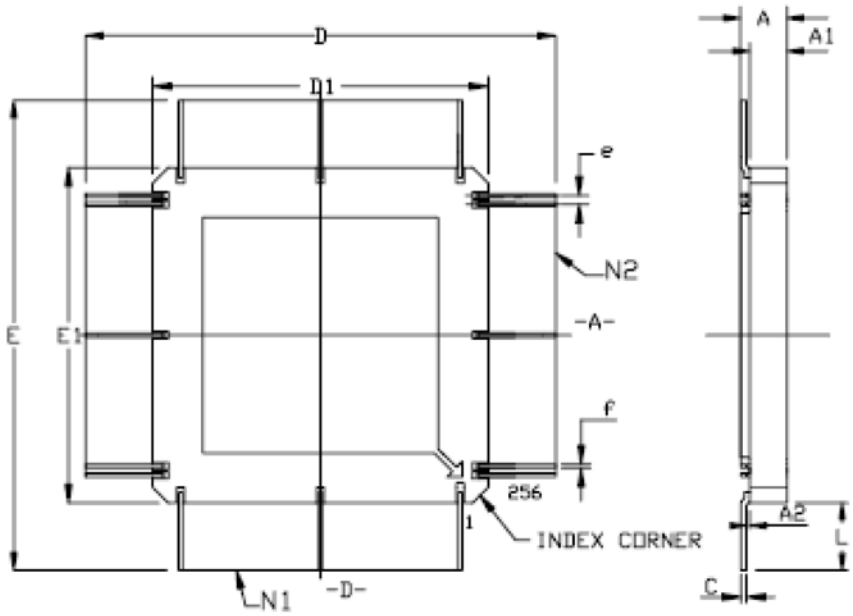




	mm		inch	
	min	max	min	max
D/E	24,8	25,2	0,976	0,992
D1/E1	22,86		0,9	
A1	1,4	1,85	0,055	0,073
A2	2,4	3,45	0,094	0,136
A	4,3	5,9	0,169	0,232
b	0,79	0,99	0,031	0,04
e	1,27		0,05	

QFP256 package

Package Description



	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	

Registers
Description

Table 29. 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 access				w=write acces										r/w=read and write access								
default value after reset	0	100			1	x = undefined or non affected by reset																	

Integer Unit
Registers

Table 30. 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[3:0]				ver[3:0]				n	z	v	c	*reserved*						ec	ef	pil[3:0]				s	ps	et	cwp[4:0]				
r				r				r/w				r/w						r	r	r/w				r/w			r/w				
0001				0001				x	x	x	x	xxxxxx						0	x	xxxx				1	1	1	00000				

Bit Number	Mnemonic	Description
31..28	impl[3:0]	Implementation or class of implementations of the architecture.
27..24	ver[3:0]	Identify one or more particular implementations or is a readable and writable state field whose properties are implementation-dependent.
23	n	indicates whether the ALU result was negative for the last instruction modifying *icc* field. 1 = negative 0 = not negative.
22	z	indicates whether the ALU result was zero for the last instruction modifying *icc* field. 1 = zero 0 = not zero.
21	v	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	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, 0 = no carry.
13	ec	determines whether the implementation-dependent oprocessor is enabled. If disabled, a coprocessor instruction will trap. 1 = enabled, 0 = disabled. If an implementation does not support a coprocessor in ardware, PSR.EC should always read as 0 and writes to it should be ignored.
12	ef	determines whether the FPU is enabled. If disabled, a floating-point instruction will trap. 1 = enabled, 0 = disabled. If an implementation does not support a hardware FPU, PSR.EF should always read as 0 and writes to it should be ignored.
11..8	pil[3:0]	identify the interrupt level above which the processor will accept an interrupt.
7	s	determines whether the processor is in supervisor or user mode. 1 = supervisor mode, 0 = user mode.
6	ps	contains the value of the S bit at the time of the most recent trap.
5	et	determines whether traps are enabled. A trap automatically resets ET to 0. When ET=0, an interrupt request is ignored and an exception trap causes the IU to halt execution, which typically results in a reset trap that resumes execution at address 0. 1 = traps enabled, 0 = traps disabled.

Bit Number	Mnemonic	Description
4..0	cwp[4:0]	comprise the current window pointer, a counter that identifies the current window into the r registers. The hardware decrements the CWP on traps and SAVE instructions, and increments it on RESTORE and RETT instructions (modulo NWINDOWS).

Table 31. Window Invalid Mask - 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																								windows							
																								7	6	5	4	3	2	1	0
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	1	0	0	0	0	0	0	1

Bit Number	Mnemonic	Description
0 < n < 7	windows[n]	Indicated wether the window is a ‘valid’ or an ‘invalid’ one. ‘0’ : valid ‘1’ : invalid

The WIM can be read by the privileged RDWIM instruction and written by the WRWIM instruction.

Table 32. Y 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																															

The Y register can be read and written with the RDY and WRY instructions.

Table 33. Trap Base Address - 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[19:0]																				tt[7:0]								*reserved*			
r/w																				r								r/w			
																												0000			

Bit Number	Mnemonic	Description
31..12	tba[19:0]	Trap Base Address This field contains the most-significant 20 bits of the trap table address.
11..4	tt[7:0]	Trap Type This eight-bit field is written by the hardware when a trap occurs, and retains its value until the next trap. It provides an offset into the trap table.

The tba field is written by the WRTBR instruction. Use of WRTBR is don’t care for tt field.

Table 34. Program Counters - 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
address[31:0]																															
0x0000 0000																															

The 32-bit PC contains the address of the instruction currently being executed by the IU.

When a trap occurs, the PC address is saved in the local register (l1). When returning from trap, l1 value is copied back to PC.

Table 35. New 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
address[31:0]																															
0x0000 0004																															

The nPC holds the address of the next instruction to be executed (assuming a trap does not occur).

When a trap occurs, the nPC address is saved in the local register (l2). When returning from trap, l2 value is copied back to nPC.

Table 36. Watch Point Address Registers
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[29:0]																														*reserved*	if
r/w																														r	r/w
xxxx xxxx																														0	0

BitNumber	Mnemonic	Description
31..2	waddr[20:0]	Defines the addresses range to be watched
0	if	Enable hit generation on instruction fetch 0 = disabled 1 = enabled

These registers are accessed using the RDASR/WRASR instructions

Table 37. Watch Point Mask registers
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[29:0]																														dl	ds
r/w																														r/w	r/w
xxxx xxxx																														0	0

BitNumber	Mnemonic	Description
31..2	wmask[29:0]	Defines which bits are to be compared to waddr. ‘0’ = comparison disabled ‘1’ = comparison enabled
1	dl	Enable hit generation on data load ‘0’ = disabled ‘1’ = enabled
0	ds	Enable hit generation on data store ‘0’ = disabled ‘1’ = enabled

These registers are accessed using the RDASR/WRASR instructions

Table 38. Register File Protection Control Register
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[2:0]			tcb[6:0]							te	di
r																				r/w			r/w							r/w	r/w
xxxx x																				000			x							0	1

Bit Number	Mnemonic	Description
11..9	cnt[2:0]	Error counter. Incremented for each corrected error
8..2	tcb[6:0]	Test checkbits
1	te	EDAC test enable ‘0’ = disabled ‘1’ = enabled
0	di	Disable EDAC function ‘1’ = disabled ‘0’ = enabled

This register is accessed using the RDASR/WRASR instructions.

Table 39. Window Registers

Type	Name	Definition
in	i7	return address
	i6	frame pointer
	i5	incoming parameter register 5
	i4	incoming parameter register 4
	i3	incoming parameter register 3
	i2	incoming parameter register 2
	i1	incoming parameter register 1
	i0	incoming parameter register 0

Table 39. Window Registers

Type	Name	Definition
local	l7	local register 7
	l6	local register 6
	l5	local register 5
	l4	local register 4
	l3	local register 3
	l2	nPC (for RETT)
	l1	PC (for RETT)
	l0	local register 0
out	o7	temp
	o6	stack pointer
	o5	outgoing parameter register 5
	o4	outgoing parameter register 4
	o3	outgoing parameter register 3
	o2	outgoing parameter register 2
	o1	outgoing parameter register 1
	o0	outgoing parameter register 0
global	g7	global register 7
	g6	global register 6
	g5	global register 5
	g4	global register 4
	g3	global register 3
	g2	global register 2
	g1	global register 1
	g0	global register 0 - always 0x00000000

Floating Point Unit Registers

Table 40. 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[1:0]		*reserved*		tem[4:0]					ns	reserved		ver[2:0]			ftt[2:0]			reserved		fcc[1:0]		aexc[4:0]					cexc[4:0]				
				nvm	ofm	ufm	dzm	nxm														nva	ofa	ufa	dza	nxa	nvc	ofc	ufc	dzc	nxc
r/w		r/w		r/w					r	r/w		r			r			r/w		r		r					r				
xx		xx		00000					x	xx		001			xxx			xx		xxx		xxxxx					00000				

Bit Number	Mnemonic	Description
31..30	rd[1:0]	Rounding Direction Defines the rounding direction used by the AT697 FPU during a floating-point arithmetic operation.

Bit Number	Mnemonic	Description
27..23	tem[4:0]	Trap Enable Mask *tem* field enables traps caused by FPops. These bits are ANDed with the bits of the *cexc* (current exception field) to determine whether to force a floating-point exception to IU. All trap enable fields correspond to the similarly named bit in the *cexc* field. 0 = trap disabled 1 = trap enabled
22	ns	Causes the FPU to produce implementation-defined results that may not correspond to ANSI/IEEE Standard 754-1985. For instance, to obtain higher performance, implementations may convert a subnormal floatingpoint operand or result to zero when NS is set.
19..17	ver[2:0]	Identify one or more particular implementations of the FPU architecture. For each SPARC IU implementation there may be one or more FPU implementations, or none. This field identifies the particular FPU implementation present.
16..14	ftt[2:0]	Floating point trap type Identify floating-point exception trap types.when floating point exception occurs, the *ftt* field encodes the type of floating-point exception until an STFSR or another FPop is executed.
11..10	fcc[1:0]	Contain the FPU condition codes. These bits are updated by floating-point compare instructions (FCMP and FCMPE). They are read and written by the STFSR and LDFSR instructions, respectively. FBfcc bases its control transfer on this field.
9..5	aexc[4:0]	Accumulate IEEE floating-point exceptions while fp_exception traps are disabled using the TEM field. After an FPop completes, the TEM and *cexc* fields are logically *and*ed together. If the result is nonzero, an fp_exception trap is generated; otherwise, the new *cexc* field is *or*'d into the *aexc* field. Thus, while traps are masked, exceptions are accumulated in the *aexc* field.
4..0	cexc[4:0]	Indicate that one or more IEEE floating-point exceptions were generated by the most recently executed FPop instruction. The absence of an exception causes the corresponding bit to be cleared.

Trap Types The *ftt* field can be read by the STFSR instruction. An LDFSR instruction does not affect *ftt field*.

Table 41. Trap Type Definition

TT	Name	Description
0	none	No trap.
1	IEEE_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	Unfinished_FPop	An unfinished_FPop indicates that an implementation's FPU was unable to generate correct results or exceptions
3	unimplemented_FPop	An unimplemented_FPop indicates that an implementation's FPU decoded an FPop that it does not implement. In this case, the *cexc* field is unchanged
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 floatingpoint 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	hardware error	A hardware_error indicates that the FPU detected a catastrophic internal error, such as an illegal state or a parity error on an *f* register access. If a hardware_error occurs during execution of user code, it may not be possible to recover sufficient state to continue execution of the user application.
6	invalid register	An invalid_fp_register trap type indicates that one (or more) operands of an FPop are misaligned, that is, a double-precision register number is not 0 mod 2, or a quadruple-precision register number is not 0 mod 4. It is recommended that implementations generate an fp_exception trap with FSR.*ftt* = invalid_fp_register in this case, but an implementation may choose not to generate a trap.

Floating Point Condition Code

Table 42. 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 generates an IEEE_exception trap.

Floating Point Exception Fields

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

Table 43. 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.
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_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 *dzc* 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 44. f registers - fx (0 < x < 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
fx[31:0]																															

Memory Interface Registers

Table 45. 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[1:0]	iobrdy	bexc	*reserved*	iows[3:0]	iop	*reserved*	prwen	*reserved*	prwdh[1:0]	prwws[3:0]	prrws[3:0]
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
x	0	0	xx	0	0	x	xxx	0	xxx xxxx	0	x	xx	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 (driven low).
29	abrdy	Asynchronous bus ready If set, the BRDY* input can be asserted without relation to the sysstem 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[1:0]	I/O bus width. Defines the data with of the I/O area (“00”=8, “10”=32).
26	iobrdy	IO area bus ready enable if set to one, an IO access will be extended until BRDY* is asserted (Driven low)
25	bexc	Bus error enable for RAM, PROM and IO transactions. If set to one, the (low) assertion of the BEXC* will generate an error response on the internal bus and causes a trap (0x01, 0x09, 0x2B) depending on the type of access.
23:20	iows[3:0]	I/O waitstates. Defines the number of waitstates during I/O accesses: “0000” = 0 waitstate, “0001” = 1 waitstates, ..., “1111” = 15 waitstates.
19	iop	I/O protection. ‘0’ : Read and write accesses to I/O area are disabled ‘1’ : Read and write accesses to I/O area are enabled.
11	prwen	Prom write enable. If set, enables write cycles to the prom area.
9:8	prwdh[1:0]	Prom width. Defines the data with of the prom area: “00” = 8 bits, “10” = 32 bits.
7..4	prwws[3:0]	Prom write waitstates. Defines the number of waitstates during prom write cycles: “0000” = 0 waitstate, “0001” = 2 waitstates, ..., “1111” = 30 waitstates.
3..0	prrws[3:0]	Prom read waitstates. Defines the number of waitstates during prom read cycles “0000” = 0 waitstate, “0001” = 2 waitstates, ..., “1111” = 30waitstates.

Note: The prom bank size is coded in the same way as the ram bank size in MCFG2. The prom bank size is used when an 8-bit prom is used with EDAC enabled - the last 25% of the prom bank is used to store the EDAC checksums and cannot be used to store instructions or data.

During power-up, the prom width (bits [9:8]) are set with value on PIO[1:0] inputs. The prom waitstates fields are set to 15 (maximum) and prom bank size is set to “0000”. External bus error and bus ready are disabled. All other fields are undefined.

Table 46. 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[2:0]			sdrcas	sdrbs[2:0]			sdrcls[1:0]		sdrcmd[1:0]		*reserved*				se	si	rambs[3:0]				*reserved*	rambrdy	rmw	ramwdh[1:0]		ramwws[1:0]		ramrws[1:0]	
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		r/w	
0	1	111			1	000			10		00		xxxx				0	0	xxxx				x	x	x	xx		xx		xx	

Bit Number	Mnemonic	Description
31	sdrref	SDRAM refresh. If set, the SDRAM refresh will be enabled.
30	trp	SDRAM tRP timing. tRP will be equal to 2 or 3 system clocks (0/1).
29..27	trfc[2:0]	SDRAM tRFC timing. tRF will be equal to 3 + field-value system clocks.
26	sdrcas	SDRAM CAS delay. Selects 2 or 3 cycle CAS delay (0/1). When changed, a LOAD-COMMAND-REGISTER command must be issued at the same time. Also sets RAS/CAS delay (tRCD).
25..23	sdrbs[2:0]	SDRAM banks size. Defines the banks size for SDRAM chip selects: “000” = 4 Mbyte, “001” = 8 Mbyte, “010” = 16 Mbyte, ..., “111”=512 Mbyte.
22..21	sdrcls[1:0]	SDRAM column size. “00” = 256 when *sdrbs* = “111”, “01” = 512 when *sdrbs* = “111”, “10” = 1024 when *sdrbs* = “111”, “11” = 4096 when *sdrbs* = “111”, = 2048 otherwise.
20..19	sdrcmd[1:0]	SDRAM command. Writing a non-zero value will generate 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 will be enabled.
13	si	SRAM disable. If set together with bit 14 (SDRAM enable), the static ram access will be disabled.

Bit Number	Mnemonic	Description
12..9	rambs[3:0]	SRAM bank size. Defines the size of each ram bank “0000” = 8 Kbyte, “0001” = 16 Kbyte, ... “1111” = 256 Mbyte.
7	rambrdy	RAM area bus ready enable. For RAM Bank 4 (RAMSN[4]). If set to one, a RAM access will be extended until BRDY* is asserted (driven low).
6	rmw	Read-modify-write. if set, Enable read-modify-write cycles on sub-word writes to 32-bit areas with common write strobe (no byte write strobe).
5..4	ramwdh[1:0]	SRAM bus width. Defines the data with of the SRAMarea: “00” = 8 bits, “1X” = 32 bits.
3..2	ramwws[1:0]	SRAM write waitstates. Defines the number of waitstates during SRAM write cycles: “00” = 0 waitstate, “01” = 1 waitstates, “10” = 2 waitstates, “11” = 3 waitstates.
1..0	ramrws[1:0]	SRAM read waitstates. Defines the number of waitstates during SRAM read cycles: “00” = 0 waitstate, “01” = 1 waitstates, “10” = 2 waitstates, “11” = 3 waitstates.

Table 47. 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
reserved					*srcrv*[14:0]															wb	*rb*	re	pe	tcb[6:0]							
r					r/w															r/w	r/w	r/w	r/w	r/w							
11101					xxx xxxx xxxx xxxx															0	0	x	x	xxxx xxxx							

Bit Number	Mnemonic	Description
26..12	srcrv[14:0]	SDRAM refresh counter reload value.
11	wb	EDAC diagnostic write bypass
10	rb	EDAC diagnostic read bypass
9	re	RAM EDAC enable. Enable EDAC checking of the RAM area
8	pe	PROM EDAC enable. Enable EDAC checking of the PROM area. At reset, this bit is initialised with the value of PIO[2]
7..0	tcb[6:0]	Test checkbits. This field replaces the normal checkbits during store cycles when WB is set. TCB is also loaded with the memory checkbits during load cycles when RB is set.

The period between each AUTO-REFRESH command is calculated as follows:

tREFRESH = ((reload value) + 1) / SDCLK frequency

Table 48. 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[14:0]															mask[14:0]														
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, enables the write protect unit
30	bp	Block protect If set, selects block protect mode
29..15	tag[14:0]	Address tag This field is compared against address(29:15)
14..0	mask[14:0]	Address mask This field contains the address mask

Table 49. 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[14:0]															mask[14:0]														
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, enables the write protect unit
30	bp	Block protect If set, selects block protect mode
29..15	tag[14:0]	Address tag This field is compared against address(29:15)
14..0	mask[14:0]	Address mask this field contains the address mask

Table 50. 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		START1[27:0]																												bp	*reserved*
r/w		r/w																												r/w	r/w
00		xxxxxxx																												x	0

Bit Number	Mnemonic	Description
29..2	START1[27:0]	Contains the first address of the protected block
1	bp	Block protect If set, selects block protect mode

Table 51. 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		END1[27:0]																												us	su
r/w		r/w																												r/w	r/w
00		xxxxxxx																												0	0

Bit Number	Mnemonic	Description
29..2	END1[27:0]	Contains the last address of the protected block
1	us	User mode If set, write protection is enabled for user mode accesses
0	su	Supervisor mode If set, write protection is enabled for supervisor mode accesses

Table 52. 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		START2[27:0]																												bp	*reserved*
r/w		r/w																												r/w	r/w
00		xxxxxxx																												x	0

Bit Number	Mnemonic	Description
29..2	START2[27:0]	Contains the first address of the protected block
1	bp	Block protect If set, selects block protect mode

Table 53. 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		END2[27:0]																												us	su
r/w		r/w																												r/w	r/w
00		xxxxxxx																												0	0

Bit Number	Mnemonic	Description
29..2	END2[27:0]	Contains the last address of the protected block
1	us	User mode If set, write protection is enabled for user mode accesses
0	su	Supervisor mode If set, write protection is enabled for supervisor mode acce[illegible]

System Registers

Table 54. 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
reserved	dsu	sdrctrl	wtpnb[2:0]			imac	nwindows[4:0]					icsz[2:0]			ilsz[2:0]		dcsz[2:0]			dlsz[1:0]		divinst	mulinst	*reserved*	memstat	fpu[1:0]		pci[1:0]		wprt[1:0]	
r/w	r	r	r			r	r					r			r		r			r		r	r	r/w	r	r		r		r	
x	1	1	100			0	00111					100			11		100			11		1	1	x	1	01		01		01	

Bit Number	Mnemonic	Description
30	dsu	Debug Support Unit present “0” = disabled “1” = present
29	sdrctrl	SDRAM controller present “0” = disabled “1” = present
28..26	wtpnt[2:0]	Number of implemented watchpoints (0 - 4)
25	imac	UMAC/SMAC instruction implemented
24..20	nwindows[4:0]	Number of register windows. The implemented number of SPARC register windows - 1
19..17	icsz[2:0]	Instruction cache size. The size (in Kbytes) of the instruction cache. Cache size = 2^(icsz).
16..15	ilsz[2:0]	Instruction cache line size. The line size (in 32-bit words) of each line. Line size = 2^(ilsz).
14..12	dcsz[2:0]	Data cache size. The size (in kbytes) of the data cache. Cache size = 2^(dcsz).
11..10	dlsz[1:0]	Data cache line size. The line size (in 32-bit words) of each line. Line size = 2^(dlsz).
9	divinst	UDIV/SDIV instruction implemented
8	mulinst	UMUL/SMUL instruction implemented
6	memstat	Memory status and failing address register present
5..4	fpu[1:0]	FPU type
3..2	pci[1:0]	PCI core type
1..0	wprt[1:0]	Write protection type

Table 55. 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
fa[31:0]																															
r																															
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx																															

Bit Number	Mnemonic	Description
31..0	fa[31:0]	Failing Address This field contains the address of the access that triggered an error response. The register is updated each time an error occurs on the internal bus.

Table 56. 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																						ee	ev	rw	hmaster[3:0]				hsize[2:0]		
r/w																						r/w	r/w	r	r				r		
xxxxxxxxxxxxxxx																						0	0	0	0	0	0	0	0	0	0

Bit Number	Mnemonic	Description
9	ee	EDAC Correctable error Set when a correctable EDAC error is detected.
8	ev	Error Valid Set when a new error occurred.
7	rw	Read/Write This bit is set if the failed access was a read cycle, otherwise it is cleared.
6..3	hmaster[3:0]	AHB master. This field contains the HMASTER[3:0] of the failed access.
2..0	hsize[2:0]	Transfer Size. This filed contains the HSIZE[2:0] of the failed transfer.

Note: Any access triggering an error response on the AHB bus will be registered in two registers; AHB failing address register and AHB status register.The failing address register will store the address of the access while the AHB status register will store the access and error type.The registers are updated when an error occur, and the EV (error valid) is not set. When the EV bit is set, interrupt 1 is generated to inform the processor about the error. After an error, the EV bit has to be reset by software.

Caches Register

Table 57. 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
reserved								ds	fd	fi	cpc[1:0]		cpte[1:0]		ib	ip	dp	ite[1:0]		ide[1:0]		dte[1:0]		dde[1:0]		df	if	dcs[1:0]		ics[1:0]	
r								r/w	r/w	r/w	r		r/w		r/w	r	r	r/w		r/w		r/w		r/w		r/w	r/w	r/w		r/w	
11110111								0	0	0	10		xx		0	x	x	00		00		00		00		x	x	00		00	

Bit Number	Mnemonic	Description
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[1:0]	Cache parity bits Indicates how many parity bits are used to protect the caches “00” = none, “01” = 1 parity bits, “10” = 2 parity bits, “11” = not used
18..17	cpte[1:0]	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 when an instruction cache flush operation is in progress.
14	dp	Data cache flush pending This bit is set when an data cache flush operation is in progress.
13..12	ite[1:0]	Instruction cache tag error counter This filed is incremented every time an instruction cache tag parity error is detected.
11.10	ide[1:0]	Instruction cache data error counter This field is incremented each time an instruction cache data sub-block parity error is detected.
9..8	dte[1:0]	Data cache tag error counter This filed is incremented every time a data cache tag parity error is detected.
7..6	dde[1:0]	Data cache data error counter This field is incremented 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[1:0]	Data Cache state Define the current data cache according to the following : “X0” = disabled “01” = frozen “11” = enabled Set to “00” at reset.

Bit Number	Mnemonic	Description
1..0	ics[1:0]	Instruction Cache state Define the current instruction cache according to the following : “X0” = disabled “01” = frozen “11” = enabled. Set to “00” at reset.

Power Down Reg.

Table 58. 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[31:0]																															
w																															
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx																															

Bit Number	Mnemonic	Description
31..0	idle[31:0]	Write only with any data. A write to this register followed by a load access will cause the system to enter power down mode

Timers Registers

Table 59. 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
tim1val[31:0]																															
r/w																															
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx																															

Bit Number	Mnemonic	Description
31..0	tim1val[31:0]	Timer 1 counter value A read access gives the decounting value of the scaler.

Table 60. 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
tim1rld																															
r/w																															
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX																															

Bit Number	Mnemonic	Description
31..0	tim1rld[31:0]	*Timer 1 reload value* A write access programs the reload value of Timer 1 counter.

Table 61. 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																													ld1	rl1	en1
r/w																													r/w		
xxxx xxxx xxxx xxxx xxxx xxxx xxxx x																													0	x	0

Bit Number	Mnemonic	Description
2	ld1	Load counter when written with ‘one’, will load the timer reload register into the timer counter register. Always reads as a ‘zero’.
1	rl1	Reload counter If rl1 is set, then the counter will automatically be reloaded with the reload value after each underflow.
0	en1	Enable counter enables the timer when set.

Table 62. 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
wdc[31:0]																															
r/w																															
1111 1111 1111 1111 1111 1111 1111 1111																															

Bit Number	Mnemonic	Description
31..0	wdc[31:0]	*Watchdog counter.* Fixes the watchdog 'Timeout'.

The 'Timeout' is the time between the loading (or re-loading) and the watchdog interrupt. 'Reset-Timeout' is greater than 'Timeout'.

Note: A read access gives the decounting value of the watchdog, the reload value itself is not stored in the processor.

Table 63. 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
tim2val[31:0]																															
r/w																															
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx																															

Bit Number	Mnemonic	Description
31..0	tim2val[31:0]	Timer 2 counter value A read access gives the decounting value of the scaler.

Table 64. 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
tim2rld[31:0]																															
r/w																															
XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX																															

Bit Number	Mnemonic	Description
31..0	tim2rld[31:0]	*Timer 2 reload value* A write access programs the reload value of Timer 1 counter.

Table 65. 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																													ld2	rl2	en2
r/w																													r/w		
xxxx xxxx xxxx xxxx xxxx xxxx xxxx x																													0	x	0

Bit Number	Mnemonic	Description
2	ld2	Load counter when written with ‘one’, will load the timer reload register into the timer counter register. Always reads as a ‘zero’.
1	rl2	Reload counter If rl2 is set, then the counter will automatically be reloaded with the reload value after each underflow.
0	en2	Enable counter enables the timer when set.

Table 66. 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																						counter value [9:0]									
r/w																						r/w									
xxxx xxxx xxxx xxxx xxx																						00 0000 0000									

Bit Number	Mnemonic	Description
9..0	counter value[9:0]	reserved counter value

A read access gives the decounting value of the prescaler.

Table 67. 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																						reload value [9:0]									
r/w																						r/w									
xxxx xxxx xxxx xxxx xxxx xxx																						00 0000 0000									

Bit Number	Mnemonic	Description
9..0	reload value [9:0]	Prescaler reload value

A write access programs the reload value of the prescaler.

A read access gives the reload value of the prescaler.

Note: The reset value for SCAR is 0. This is not a legal value, it is however equivalent to a value of 3 and leads to a division rate of 4.

UARTs Registers

Table 68. 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																								rtd1[7:0]							
r/w																								r/w							
xxxx xxxx xxxx xxxx xxxx xxxx																								xxxx xxxx							

Bit Number	Mnemonic	Description
7..0	rtd1[7:0]	Received or Transmitted Data of UART1

rtd1 field has 2 meanings:

- A write access enables the sending of the written 8-bit data on UART 1.
- A read access provides the received 8-bit data on UART1.

Table 69. 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																									fe1	pe1	ov1	br1	th1	ts1	dr1
r/w																									r/w						
xxxx xxxx xxxx xxxx xxxx xxxx x																									0	0	0	0	1	1	0

Bit Number	Mnemonic	Description
6	fe1	Framing error Indicates that a framing error was detected.
5	pe1	Parity error indicates that a parity error was detected.
4	ov1	Overrun Indicates that one or more character have been lost due to overrun.
3	br1	Break received Indicates that a BREAK has been received.
2	th1	Transmitter hold register empty Indicates that the transmitter hold register is empty.
1	ts1	Transmitter shift register empty Indicates that the transmitter shift register is empty.
0	dr 1	Data ready Indicates that new data is available in the receiver holding register.

Table 70. 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																							ec1	lb1	fl1	pe1	ps1	ti1	ri1	te1	re1
r/w																							r/w								
xxxx xxxx xxxx xxxx xxxx xxx																							0	x	0	x	x	x	x	0	0

Bit Number	Mnemonic	Description
8	ec1	External Clock if set, the UART scaler will be clocked by PIO[3]
7	lb1	Loop back If set, loop back mode will be enabled.
6	fl1	Flow control If set, enables flow control using CTS/RTS.
5	pe1	Parity enable If set, enables parity generation and checking.
4	ps1	Parity select Selects parity polarity ‘0’ = even parity ‘1’ = odd parity
3	ti1	Transmitter interrupt enable If set, enables generation of transmitter interrupt.
2	ri1	Receiver interrupt enable If set, enables generation of receiver interrupt.
1	te1	Transmitter enable If set, enables the transmitter.
0	re1	Receiver enable If set, enables the receiver.

Table 71. 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																				scaler value1 [11:0]											
																				r/w											
xxxx xxx xxxx x x xxxx																				xxxx xxxx xxxx											

Table 72. UART 2 Data Register - UAD2
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																								rtd2 [7:0]							
r/w																								r/w							
xxxx xxxx xxxx xxxx xxxx xxxx																								xxxx xxxx							

Bit Number	Mnemonic	Description
7..0	rtd2[7:0]	Received or Transmitted Data of UART2

rtd1 field has 2 meanings :

A write access enables the sending of the written 8-bit data on UART 2.

A read access provides the received 8-bit data on UART2.

Table 73. 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																									fe2	pe2	ov2	br2	th2	ts2	dr2
r/w																									r/w						
xxxx xxxx xxxx xxxx xxxx xxxx x																									0	0	x	x	1	1	0

Bit Number	Mnemonic	Description
6	fe2	Framing error Indicates that a framing error was detected.
5	pe2	Parity error indicates that a parity error was detected.
4	ov2	Overrun Indicates that one or more character have been lost due to overrun.
3	br2	Break received Indicates that a BREAK has been received.
2	th2	Transmitter hold register empty Indicates that the transmitter hold register is empty.
1	ts2	Transmitter shift register empty Indicates that the transmitter shift register is empty.
0	dr2	Data ready Indicates that new data is available in the receiver holding register.

Table 74. 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																							ec2	lb2	fl2	pe2	ps2	ti2	ri2	te2	re2
r/w																							r/w								
xxxx xxxx xxxx xxxx xxxx xxx																							0	x	0	x	x	x	x	0	0

Bit Number	Mnemonic	Description
8	ec2	External Clock if set, the UART scaler will be clocked by PIO[3]
7	lb2	Loop back If set, loop back mode will be enabled.
6	fl2	Flow control If set, enables flow control using CTS/RTS.
5	pe2	Parity enable If set, enables parity generation and checking.
4	ps2	Parity select Selects parity polarity “0” = even parity “1” = odd parity
3	ti2	Transmitter interrupt enable If set, enables generation of transmitter interrupt.

Bit Number	Mnemonic	Description
2	ri2	Receiver interrupt enable If set, enables generation of receiver interrupt.
1	te2	Transmitter enable If set, enables the transmitter.
0	re2	Receiver enable If set, enables the receiver.

Table 75. 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																				scaler value2 [11:0]											
r/w																				r/w											
xxxx xxxx xxxx xxxx xxxx																				xxxx xxxx xxxx											

Interrupt Registers

Table 76. 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[14:0]															*reserved*	imask[14:0]															*reserved*
I/O7	PCI	I/O6	I/O5	DSU	I/O4	Timer2	Timer1	I/O3	I/O2	I/O1	I/O0	UART1	UART2	AMBA		I/O7	PCI	I/O6	I/O5	DSU	I/O4	Timer2	Timer1	I/O3	I/O2	I/O1	I/O0	UART1	UART2	AMBA	
r/w															r/w	r/w															r/w
xxxx xxxx xxxx xxx															x	000 0000 0000 000															x

Bit Number	Mnemonic	Description
31..17	ilevel[14:0]	Interrupt level indicates whether an interrupt belongs to priority level 1 (ilevel[n]=1) or level 0 (ilevel[n]=0).
15..1	imask[14:0]	Interrupt mask indicates whether an interrupt is masked or enabled '0' = masked '1' = enabled After reset, the interrupt mask register is set to all zeros while the remaining control registers are undefined.

Table 77. 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[14:0]															*reserved*
																I/O7	PCI	I/O6	I/O5	DSU	I/O4	Timer2	Timer1	I/O3	I/O2	I/O1	I/O0	UART1	UART2	AMBA	
r/w																r															r/w
xxxx xxxx xxxx xxxx																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	x

Bit Number	Mnemonic	Description
15..1	ipend[14:0]	Interrupt pending indicates whether an interrupt is pending "1" = interrupt pending "0" = interrupt not pending

When the IU acknowledges the interrupt, the corresponding pending bit is automatically cleared.

Table 78. 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[14:0]															*reserved*
																I/O7	PCI	I/O6	I/O5	DSU	I/O4	Timer2	Timer1	I/O3	I/O2	I/O1	I/O0	UART1	UART2	AMBA	
r/w																r/w															r/w
xxxx xxxx xxxx xxxx																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	x

Bit Number	Mnemonic	Description
15..1	iforce[14:0]	Interrupt force indicates whether an interrupt is being forced '1' = interrupt forced '0' = interrupt not forced

Interrupt can be forced by setting a bit in the interrupt force register. IU acknowledgement will clear the force bit.

Table 79. 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[14:0]															*reserved*
																I/O7	PCI	I/O6	I/O5	DSU	I/O4	Timer2	Timer1	I/O3	I/O2	I/O1	I/O0	UART1	UART2	AMBA	
r/w																r															r/w
xxxx xxxx xxxx xxxx																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	x

Bit Number	Mnemonic	Description
15..1	iclear[14:0]	Interrupt clear If written with a ‘1’, will clear the corresponding bit(s) in the interrupt pending register. Aread return zero.

Table 80. Secondary Interrupt Mask Register - SITM
Address = 0x800000B0

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
simask[31:0]																															
r/w																															
0000 0000 0000 0000 0000 0000 0000 0000																															

Bit Number	Mnemonic	Description
32..0	simask[31:0]	Second interrupt mask indicates whether an interrupt is masked (simask[n] = ‘0’) or enabled (simask[n] = ’1’) After reset, the interrupt mask register is set to all zeros while the remaining control registers are undefined.

.

Table 81. Secondary Interrupt Pending Register - SITP
Address = 0x800000B4

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
sipend[31:0]																															
r/w																															
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx																															

Bit Number	Mnemonic	Description
32..0	sipend[31:0]	Second interrupt pending indicates whether a iterrupt is pending (sipend[n] = ‘1’)

.

Table 82. Secondary Interrupt Status Register - SITS
Address = 0x800000B8

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																										ip	irl[4:0]				
r/w																										r/w	r/w				
xxxx xxxx xxxx xxxx xxxx xxxx xx																										x	x xxxx				

Bit Number	Mnemonic	Description
5	ip	Second interrupt pending If set, then irl[4:0] is valid. If cleared, no unmasked interrupt is pending.
4..0	irl[4..0]	Second request level Indicates the highest unmasked pending interrupt.

Table 83. Secondary Interrupt Clear Register - SITC
Address = 0x800000BC

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		5		3	2	1	0
siclear[31:0]																															
r/w																															
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx																															

Bit Number	Mnemonic	Description
31..0	siclear[31:0]	Second interrupt clear if written with a ‘1’, will clear the corresponding bit(s) in the interrupt pending register.

General Purpose Interface Registers

Table 84. 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[7:0]								lowdat[7:0]								iodata[15:0]															
r/w																r/w															
xxxx xxxx								xxxx xxxx								xxxx xxxx xxxx xxxx															

Bit Number	Mnemonic	Description
15..0	iodata[15:0]	I/O port data
23..16	meddat[7:0]	Meddium Data Corresponding to the data D[15:8]
31..24	lowdat[7:0]	Low Data Corresponding to the data D[7:0]

when read, returns the current value of the I/O port;

when written, value is driven on the I/O port signals (if enabled as Output)

Table 85. 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	iodir[15:0]															
r/w														r/w	r/w	r/w															
xxxx xxxx xxxx xx														00 0000 0000 0000 0000																	

Bit Number	Mnemonic	Description
17	meddir	Defines the direction of D[15..8]
16	lowdir	Defines the direction of D[7..0]
15..0	iodir[15:0]	I/O port direction Defines the direction of I/O ports 15 - 0. ‘1’ = output ‘0’ = input

Table 86. 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[4:0]					en2	le2	pl2	isel2[4:0]					en1	le1	pl1	isel1[4:0]					en0	le0	pl0	isel0[4:0]				
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[4:0]	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[4:0]	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[4:0]	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.
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[4:0]	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 7. 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[4:0]					en6	le6	pl6	isel6[4:0]					en5	le5	pl5	isel5[4:0]					en4	le4	pl4	isel4[4:0]				
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[4:0]	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[4:0]	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).
12..8	isel5[4:0]	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[4:0]	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

Table 88. 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 [15:0]																vendor id [15:0]															
r																r															
0x1202																0x1438															

Bit Number	Mnemonic	Description
31..16	device id [15:0]	This field identifies the particular device. This identifier is allocated by the vendor.
15..0	vendor id [15:0]	This field identifies the manufacturer of the device. Valid vendor identifiers are allocated by the PCI SIG to ensure uniqueness. 0FFFFh is an invalid value for Vendor ID.

Table 89. 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[1:0]		stat8	stat7	stat6	stat5	stat4	stat3	*reserved*								com10	com9	com8	com7	com6	com5	com4	com3	com2	com1	com0
rr	rr	rr	rr	rr	r		rr	r	r	r	r	r	r								r/w	r/w	r/w	r	r/w	r	r/w	r	r/w	r/w	r/w
0	0	0	0	0	01		0	1	0	0	0	0	0x0000 0000								0	0	0	0	0	0	0	0	0	0	0

Note: 1. rr = Read and Reset by writing 1

Bit Number	Mnemonic	Description
31	stat15	Parity error detected. This bit must be set by the device whenever it detects a parity error, even if parity error handling is disabled (as controlled by bit 6 in the Command register).
30	stat14	SERR asserted. This bit must be set whenever the device asserts **SERR***. Devices who will never assert **SERR*** do not need to implement this bit.
29	stat13	Master has terminated master abort. This bit must be set by a master device whenever its transaction except for Special Cycle) is terminated with Master-Abort. All master devices must implement this bit.
28	stat12	Master has terminated target abort This bit must be set by a master device whenever its transaction is terminated with Target-Abort. All master devices must implement this bit.
27	stat11	Target signal target abort. This bit must be set by a target device whenever it terminates a transaction with Target-Abort. Devices that will never signal Target-Abort do not need to implement this bit.
26..25	stat10_9[1:0]	Devsel timing. These bits encode the timing of **DEVSEL***. Three allowable timings for assertion of **DEVSEL*** are specified. These are encoded as 00 for fast, 01 for medium, and 10 for slow (11b is reserved). These bits are read-only and must indicate the slowest time that a device asserts **DEVSEL*** for any bus command except Configuration Read and Configuration Write.

Bit Number	Mnemonic	Description
24	stat8	Master received/asserted PERR This bit is only implemented by bus masters. It is set when three conditions are met: 1) the bus agent asserted **PERR*** itself (on a read) or observed **PERR*** asserted (on a write); 2) the agent setting the bit acted as the bus master for the operation in which the error occurred; 3) the Parity Error Response bit (Command register) is set.
23	stat7	Target supports fast back2back This optional read-only bit indicates whether or not the target is capable of accepting fast back-to-back transactions when the transactions are not to the same agent. This bit can be set to 1 if the device can accept these transactions and must be set to 0 otherwise.
22	stat6	User definable features
21	stat5	66 MHz capability This optional read-only bit indicates whether or not this device is capable of running at 66 MHz as defined in Chapter 7. A value of zero indicates 33 MHz. A value of 1 indicates that the device is 66 MHz capable
20	stat4	Power management capability. This optional read-only bit indicates whether or not this device implements the pointer for a New Capabilities linked list at offset 34h. A value of zero indicates that no New Capabilities linked list is available. A value of one indicates that the value read at offset 34h is a pointer in Configuration Space to a linked list of new capabilities.
10	com10	Interrupt command. This bit disables the device/function from asserting **INTx***. A value of 0 enables the assertion of its **INTx*** signal. A value of 1 disables the assertion of its **INTx*** signal. This bit's state after **RST*** is 0.
9	com9	Master can generate fast back2back. This optional read/write bit controls whether or not a master can do fast back-to-back transactions to different devices. Initialization software will set the bit if all targets are fast back-to-back capable. A value of 1 means the master is allowed to generate fast back-to-back transactions to different agents. A value of 0 means fast back-to-back transactions are only allowed to the same agent. This bit's state after **RST*** is 0.
8	com8	Enable SERR driver -This bit is an enable bit for the **SERR*** driver. A value of 0 disables the **SERR*** driver. A value of 1 enables the **SERR*** driver. This bit's state after **RST*** is 0. All devices that have an **SERR*** pin must implement this bit. Address parity errors are reported only if this bit and bit 6 are 1.
7	com7	Address/Data stepping on PCI bus
6	com6	Enable Parity Check This bit controls the device's response to parity errors. When the bitis set, the device must take its normal action when a parity error is detected. When the bit is 0, the device sets its Detected Parity Error status bit (bit 15 in the Status register) when an error is detected, but does not assert **PERR*** and continues normal operation. This bit's state after **RST*** is 0. Devices that check parity must implement this bit. Devices are still required to generate parity even if parity checking is disabled.
5	com5	VGA palette snooping This bit controls how VGA compatible and graphics devices handle accesses to VGA palette registers. When this bit is 1, palette snooping is enabled (i.e., the device does not respond to palette register writes and snoops the data). When the bit is 0, the device should treat palette write accesses like all other accesses. VGA compatible devices should implement this bit.
4	com4	Enable memory write and invalidate. This is an enable bit for using the Memory Write and Invalidate command. When this bit is 1, masters may generate the command. When it is 0, Memory Write must be used instead. State after **RST*** is 0. This bit must be implemented by master devices that can generate the Memory Write and Invalidate command.
3	com3	Enable special cycles Controls a device's action on Special Cycle operations. A value of 0 causes the device to ignore all Special Cycle operations. A value of 1 allows the device to monitor Special Cycle operations. State after **RST*** is 0.
2	com2	Enable PCI master Controls a device's ability to act as a master on the PCI bus. A value of 0 disables the device from generating PCI accesses. A value of 1 allows the device to behave as a bus master. State after **RST*** is 0.
1	com1	Enable target memory command response Controls a device's response to Memory Space accesses. A value of 0 disables the device response. A value of 1 allows the device to respond to Memory Space accesses. State after **RST*** is 0.

Bit Number	Mnemonic	Description
0	com1	Enable target IO command response Controls a device's response to I/O Space accesses. A value of 0 disables the device response. A value of 1 allows the device to respond to I/O Space accesses. State after **RST*** is 0.

Table 90. 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 [23:0]																								revision id [7:0]							
r																								r							
0x0B4000																								0x10							

Bit Number	Mnemonic	Description
31..8	class code [23:0]	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.
7..0	revision id [7:0]	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 91. Bist, Header type, Latency, Cache line 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[7:0]								header type [7:0]								latency timer [7:0]								cache line size [7:0]							
r/w								r/w								r/w								r/w							
0x00								0x00								0x00								0x00							

Bit Number	Mnemonic	Description
31..24	bist[7:0]	bist7 : Return 1 if device supports BIST. Return 0 if the device is not BIST capable. bist6 : Write a 1 to invoke BIST. Device resets the bit when BIST is complete. Software should fail the device if BIST is not complete after 2 seconds. bist[3..0] : A value of 0 means the device has passed its test. Non-zero values mean the device failed. Device-specific failure codes can be encoded in the non-zero value.
23..16	header type [7:0]	header 7 : multi-function device “0” : device is single function “1” : device is multi-function header[6..0] : header second part layout
15..8	latency timer [7:0]	this field specifies the value for latency timer in PCI bus clock unit
7..0	cache line size [7:0]	Specifies the cache line size

Table 92. 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
MEMBAR1 [27:0]																												pref1	type1[1:0]		msi1
r/w																													r		r
0x000 0000																												1	00		0

Bit Number	Mnemonic	Description
31..4	MEMBAR1[27:0]	Memory base address.
3	pref1	Prefetchable indicates there are no side effects on reads. The device returns all bytes on reads regardless of the byte enables.
2..1	type1[1:0]	“00” Base register is 32 bits wide and mapping can be done anywhere in the 32-bit Memory Space. “10” Base register is 64 bits wide and can be mapped anywhere in the 64-bit address space. “11” & “01” Reserved
0	msi1	“0” : Indicates that base address mapes Memory Space

Table 93. 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
MEMBAR2[27:0]																												pref2	type2		msi2
r/w																													r		r
0x000 0000																												1	00		0

Bit Number	Mnemonic	Description
31..4	MEMBAR2[27:0]	Memory base address.
3	pref2	Prefetchable indicates there are no side effects on reads. The device returns all bytes on reads regardless of the byte enables.
2..1	type2	“00” Base register is 32 bits wide and mapping can be done anywhere in the 32-bit Memory Space. “10” Base register is 64 bits wide and can be mapped anywhere in the 64-bit address space. “11” & “01” Reserved
0	msi2	“0” : Indicates that base address mapes memory space

Table 94. 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
IOBAR[29:0]																														*reserved*	msi
r/w																														r	r
0x0 000 0000																														0	1

Bit Number	Mnemonic	Description
31..2	IOBAR[29:0]	Memory base address.
0	msi	“1” : Indicates that base address mapes I/O Space

Table 95. 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
subsystem id [15:0]																svi[15:0]															
r																r															
0x0001																0x1438															

Bit Number	Mnemonic	Description
31..16	sid[15:0]	subsystem id
15..0	svi[15:0]	subsystem vendor id

Table 96. PCI Capabilities Pointer Register - PCICP
Address = 0x80000134

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																								pointer[7:0]							
r																								r							
0x0000 00																								0xDC							

Bit Number	Mnemonic	Description
7..0	pointer[7:0]	index for the extended capabilities registers

Table 97. 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[7:0]								min_gnt[7:0]								int_pin[7:0]								int_line[7:0]							
r/w								r/w								r								r/w							
0								0								0								0							

Bit Number	Mnemonic	Description
31..24	maxlat[7:0]	maxlat field specifies how often the processor need to gain access to the PCI bus. Units are 0.25 micosecond
23..16	mingnt[7:0]	min_gnt identifies the length of burst period, assuming a 33MHz clock. Units are 0.25 micosecond
15..8	intpin[7:0]	indicates which interrupt pin the processor uses - Always 0 due to absence of PCI interrupt management.
7..0	intlin[7:0]	Indicates the interrupt line register to which the core is connected to. - Always 0 due to absence of PCI interrupt management.

Table 98. PCI Retry _trdy - PCIRT
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[7:0]								trdy[7:0]							
																r/w								r/w							
0																0x80								0x80							

Bit Number	Mnemonic	Description
15..8	retry[7:0]	Indicates the number of retry the core will performe when configured as master
7..0	trdy[7:0]	Indicates the number of PCI clock the processor configured as master will wait for TRDY

Table 99. PCI Configuration Write Register - PCICW
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[3:0]			
r/w																												r/w			
0x0000 00																												0x0			

Bit Number	Mnemonic	Description
3..0	ben[3:0]	Byte enables for writes to the PCI core configuration space '0' = enabled '1' = disable

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 less significant byte (LSB)

Table 100. 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
start address [31:0]																															
r/w																															
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Bit Number	Mnemonic	Description
31..0	start address [31:0]	PCI start address for PCI initiator transactions in APB and DMA mode.

Table 101. PCI Initiator Write Register - PCIIW
Address = 0x8000014C

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[3:0]			
r/w																												r/w			
0x0000 000																												0x0			

Bit Number	Mnemonic	Description
3..0	ben[3:0]	Byte enables for writes to the PCI core configuration space ‘0’ = enabled ‘1’ = disabled

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 less significant byte (LSB)

Table 102. 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																			b2b	com[3:0]				wdcnt[7:0]							
r/w																				r/w				r/w							
0x0000 0																			0	0x0				0x00							

Bit Number	Mnemonic	Description
12	b2b	Use back2back-mode. Can be written to 1, if this transaction is to the same target, as the last one. Note: works only, if the core is enabled for back2back mode.
11..8	com[3:0]	PCI command to be used in DMA mode.
7..0	wdcnt[7:0]	Word count. Minimum number of words for the burst.

Table 103. 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[3:0]				act	xff	xfe	rfe	ss[3:0]			
r/w																			r	r				r	r	r	r	r			
0x0000 0																			x	0x0				0	0	1	1	0x0			

Bit Number	Mnemonic	Description
12	sys	Value of the SYSEN* pin 0 : Host mode 1 : Satellite mode
11..8	dmas[3:0]	DMA state
7	act	PCI core active 1 = active 0 = inactive
6	ff	If set 1, the transmitter fifo is full
5	xfe	If set 1, the transmitter fifo is empty
4	rfe	If set 1, the receiver fifo is empty
3..0	ss[3:0]	Slave status

Table 104. 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
reserved																								commsb[1:0]		*reserved*					mod
r/w																								r/w		r/w					r/w
0x0000 00																								01		00000					0

Bit Number	Mnemonic	Description
7.6	commsb[1:0]	Specifies the two most significant bits of the command used by AHB slave interface. '00' = IO write '01' = memory read/write '10' = configuration read/write '11' = memory read-line/write-invalidate
0	mod	'0' = command source mode '1' = AHB slave - DMA mode '0' = APB mode

Table 105 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[7:0]								*reserved*								tpa2[7:0]								*reserved*							
r/w								r/w								r/w								r/w							
0x40								0x00								0x90								0x00							

Bit Number	Mnemonic	Description
31..24	tpa1[7:0]	Target page address defines the 8 most significant bits of the 16 MByte memory page on which PCI addresses are mapped
15..8	tpa2[7:0]	Target page address defines the 8 most significant bits for the second memory BAR.

Table 106. PCI Target Status-Command 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																							frty	errmem	xff	xfe	rfe	tms[3:0]			
r/w																							r/w	r/w	r/w	r/w	r/w	r/w			
0x0000 00																							0	0	0	1		0000			

Bit Number	Mnemonic	Description
8	frty	Force Retry Set automatically during long delayed read to prevent the read from h[illegible] overw[illegible]en (Debug Purpose only) Cleared by writing a 1.
7	errmem	Reception Fifo parity error ‘0’ = Do not save data with parity error ‘1’ = Data with parity error is saved to memory
6	xff	TXMT Fifo full ‘1’ = force transmistion to abort
5	xfe	TXMT Fifo empty ‘1’ = flushes TXMT Fifo
4	rfe	TRCV Fifo empty ‘1’ = flushes TRCV Fifo
3..0	tms[3:0]	Target AHB master st[illegible] ‘1111’ = reset the state [illegible]

Table 107. PCI Interrupt Enable Register - PCIITE
Address = 0x80000164

31	30	29	28	27	26	25	24	23	[illegible]	21	[illegible]	[illegible]	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																								dmaer	imier	cmfer	imper	tier	tbeer	tper	syser
r/w																								r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
0x0000 00																								0	0	0	0	0	0	0	0

Bit Number	Mnemonic	Description
7	dmaer	DMA end of transfer ‘0’ = disable ‘1’ = enable
6	imier	Initiator error ‘0’ = disable ‘1’ = enable
5	cmfer	PCI core error ‘0’ = disable ‘1’ = enable
4	imper	Initiator Parity error ‘0’ = disable ‘1’ = enable
3	tier	Target error ‘0’ = disable ‘1’ = enable

Bit Number	Mnemonic	Description
2	tbeer	Target byte enable error ‘0’ = disable ‘1’ = enable
1	tper	Target parity error ‘0’ = disable ‘1’ = enable
0	syser	System error asserted on PCI bus ‘0’ = disable ‘1’ = enable

Table 108. PCI Interrupt Pending Register - PCIITP(1)
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																								dmaer	imier	cmfer	imper	tier	tbeer	tper	syser
r/w																								r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
0x0000 00																								0	0	0	0	0	0	0	0

Bit Number	Mnemonic	Description
7	dmaer	DMA end of transfer ‘0’ = not pending ‘1’ = pending
6	imier	Initiator error ‘0’ = not pending ‘1’ = pending
5	cmfer	PCI core error ‘0’ = not pending ‘1’ = pending
4	imper	Initiator Parity error ‘0’ = not pending ‘1’ = pending
3	tier	Target error ‘0’ = not pending ‘1’ = pending
2	tbeer	Target byte enable error ‘0’ = not pending ‘1’ = pending
1	tper	Target parity error ‘0’ = not pending ‘1’ = pending
0	syser	System error asserted on PCI bus ‘0’ = not pending ‘1’ = pending

Note: 1. Bits are cleared when written with a 1. Writing a 0 to the register has no effect.

Table 109. 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																								dmaer	imier	cmfer	imper	tier	tbeer	tper	syser
r/w																								r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
0x0000 00																								0	0	0	0	0	0	0	0

Bit Number	Mnemonic	Description
7	dmaer	DMA end of transfer ‘0’ = not forced ‘1’ = forced
6	imier	Initiator error ‘0’ = not forced ‘1’ = forced
5	cmfer	PCI core error ‘0’ = not forced ‘1’ = forced
4	imper	Initiator Parity error ‘0’ = not forced ‘1’ = forced
3	tier	Target error ‘0’ = not forced ‘1’ = forced
2	tbeer	Target byte enable error ‘0’ = not forced ‘1’ = forced
1	tper	Target parity error ‘0’ = not forced ‘1’ = forced
0	syser	System error asserted on PCI bus ‘0’ = not forced ‘1’ = forced

Table 110. PCI Data Register - PCID
Address = 0x8000017C

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
dat[31:0]																															
r/w																															
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx																															

Bit Number	Mnemonic	Description
31..0	dat[31:0]	data writen/read to/from Fifo

Table 111. PCI Burst End Register - PCIBE
Address = 0x80000174

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
dat[31:0]																															
r/w																															
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx																															

Bit Number	Mnemonic	Description
31..0	dat[31:0]	Last data of a burst in APB mode

Table 112. 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[31:0]																															
r/w																															
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx																															

Bit Number	Mnemonic	Description
31..0	addr[31:0]	Defines the start address of a DMA transaction

Table 113. 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
0x0000 000																												1	1	1	1

Bit Number	Mnemonic	Description
3	p3	Round robin level for agent 3
2	p2	Round robin level for agent 2
1	p1	Round robin level for agent 1
0	p0	Round robin level for agent 0

DSU Registers

Table 114. Trace Buffer Control Register - TBC

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 (AHB index [8:0])									*reserved*			ICNT (Inst Index [8:0])									
r/w					r/w	r/w	r/w				r/w												r/w									
000000					x	x	x	xxx			x xxxx xxxx									xxx			xxxx xxxx								

Bit Number	Mnemonic	Description
26	af	AHB trace buffer freeze If set, the trace buffer will be frozen when the processor enters in debug mode
25	ta	Trace AHB enable
24	ti	Trace instruction enable
20..12	BCNT	AHB trace index counter (AHB Index [8:0])
8..0	ICNT	Instruction trace index counter (Inst Index [8:0])

Table 115. 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[11:0]									re	dr	lr	ss	pe	ee	eb	dm	de	bz	bx	bd	bn	bs	bw	be	ft	bt	dm	te
																	r	r	r												

Bit Number	Mnemonic	Description
31..20	dcnt[11:0]	Trace buffer delay counter
19	re	Reset error mode if set, will clear the error mode in the processor.
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 AHB transfer.
16	ss	Single step If set, the processor will execute one instruction and the 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 external DSUEN signal (read-only)
13	eb	value of the external DSUBRE signal (read-only)
12	dm	Debug mode Indicates when 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.

Bit Number	Mnemonic	Description
10	bz	Break on error traps If set, will force the processor into debug mode on all *except* the following traps: priviledged_instruction, fpu_disabled, window_overflow, window_underflow, asynchronous_interrupt, ticc_trap.
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 to debug mode when an DSU breakpoint is hit.
7	bn	Break now Force processor into debug mode. If cleared, the processor will resume execution.
6	bs	Break on S/W breakpoint If set, debug mode will be forced when an 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).
4	be	Break on error if set, will force the processor to debug mode when the processor would have entered error condition (trap in trap).
3	ft	Freeze timers If set, the scaler in the LEON timer unit will be stopped during debug mode to preserve the time for the software application.
2	bt	Break on trace 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. Enables the trace buffer.

Table 116. 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	*reserved*	th	ts	dr
r/w																									r	r/w	r	r/w	r	r	r
xxxx xxxx xxxx xxxx xxxx xxxx x																									0	x	0	x	0	0	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 117. DSU Trap Register - DTR
Address = 0x9008001C

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																			*em*	*trap type* [7:0]								*reserved*			
																												0000			

Bit Number	Bit Mnemonic	Description
12	em	Error Mode. Set if the trap would have cause the processor to enter error mode
11..4	trap type [7:0]	8-bit SPARC trap type

Table 118. 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
BADD1[29:0]																														*reserved*	*ex1*

Bit Number	Bit mnemonic	Description
31..2	BADD1[29:0]	Breakpoint address
0	ex1	Enables break on executed instruction

Table 119. 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
BMA1[29:0]																														*ld1*	*st1*

Bit Number	Bit mnemonic	Description
31..2	BMA1[29:0]	Breaking Mask
1	ld1	Enables break on AHB load
0	st1	Enables break on AHB write

Table 120. 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
BADD2[29:0]																														*reserved*	*ex2*

Bit Number	Bit mnemonic	Description
31..2	BADD2[29:0]	Breakpoint address
0	ex2	Enables break on executed instruction

Table 121. 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
BMA2[29:0]																														*ld2*	*st2*

Bit Number	Bit mnemonic	Description
31..0	BMA2[29:0]	Breaking Mask
1	ld2	Enables break on A[illegible] load
0	st2	Enables break on AHB write

Table 122. 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	re
r/w																														r	r/w
xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx																														0	x

Bit Number	Mnemonic	Description
1	bl	Baud rate locked Is automatically set when the baud rate is locked.
0	re	Receiver enable If set, enables both the transmitter and receiver.

Table 123. DSU UART Scaler Reload Register - DSUUS

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																		scaler reload value [12:0]													
r/w																		r/w													
xxxx xxxx xxxx xxxx xx																		xx xxxx xxxx xxxx													

Bit Number	Mnemonic	Description
13..0	Scaler reload value [13:0]	Scaler reload Value[1]

Note: 1. The best scaler value for manually programming the baudrate can be calculated as follows:

$$scaler = \frac{\frac{sdclk\ frequency \times 10}{baudrate \times 8} - 5}{10}$$

Electrical Characteristics

Electrical Characteristics for this product have not yet been finalized. Please consider all values listed here as preliminary and non contractual

Absolute Maximum Ratings

Operating Temperature .....-55 °C to +125 °C

Storage Temperature .....-65 °C to +150 °C

Voltage on VDD with respect to Ground .....-0.3 V to + 2.0 V

Voltage on VCC with respect to Ground .....-0.3 V to + 4.0 V

DC current VCC (VDD) and VSS 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

Notes: 1. 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 124. DC characteristics

Symbol	Parameter	Min	Typ	Max	Unit	Test Conditions
VDD	Core Power Supply	1.65	1.8	1.95	V	
VCC	I/O Power Supply Voltage	3	3.3	3.6	V	
IILpu	Low Level Input Pull-up Current	100		500	uA	Vin = VSS
IIHpd	High Level Input Pull-downCurrent	100		500	uA	Vin = VCC (max)
IIL	Low Level Input Leakage Current	-1		1	uA	Vin = VSS
IIH	High Level Input Leakage Current	-1		1	uA	Vin = VCC (max)
IOZ	High Impedance Current	100		500	uA	Vin = VSS or VCC (max)
VIL TTL	Low Level Input Voltage			0.8	V	
VIL CMOS				30%VCC	V	
VIH TTL	High Level Input Voltage	2			V	
VIH CMOS		70%VCC			V	
VOL	Low Level Output Voltage			0.4	V	VCC = VCC(min) IOL = 2, 4, 8, 16mA
VOL pci	Low Level Output Voltage for PCI buffers			0.1 VCC	V	VCC = VCC(min) IOL = 1.5mA
VOH	High Level Output Voltage	VCC - 0.4			V	VCC = VCC(min) IOH = 2, 4, 8, 16mA
VOH pci	High Level Output Voltage for PCI buffers	0.9 VCC			V	VCC = VCC(min) IOH = 0.5mA
ICCSb	Standby Current			5	mA	VCC = VCC(max) no clock active

Power "On/Off" Sequence

The AT697E is based on the Atmel 0.18 µm CMOS process. As VDD (1.8V) and VCC (3.3V) power supplies are electrically isolated, there is no specified sequence in which the power rails may be activated or deactivated.

Power Consumption

The power dissipation is the sum of three basic contributions : P = Pcore + Pio + Ppci

- Pcore represents the contribute due to the internal activity.
- Pio represents the contribute due to the IO pads and output load current, except the PCI bus.
- Ppci represents the contribute due to the PCI pads and output load current.

The following table gives the estimated current consumption for different conditions. The values are coming from estimation and calculation and not from real measurement.

Table 125. Power Dissipation

Mode	Typical conditions			Worst Conditions		
	P Core (1.8V) in W	P I/O (3.3V) in W	P PCI (3.3V) in W	P Core (1.8V) in W	P I/O (3.3V) in W	P PCI (3.3V) in W
Operating (100MHz)	0.6	0.2	0.1	0.8	0.3	0.2

Typical conditions : 25°C, 1.8V core, 3.3V I/O, High I/O and core activity

Worst conditions : 125°C, 1.95 V core, 3.6V I/O; High I/O and core activity

In idle mode (100 MHz external clock), the core power consumption is 0.5W in typical conditions and 0.7W in worst case conditions.

Decoupling capacitance

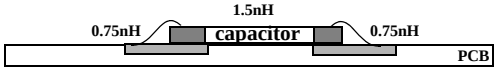
Two main frequencies are involved in the AT697 processor environment :

- 33MHz from the PCI interface
- 100MHz from the master clock of the processor (either from PLL or directly from resonator input)

The following hypothesis is taken for the calculation of the decoupling capacitance :

- 1.5nH is issued from the connection of the capacitor to the PCB
- 1.5nH is issued from the capacitor intrinsic inductance

Figure 50. Capacitor description



This hypothesis corresponds to a capacitor connected to two micro-vias on a PCB.

The filter defined by the self and the decoupling capacitor shall be able to filter the characteristic frequencies of the application. Each frequency to filtre is defined by :

$$fc = \frac{1}{2\pi\sqrt{LC}}$$

- L : the inductance equivalent to the global inductance on the VSS/VDD (VSS/VCC) line.
- C : the decoupling capacitance.

For a processor running at 100MHz with a PCI interface at a characteristic frequency of 33MHz and considering that power supply pins ar grouped by multiple of four, the decoupling capacitance to set are :

- 33nF for 33MHz decoupling
- 3nF for 100MHz decoupling

Capacitance

Parameter	Description	MAX
C_{IN}	Standard Input Capacitance	5pF
C_{IO}	Standard Input/Output Capacitance	5pF
C_{INp}	PCI Input Capacitance	7pF
C_{IOp}	PCI Input/Output Capacitance	7pF

Rating

AC Characteristics

The AT697 processor implements a single event transient 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 skew condition natural and maximum.

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 : 30pF

Table 126. AC Characteristics - Natural Skew

Parameter	Min (ns)	Max (ns)	Comment	Reference edge ('+' for rising edge)
t1	10		CLK Period with PLL disable	
t1_p	40	57	CLK Period with PLL enable	
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	3.5	7	SDCLK output delay - PLL disabled	CLK
t5		1.10^{7}	PLL setup time	
t6	1*t3		Reset Pulse Width	
t10	1.5	7	A[27:0] output delay	SDCLK +
t11	1.5	8	D[31:0] and CB[7:0] output delay	SDCLK +
t12	2	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			D[31:0] and CB[7:0] hold time during write	SDCLK +
t15	1.5	8	OE*, READ and WRITE* output delay	SDCLK +
t16	1	5	ROMS*[1:0] output delay	SDCLK +
t17	1.5	6	RAMS*[4:0], RAMOE*[4:0] and RWE*[3:0] output delay	SDCLK +
t18	1.5	6	IOS* output delay	SDCLK +
t19	5		BRDY* setup time	SDCLK +
t20	0		BRDY* hold time	SDCLK +
t21	2.5	8.5	SDCAS* output delay	SDCLK +
t22	2.5	8.5	SDCS*[1:0], SDRAS*, SDWE* and SDDQM*[3:0] output delay	SDCLK +
t23	6		BEXC* setup time	SDCLK +
t24	0		BEXC* hold time	SDCLK +
t25	2.5	9	PIO[15:0] output delay	SDCLK +
t26	6		PIO[15:0] setup time	SDCLK +

Parameter	Min (ns)	Max (ns)	Comment	Reference edge ('+' for rising edge)
t27	0		PIO[15:0] hold time during load	SDCLK +
t28			PIO[15:0] hold time during write	SDCLK +
t101	30		PCI_CLK Period	
t102	13.5		PCI_CLK Low and High pulse width	
t110	2	12	A/D[31:0] and C/BE[3:0] output delay	PCI_CLK +
t111	7		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	2	11	FRAME*, PAR, PERR*, SERR*, STOP* and DEVSEL* output delay	PCI_CLK +
t114	2	11	IRDY* and TRDY* output delay	PCI_CLK +
t115	2	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	10		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 +

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 : 30pF

Table 127. AC Characteristics - Maximum Skew

Parameter	Min (ns)	Max (ns)	Comment	Reference edge ('+' for rising edge)
t1	11		CLK Period with PLL disable	
t1_p	44	57	CLK Period with PLL enable	
t2	4.95		CLK Low and High pulse width - PLL disabled	
t2_p	20		CLK Low and High pulse width - PLL enabled	
t3	11		SDCLK Period	
t4	3.5	7	SDCLK output delay - PLL disabled	CLK
t5		1.10^{7}	PLL setup time	
t6	1*t3		Reset Pulse Width	
t10	1.5	8	A[27:0] output delay	SDCLK +
t11	1.5	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			D[31:0] and CB[7:0] hold time during write	SDCLK +
t15	1.5	8.5	OE*, READ and WRITE* output delay	SDCLK +
t16	1.5	6.5	ROMS*[1:0] output delay	SDCLK +
t17		7	RAMS*[4:0], RAMOE*[4:0] and RWE*[3:0] output delay	SDCLK +
t18	1.5	6.5	IOS* output delay	SDCLK +
t19	5		BRDY* setup time	SDCLK +
t20	0		BRDY* hold time	SDCLK +
t21	3	9.5	SDCAS* output delay	SDCLK +
t22	3	9.5	SDCS*[1:0], SDRAS*, SDWE* and SDDQM*[3:0] output delay	SDCLK +
t23	6		BEXC* setup time	SDCLK +
t24	0		BEXC* hold time	SDCLK +
t25	2.5	9	PIO[15:0] output delay	SDCLK +
t26	6		PIO[15:0] setup time	SDCLK +
t27	0		PIO[15:0] hold time	SDCLK +
t28			PIO[15:0] hold time during write	SDCLK +
t101	30		PCI_CLK period	
t102	13.5		PCI_CLK low and high pulse width	

Parameter	Min (ns)	Max (ns)	Comment	Reference edge ('+' for rising edge)
t110	2	13	A/D[31:0] and C/BE[3:0] output delay	PCI_CLK +
t111	7		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	2	11	FRAME*, PAR, PERR*, SERR*, STOP* and DEVSEL* output delay	PCI_CLK +
t114	2	11.5	IRDY* and TRDY* output delay	PCI_CLK +
t115	2	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	10		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 +

Timing Derating

Depending on the capacitance load on each pin, the timing figures change. The following figures summarize the timing derating versus the load capacitance.

Figure 51. Timing derating

The timing derating figures will be included in next release

Timing Diagrams - *Will be updated for production release*

Diagram List

- Clock Input without PLL
- Clock Input with PLL
- Reset Sequence

- Fetch, Read and Write from/to 32-bit PROM - 0 Waitstate
- Fetch, Read and Write from/to 32-bit PROM - n Waitstates
- Fetch, Read and Write from/to 32-bit PROM - n Waitstates + BRDY*

- Fetch from 8-bit PROM with EDAC disabled - n Waitstates
- Word Write to 8-bit PROM with EDAC disabled - n Waitstates
- Byte and Half Word Write to 8-bit PROM with EDAC disabled - n Waitstates
- Fetch from 8-bit PROM with EDAC enabled - n Waitstates

- Fetch, Read and Write from/to 32-bit SRAM - 0 Waitstate
- Fetch, Read and Write from/to 32-bit SRAM - n Waitstates

- Burst of RAM Fetches and RAM Write Sequence - 0 Waitstate
- Burst of RAM Fetches and RAM Write Sequence - n Waitstates

- 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 Waitstate
- Fetch from ROM, Read and Write from/to 32-bit I/O - n Waitstates
- Fetch from ROM, Read and Write from/to 32-bit I/O - n Waitstates + BRDY*

Figure 52. Clock Input without PLL (***PRELIMINARY***)

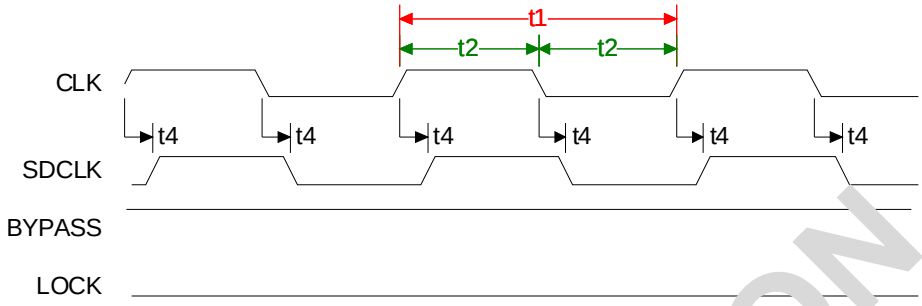


Figure 53. Clock Input with PLL (***PRELIMINARY***)

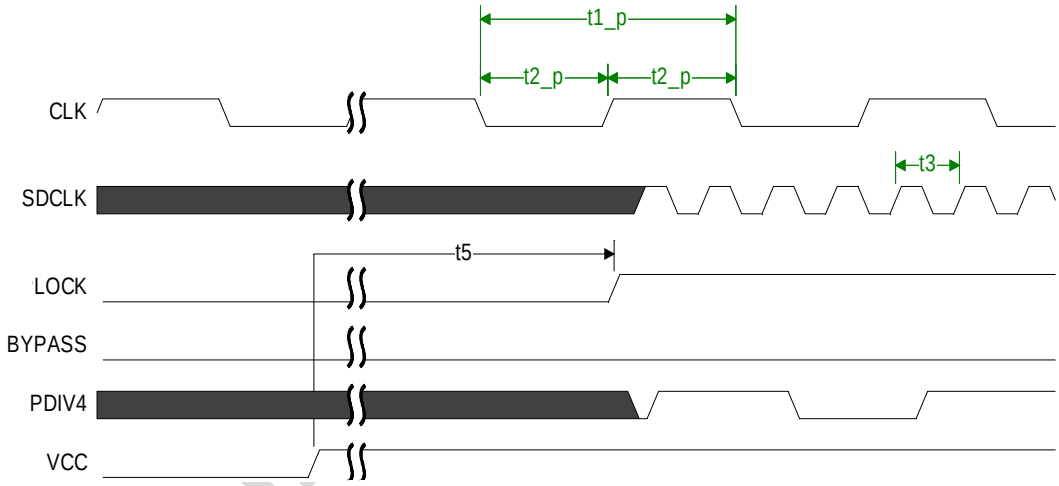


Figure 54. Reset Sequence(***PRELIMINARY***)

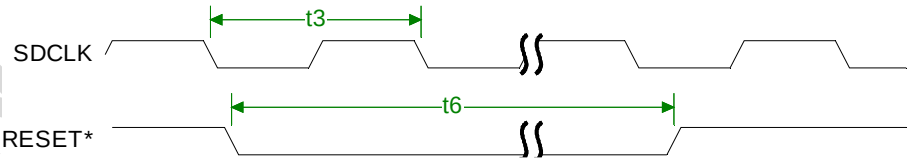


Figure 55. Fetch, Read and Write from 32-bit PROM - 0 Waitstate(***PRELIMINARY***)

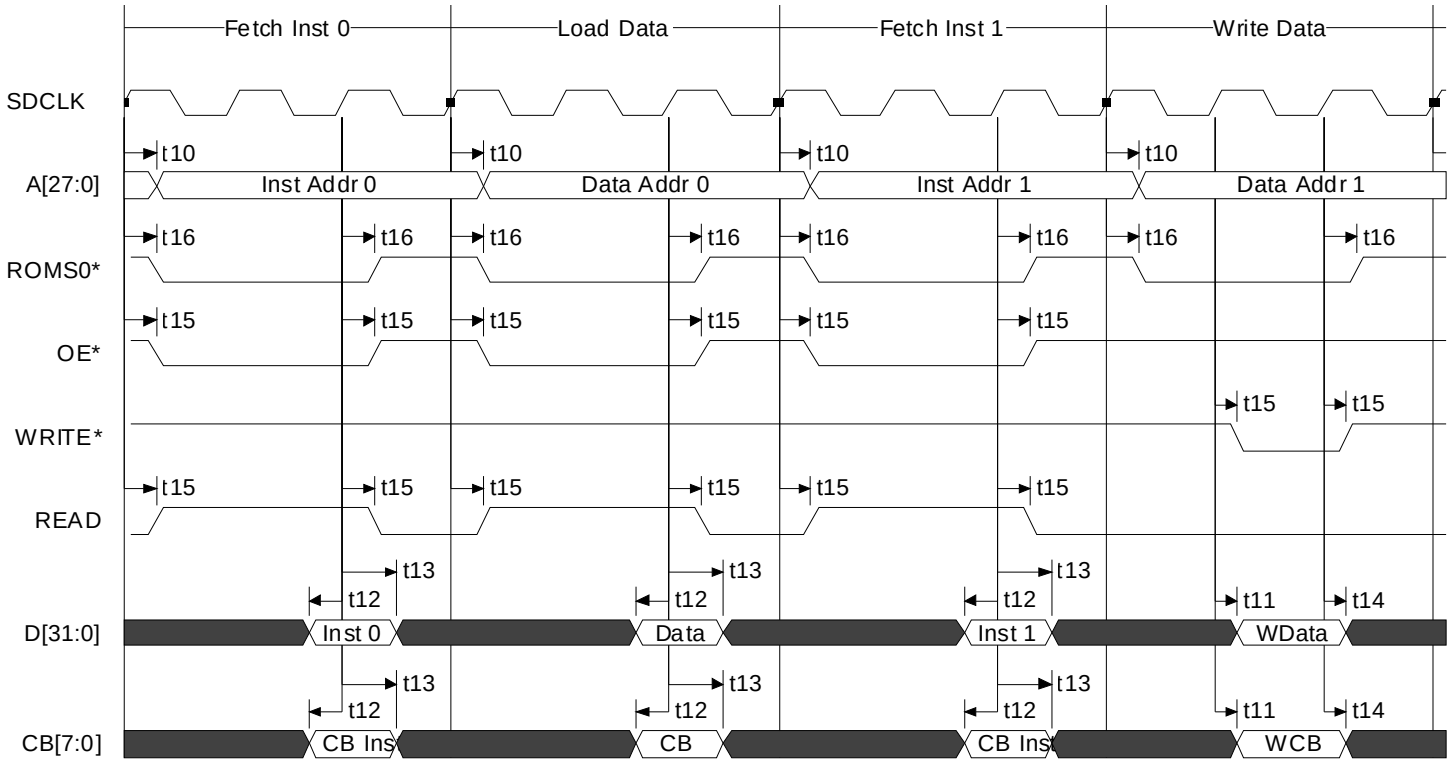


Figure 56. Fetch, Read and Write from 32-bit PROM - n Waitstates(***PRELIMINARY***)

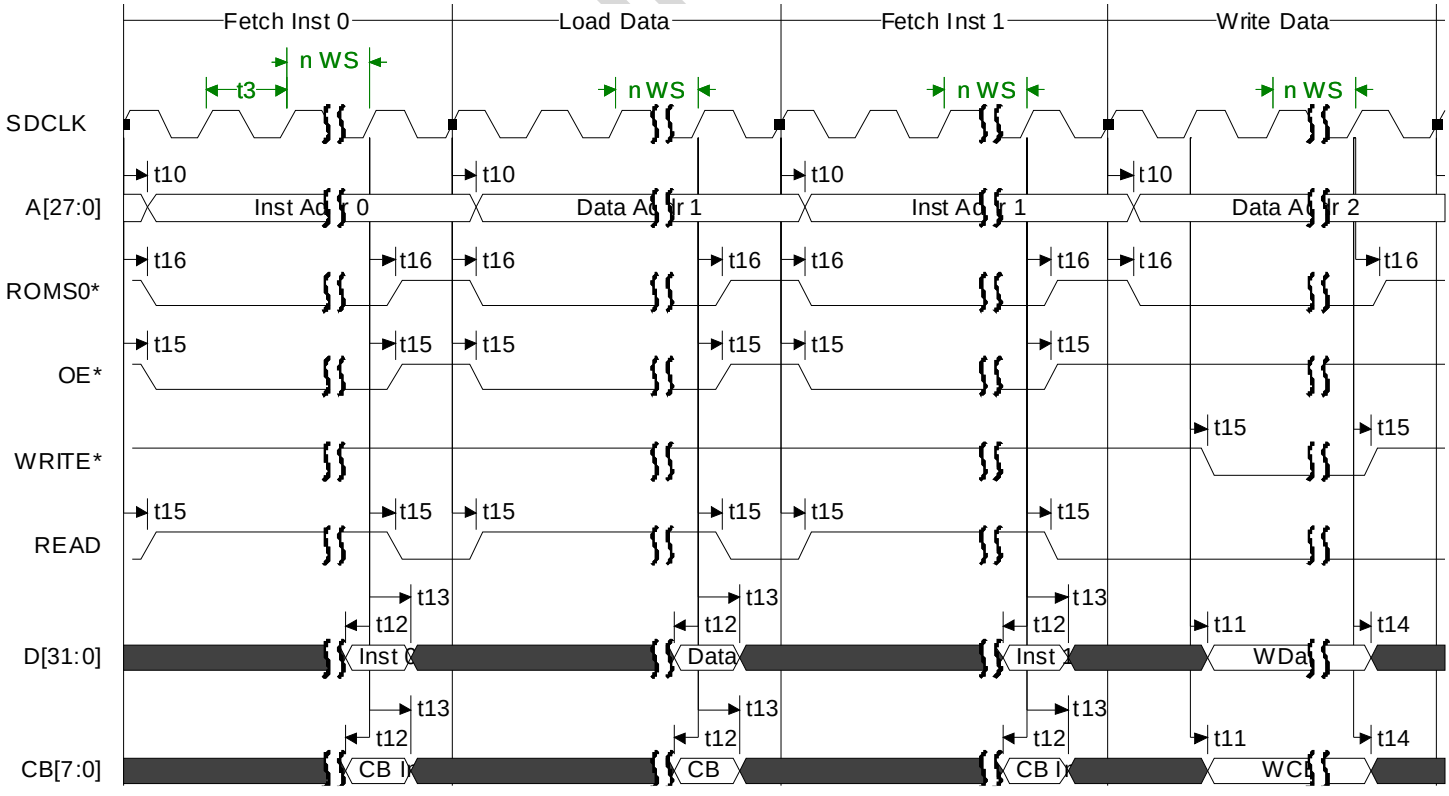


Figure 57. Fetch, Read and Write from 32-bit PROM - n Waitstates + BRDY*(***PRELIMINARY***)

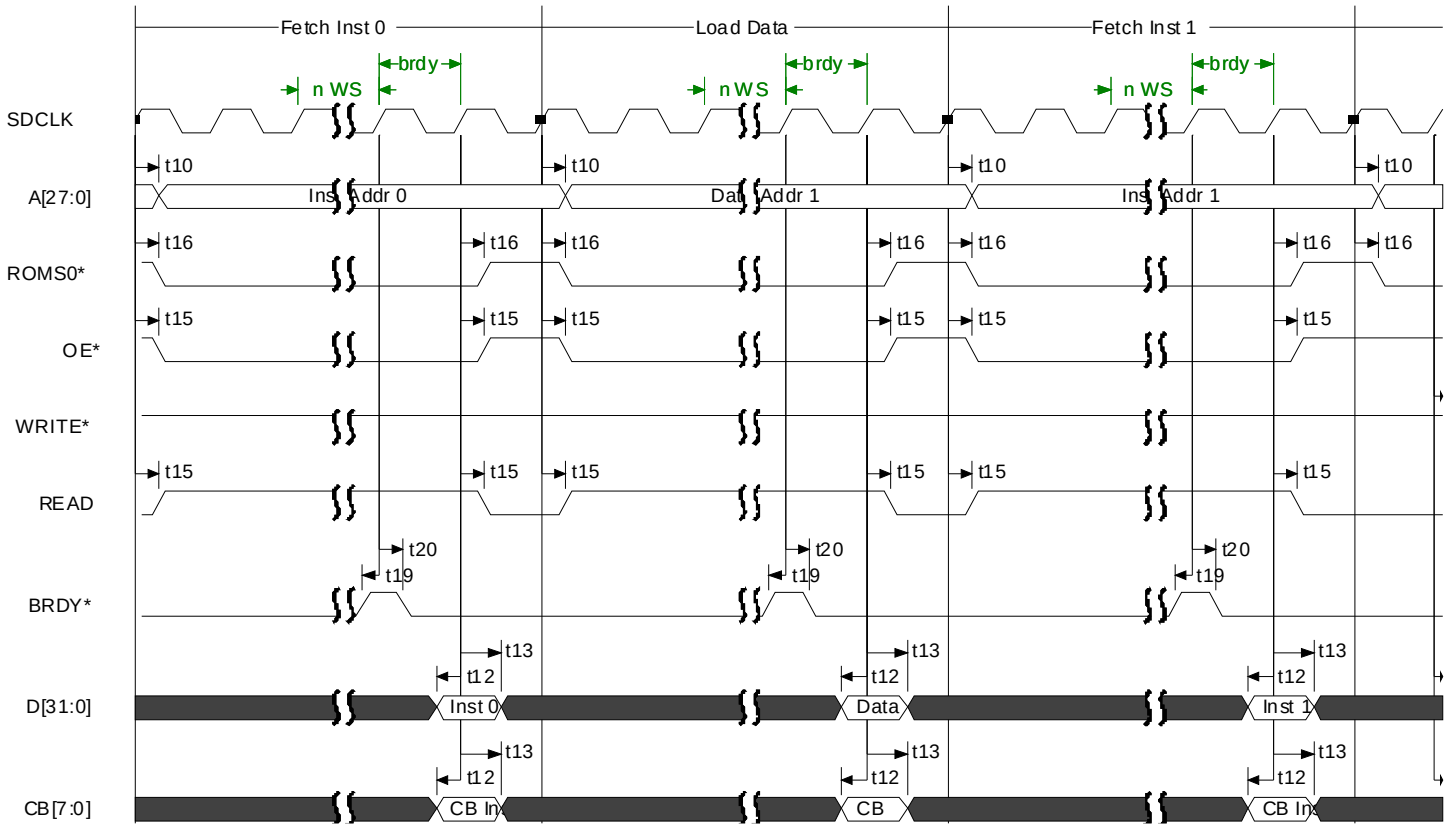


Figure 58. Fetch from 8-bit PROM with EDAC disabled - n Waitstates(***PRELIMINARY***)

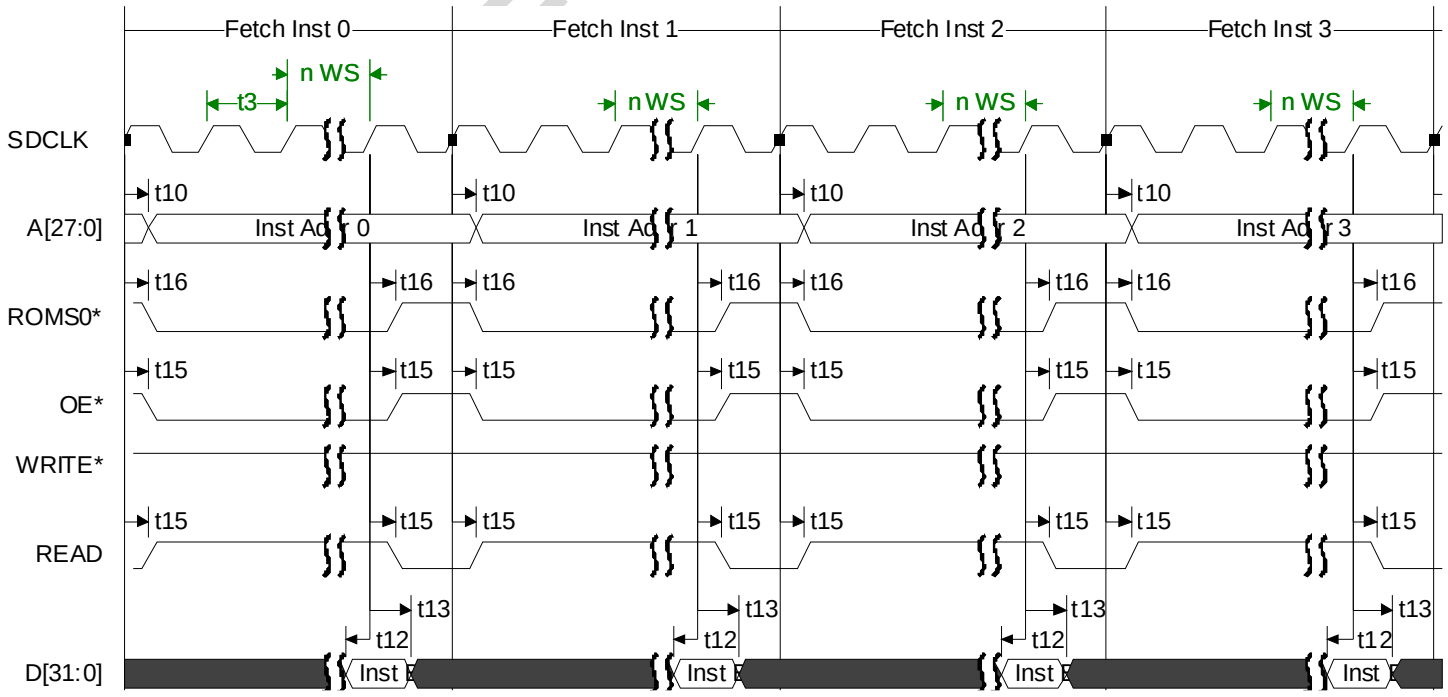


Figure 59. Word Write to 8-bit PROM with EDAC disabled - n Waitstates (***PRELIMINARY***)

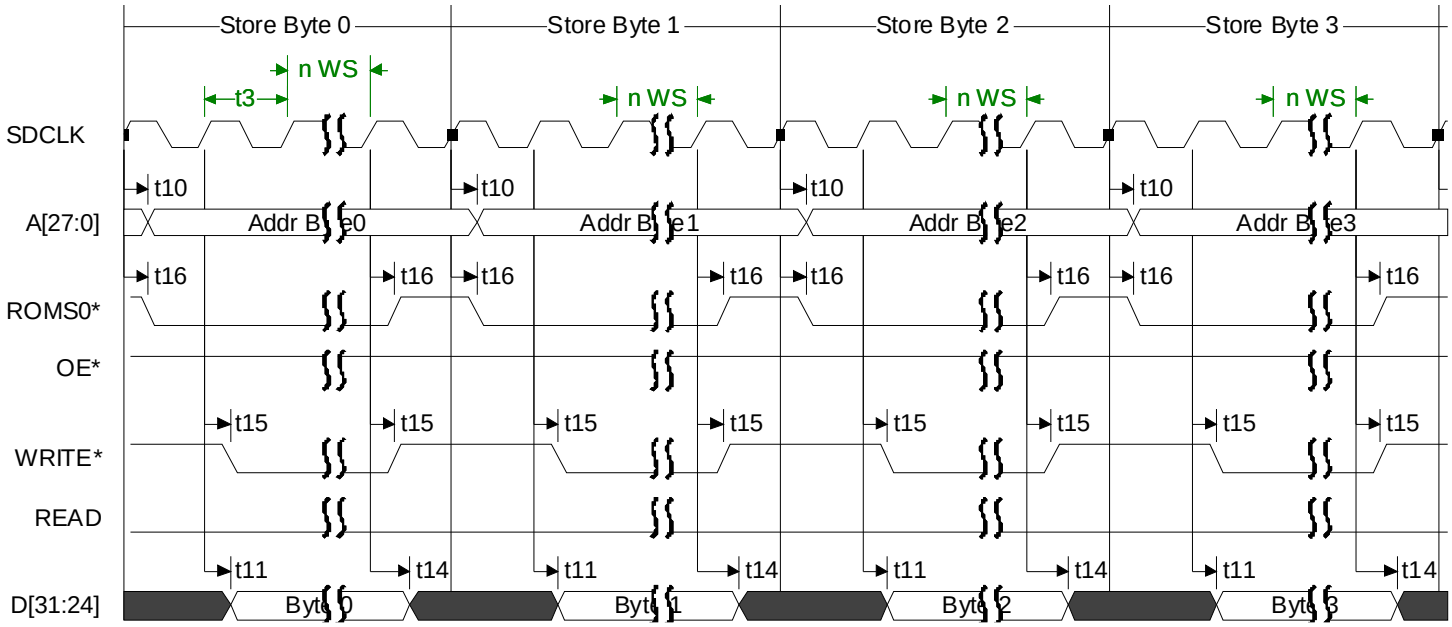


Figure 60. Byte and Half Word Write to 8-bit PROM with EDAC disabled - n Waitstates (***PRELIMINARY***)

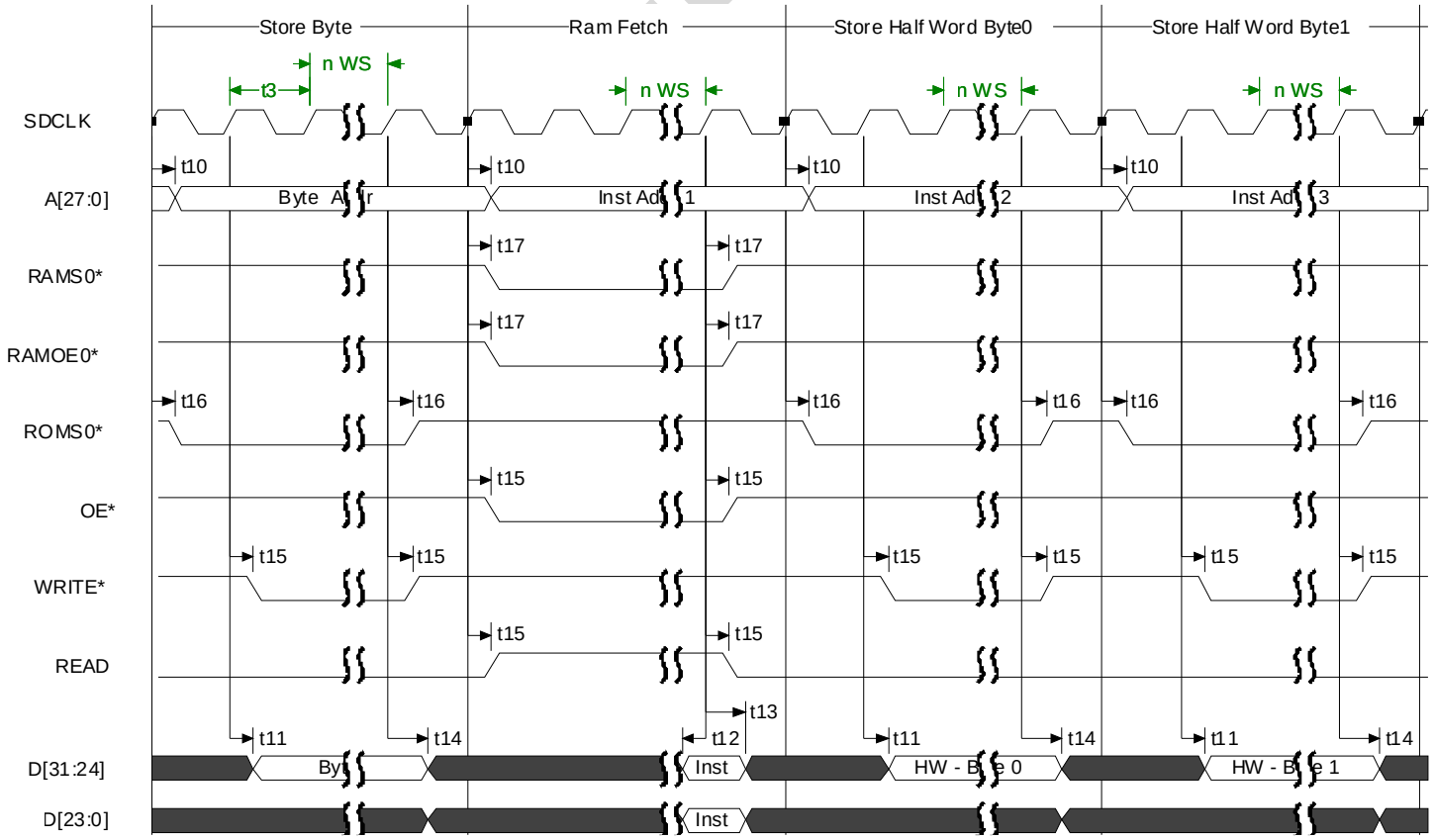


Figure 61. Fetch from 8-bit PROM with EDAC enabled - n Waitstates(***PRELIMINARY***)

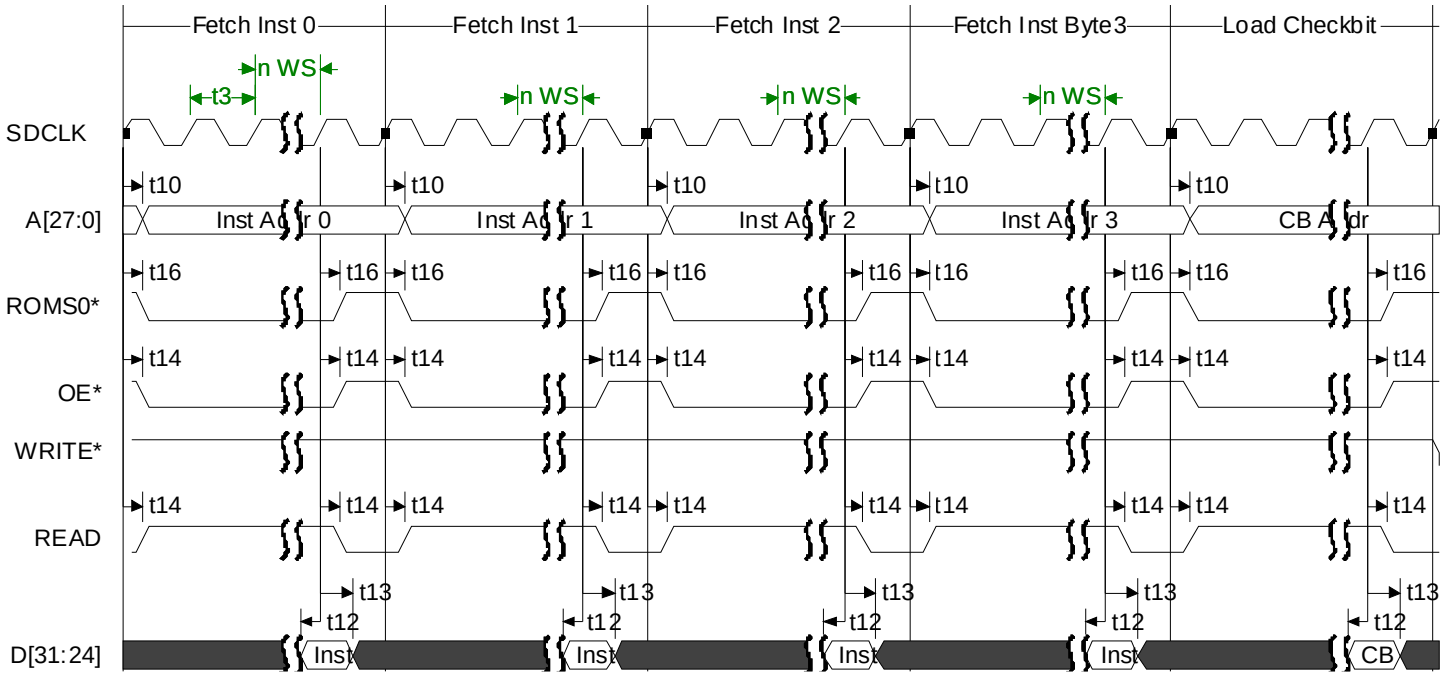


Figure 62. Fetch, Read and Write from/to 32-bit SRAM - 0 Waitstate(***PRELIMINARY***)

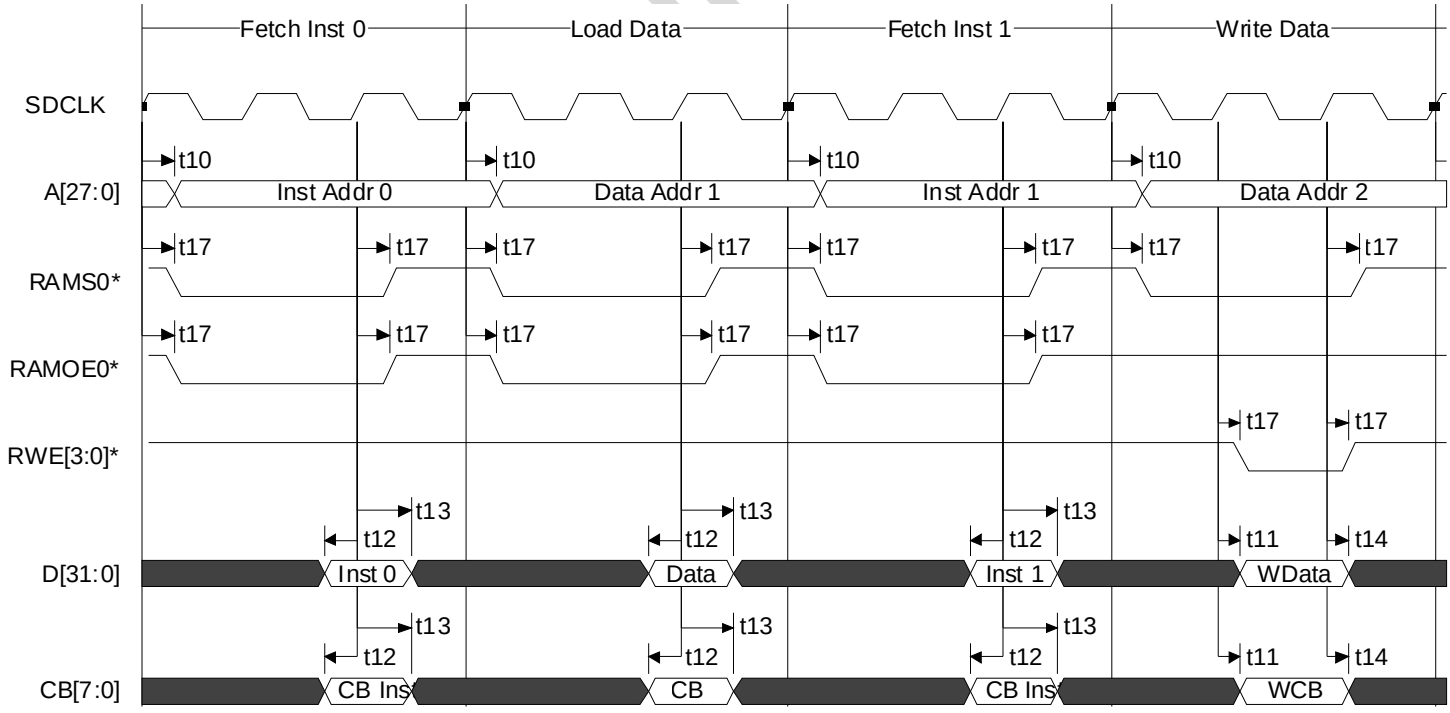


Figure 63. Fetch, Read and Write from/to 32-bit SRAM - n Waitstates(***PRELIMINARY***)

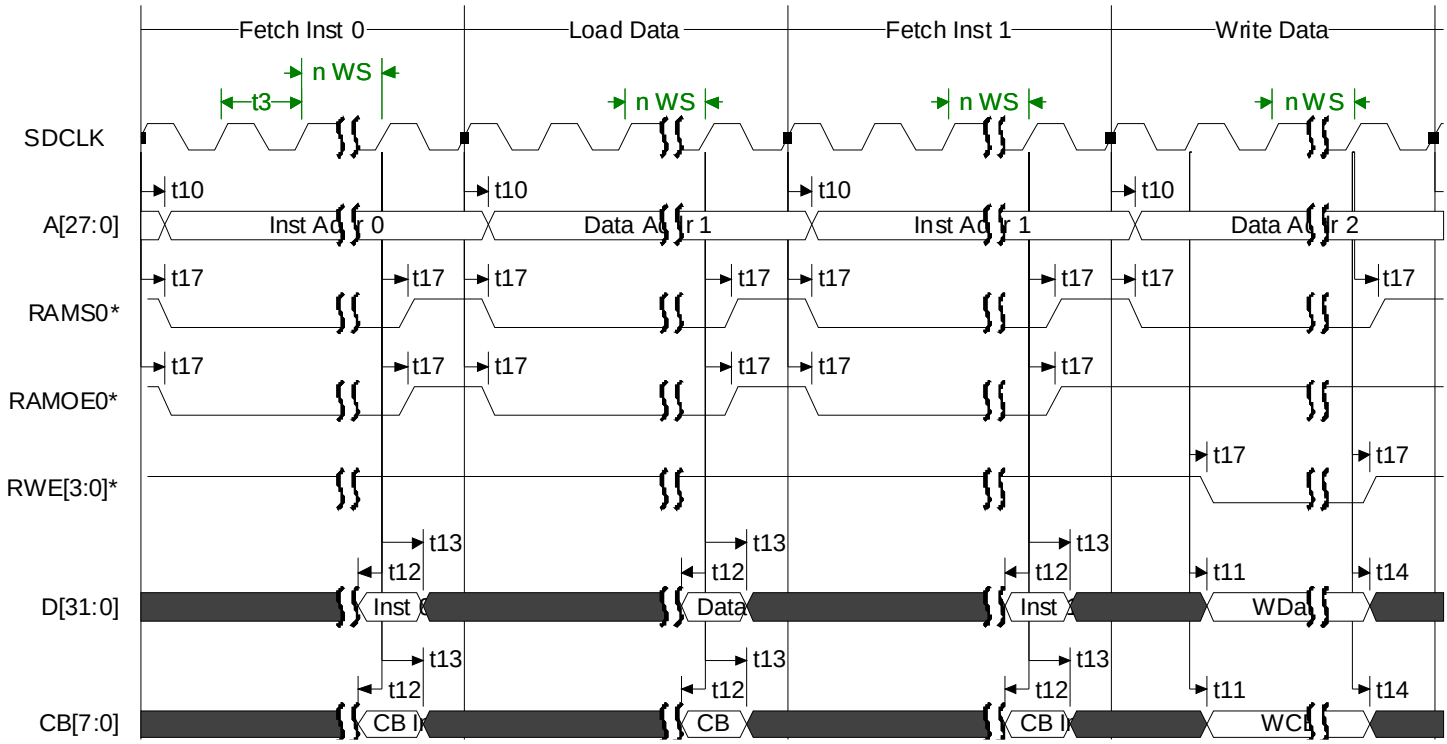


Figure 64. Burst of RAM Fetches and RAM Write Sequence - 0 Waitstate(***PRELIMINARY***)

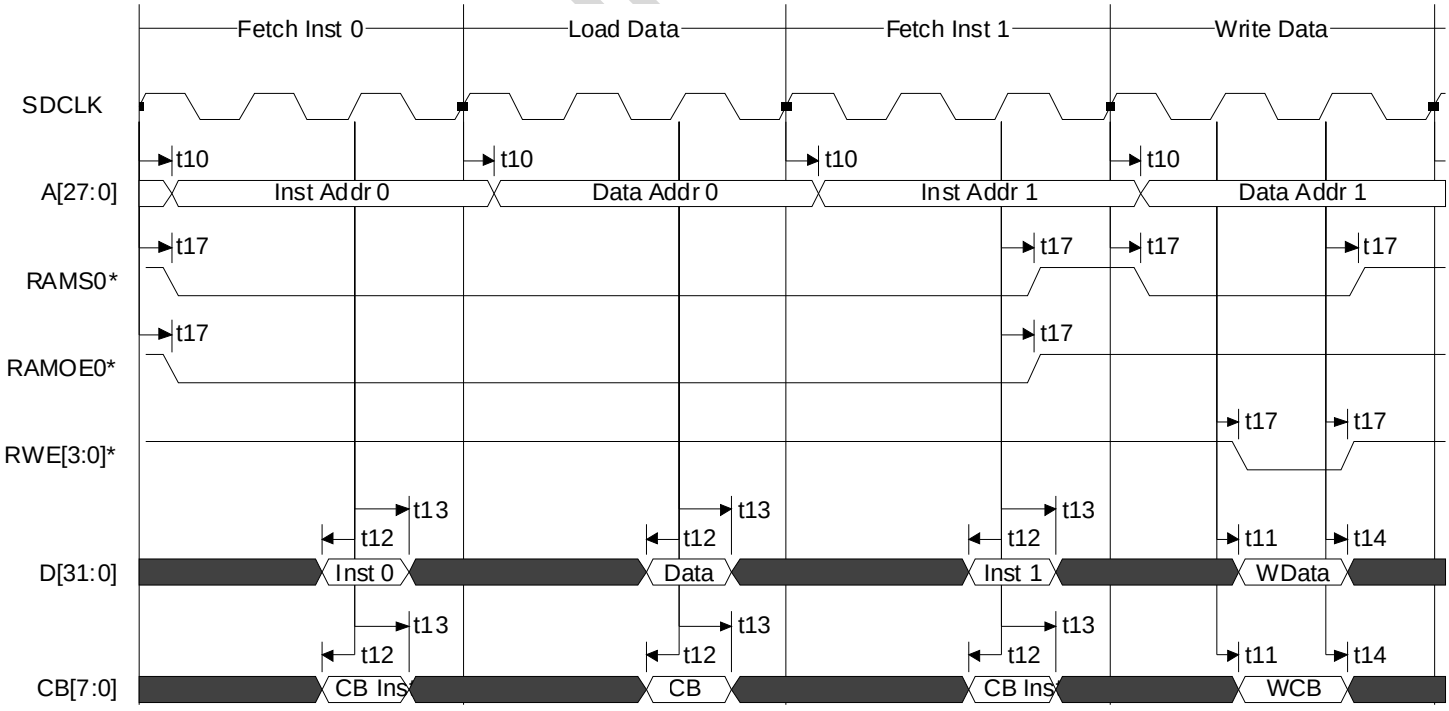


Figure 65. Burst of RAM Fetches and RAM Write Sequence - n Waitstates(***PRELIMINARY***)

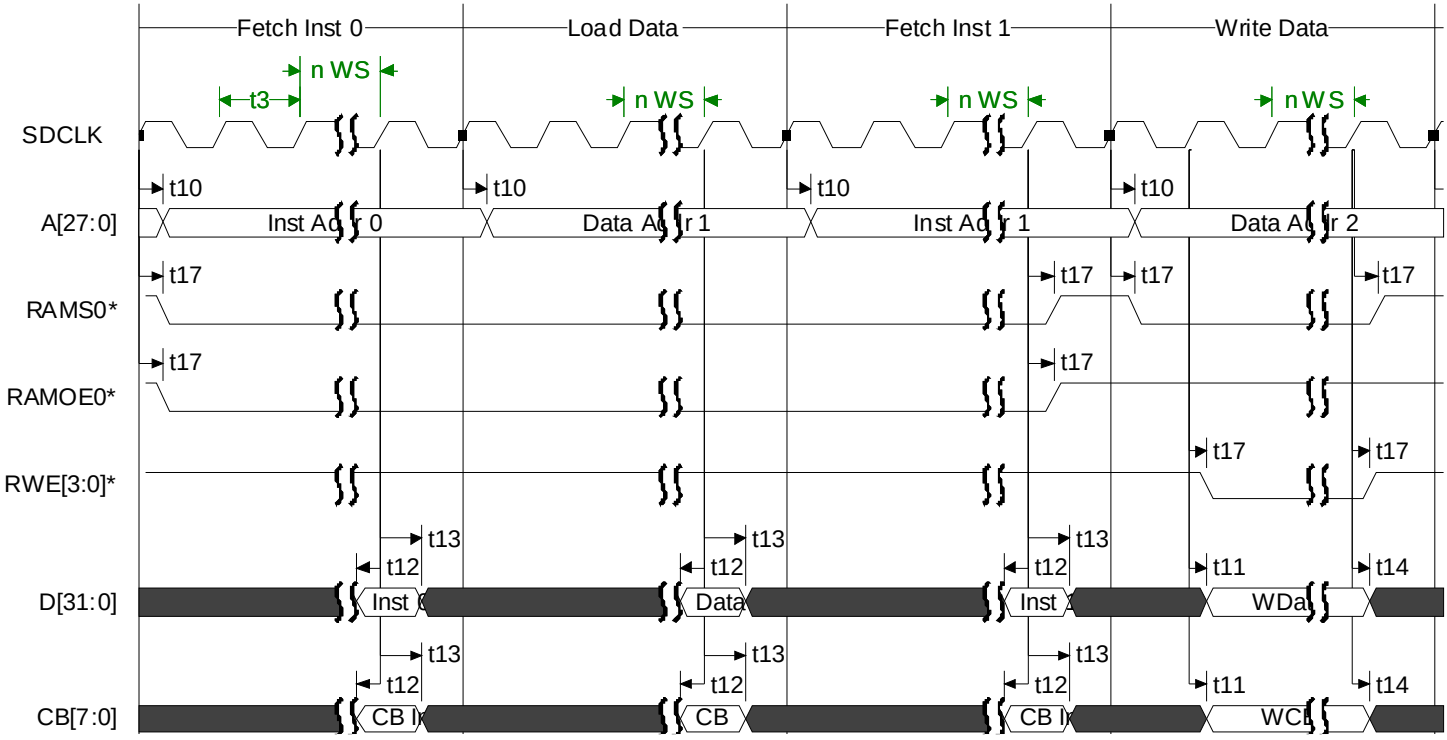


Figure 66. SDRAM Read (or Fetch) with Precharge - Burst length = 1; CL = 3(***PRELIMINARY***)

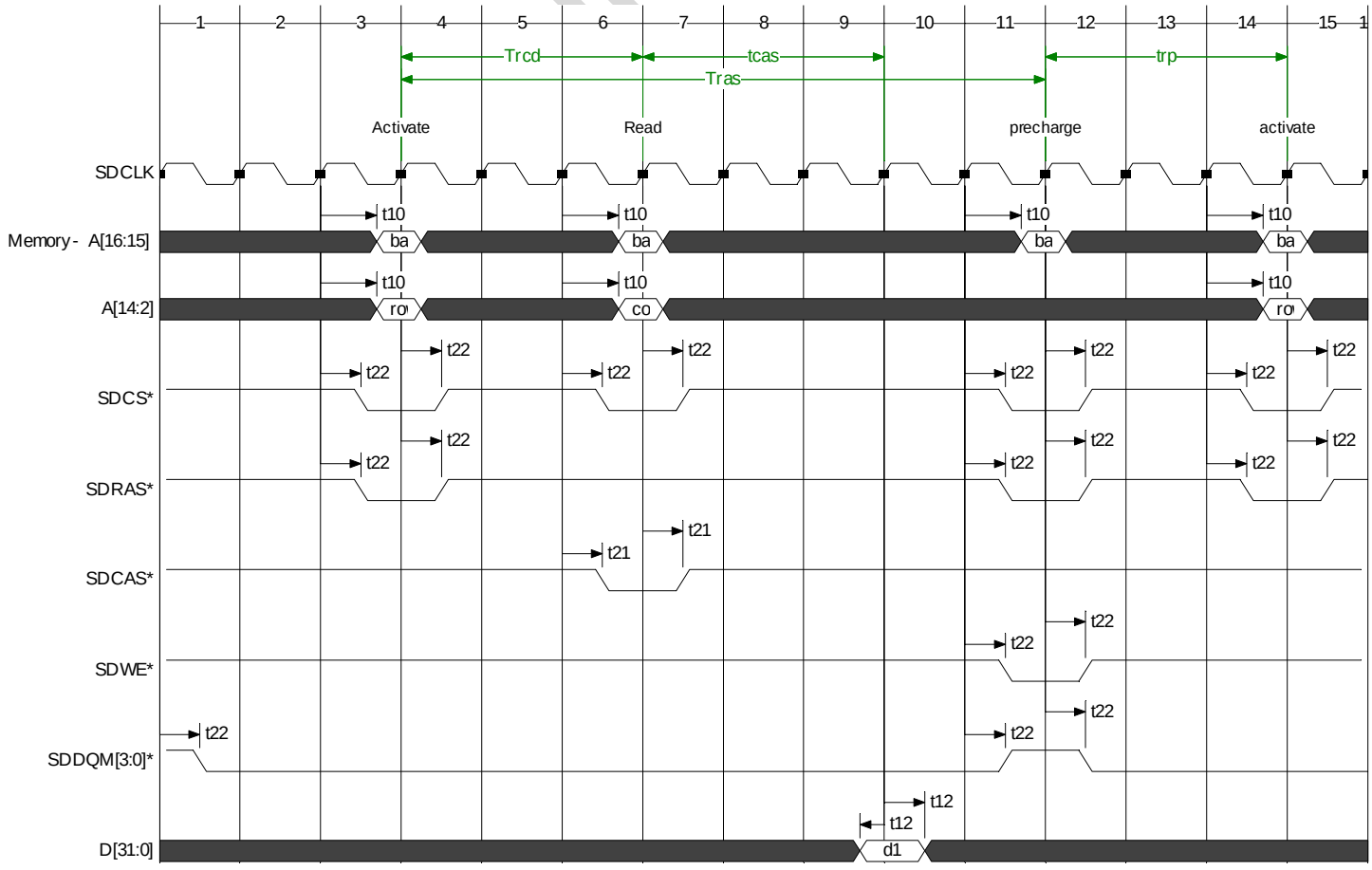


Figure 67. SDRAM Write with Precharge - Burst length = 1; CL = 3(***PRELIMINARY***)

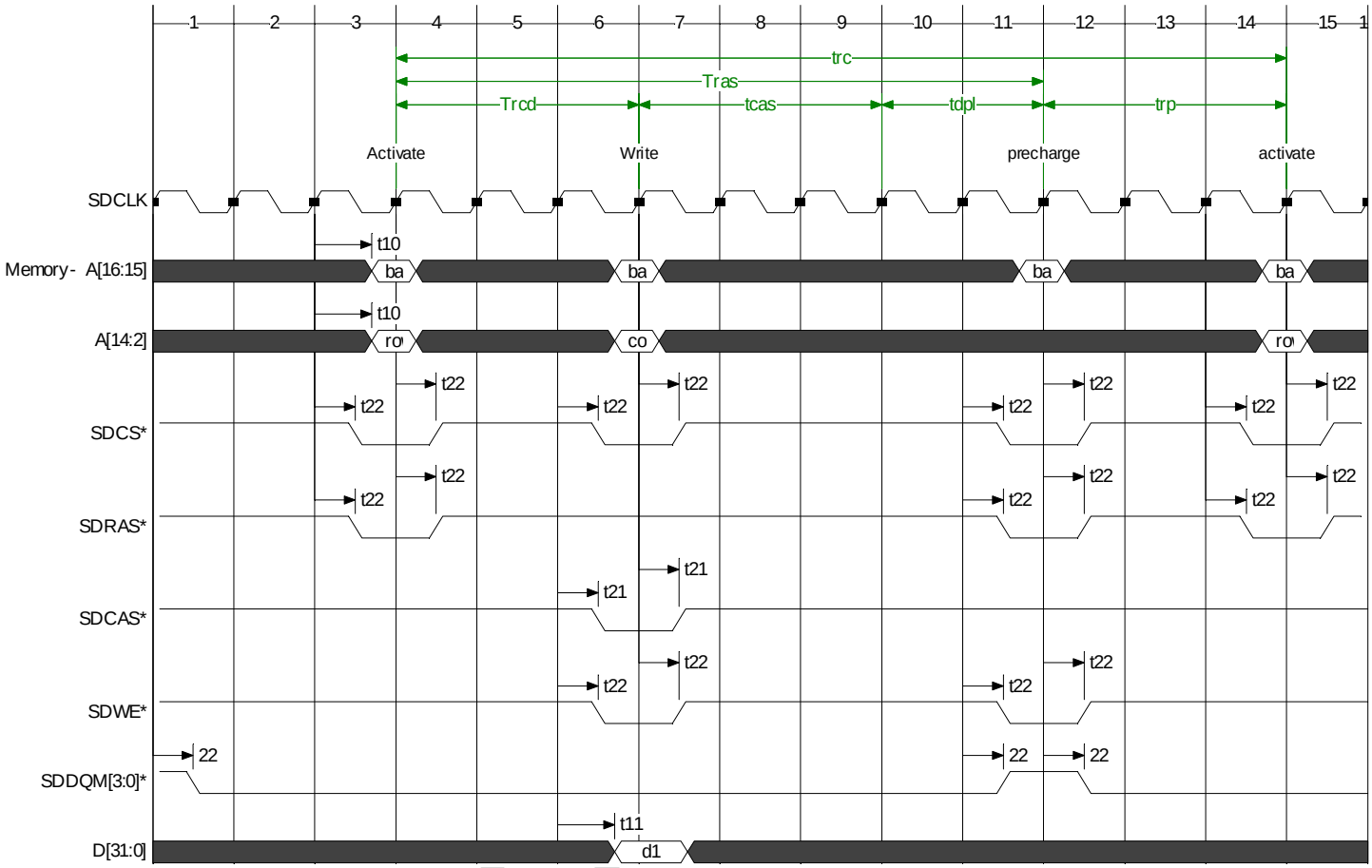


Figure 68. Fetch from ROM, Read and Write from/to 32-bit I/O - 0 Waitstate(***PRELIMINARY***)

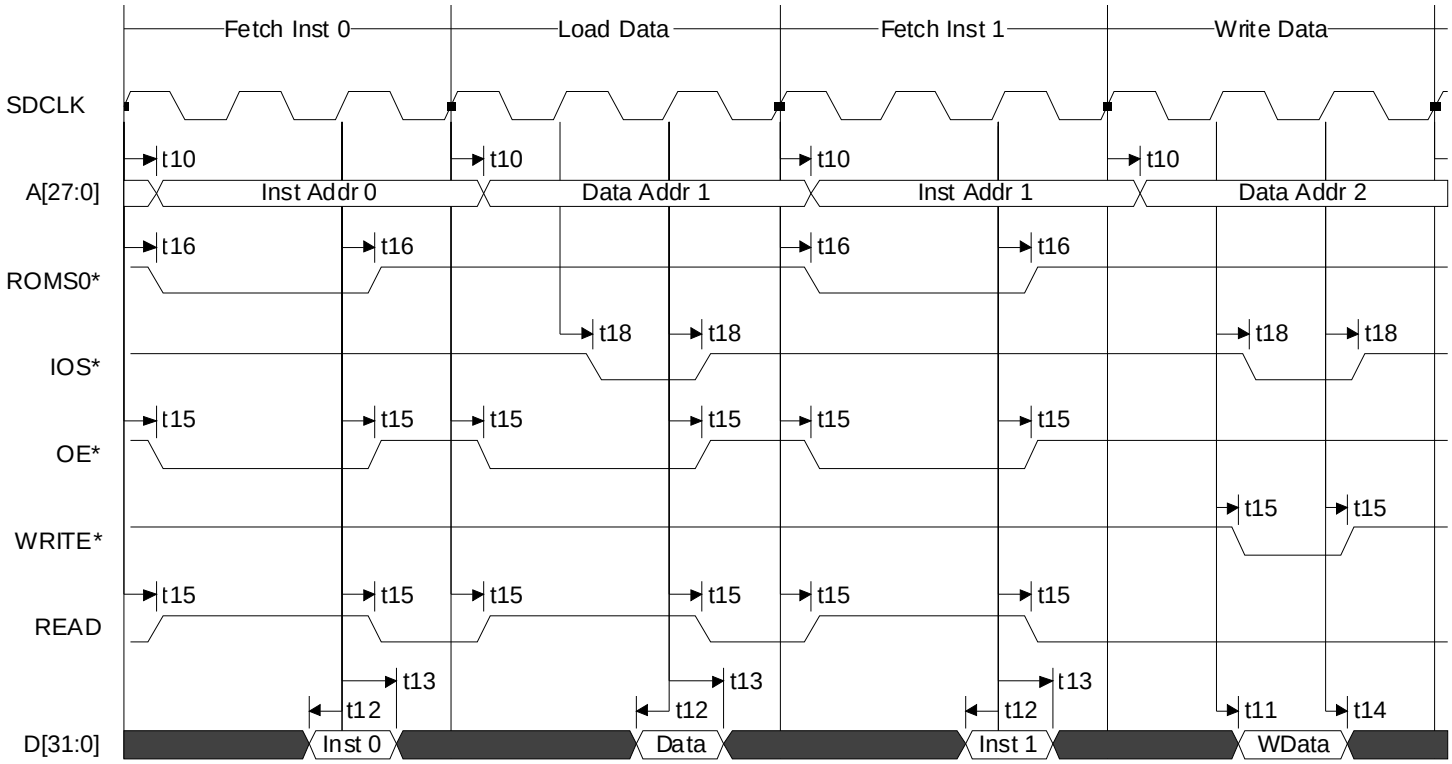


Figure 69. Fetch from ROM, Read and Write from/to 32-bit I/O - n Waitstates(***PRELIMINARY***)

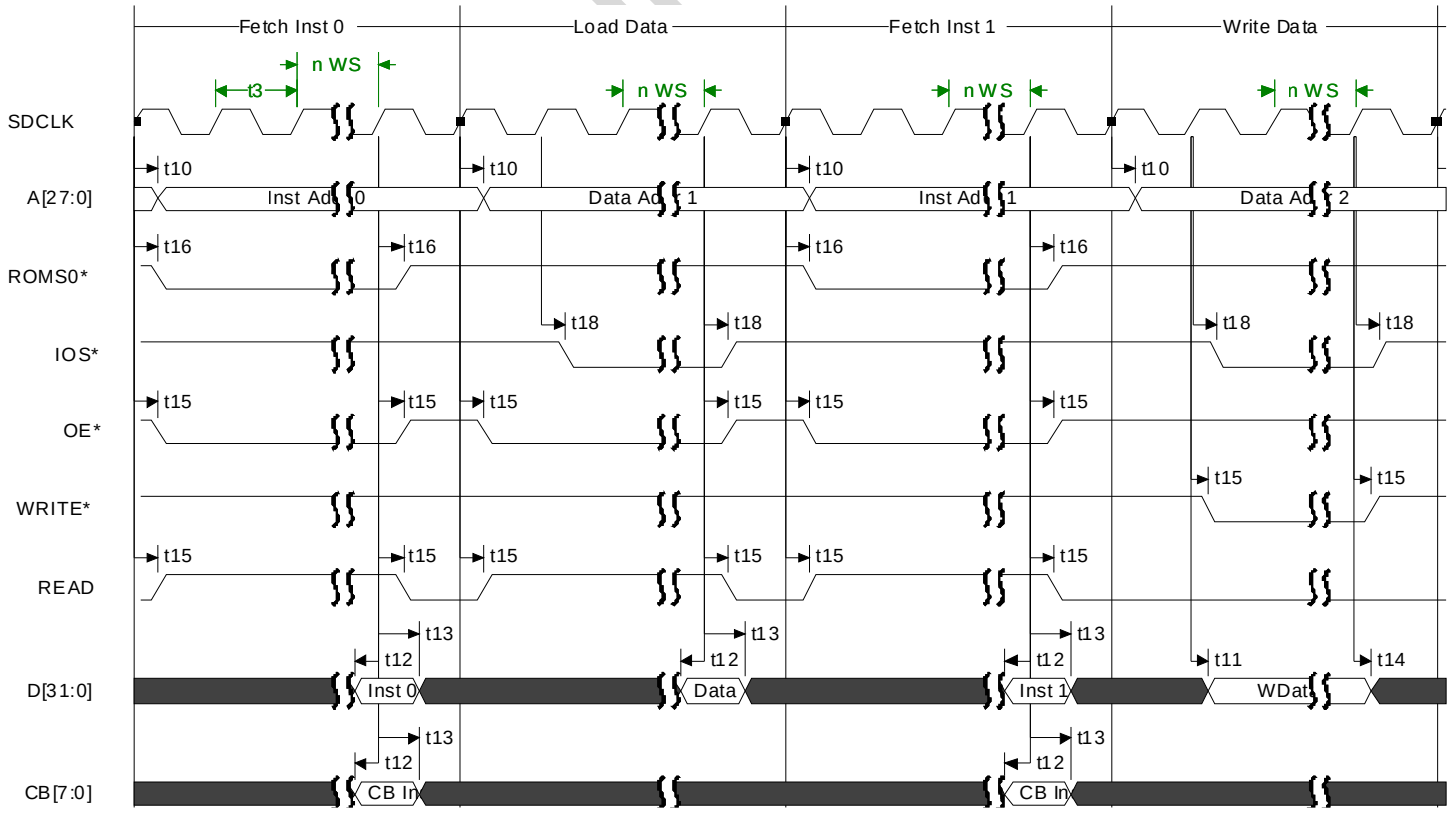
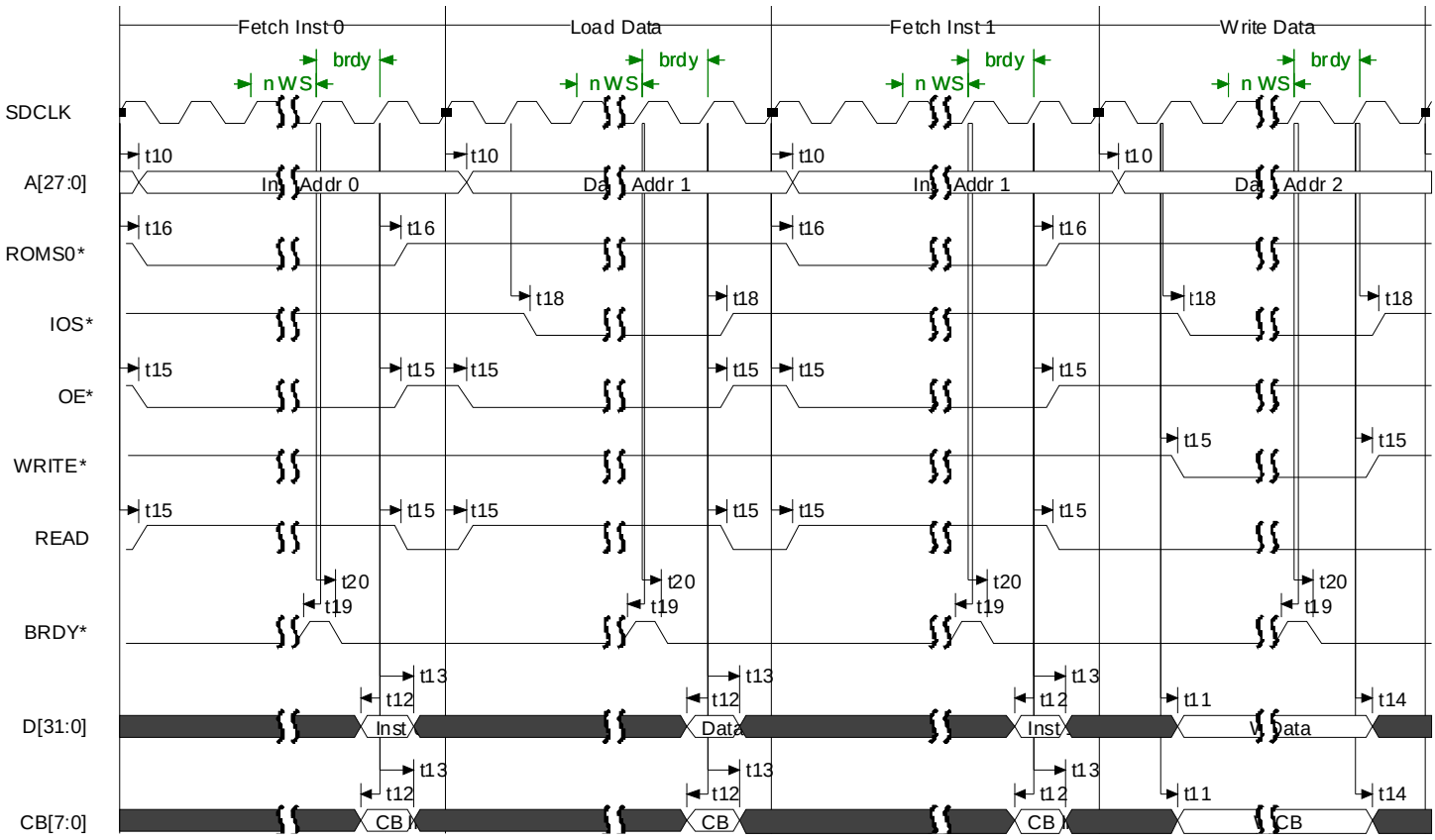


Figure 70. Fetch from ROM, Read and Write from/to 32-bit I/O - n Waitstates + BRDY*(***PRELIMINARY***)



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 128. Summary of the new/modified features

Feature	AT697F	AT697E
Write protection scheme	Start/End addresses and MASK based	MASK based only
BRDY* capability over ROM 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 detailled information on the functional bugs status.

Register modifications

Table 129. Summary of the register changes

Register	Address	AT697F Description	AT697E Description
MCFG1	0x80000000	bit 30 - PROM bus ready enable bit 29 - Asynchronous bus read enable	bit 30 - reserved bit 29 -reserved
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
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 - reload 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

Register	Address	AT697F Description	AT697E Description
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	Renaming of IOIT	IOIT
IOIT2	0x800000AC	IO Port Interrupt Register Configuration of IO interrupt for interrupt 4, 5, 6 and 7	not available
PCIID1	0x80000100	device id : 0x1E0F vendor id : 16E3	device id : 0x1202 vendor id : 0x1438
PCIID2	0x80000108	class code : 0xB4000 revision id : 0x10	class code : 0xB revision id : 0x01
PCISID	0x8000012C	subsystem id : 0x2103 subsystem vendor id : 0x16E3	subsystem id : 0x1 subsystem vendor id : 0x143E
PCIIS	0x80000154	bit 12 - SYSEN* state	bit 12 - reserved
PCIIC	0x80000158	bit 3 - reserved bit 2 - reserved bit 1 - reserved	bit 3 - PERR retry enable bit 2 - Double write configuration bit 1 - Double read configuration
PCITSC	0x80000160	bit 8 - force retry	bit 8 - reserved
TBC	0x90000004	bit 26 - AHB trace buffer freeze	bit 26 - reserved

Pin Modifications

Table 130. Summary if the pin changes

Pin Name	Pin Number	AT697F Description	AT697E Description
LFT	M16	Not Connected - The PLL filter is internal	PLL filter

Ordering Information

Table 131. 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
AT697F-KG-E	1.8V / 3.3V	+25°C	100	MQFP 256	Engineering Samples

Datasheet Revision History

7703A - 05/08

1. Document creation.

7703B - 12/08

1. ADVANCE INFORMATION DATASHEET Document.

7703C - 6/09

1. AB 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

TABLE OF CONTENTS

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
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
aerospace@nto.atmel.com

Sales Contact
www.atmel.com/contacts

Literature Requests
www.atmel.com/literature

7703C–AERO–6/09