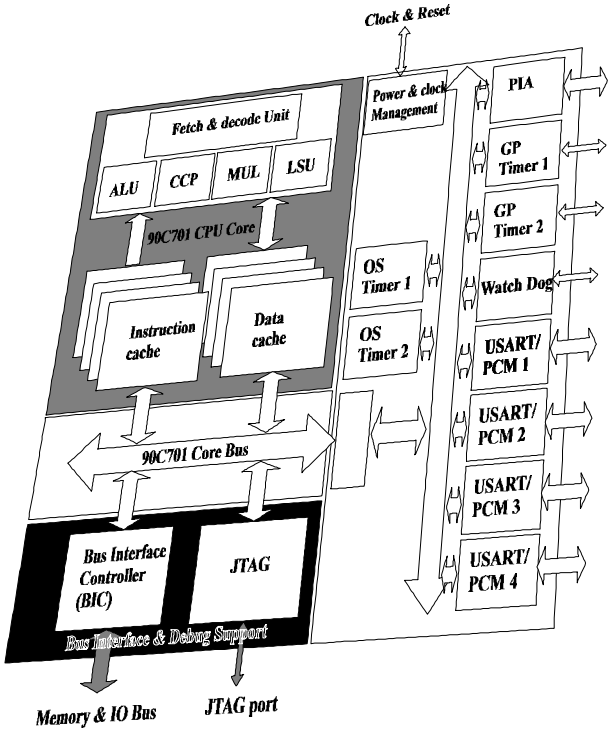
90C701
for Advanced Communication Systems

Preview - November 1995



TEMIC
Semiconductor

TEMIC / MATRA MHS FAX-IT
We Want Your Comments

FAX (+33) 1-30 60 71 57
e-mail : c701.preview@matramhs.fr

TEMIC / MATRA MHS SPARClet™ Applications Engineering provides a Fax number and an e-mail address for your comments about the content of the Advanced Communication Controller 90C701 Preview. We welcome your suggestions.

When referring to items in the preview, please reference the section number, page number, line number, and, if applicable, figure number and table number.

When sending a FAX or e-mail, please provide your name, company name, FAX number, and phone number.

The information contained herein is subject to change without notice. No responsibility is assumed by MATRA MHS SA for using this publication and/or circuits described herein: nor for any possible infringements of patents or other rights of third parties which may result from its use.

About this Preview

The 90C701 is the first in the SPARClet™ microcontrollers family and provides the basis for developing further derivative compatible products already forecasted.

The goal of this preview is to define the features and functionality of the 90C701 microcontroller for project leaders, system architects, hardware and software designers. The 90C701 preview has been organized accordingly. This document is composed of three main chapters:

Chapter I : Product Overview

The product overview lists the main features of the product without going to detailed functionality.

Chapter II : Product Architecture

The product architecture chapter includes the sections called "SPARClet™ Architecture" and "The 90C701 as a SPARClet™ Implementation". Main SPARClet™ implementation dependant features are described.

Chapter III : Product Description

The product description chapter is organized around the "90C701 Programming Model" and the "90C701 Operations and Registers Description" sections. These two parts give a detailed information on memory organization, instructions set, registers organization and associated operations.

The following additional reading are suggested.

- The SPARC Architecture Manual Version 8, SPARC International, Inc.
- SPARC-V8 Embedded (V8E) Release 1 Architecture Specification.

This can provide background to the information in this preview.

Table of Content

Table of Content

List of Figures

List of Tables

Chapter I

Product Overview

90C701 : Advanced Communication Controller

1 90C701 Microcontroller Overview

The 90C701 is an embedded SPARC processor with integrated communication peripherals. Built around a SPARClet™ CPU core, it includes the most frequently needed peripherals in advanced communication applications. The 90C701 is specially adapted for communication applications such as digital cellular base stations, bridges, routers, optical frame relay, ISDN adapters, and communication card controllers.

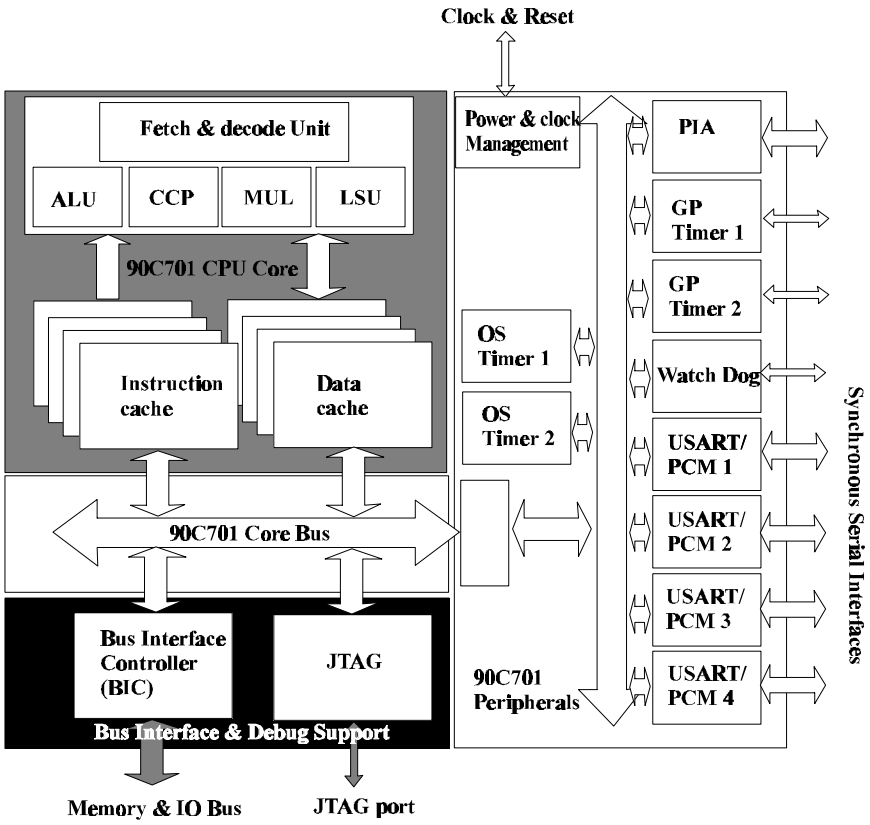


Figure 1. 90C701 block diagram

The 90C701 consists of a high-performance RISC fixed-point processor with integrated memories and devices controller, peripheral interface adapter, timers, USARTs, and JTAG port controllers.

1.1 The CPU Core

The CPU core integrates four execution units including a hardware multiplier (MUL), communication coprocessor (CCP), arithmetic and logic unit (ALU) and load-store unit (LSU). The 8 KB Data cache and 16 KB Instruction cache are integrated also and both use a four-way associative organization scheme. The CPU executes one instruction per cycle. Accordingly, all operations which require several cycles to be completed, are executed concurrently. For example, if the 90C701 is waiting for data coming back from the memory while the CPU is multiplying, both processing and transactions are done in parallel.

1.2 The Core Bus

The 90C701 core bus is the main link between the instructions cache, the data cache, the I/O-memory interface units and the on-chip peripherals bridge. Using a split cycle bus protocol, each transaction is tagged and exploits the full bandwidth of the core bus, even in presence of wait states. According to the SPARClet™ core bus protocol, the CPU performs the requests in order but the results comes back out of order.

1.3 Bus Interface & Debug Support

The Bus Interface Controller (BIC) supports external I/O devices and memory banks of different speeds through a user-programmable interface. The I/O-memory bus is organized through a 32-bit data bus, addressing for 256 MB of external memory and peripheral devices. The 90C701 controls directly DRAM, SRAM, ROM and I/O devices in a 48 MB address space. The BIC supports also multimaster configuration.

Debug is supported by the JTAG port (IEEE 1149.1 compliant). This features a TAP (Test Access Port) which provides the support for accessing internal SPARClet™ core bus agents as well as standard Boundary SCAN functions.

1.4 On-Chip Peripherals

The 90C701 provides four synchronous-asynchronous serial interfaces. According to the available CPU load budget in the application, the transmission speeds can be in the range from 2 Mbits to 8 Mbits. For example, four transmissions at 2 Mbits can be achieved in full duplex, requiring only 30 % of the CPU load at 40 MHz. The physical interface of the serial port supports the PCM, UART, and USART signals, through a user-programmable interface.

Timers, Watchdog, and PIA are also on-chip peripherals. The OS Timers provide the time base to support the task and event scheduling activities of real-time operating systems. The General Purpose timers support several modes such as PWM (pulse width modulation). The Peripheral Interface Adapter (PIA) features up to 10 user-programmable general purpose input/output ports.

1.5 Features

- Fully static SPARClet™ CPU core
- On-chip clock frequency multiplier
- Industrial range operating frequency
  - 30 MHz at 3.3V (+/- 10%)
  - 50 MHz at 5V (+/- 10%)
- V8 compliant SPARC Processor
  - Little & Big Endian data supported
  - Transparent power management system
  - Mulitply and Accumulate instruction
  - Bit scanning and bit shuffling instructions
  - 8 Register Windows
  - Alternate Window Registers
- Instruction Cache
  - 16 KBytes
  - four-way associativity
  - eight-words line size
  - Lockable by bank
  - Full LRU replacement algorithm
- Data Cache
  - 8 KBytes
  - four-way associativity
  - four-words line size
  - Lockable by bank
  - Write through and copy back support
  - Full LRU replacement algorithm
  - 8 entries store buffer
  - No write allocation
- Bus Interface Controller (BIC)
  - 256 Mbytes address space
  - DRAM interface with programmable refresh
  - SRAM interface
  - ROM interface
  - Multimaster bus support
  - 8-bit boot feature
  - Control signals generated for 48 Mbytes.
- Communication Coprocessor (CCP)
  - Coder/Decoder/CRC
  - supported protocols : HDLC,V.110, proprietary
  - 50 Mbit per second max. @ 50MHz
- Peripheral Interface Adapter
  - 10-bit bidirectional port
  - Lockable directions for secure design
- Timers
  - 2 General Purpose Timers
  - 2 Operating System Timers
  - 1 Watchdog
- USART/PCMs
  - 4 supported
- Distributed interrupt control logic
  - software programmable interrupt levels
- JTAG with Boundary Scan
- 208-pin PQFP and 240-pin PGA packages
- 0.6 µm, 3 metal layers CMOS technology

Chapter II

Product Architecture

2 SPARClet™ Architecture

The SPARClet™ architecture is a SPARC V8 RISC based processor. Enhancements have been made to merge data processing and real-time control execution on the same cost-effective central processing unit. Combining parallel operational units and superscalar techniques, SPARClet ™ provides the best trade-off regarding the price/performance ratio.

SPARClet™ is particularly well adapted for emerging advanced communication systems which require high-performance embedded computing devices to support new applications such as real-time speech recognition or image processing. SPARClet ™ is a general purpose architecture including Digital Signal Processing functions specially designed to address these requirements.

2.1 Performance Challenge

The performance of a processor can be defined as the time required to accomplish a specific task and is expressed as the product of two factors:

Time per Task = CPI * IPT

CPI = Cycle Per Instruction

IPT = Instruction per Task

Performance can be improved by reducing any of these two factors. RISC-type designs strive to improve performance by minimizing the first factor. In the following sections, the SPARClet ™ advantage in these performance-related factors, is highlighted.

2.1.1 Cycle Per Instruction (CPI)

One of the main benefits of using SPARClet ™ is its high performance/power consumption ratio (Mips/mWatt) as represented by the number of CPI (Cycles Per Instruction).

As with other RISC processors, SPARClet ™ exploits the instruction pipeline and the load/store popular architecture. Accordingly, the SPARClet ™ instruction pipeline works by dividing the execution of each instruction into four stages as shown in Figure 2 :

Fetch (F)	Decode (D)	Execute (E)	WriteBack (WB)

Figure 2. RISC instruction pipeline

According to the RISC concept one instruction is fetched and decoded each cycle. However, enhancements have been made in the pipeline control to exploit the natural parallelism of the executed operations. In the SPARClet ™ architecture all instructions requiring several cycles, such as multiply, load/store, and co-processor instructions, operate in parallel with the arithmetic and logic instructions. Accordingly, after the fetch stage the instruction is broadcast to the different execution units (including the fetch unit, which is responsible for control transfer instructions). Each execution unit is responsible for decoding, executing, and writing results in the register file. Consequently, results are written back out of order.

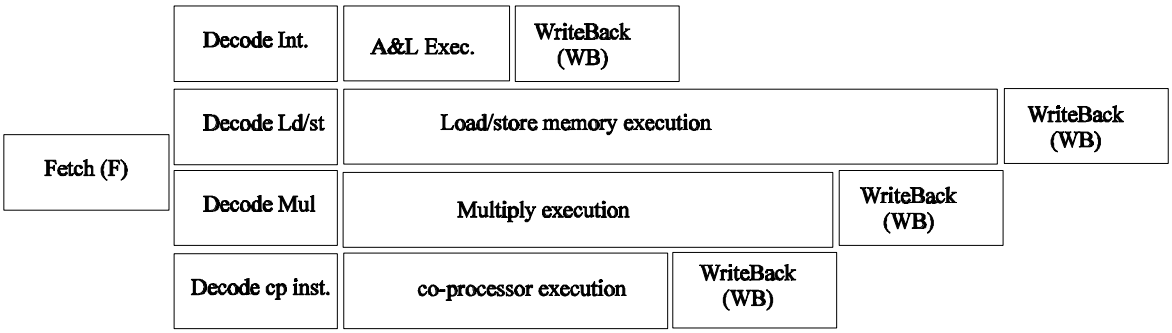


Figure 3. SPARClet™ instruction pipeline

Load instructions read data from memory into processor registers for later processing by subsequent instructions. Because memory typically operates at much slower speeds than the processor, the loaded operand is not immediately available for subsequent instructions in a processor with an instruction pipeline. The technique used in many RISC designs to handle this data dependency is to rely on the compilers to handle the inherent latency and respect the load delay. Usually, to keep one CPI, the load delay must be one instruction. In SPARClet™ the latency or the duration of load delay covers the execution of several instructions.

Accordingly, the SPARClet™ CPU needs one effective-cycle per instruction while traditional RISC based architecture will need 2 or 3 cycles in average to operate in applications. Wait states can be inserted for only two reasons : resource conflicts (a resource is already busy executing an operation and instruction needs it). and data dependencies (an instruction needs a data which is not yet available).

2.1.2 Instructions Per Task (IPT)

The number of executed instructions depends on the optimizing techniques used in compilers, such as register allocation, redundancy elimination, replacement algorithms with faster operation (Multiply-and-accumulate, bit shuffling or scanning), loop optimization, and pipeline scheduling. The SPARClet™ architecture contributes to the reduction of instructions per task in two manners:

New instructions to support digital signal algorithms:

Multiply-and-accumulate instruction (MAC):

Accumulation is executed without an extra cycle. The speed of the MAC is the speed of the multiplier.

Bit scanning instruction (SCAN):

The SCAN instruction is particularly well suited to data normalization, priority encoding and run, length and coding algorithms. It replaces 30 scalar instructions.

Bit shuffling instruction (SHUFFLE):

The SHUFFLE instruction executes bit, couples,digits,byte, nibble, and half-word swapping, and supports efficiently the data endianess issue.

Pipeline scheduling:

In pipeline scheduling techniques the compilers schedule and reorganize instructions to ensure that pipeline delay slots are filled with useful instructions as illustrated earlier in the description of load delays.

The following task (dot product) shows the benefit of the instruction reordering generated by the compiler. In this example, some of the 5 instructions in the inner loop require multiple cycles operations, such as the four-cycles latency Multiply-and-Accumulate (UMAC) instruction and the two-cycles latency Load (LD) instruction. A regular RISC processor will need 13 cycles to execute the loop. SPARClet™ will do the same loop with only five cycles. Figure 4 shows the instruction scheduling for the inner loop of the dot product algorithm..

```
            ld    [%g1+%l0], %l1
            ld    [%g2+%l0], %l3
            subcc %l0, 4, %l0
Loop:       umac  %l1, %l3, %o1
            ld    [%g1+%l0], %l1
            ld    [%g2+%l0], %l3
            bne   Loop
            subcc %l0, 4, %l0       ; always executed (delay slot)
```

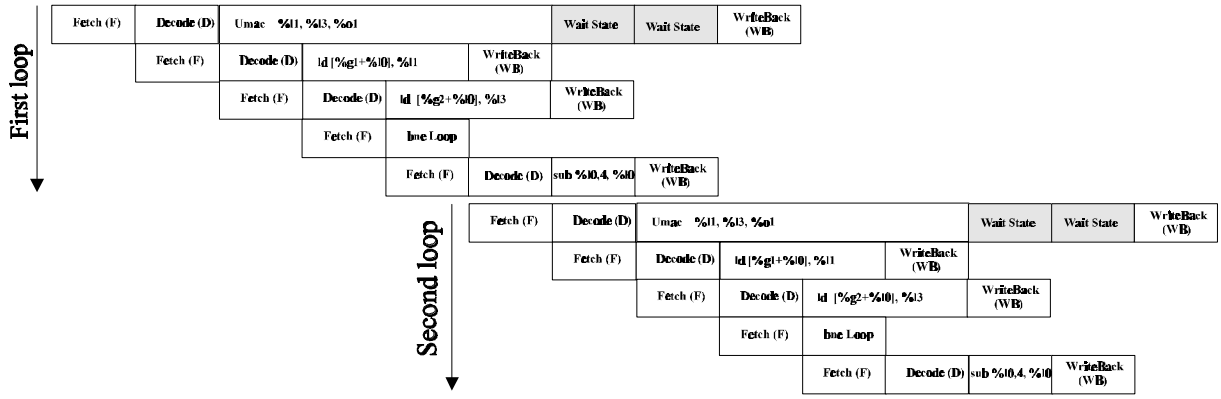


Figure 4. SPARClet™ Pipeline Scheduling - dot product inner loop

In this example, the MAC instruction is ready to write back its result at the same time as load and subcc instructions. Two wait states have to be added to the MAC instruction to return the result to the register file. This resource conflict is described in the "Product Architecture" Chapter . Even with these two wait states, the SPARClet™ architecture allows the processor to run at one CPI for this application.

2.2 Operating System Support

The performance gains obtained by providing support for operating systems are often subtle. The SPARClet™ architecture supplies appropriate operating system support to enhance performance in this domain. For example SPARClet™ supports Multiprocessor synchronization instructions. One of them performs an atomic read-then-set-memory operation; another performs an atomic exchange-register-with-memory operation. Enhancement has been done in order to provide fast trap handlers. In this domain extra features are supported by the SPARClet ™, Single Vector Trapping and Alternate Window Registers. The interrupt response has been

Single Vector Trapping

Single vector trapping can spare code space. Due to the SVT, the traps table size is reduced from 4KB to four-words. After a trap has been taken, its Trap Type can be determined by reading the Trap Type field, *tt*, of the Trap Base Register (TBR). This can be used by software to determine subsequent processing of the trap.

Alternate Window Registers

The Alternate Window Registers (AWR) consist of a separate set of 32 registers. The associated mechanism allows routines which manipulate large amounts of data (such as trap handlers or software direct memory access handlers) to switch context faster. This capability reduces the interrupt latency. Each level of interrupt (IRL) can be associated to alternate window registers or not. The alternate window registers avoid time loss in save/restore context operations.

Thanks to the SPARClet™ architecture, another real-time improvement has been accomplished regarding the interrupt response time. The SPARClet ™ CPU can process an interrupt routine while it is executing memory loads/stores or multiplications. This characteristic allows a fast response time, as well as better determinism in the behavior of the real-time applications. The typical time from the interrupt detection (after de-glitching by the input handler in case of an external interrupt) to the trap handler's first instruction fetch is 6 to 7 clock cycles.

3 The 90C701 as a SPARClet™ Implementation

3.1 The 90C701 CPU Core

The 90C701 CPU core consists of two main subsystems, the control block and the execution units (Figure 5). The control block includes the Fetch and Decode unit and the three ports registers file and all the CPU control registers. The fetch and decode unit gathers all the sequencing functions.

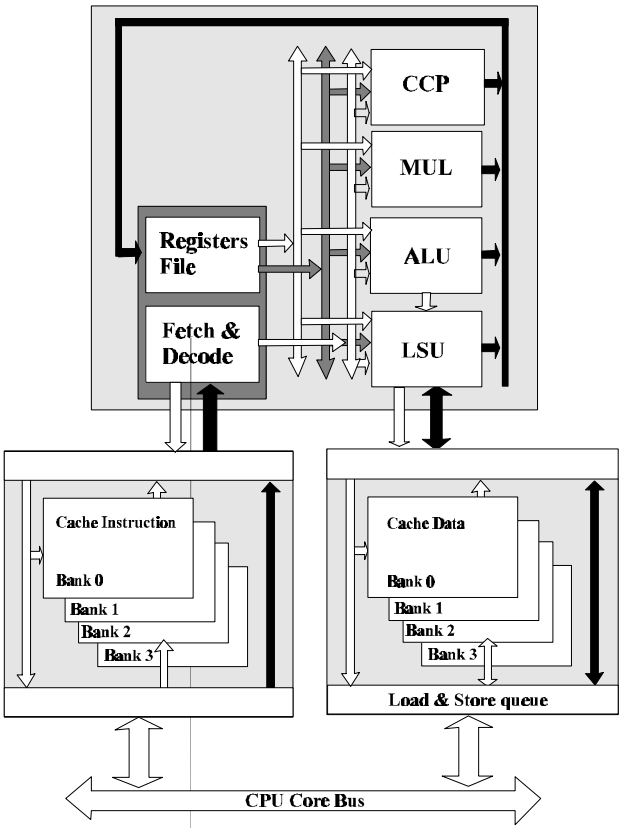


Figure 5. 90C701 CPU Core

The register file is a triple port static RAM array. The 136 32-bit registers are divided into a set of 128 registers and a set of 8 global registers. The 128 registers are grouped into eight overlapping sets of 24 registers called register windows. Each window shares eight registers with its two adjacent windows. The Alternate Window Registers are an additional 32 registers window entered on specific conditions detailed in the "Product Description" Chapter.

The 90C701 CPU control registers include the Processor State Register (PSR), the Window Invalid Mask (WIM) register, the Trap Base Register (TBR), the Program Counters (PC, nPC), and the Ancillary State Registers (ASRs). A brief description of these registers is given in "Product Description" Chapter .

Four internal buses are used to control the execution units, while one bus is used by the control block to send the operation to the execution unit which will operate the decoded instruction. At the same time two buses are used to provide the two source operands to the associated execution unit. At this time the selected execution unit is responsible for the processing and the reporting activities of the instruction. Concurrent processing is fully supported by a dedicated hardware interlock mechanism which prevents any out-of-order data update. When the operation is completed, the execution unit writes back the result to the register file, sharing the destination bus with the other execution units.

Instructions are accessed by the processor from memory and are executed, annulled, or trapped. Instructions are encoded in three 32-bit formats and can be partitioned into five categories. The categories are the load/store instructions, the integer arithmetic instructions, the control-transfer instructions, Read/Write state registers, and Communication co-processor instructions. These instructions are presented in the next Chapter. These five instruction categories are implemented on four execution units which will be described below. These are the arithmetic and logic unit, the multiplier unit, the load/store unit, and the communication co-processor.

3.1.1 Resource Conflicts

In a running situation (as already discussed in the chapter SPARClet ™ Architecture), wait states can be generated according to the resource conflict and/or to the data dependency. In most such cases, the compiler can use techniques such as instruction reordering to minimize the performance impact due to the data dependency between instructions.

Since the register file has only two read ports, some instructions, which need to access more than two operands in the same cycle, will generate wait states. The following table illustrates this situation.

Operand fetch in register file	*Instruction example*	*Wait state*
no operand	`bicc, sethi, cppull,..`	No
rs1 only	`add %r1, imm, %r2`	No
rs1 and rs2	`add %r1, %r2, %r3`	No
rs1 and rd	`st %r2,[%r1+imm]`	No
rs1 and rd and rd+1	`std %r2,[%r1+imm]`	(*)
rs1 and rs2 and rd	`st %r3,[%r1+%r2]`	(*)
rs1 and rs2 and rd and rd+1	`std %r4,[%r1+%r2]`	(*)

Table 1. Resource conflicts on register file fetch

(*) Note: These instructions generate wait state only if following instructions use the second read port of the register file.

Another source of conflict can be the write-back port of the register file. When the operation is completed, each execution unit will ask to access to this port to write its result in the registers file. In case of several write-back port requests during the same cycle, the execution units have to be prioritized according to instruction classes:

Instruction class	*Priority*
Control-transfer	1
Integer Arithmetic	2
Load/store	3
Communication Co-processor	4
Multiplier	5

Level 1 is the highest priority.

There are also possible conflicts within execution units, especially with the communication coprocessor, the multiplier, and the load/store units. In this case, the availability of the operators to execute the next operation is the main reason of wait states insertion.

3.1.2 Execution Units

The execution units have been designed to support four classes of instructions. All run in parallel. The first class consists of instructions regarding external CPU accesses, such as load and store instructions, excepting the address generation. The second class is associated to the dedicated processing, such as multiplications, multiply-and-accumulate and read and write Y register. The third class, is specific to the 90C701 communication oriented co-processor instructions. The last class includes the traditional arithmetic and logic instructions : in other words, all the scalar instructions and the address generation function (JMPL, RETT, STORE, LOAD).

3.1.3 Load/Store Unit

The load/store unit is in charge of all the transactions between the CPU core, the data cache, and memory-I/O external devices. The load/store unit can bufferize up to four ongoing loads once all information has been sent to the data cache while it is waiting for data to come back. The load instructions take one cycle to send an address. The store instructions can take from one to three cycles according to resource conflict cases as shown above.

3.1.4 Arithmetic and Logical Unit

The ALU performs all integer arithmetic and logical instructions, which are generally triadic-register-address instructions. The ALU computes a result that is a function of two source operands, and either writes the result into the destination register r[rd] or discards it. One of the source operands is always r[rs1]. The other source operand depends on the *immediate operand bit (i)* in the instruction. If i=0 , the operand is r[rs2], but if i=1, the operand is the constant simm13 sign-extended to a width of 32 bits. The ALU is also responsible for computing a 32-bit, byte-aligned memory address for the Load and store instructions.

3.1.5 Communication Coprocessor

The coprocessor supports HDLC and V110 communication protocols: they require the following functions:

- Coding messages, interleaving them with a byte granularity and transmitting the resulting bit stream on a serial interface.
- Receiving a bit stream from a serial interface, allocating its bytes to different messages and decoding them.

The coding and interleaving functions are executed by the communication coprocessor, as well as the reverse operations (decoding and byte allocation). It sustains a 1-bit per cycle throuput for medium size messages. Serial bit stream transmission and reception are handled by an on-chip PCM interface, capable to manage up to 8-Mb/sec full duplex data streams. Each PCM features 2 FIFO's of 12-Bytes each to smoothen transmission and reception processes.

Synchronization between the coprocessor and the PCM peripheral is ensured by an interrupt mechanism, with minimal context switching overhead, which supports 64-bit DMA transfers in less than 20 cycles. The Alternate Window (detailed in Chapter Product Description), provides the DMA pointers and counters for up to three full duplex channels.

Coding functions are user-programmable for both message modification ("zero" insertion or suppression) and CRC computation (polynomial coefficients). This allows to support several protocol families on the same silicon part. Real-time compromises (latency between consecutive DMA interrupts, versus DMA throuput) are under user's control: he can program the PCM FIFO's threshold for DMA interrupt generation. As a result, a 90C701 running at 50 MHz could manage a full duplex 4-Mb/sec HDLC stream, and keep 45 Mips available for additional users' needs.

3.1.6 Instruction Cache

The Fetch and Decode Unit processes instructions at a maximal rate of one CPI. Therefore, the Instruction Cache is able to fetch the code with the same performance level, in order to maximise the global throughput of the core.

Therefore, the 16 Kbytes Instruction Cache is split in 4 banks, according to a set associative scheme. Up to 3 banks can be locked simultaneously, avoiding "miss penalty" for critical routines smaller than 4 Kbytes (which fits for most embedded applications, for example critical trap handlers). The Instruction Cache features are summarized in the following table.

Feature	*Benefits*
16-bytes (4 banks *128 lines * 8 words)	Good trade-off for embedded applications
four-way associative	High hit rate
eight-words line size	Performance (bus traffic)
Lockable by bank	Critical routines deterministic behavior
Least Recently Used (LRU)	Performance

Table 2. Instruction Cache Features

Address decoding

The input address supplied by the Fetch and Decode Unit is decoded as follows :

Table 3. Instruction Cache Address Decoding

28-12	11-5	4-2	1-0
Tag Address	Line number	Word number	0 0

A Tag register is associated with each line of each bank (512 tag registers overall for the Instruction Cache).

Table 4. Instruction Cache Tag Register

18-2	1	0
Tag Address	P	V

P : Privilege associated with the line (0:User, 1:Supervisor)
V : Valid bit

If the 18 most significant bits of the input address bits matches with the Tag Address field of one out of the 512 Tag Registers, the required address is present in the cache (Cache Hit). Otherwise, the Load/Store Unit (LSU) will have to wait for the data to be accessed in external memory (Cache Miss).

LRU algorithm

In order to support the LRU algorithm, a register of 8 bits has been dedicated for each line.

Table 5. LRU Support Register

7-6	5-4	3-2	1-0
LRU			MRU

For each line, it contains 4 fields showing the Least Recently Used to Most Recently Used Bank numbers order. For example "3120" in LRU register for line 3 means that line 3/bank 0 was used the most recently, then bank2, then bank1 and bank 3 was the Least Recently Used.When a line has to be reloaded (In case of a Cache Miss), the Least Recently Used bank will be updated from main memory.

All LRU control bytes have to be initialized at 0xe4 ("3210" in LRU register).

Lock mechanism

The lock status is controlled by a 3 bit Instruction Cache Control Register (ICCR) :

Table 6. Instruction Cache Control Register

2-1	0
_lock	*_enable*

ICCR_enable When set, the Instruction Cache is enabled (initialized at 0 after Reset)

ICCR_lock Defines which banks are locked :
0 : no locked bank
1 : bank 3 is locked
2 : banks 3 and 2 are locked
3 : banks 3,2 and 1 are locked
Bank 0 is not lockable.

Instruction Cache Controller Address Space

The Cache Tag registers, the LRU registers, and the Control register can be read or updated using regular Store instructions in Supervisor mode.The address given in operand should be formatted as follows :

Table 7. Instruction Cache Controller Address Decoding

31-19	18	17-16	15	14	13	12	11-5	4-2	1-0
1 100 000 000 000	Op	Reg	B3	B2	B1	B0	Line number	Word number	Byte number

Bits 31-19 are the Cache Controller Base address, fixed as shown on the table.

The Op field tells if it is an update ("0") or a check ("1").In case of a check, the data value given in operand in the Store instruction will be compared to the checked register content and the operation will return a Bus Error if the values are different.

Reg shows which register is to be checked/updated :

00 : Control Reg
01 : Cache memory
10 : Cache Tag
11 : LRU register

B3..B0 : bank number (used for Cache Memory and Cache Tag). A "1" indicates that the corresponding bank is selected. Any combination is legal.

Line number : used for Cache Memory , LRU register and Cache Tag.
Word number : for Cache Memory, indicates which word in the selected line is to be checked/updated
Byte number : for partial store, indicates which byte of the word is to be checked/updated (Used for Cache Memory and the Cache Tag).

3.1.7 Data Cache

The Fetch and Decode Unit processes instructions at a maximal rate of one CPI. Therefore, the Data Cache can be accessed (read or written) once per clock cycle, to maximize the global throughput of the core. The 8Kbytes Data Cache is split in four banks, according to a set associative scheme. Up to three banks can be locked, avoiding "miss penalty" for critical sets of data (constant tables for example) .The different Data Cache parameters are summarized in the following table.

Feature	Advantage
8 Kbytes (4 banks * 128 lines * 4words)	Good trade-off for Embedded applications
four-way set associative	Allows four different contexts to use the same data set.
four-words line size	Performance (bus traffic)
Lockable by bank	Performance/Real-Time support
Copy back	Performance (bus traffic)
Write through	Real-Time support
Least Recently Used (LRU)	Performance

Table 8. Data Cache Features

In addition to standard caching functionalities, the Data Cache provides transaction queues (for load and store requests), so that multiple transactions can be handled simultaneously. Responses to load requests do not necessarily come back in order, and may pass missing loads being processed. This ability is called "hit-under-miss".

To maximize the performance gain of this enhancement, the Fetch and Decode Unit can generate several Data Cache accesses and continue the processing flow without waiting for the Data Cache responses, as long as no data dependency between consecutive instructions occured.

To support the "hit-under-miss" mechanism, the Data Cache and the Integer Unit respectively include a Store Buffer (eight entries) and a Load Buffer (four entries). The associated benefit is a Data Cache half as big as the Instruction Cache, with negligible impact on "miss penalty".

As the Data Cache implementation is very similar with the Instruction Cache, we will limit the following description to the differences.

Address Decoding

The input address is decoded as follows :

Table 9. Data Cache Address Decoding

28-11	10-4	3-2	1-0
Tag Address	Line number	Word number	Byte number

The Tag registers (one per line of each bank) :

Table 10.Data Cache Tag Register

22-4	3	2	1	0
Tag Address	D	F	P	V

D : Dirty line (line inconsistent with the memory content, happens when copy back is used and the update not done).

F : Full support (When this line has been allocated, the memory controller guaranteed that an error at the given address will never occur)

P : Privilege (0 for user)

V : line valid

Data Cache Control register (DCCR)

Two fields have been added to the Data Cache control register:

Table 11.Data Cache Control register (DCCR)

4	3	1-2	0
_copy_back	*_overtake_store*	*_lock*	*_enable*

DCCR_ copy_back Enables the copy back when set

DCCR_overtake_store allows missed cacheable loads to overtake pending stores when the targeted addresses are not in conflict.

DCCR_enable When set, the Instruction Cache is enabled (initialized at 0 after Reset)

DCCR_lock Defines which banks are locked :
0 : no locked bank
1 : bank 3 is locked
2 : banks 3 and 2 are locked
3 : banks 3,2 and 1 are locked
Bank 0 is not lockable.

Data Cache Controller Address Space

The Cache Tag registers, the LRU registers, and the Control register can be read or updated using regular Store instructions in Supervisor mode.The address given in operand should be formatted as follows :

Table 12.Data Cache Controller Address Decoding

31-18	17	16-15	14	13	12	11	10-4	3-2	1-0
11 001 000 000 000	Op	Reg	B3	B2	B1	B0	Line number	Word number	Byte number

Bits 31-18 are the Data Cache Controller Base address, fixed as shown on the table.

The Op field tells if it is an update ("0") or a simple check ("1").In case of a check, the data value given in operand in the Store instruction will be compared to the checked register content and the operation will return a Bus Error if the values are different.

Reg shows which register is to be checked/updated :

00 : Control Reg
01 : Cache memory
10 : Cache Tag
11 : LRU register

B3..B0 : bank number (used for Cache Memory and Cache Tag. A "1" indicates that the corresponding bank is selected. Any combination is legal)

Line number : used for Cache Memory , LRU register and Cache Tag
Word number : for Cache Memory, indicates which word in the selected line is to be checked/updated
Byte number : for partial store, indicates which byte of the word is to be checked/updated (Used for Cache Memory and the Cache Tag).

3.2 The 90C701 Core Bus

The 90C701 core bus is the central link between the CPU core, memory, and peripherals. It can support high bandwidth (32-bit word per cycle, 200 MB/s), even in presence of wait states through a split cycle protocol. In other words a split cycle protocol allows interleaving accesses between all the bus agents, which can be CPU (caches), memory, or peripherals. Each transaction has an associated signature. All messages belonging to a transaction are sent with the associated signature. This core bus provides the right support for critical word first block transfers through out-of-order responses and word hint.

Other features include one cycle latency for bus acquisition through self-arbitration and a fast reaction time through self slave selection mechanisms.

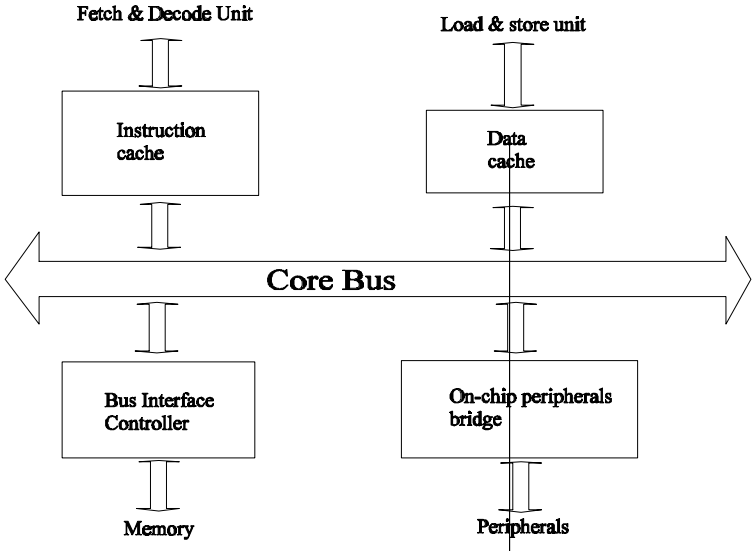


Figure 6. Core bus interconnection

All these characteristics contribute to the performance of the SPARClet ™ 90C701 processor, allowing one effective cycle per instruction even in presence of wait states due to the natural latency of memories and/or peripherals. The following figure shows how the 90C701 core bus interleaves device and memory accesses.

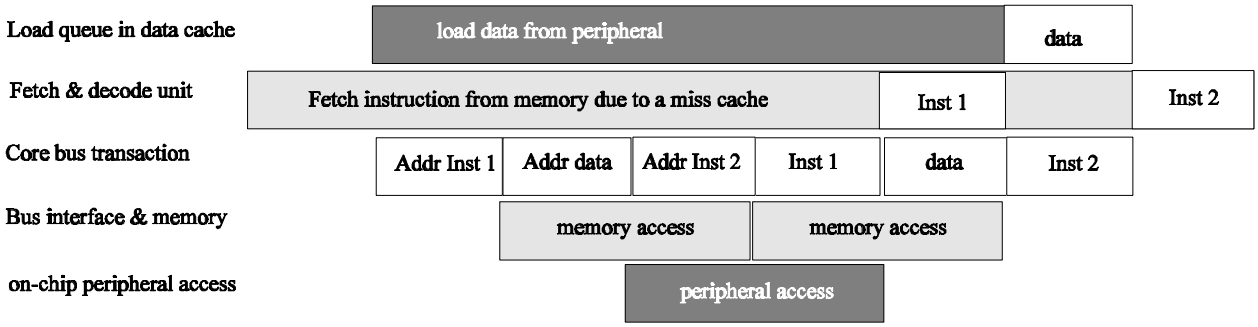


Figure 7. Example of interleaved transactions on core bus

In this example SPARClet™ is retrieving two instructions and one data in eight cycles. A regular RISC processor using traditional bus protocol will retrieve the two instructions and data in 16 cycles.

3.3 90C701 On-Chip Peripherals

3.3.1 Bus Interface Controller (BIC)

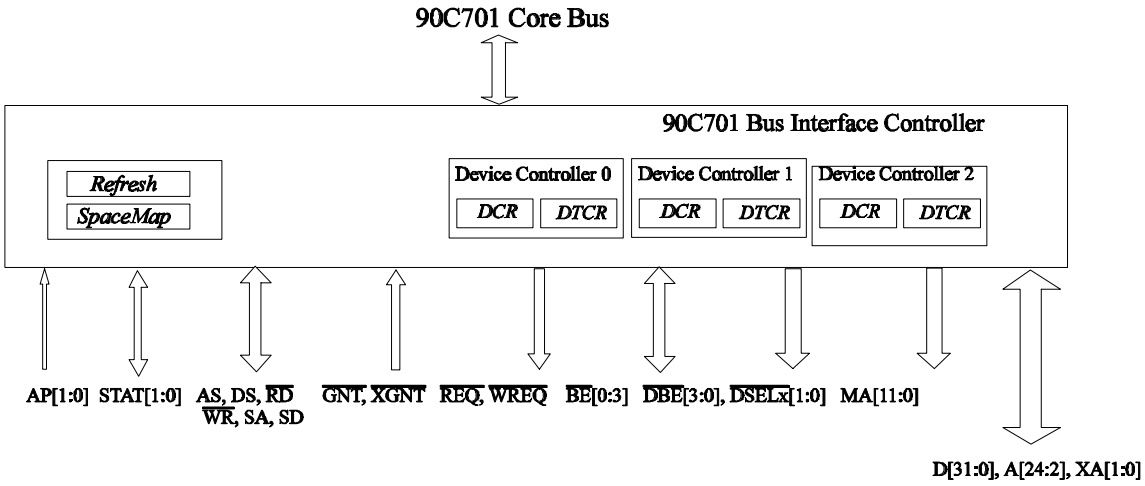


Figure 8. 90C701 Bus Interface Controller

The 90C701 Bus Interface Controller (BIC) provides direct handling of DRAM, SRAM, and ROM memories as well as external I/O devices. The BIC features include :

- up to 256 MB accessing
- 48 MB directly controlled by the 90C701
- three independent device controllers (2 banks of 8 MB per device controller)
- No-multiplexed address/data bus
- 8/16/32 bit data accesses
- 8-bit boot access
- large scope of devices supported (SRAM, ROM, DRAM, FIFO, I/O couplers)
- Direct Memory Access supported
- CAS before RAS refresh for selected devices
- Self-refresh support
- User/supervisor address space mapping
- External clock synchronous

Within the 256-MB Memory and I/O addressable Space (MIOS), the Bus Interface Controller directly control 48-MB without external glue logic. This direct decoded area is included in one of the four 64-MB addressable memory pages.

The 48-MB direct address space is splitted into three segments of 16-MB. Three independent device controllers in the BIC are responsible for mapping memories and I/O devices within these segments.

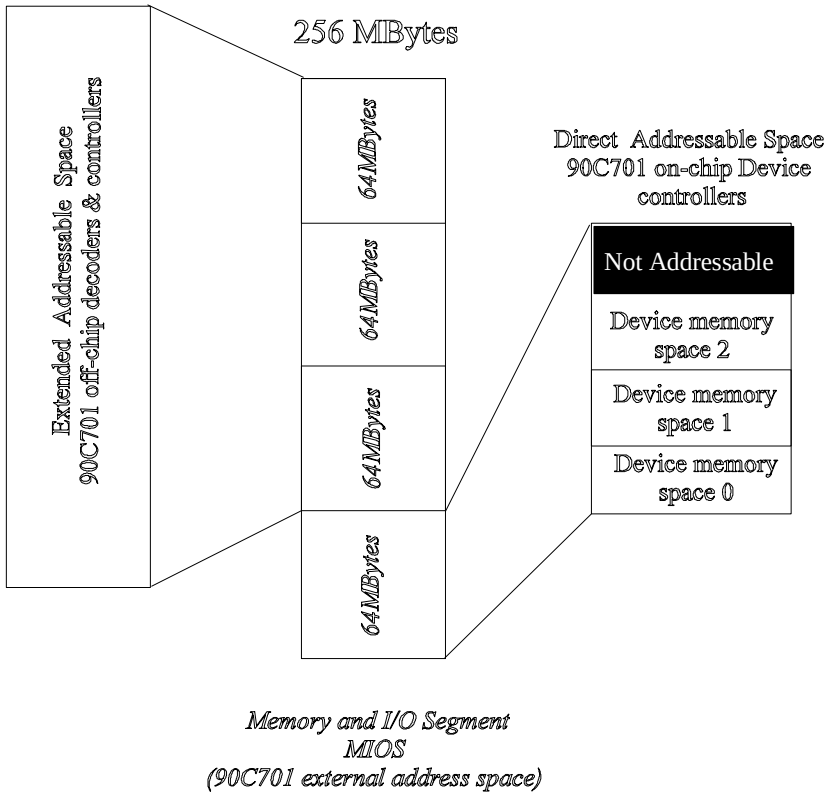


Figure 9. Memory and I/O Addressing Space (MIOS)

Each device controller provides a pair of programmable chip selects and all share the byte enable control signals. The table below shows three possible configurations using the BIC.

Device Controller	*Device*	*Configuration*
0	ROM	2 banks of 8MB
1	DRAM	2 banks of 8MB
2	Motorola type I/O device	2 I/O devices
0	ROM, DRAM	1 bk DRAM, 1 bk ROM
1	Intel type I/O device	1 I/O device
2	Motorola type I/O device	2 I/O devices
0	ROM, SRAM	2 banks of 8MB
1	DRAM	1 bk ROM, 1bk SRAM
2	Intel type I/O device	1 I/O devices

Table 13.Possible System Configurations

The waveform of the two signals (per device controller) is programmable. To control their timing, two BIC control registers per device controller, called DCR and DTCR, are available. The device control register (DCR) is dedicated to the general control of the device. The device timing control register (DTCR) contains timing information for the device accesses. The description of the register associated operations is provided in"Product Description" Chapter .

3.3.2 PCM/USART

The PCM/USART module is software-configurable as a PCM or USART interface.

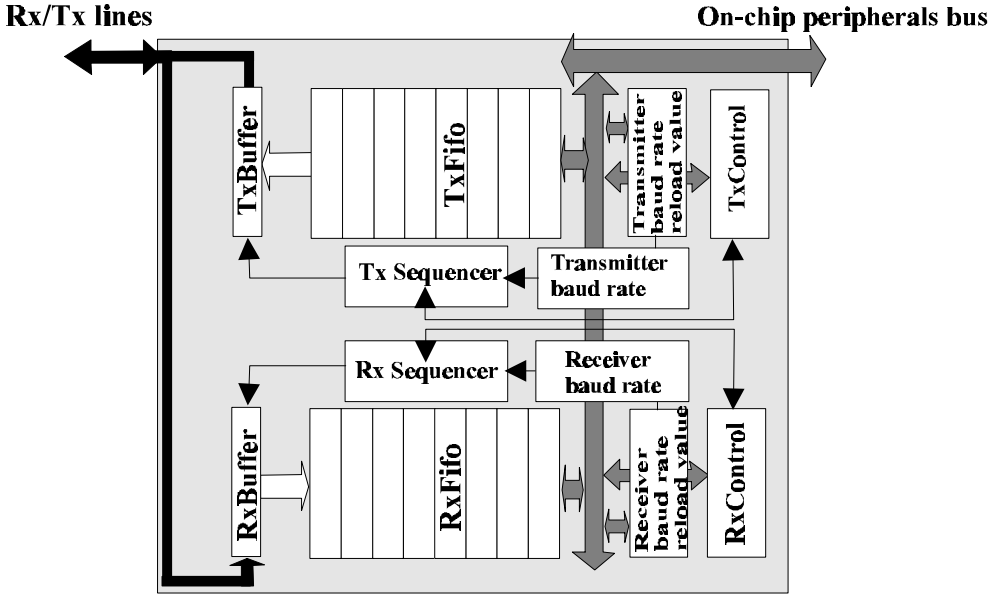


Figure 10.PCM/USART Block Diagram

When set up as a Universal Synchronous Asynchronous Receiver Transmitter (USART), each module features :

- 5- to 8-bit character length selection
- Parity bit option (Even, Odd, One, Zero, No Parity)
- 1 or 2 stop bits (asynchronous mode)
- Break character detection
- Parity, overrun and framing error detection
- Internal/External Clock
- Receiver and Transceiver Fifos
- Theoritical Speed up to 8 Mb/s at 50MHz

Running as a PCM interface, coupled with the internal Communication coprocessor, it supports the HDLC or V110 protocols up to 8 Mbits/s (Full Duplex).

3.3.3 General Purpose Timer

This is a general purpose timer based on two 16-bit counters : the counter and the scaler. It can generate interrupts and external waveforms. The timer is triggered by external events or system clock. The timer is controlled by six registers: the Input Handler, the Scaler, the Scaler Reload Register, the Counter, the Counter Reload Register, and the Shaper. The Input Handler rules the external pins configuration: edge or level counting, active edge, etc. The Shaper allows the generation of programmable duty cycles, thus providing the PWM capability.

The Input Handler contains the attributes of the external counting events.

3.3.4 OS Timers

This 32-bit decremental timer generates a trap at "0" detection. Depending on the Reload value, it can generate time reference intervals from one to 2^{32} clock cycle (86 s at 50 MHz). It serves to support the Operating System task scheduling.

3.3.5

The watchdog is an additional feature of the OsTimer, with a reduced functionality:

Load and Reset commands performed on the Counter Register will reload the watchdog while the Command Register content is discarded. The watchdog output signal has to be wired externally to the RESET_ input (possibly through external circuitry) or also to the reset inputs of an external peripheral).

As we use a 32-bit decremental timer, the Watchdog duration can be tweaked up to 2^{32} clock cycles. (86 s at 50 MHz)

3.3.6 Peripheral Interface Adapter (PIA)

This cell allows the attributes of a single port pin to be programmed. This is done by using a dedicated command register which determines :

- if the port is input or output
- any filtering functions on the port (polarity, noise reduction, level or edge detection and masking) are ruled by the same Input Handler as the timer's one.
- the interrupt level associated to the port

So any PIA external pin can be used as an input, an output, or as an external hardware interrupt.

Chapter III

Product Description

4 90C701 Programming Model

4.1 SPARC Compliance

The SPARClet™ architecture is compliant to SPARC V8 [1], and follows some of the recommendations proposed in SPARC V8 complement (the SPARC-V8 Embedded architecture specification) [2].

4.1.1 The 90C701 and the SPARC V8

SPARClet™ implements the SPARC V8 architecture specification as described in the architecture manual. This means that SPARC compatibility is respected and any developed SPARC V8 tool is directly applicable to SPARClet™. Some SPARClet™ implementation dependent features have been proposed in the 90C701, and those will be highlighted in this chapter.

In the following section an **[V8SID]** flag will refer to the V8 SPARClet™ Implementation Dependent features. For more details about the V8 architecture specifications refer to the official SPARC V8 manual.

4.1.2 The 90C701 and the SPARC V8 Complement - SPARC V8E

The SPARC V8 complement (SPARC V8E) was released by SPARC International in 1994. The SPARC V8E has better performance than the SPARC V8 to support real-time and embedded applications. The V8E specification recommends implementation and architecture enhancements at several levels, such as instructions, real-time I/O, tracing, and emulation techniques.

The SPARClet™ architecture follows some of these recommendations. In the following section a **[V8E]** flag will refer to the SPARClet™ V8E features. For more details about the V8E architecture specifications refer to the official SPARC V8E release 1 document.

[1] The SPARC Architecture Manual Version 8, SPARC International, Inc 535 Middlefield Road, Suite 210 Menlo Park, California 94025.

[2] SPARC-V8 Embedded (V8E) Release 1 Architecture Specification.

4.2 Memory Organization

The 90C701 can access to a 4-GByte address space. 256-MB of the MIOS (decribed previously) are offered to support external devices such as ROM, SRAM, DRAM, and peripheral devices.

The 4-GByte addressing space is divided into four segments:

Address	*Name*	*Usage*	*Space*
0x3FFFFFFF-0x00000000	CMS	Cacheable Memory Segment	1 GB
0x7FFFFFFF-0x40000000	NCMS	Not Cacheable Memory Segment	1 GB
0xBFFFFFFF-0x80000000	IOS	Input/Output Segment	1 GB
0xFFFFFFFF-0xC0000000	SCS	System Control Segment	1 GB

Only 512-MB can be physically addressed per segment. According to address bit A29, one location (@) within this 512-MB space can be accessed by two logical address locations *@+0x20000000* and *@+0x00000000*, as shown in the following figure.

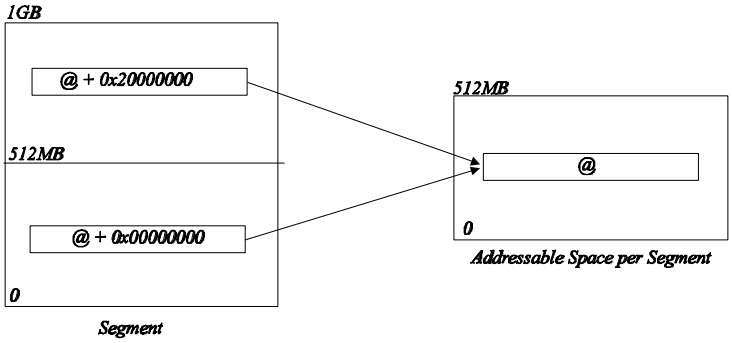


Figure 11.Segment Organization

4.2.1 System Control Segment (SCS)

The system Control Segment (SCS) is only accessible in supervisor mode and includes the instruction and data cache control registers and the bus interface controller registers.

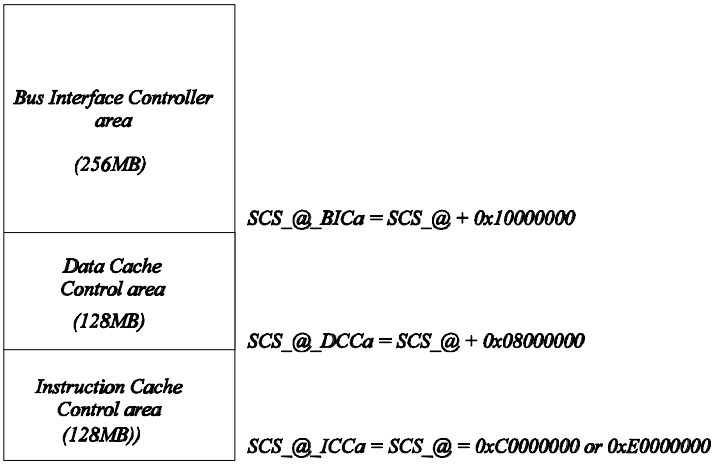


Figure 12.System Control Segment

4.2.2 Input/Output Segment (IOS)

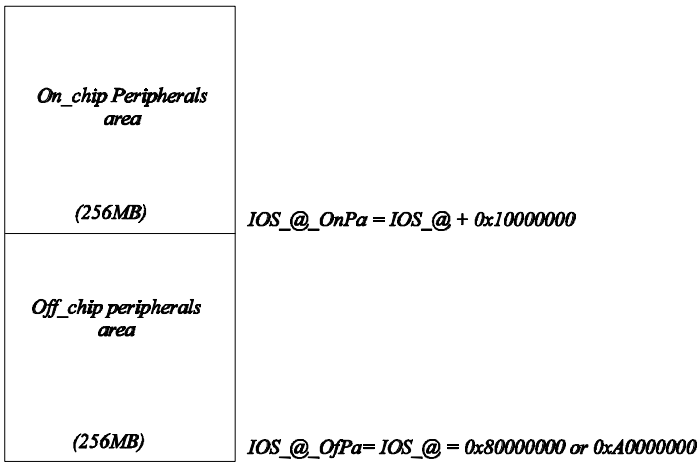


Figure 13.Input/Output Segment

The Input/Output Segment (IOS) has two subsegments. The higher subsegment is allocated to the on-chip peripherals. The following table shows the 90C701 on-chip peripherals mapping.

Offset in IOS_@_OnPa	*On-chip peripheral*	*Designation*
0x2600000	USART/PCM3	Synchronous Serial Interface
0x2400000	USART/PCM2	Synchronous Serial Interface
0x2200000	USART/PCM1	Synchronous Serial Interface
0x2000000	USART/PCM0	Synchronous Serial Interface
0x1E00000	Watchdog	System Watchdog
0x1C00000	OSTimer1	Operating System Timer
0x1A00000	OSTimer0	Operating System Timer
0x1800000	GPTimer1	General Purpose Timer
0x1600000	GPTimer0	General Purpose Timer
0x1400000	PIA9	Peripheral Interface Adapter
0x1200000	PIA8	Peripheral Interface Adapter
0x1000000	PIA7	Peripheral Interface Adapter
0x0E00000	PIA6	Peripheral Interface Adapter
0x0C00000	PIA5	Peripheral Interface Adapter
0x0A00000	PIA4	Peripheral Interface Adapter
0x0800000	PIA3	Peripheral Interface Adapter
0x0600000	PIA2	Peripheral Interface Adapter
0x0400000	PIA1	Peripheral Interface Adapter
0x0200000	PIA0	Peripheral Interface Adapter
0x0000000	CLK Ctrl	Clock Management

Table 14.90C701 on-chip peripherals mapping

The lower subsegment is dedicated to the off-chip peripherals. These are decoded by the Bus interface controller. This segment is one way to address the resources available in the **MIOS** area, which has been described in the previous chapter **90C701 On-Chip Peripherals** section **Bus Interface Controller (BIC).**

4.2.3 Not Cacheable Memory Segment (NCMS)

This segment is the second way to address the memory available in the MIOS area. Note, however, that all memory accesses will be not cacheable. The caches will be ignored in this area .

4.2.4 Cacheable Memory Segment (CMS)

This is the third an last way to address the MIOS area. All the accesses will go through the two caches before going to the available memory.

The 512-MB of these two last segments (NCMS/CMS) is divided in two 256-MB subsegments. The lower subsegment is reserved for 8-bit MIOS resource accesses (only load/fetch accesses authorized). The higher subsegment is dedicated to 32-Bit MIOS resource accesses. The lower subsegment can be used to boot on 8-bit boot ROM.

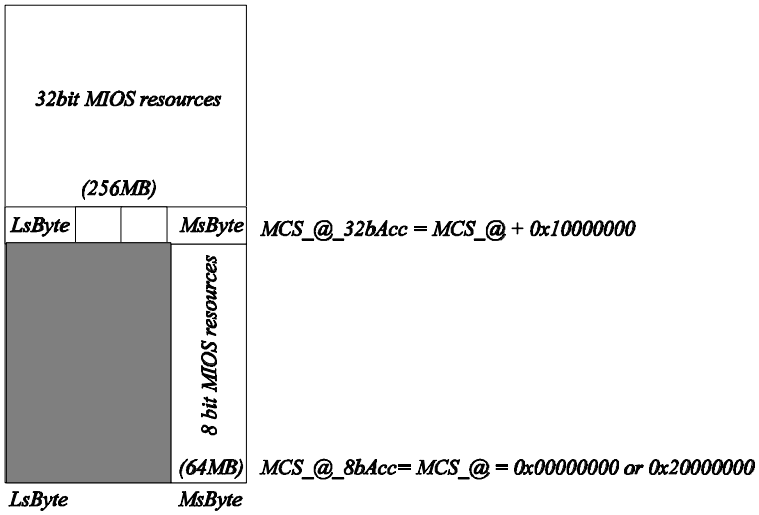


Figure 14.Cacheable Memory Segment

4.2.5 Logical to Physical MIOS Addresses Translation

The redundant mapping from the logical 4-GByte to the physical 256-MB MIOS space, provides the best flexibility for the system configuration. All possible views are offered in the MIOS area. i.e. SRAM, ROM, I/O, 8-bit, 32-bit, cacheable, not cacheable. The following figure summarizes the three logical addressing spaces of the MIOS.

(1) MIOS is viewed as off-chip peripherals area
(2) MIOS is viewed as not cacheable 32-bit and 8-bit area
(3) MIOS is viewed as cacheable 32-bit and 8-bit area.

A complete system can be configured following a combination of these three views.

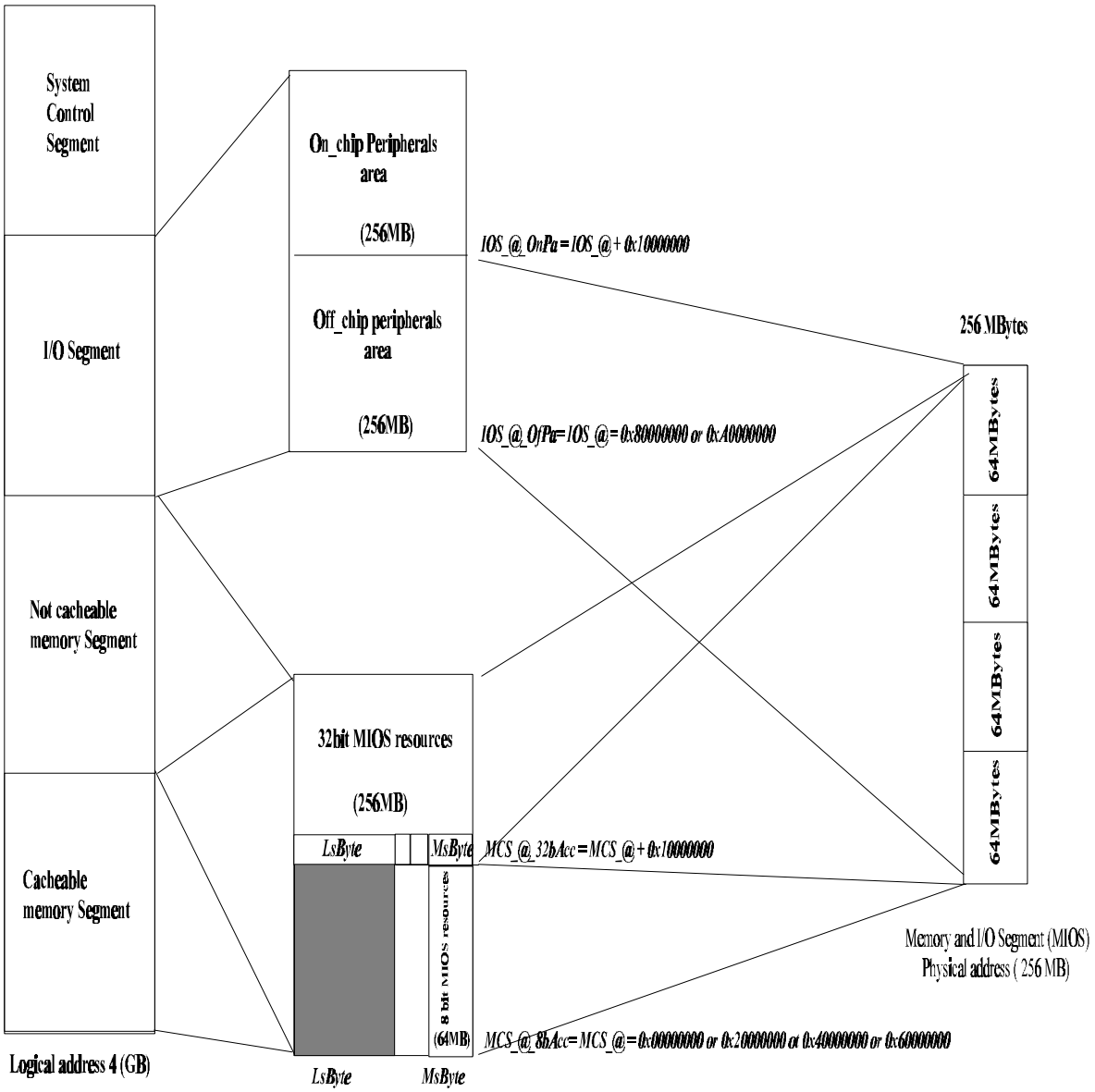


Figure 15.Logical to Physical MIOS Addresses Translation.

Boot at Reset

At reset, the 90C701 will fetch first instructions starting at the address 0x0. The processor addresses the boot ROM, assuming the availability of an 8-bit boot ROM device even if the physical ROM is 32-bit data width. The Device controller will handle the generation of the four addresses to the 8-bit boot ROM to rebuild the 32-bit word fetched by the 90C701. If a 32-bit boot ROM is used, then consecutive byte addresses of the bootstrap have to be written to the ROM device, taking account of word alignment as shown in the following example :

```
Bootstrap

0x0                     sethi %hi(_Start),%g1
0x4                     jmp   %g1 + %lo(_Start)
0x8                     nop
:
:
0x10000030(_Start)      nop     ;first instruction of the 32bit code segment
0x10000034              sethi %hi(0), %l0


BootROM (physical implementation)

0x0                     first byte of sethi %hi(_Start),%g1
0x4                     second byte of sethi %hi(_Start),%g1
0x8                     third byte of sethi %hi(_Start),%g1
0xC                     fourth byte of sethi %hi(_Start),%g1

0x10                    first byte of jmp  %g1 + %lo(_Start)
0x14                    second byte of jmp  %g1 + %lo(_Start)
0x18                    third byte of jmp  %g1 + %lo(_Start)
0x1C                    fourth byte of jmp  %g1 + %lo(_Start)

0x20                    first byte of nop
0x24                    second byte of nop
0x28                    third byte of nop
0x2C                    fourth byte of nop

0x30                    nop     ;(_Start label)
0x34                    sethi %hi(0), %l0
```

4.3 Data Types and Alignment

The 90C701 recognizes two fundamental data types:

Signed Integer : 8, 16, 32, and 64 bits.
Unsigned Integer : 8, 16, 32, and 64 bits

The format widths are defined as:

Byte : 8-bit
Halfword : 16-bit
Word/Singleword : 32-bit
Tagged Word : 32-bit (30-bit value plus 2 tag bit)
Doubleword : 64-bit

Halfword accesses must be aligned on a 2-byte boundary; word accesses (which include instruction fetches) must be aligned on a 4-byte boundary; and double-word accesses must be aligned on an 8-byte boundary. An improperly aligned address causes a load or store instruction to generate a mem_address_not_aligned trap. SPARC V8 is a big-endian architecture. However, SPARClet ™ is capable of handling **big and little endian architectures**. This feature improves flexibility and performance in embedded systems using different types of peripherals. The addressing mode can be selected with the field LE in the Processor State Register.

4.4 Registers

The 90C701 includes two types of registers: general-purpose or "working" data registers and control/status registers. The 90C701 CPU general-purpose registers are called *r* registers, and the communication coprocessor working registers are called *cp* registers. The 90C701 CPU control/status registers include:

Processor State Register (PSR) [V8SID]
Window Invalid Mask (WIM)
Trap Base Register (TBR)
Multiply/Divide Register (Y)
Program Counters (PC, nPC)
Ancillary State Registers (ASR0, 1, 15, 17, 18, 19, 20, 21, 22)[V8SID][V8E]
Communication coprocessor State Register (CSR) [V8SID]

The 90C701 contains 176 general-purpose 32-bit *r* registers. They are partitioned into 8 *global* registers, plus 8 *sets* composed of 8 *local* and 8 *in* registers, plus 32 *alternate* registers. The *global* register r[0] produces the value zero. If the destination field indicates a write into r[0], no register is modified and the result is discarded.

A register window comprises the 8 *in* and 8 *local* registers of a particular register set, together with the 8 *in* registers of an adjacent registers set, which are addressable from the current window as *out* registers. See the figure in the Chapter **SPARClet™ Architecture.**

The 32 *alternate* registers are viewed as an *alternate window* registers [V8E] and provide the support for fast context switching on interrupt. The usage of the Alternate window registers is based on the value of the AW bit (Alternate Window Bit) of the PSR (if AW=1, the alternate set of registers is used). The alternate window registers are organized as shown in the following figure. The two first registers are used to save the PC, and nPC values.

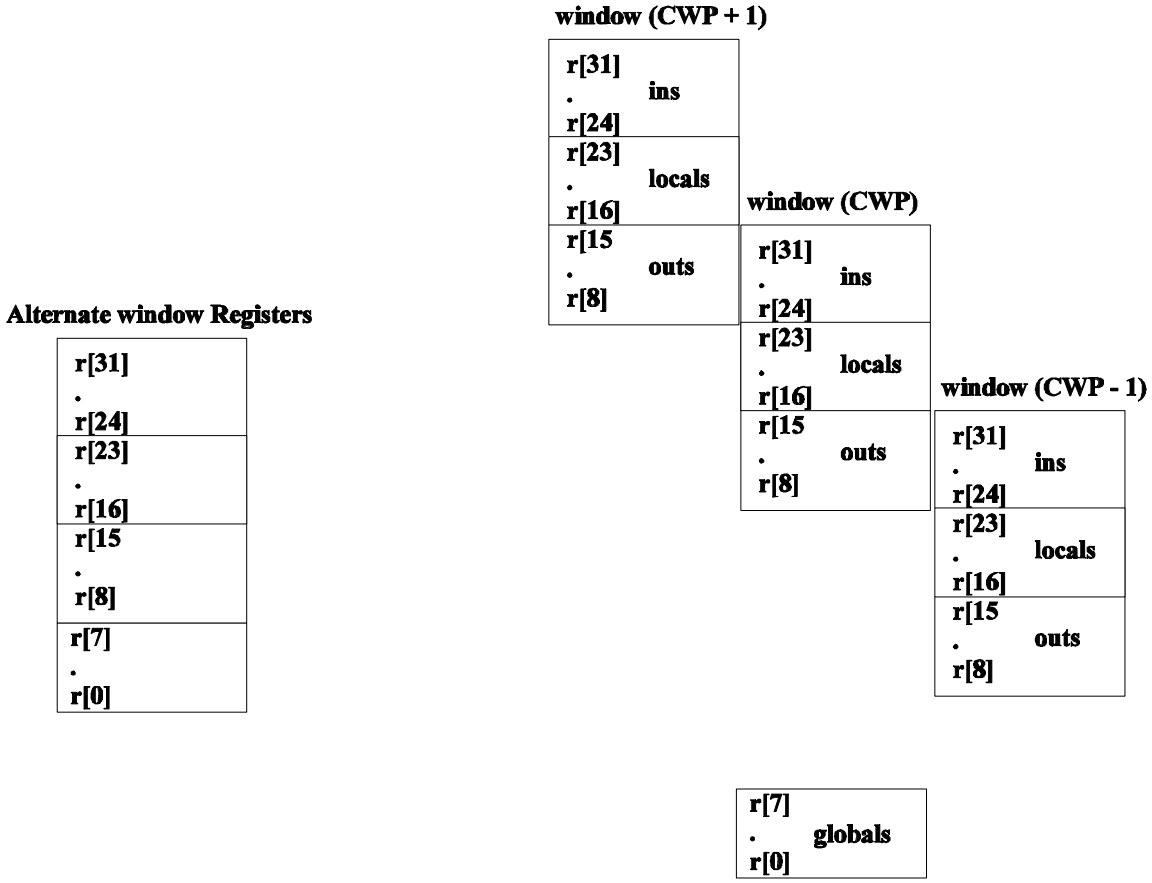


Figure 16.Alternate Window Registers

Because WIM, TBR, Y, PC, nPC registers have no addition to SPARC V8 definition, they will not be described here. Only the PSR, the ASRs, and the CSR will be described in the following sections.

4.4.1 Processor State Register (PSR)

Two bits have been added to the V8 *PSR_reserved* field. The bit LE (Little Endian), when set, inverts the two less significant bits of the addresses. The bit AW (Alternate Window) , when set, allows the alternate window mode. The bit EE (Enable Extension), when asserted, enables the AW , LE associated functionalities and extends the trap model using the type registered in *ASR17_trap_mode*l field. Otherwise the default trap model is the precise traps.

31-28	27-24	23-20	19-17	16	15	14	13	12	11-8	7	6	5	4-0
IMPL	*VER*	*ICC*	*reserved*	*AW*	*LE*	*EE*	*EC*	*EF*	*PIL*	*S*	*PS*	*ET*	*CWP*

Table 15.90C701 Processsor State Register (PSR)

4.4.2 Ancillary State Registers (ASRs)

The 90C701 implements nine specific ancillary state registers:

ASR	*Description*
0	Copy of Y register
1	Implementation Extension Register [V8E]
15	Nop register. RDASR from ASR15 creates a nop equivalent instruction.
17	Copy of ASR1 register (IER)
18	Performance Counting register. This register can be used for measure.
19	Stop register. RDASR from ASR19 stops the processor.
20	Fault address register. Contains the address of the fault creating instruction.
21	Fault Status register
22	Alternate Window Configuration register

The following figures show the different fields of ASR17, 18, 21, and 22.

Implementation Extension Register (IER-ASR17)

31-14	13-12	11-2	1	0
Reserved	*_trap_model*	*_trap_base_offset*	*Reserved*	*_single_vector_trap*

Table 16. Implementation Extension Register (ASR17)

ASR17_single_vector_trap — Bit 0. When asserted all traps target one single vector. The address is the addition of *TBR_trap_base_address* and *ASR17_trap_base_offset*

ASR17_trap_base_offset — Bit 2 through 11. Offset to be added to *TBR_trap_base_address* when single vector trap mode is selected.

ASR17_trap_model — Bit 12 through 13. This field indicates whether all traps must be precise, or if deferred traps or/and interrupting traps have been allowed.

Performance Counting register (PCR-ASR18)

31-16	15 -0
_timer_control	*_timer_count*

Table 17.Performance Counting Register (ASR18)

ASR18_timer_count Bit 0 through 15. This field is used as a 16-bit timer.

ASR18_timer_control Bit 16 through 31. This field is dedicated to the control of the timer. *(Operations will be described in a forthcoming document.*

Fault Status Register (FSR - ASR21)

31-7	6	5	4	3-2	1	0
reserved	*_store*	*_privilege_violation*	*_full_support*	*_reserved*	*_double_fault*	*_error*

Table 18.Fault Status Register (ASR21)

ASR21_error Bit 0. When asserted an error has occurred.

ASR21_double_fault Bit 1. Indicates that a trap occured while ET=0 (Traps disabled).

ASR21_full_support Bit 4. Indicates that the Fault Address Register content is correct.

ASR21_privilege_violation Bit 5. When set, a user/supervisor access violation has been detected.

ASR21_store Bit 6. Indicates that the fault was created by a store instruction

note : LDSTB and SWAP are classified as Load instructions.

Alternate Window Configuration Register (AWCR - ASR22)

31-16	15	14-12	11-4	3-0
_IRL_mask	*_double_fault_mask*	*Reserved*	*_trap_type*	*Reserved*

Table 19.Alternate Window Configuration Register (ASR22)

ASR22_trap_type Bit 4 through 12. The trap type as defined in V8.

ASR22_double_fault_mask Bit 15. When asserted it authorizes the double fault detection (so avoiding the Error Mode).

ASR22_IRL_mask Bit 16 through 31. 1 bit for one interrupt level (15). When set, the interrupt level will be associated to the alternate window registers.

Communication coprocessor State Register (CSR)

The CSR register fields contain Communication Coprocessor mode and status information. The CSR is read and written by the CHRDCXT and CHWRCXT instructions.

31-26	25-20	19-14	13	12	11-9	8	7-5	4	3	2	1	0
_outbuf _count	*_inbuf _count*	*_inreg _count*	*_inbuf _asis*	*_inreg _asis*	*_mode*	*Reserved*	*_coder _decoder _count*	*_decode _error*	*_end _of _message*	*_outreg _full*	*_inreg _empty*	*_cp _freeze*

Table 20.Coprocessor State Register

CSR_outbuf_count — Bits 31 through 26 are the indication of significant bits in coprocessor output buffer.

CSR_inbuf_count — Bits 25 through 20 are the indication of the significant bits in coprocessor input buffer.

CSR_inreg_count — Bits 19 through 14 are the indication of significant bits in coprocessor input register.

CSR_inbuf_asis — Bit 13 indicates that the corresponding data chunk in the input buffer has to be processed without translation (for start and end flags in HDLC mode for example).

CSR_inreg_asis — Bit 12 indicates that the corresponding data chunk in the input register has to be processed without translation (for start and end flags in HDLC mode for example).

CSR_mode — Bits 9 through 11 are used to specify the type of operation to be performed on the bit stream by the co-processor; i.e.. HDLC decode/encode, V110 decode/encode.

CSR_coder_decoder_count — Bits 5 through 7 indicates a count used for the coder and decoder processing.

CSR_decode_error — Bit 4 indicates that an abnormal end of message has been detected; i.e. 7 consecutive "1" have been seen on the input bit stream while decoding HDLC.

CSR_end_of_message — Bit 3 indicates that an end of message condition has been detected while HDLC decoding.

CSR_outreg_full — Bit 2 indicates if the output register does or does not contain significant data (32 bits).

.

CSR_inreg_empty — Bit 1 indicates if the input register contains significant data or garbage. The number of significant bits is given by the CSR_inreg_count field.

CSR_cp_freeze — Bit 0 when asserted the coprocessor context is frozen.

4.5 Branching Control

In addition to the SPARC V8 control-transfer instructions (CTI), the SPARClet ™ architecture enhances the performance of the "branch on integer condition codes" instructions (Bicc).

To reduce the number of cycles per instruction in the intensive processing loop, all 90C701 **Bicc** instructions will **predict** the issue of the transfer if the condition code is generated during the previous cycle. In that case, the prediction is made "backward", i.e. predicted as taken for backward branches and as not taken for forward branches. Missing on a prediction will cost 1 wait state.

The 90C701 implements also the "branch on coprocessor condition codes" [V8SID] instructions.

4.6 Interrupts, Traps, and Exceptions

Traps are controlled by two registers: exception and interrupt requests via the enable traps (ET) field in the PSR, and interrupt requests via the processor interrupt level (PIL) field in the PSR.

We suggest that designers refer to the V8 manual to analyze the ET and PIL control schemes. The 90C701 implements three "implementation-dependent-exceptions". Two are reporting errors in the communication co-processor and one is reporting a double fault situation. The following table shows all exceptions and interrupt request priority and tt values of the 90C701.

double_fault — This trap occurs when an exception appears while ET=0 if the current window is not the Alternate Window Register. This exception is helpful in the software development phase (It avoids entering in Error Mode).

cp_push_error — This exception occurs when the CPPUSH instruction is executed and the *CSR_inreg_empty* is not set (=0). This trap means that an initiative to push data to the co-processor input register has been taken while this register is already occupied by data.

cp_pull_error — This exception occurs when the CPPULL instruction is executed and the *CSR_outreg_full* is not set (=0). This trap means that an initiative to pull data from the co-processor output register has been taken while this register has no data.

The 31 different interrupt sources of the 90C701 can be mapped on the 15 available interrupt levels. A specific field is reserved in the peripheral control registers to associate the IRL (Interrupt Request Level) value to the interrupt source. The 90C701 interrupt sources are listed in the following table.

Exception or Interrupt Request	*Priority*	*tt*
reset	1	NA
double_fault	2	0x6F
data_store_error	2	0x2B
instruction_access_error	2	0x3B
instruction_access_exception	5	0x01
privileged_instruction	6	0x03
illegal_instruction	7	0x02
cp_disabled	8	0x24
window_overflow	9	0x05
window_underflow	9	0x06
mem_address_not_aligned	10	0x07
data_access_error	12	0x29
data_access_exception	13	0x09
tag_overflow	14	0x0A
cp_push_error	15	0x64
cp_pull_error	15	0x65
trap_instruction	16	0x80 - 0xFF
interrupt_level_15	17	0x1F
interrupt_level_14	18	0x1E
interrupt_level_13	19	0x1D
interrupt_level_12	20	0x1C
interrupt_level_11	21	0x1B
interrupt_level_10	22	0x1A
interrupt_level_9	23	0x19
interrupt_level_8	24	0x18
interrupt_level_7	25	0x17
interrupt_level_6	26	0x16
interrupt_level_5	27	0x15
interrupt_level_4	28	0x14
interrupt_level_3	29	0x13
interrupt_level_2	30	0x12
interrupt_level_1	31	0x11

Table 21.Exception and Interrupt Request Priority and tt Values

Source	*Peripherals*
usart_pcm_0_tx_error usart_pcm_0_rx_error usart_pcm_0_tx_char_ready usart_pcm_0_rx_char_received	USART/PCM 0
usart_pcm_1_tx_error usart_pcm_1_rx_error usart_pcm_1_tx_char_ready usart_pcm_1_rx_char_received	USART/PCM 1
usart_pcm_2_tx_error usart_pcm_2_rx_error usart_pcm_2_tx_char_ready usart_pcm_2_rx_char_received	USART/PCM 2 (90C701B)
usart_pcm_3_tx_error usart_pcm_3_rx_error usart_pcm_3_tx_char_ready usart_pcm_3_rx_char_received	USART/PCM 3 (90C701B)
os_timer_0_event_done	OSTIMER 0
os_timer_1_event_done	OSTIMER 1
watchdog_event_done	WATCHDOG
gp_timer_0_event_done	GPTIMER 0
gp_timer_1_event_done	GPTIMER 1
pia_b0_edge_level pia_b1_edge_level pia_b2_edge_level pia_b3_edge_level pia_b4_edge_level pia_b5_edge_level pia_b6_edge_level pia_b7_edge_level pia_b8_edge_level pia_b9_edge_level	PIA

Table 22.90C701 Interrupt sources

4.7 90C701 Additional Instructions

The 90C701 implements the following application specific instructions: SCAN, SHUFFLE, Write Communication Coprocessor Register, Read Communication Coprocessor Register, Push data to Communication coprocessor, Pull data from Communication Coprocessor, Multiply and Accumulate Instructions and Branch on Communication Coprocessor condition code.

Table 23. 90701 Additional Instruction Set

Mnemonic	Operands	Description
UMAC	reg_source1, reg_source2(or imm), reg_dest	Unsigned multiply and accumulate
SMAC	reg_source1, reg_source2(or imm), reg_dest	Signed multiply and accumulate
UMACd	reg_source1, reg_source2(or imm), reg_dest	Unsigned double operand Multiply and Accumulate using Y register to hold the most significant bits
SMACd	reg_source1, reg_source2(or imm), reg_dest	Signed double operand Multiply and Accumulate using Y register to hold the most significant bits
UMULd	reg_source1, reg_source2(or imm), reg_dest	Unsigned double operand Multiply using Y register to hold the most significant bits
SMULd	reg_source1, reg_source2(or imm), reg_dest	Signed double operand Multiply using Y register to hold the most significant bits
SCAN	reg_source1, reg_source2(or imm), reg_dest	Identify most significant set/cleared bit in data item
SHUFFLE	reg_source1, reg_source2(or imm5), reg_dest	Bit, couples, nibble, bytes or half words swapping
CPWRCXT	reg_source1, context_reg	Update a co-processor context register from IU register
CPRDCXT	context_reg, reg_dest	Save a co-processor context register into IU register
CPPUSH	reg_source1,reg_source2	Pushes a chunk of data into coprocessor input register
CPPUSHA	reg_source1,reg_source2	Pushes a chunk of data into coprocessor input register with Asis bit set.
CPPULL	reg_dest	Pulls a 32 bits of data from coprocessor output register
CBccc	label	Branch on Coprocessor Condition Codes

The following SPARC V8 Instructions are not implemented : MULScc , UMULcc , SMULcc , UDIV , UDIVcc , SDIV , SDIVcc , FLUSH , All FP instructions , RDASR and WRASR for unimplemented ASRs

The following SPARC V8E Instruction is not implemented : DIVScc

4.7.1 SCAN instruction

Table 24. SCAN instruction

opcode	op3	operation
SCAN	101 100	Scan for first occurence of "1" or "0" bit

Format (3) :

31-30	29-25	24-19	18-14	13	12-5	4-0
10	Destination (rd)	opcode (op3)	Source 1 (rs1)	0	Unused (0)	Source 2 (rs2)
10	Destination (rd)	opcode (op3)	Source 1 (rs1)	1	13 bit immediate	

Description:

The SCAN instruction returns the location of the first bit in %rs1 that differs of the value of the most significant bit of %rs1 or the location of the first "1" or "0" bit of source register %rs1. SCAN works as follows:

1) The value of %rs1 is xored on a bit-wise basis with the mask obtained by shifting right by one bit and sign extending the content of %rs2 (or imm13 if the immediate bit is set).

2) The number of the bit position of the first "1" in the resulat from 1) above is returned to the destination register %rd. Bit numbers range from 0 for the most significant bit to 31 for the least significant bit. A "1" in the most significant bit (MSB) position returns a value of 0, while the first "1" in the least significant bit (LSB) position returns a value of 31.

Suggested assembly language syntax:

```
scan        %rs1,%rs2 (or imm13),%rd
```

Traps : None

4.7.2 SHUFFLE instruction

Table 25. SHUFFLE instruction

opcode	op3	operation
SHUFFLE	101 101	Swaps the adjacent bits, couples, nibbles, bytes or half words

Format (3) :

31-30	29-25	24-19	14-18	13	12-5	4-0
10	Destination (rd)	opcode (op3)	Source 1 (rs1)	0	Unused (0)	Source 2 (rs2)
10	Destination (rd)	opcode (op3)	Source 1 (rs1)	1	13 bit immediate	

Description :

Using the 5 less significant bits of %rs2 or imm13 as operand, this instruction swaps:

- adjacent bits of %rs1 (if rs2[0] is set)
- then adjacent couples of bits of the result (if rs2[1] is set)
- then adjacent nibbles of the result (if rs2[2] is set)
- then adjacent bytes of the result (if rs2[3] is set)
- then adjacent half words of the result (if rs2[4] is set).

The final result is stored in the the %rd register.This instruction can be used to switch from little endian to big endian or inversely, or to perform any kind of 32 bits data shuffling. In case %rs2 is used as operand, rs2[31:5] bits have to be set as "0".

Suggested assembly language syntax :

`shuffle` *%rs1,%rs2 (or imm13),%rd*

Traps : None

4.7.3 MAC instructions

Table 26. MAC instructions

opcode	op3	operation
UMAC	111110	Unsigned multiply and accumulate
UMACd	101110	Unsigned multiply and accumulate with double operand
SMAC	111111	Signed multiply and accumulate
SMACd	101111	Signed multiply and accumulate with double operand
UMULd	001001	Unsigned multiply with double operand
SMULd	001101	Signed multiply with double operand

Format (3):

31-30	29-25	24-19	14-18	13	12-5	4-0
10	Destination (rd)	opcode (op3)	Source 1 (rs1)	0	Unused (0)	Source 2 (rs2)
10	Destination (rd)	opcode (op3)	Source 1 (rs1)	1	13 bit immediate	

Description :

The MAC instructions perform a multiplication of the two source operands and accumulate the result in the Y register (most significant bits) concatenated with the rd register (less significant bits).

SMAC and UMAC perform the following computation:

Y..rd = Y..rd + (rs1 * rs2 (or imm13,depending of the immediate bit value))
The computation is signed for SMAC and unsigned for UMAC.

UMACd and SMACd perform the following computation:

Y[8::0]..r(d + 1)..rd = Y[8::0]..r(d + 1)..rd + (rs1 * rs2 (or imm13))

Y[31] is set if an overflow (for SMACd) or a carry (for UMACd) occurred in the previous equation, and stays untouched otherwise.
UMUL and UMULd work the same way as UMAC and UMACd with Y and rd initialized at 0x0 before starting the computation.

Suggested assembly language syntax :

```
umac    %rs1,%rs2 (or imm13),%rd
umacd   %rs1,%rs2 (or imm13),%rd
smac    %rs1,%rs2 (or imm13),%rd
smad    %rs1,%rs2 (or imm13),%rd
umul    %rs1,%rs2 (or imm13),%rd
umuld   %rs1,%rs2 (or imm13),%rd
smul    %rs1,%rs2 (or imm13),%rd
smuld   %rs1,%rs2 (or imm13),%rd
```

Traps: None

Note : ".." stands for the concatenation

4.7.4 CPRDCXT / CPWRCXT: Read / Write an Communication Coprocessor Context Register

Table 27. CPRDCXT / CPWRCXT Instruction Set

opcode	op3	opc	operation
CPRDCXT	110 110	000000100	Read Coprocessor Context Register
CPWRCXT	110 110	000000011	Write Coprocessor Context Register

Format (3):

31-30	29-25	24-19	18-14	13-5	4-0
10	Destination (rd)	opcode (op3)	Source 1 (rs1)	CP Opcode (opc)	Source 2 (rs2)

Description:

These 2 instructions are used to read/write the Communication coprocessor internal register set : %CSR, %FIFO, %POLY and %CRC from/to a window register. Note that the CPWRCXT %rs1,%FIFO is not suitable to load the input of the Coder/decoder, as it would not write the InReg Count in the CSR register. The CPPUSH instruction has been dedicated to this purpose.

Suggested assembly language syntax:

```
cprdcxt     %cpreg,%rd
cpwrcxt     %rs1,%cpreg
```

Traps: cp_disabled

4.7.5 CPPUSH[a] :

Table 28.CPPUSH[a] Instruction Set

opcode	op3	opc	operation
CPPUSH	110 110	000000000	Pushes a bit sequence of up to 32 bits in the coder/decoder input register
CPPUSHa	110 110	000000001	Pushes a bit sequence of up to 32 bits in the coder/decoder input register with Asis bit set

Format (3) :

31-30	29-25	24-19	18-14	13-5	4-0
10	Destination (rd)	opcode (op3)	Source 1 (rs1)	CP Opcode (opc)	Source 2 (rs2)

Description:

This instruction writes a stream of up to 32 bits from %rs1 to %InReg . It simultaneously writes the number of significative bits in %InReg (less or equal to 32 bits) from the 6 less significative bits of %rs2 to the %InReg Count field of the CSR. The CPPUSHa instruction sets the %InReg Asis bit of the %CSR, thus indicating that the content of %InReg has to be processed with no translation.If %InReg is not detected as empty when this instruction is executed, the coprocessor will generate a coprocessor exception.

Suggested assembly language syntax :

```
cppush      %rs1,%rs2
cppusha     %rs1,%rs2
```

Traps : cp_disabled
cp_push_error

4.7.6 CPPULL

Table 29.CPPULL Instruction Set

opcode	op3	opc	operation
CPPULL	110 110	000000010	Pulls a 32 bits processed sequence out of the coder/decoder output register

Format (3):

31-30	29-25	24-19	18-14	13-5	4-0
10	Destination (rd)	opcode (op3)	Source 1 (rs1)	CP Opcode (opc)	Source 2 (rs2)

Description :

The CPPULL transfers the content of the %OutReg register to a window register. The OutReg Full field of the CSR register must be set when this instruction is executed otherwise a Coprocessor Exception will be generated.

Suggested assembly language syntax :

```
cppull          %rd
```

Traps : cp_disabled
cp_pull_error

4.7.7 CBccc

Table 30.CBccc Instruction Set

opcode	op2	cond	operation
CBccc	111	xxxx depends on the condition	Branch on HDLC coprocessor condition code

Format (2):

31-30	29	29-25	24-22	21-0
0	a	Test Condition	Opcode (op2)	22-Bit displacement

Description:

The CBccc performs a branch to the defined label if the condition code is true. As for the other branches , the "annul bit" can be set or not to control the execution of the instruction located in the delay slot that immediately follows the transfer instruction .The address of the targeted label is calculated as:

PC (actual value of the Program Counter) + (sign extnd (disp 22) * 4)

Suggested assembly language syntax:

`cbn{,a}`	- never taken branch
`cbe{,a}`	- branch if %InReg is empty
`cbf{,a}`	- branch if %OutReg full
`cbef{,a}`	- branch if %InReg is empty or %OutReg full
`cbr{,a}`	- branch if the coprocessor is running
`cber{,a}`	- branch if %InReg is empty or if the coprocessor is running
`cbfr{,a}`	- branch if %OutReg is full or if the coprocessor is running
`cbefr{,a}`	- branch if %InReg is empty or %OutReg full or if the coprocessor is running
`cba{,a}`	- branch always taken
`cbne{,a}`	- branch if %InReg is not empty
`cbnf{,a}`	- branch if %OutReg is not full
`cbnef{,a}`	- branch if %InReg is not empty and OutReg is not full
`cbnr{,a}`	- branch if the coprocessor is not running
`cbner{,a}`	- branch if %InReg is not empty and the coprocessor is not running
`cbnfr{,a}`	- branch if %OutReg is not full and the coprocessor is not running
`cbnefr{,a}`	- branch if %InReg is not empty and %OutReg is not full and the coprocessor is not running

Traps: cp_disabled

5 90C701 Operations and Register Description

5.1 Communication Coprocessor

The coprocessor is designed to enhance the 90C701's performance when supporting HDLC or V110 communication protocols. It processes bit stream by blocks of up to 32 bits. The coprocessor is composed of two independent entities: the CODER/DECODER and the CRC generator.

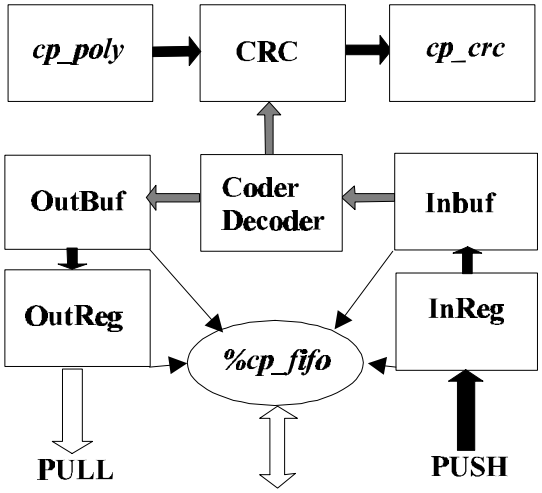


Figure 17.Communication Coprocessor Block Diagram

The CODER/DECODER run continuously based on the transcoding sequence programmed in *cp_StateReg* (either HDLC or V110) : this is called the background process. The CPPUSH or CPPUSHA instruction loads *cp_InReg* with a new block of up to 32 bits of data to be processed (CPPUSHA sets the Asis bit up, which means that the bit stream has to be processed as is , without any translation. This keeps the frame start and stop sequences unchanged).

In parallel, the count of valid bits entered in *cp_InReg* is written to the corresponding field in *CSR* (Coprocessor State Register, see description Section 4.4). As soon as *cp_InBuf* is empty, the contents of *cp_InReg* is shifted to *cp_InBuf*. Then, the bits are processed LSB first by the CODER/DECODER at a 1-bit per cycle rate. The CODER/DECODER outputs the processed bit stream to the *cp_OutBuf* register. It is then transferred to *cp_OutReg* so long as *cp_OutReg* is detected as not full .

The translated bit sequence can then be extracted from *cp_OutReg* to a register file's window register by the CPPULL instruction. As the code sequence has to be optimized so that the program extracts the *cp_Outreg* content just after this one has been updated, it uses a Bccc (Branch on Coprocessor Condition Code) just before the CPPULL to make sure the coding/decoding process has been completed. Note that the output length (conversely to the input length) is fixed at 32 bits.

The CRC computation is performed automatically in parallel to the coding/decoding operation, and the result is available in the *cp_crc* register. The CRC calculation consists of a polynomial division of *cp_InReg* content by the content of cp_*poly* register. However, if a CRC computation is needed without the corresponding CODER/DECODER operation, a special opcode field in the *cp_StateReg* register allows to perform CRC computation only.

Registers Description

- FIFO register (*cp_fifo*) : depending on the executed instruction, this register targets either InReg or OutReg.
  - A write to *cp_fifo* (CPWRCXT) actually writes the source operand to *cp_Inreg* which previous content is transferred to *cp_Inbuf*, which previous content is transferred to *cp_OutBuf*, which previous content is transferred to *cp_OutReg*. It intends to restore the coprocessor after a routine switch
  - A read from *cp_fifo* (CPRDCXT) transfers *cp_OutReg* to the destination register.

  So the communication coprocessor does not have its own register file and uses directly the main one.

- POLY register (*cp_poly*) : contains the divider the input bit sequence is to be divided by polynomial division during the Crc computation.

- CRC register (*cp_crc*) will host the result (partial remainder) of the redundancy check.

5.2 Bus Interface Controller

The 90C701 Bus Interface Controller acts in two modes according to the request area. The first mode is acting when the generated address is included in the 48Mbytes direct selectable pages. In this mode the selected device controller will handle the generation of the parametrized chip selects accordingly with the contents of DCR and DTCR registers. The second mode is acting when the request access is outside the 48Mbytes but inside the 256 Mbytes Memory & I/O segment (MIOS). In this mode the bus interface controller of the 90C701 will act as traditional bus controller and generated transactions on the external IoBus.

Mode 1: Direct Control of the attached devices

In this mode the selected device controller will generate the dedicated timing for the $\overline{\text{DSELx}}$[1:0], and $\overline{\text{DBE}}$[3:0] signals. The two following figures shows two typical access timings. One can be devoted to I/O device or static RAM, and the second to DRAM. Differences are coming from the address multiplexing mode.

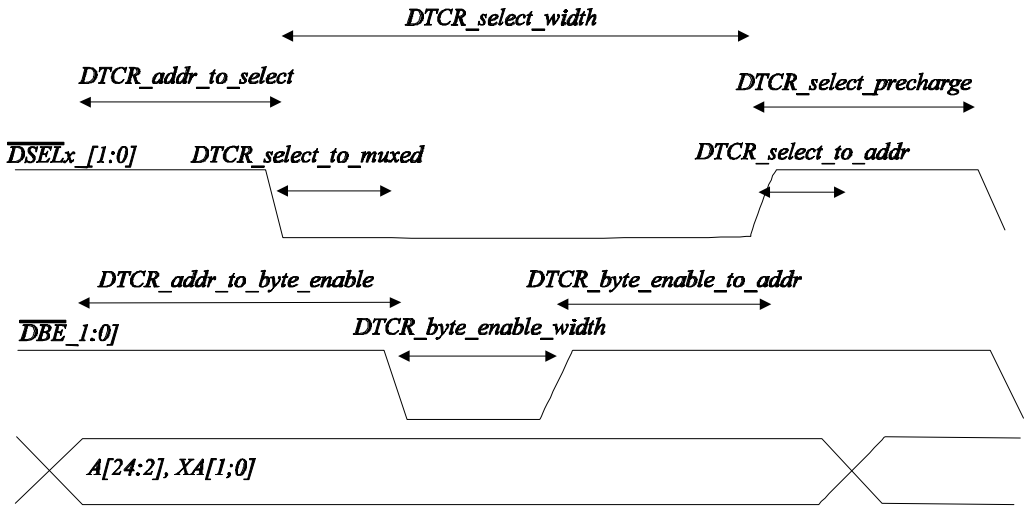


Figure 18.$\overline{\text{DSEL}}$ and $\overline{\text{DBE}}$ timings when Addresses are not multiplexed

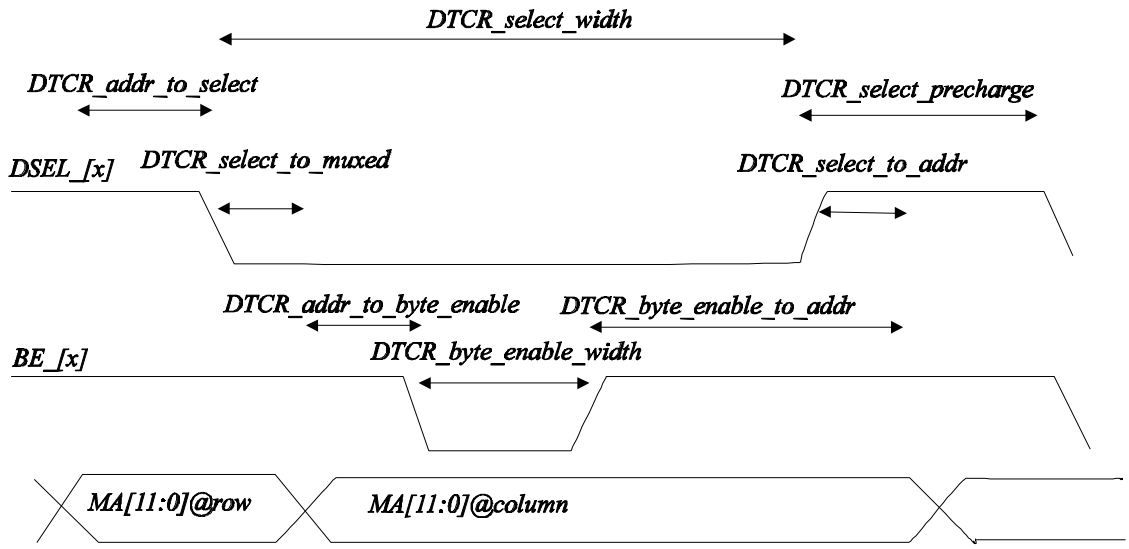


Figure 19.$\overline{\text{DSEL}}$ and $\overline{\text{DBE}}$ timings when Addresses are multiplexed.

Device Control Register (DCR)

26	25	24	23	22	21-20	19	18	17	16
_configuration_lock	*_programmable_output_1_lock*	*_programmable_output_0_lock*	*_refresh*	*_mux_addr_enable*	*_page_mode*	*_select_advanced*	*_select_delayed*	*_advanced_byte_enable*	*_delayed_byte_enable*

15-12	11-8	7-4	3	2	1	0
_device_select_1	*_device_select_0*	*_byte_enable*	*_programmable_output_1_enable*	*_programmable_output_0_enable*	*_programmable_output_1_value*	*_programmable_output_0_value*

Table 31.Device Control Register (DCR)

DCR_configuration_lock Bit 26 when asserted the device configuration is locked

DCR_programmable_output_1_lock Bit 25 when asserted the signal polarity fields are locked (*DCR_programmable_output_1_lock, DCR_programmable_output_1_enable, DCR_programmable_output_1_value)*

DCR_programmable_output_0_lock Bit 24 when asserted the signal polarity fields are locked (*DCR_programmable_output_0_lock, DCR_programmable_output_0_enable, DCR_programmable_output_0_value)*

DCR_refresh	Bit 23, when asserted means that the controlled device is a refreshable device. The refresh mode is enable.
DCR_mux_addr_enable	Bit 22, when asserted the device is expecting its addresses from MA[11:0] on the IoBus with the corresponding timings figure 19.
DCR_page_mode	Bit 20 through 21. Different multiplexed addresses configurations if *DCR_mux_addr_enable is asserted.*
DCR_select_advanced	Bit 19, when asserted the signal $\overline{\text{DSELx}}$[1:0] of the IoBus is advanced of one clock half cycle. Ignored if (x1) clock.
DCR_select_delayed	Bit 18, when asserted the signal $\overline{\text{DSELx}}$[1:0] of the IoBus is delayed of one clock half cycle. Ignored if (x1) clock.
DCR_advanced_byte_enable	Bit 17, when asserted the signal $\overline{\text{DBE}}$[3:0] of the IoBus is advanced of one clock half cycle. Ignored if (x1) clock.
DCR_delayed_byte_enable	Bit 16, when asserted the signal $\overline{\text{DBE}}$[3:0] of the IoBus is delayed of one clock half cycle. Ignored if (x1) clock.
DCR_device_select_1	Bits 12 through 15, specify the conditions to select the $\overline{\text{DSELx}}$[1] control line according to the address and mode (Read/Write).
DCR_device_select_0	Bits 8 through 11, specify the conditions to select the $\overline{\text{DSELx}}$[0] control line according to the address and mode (Read/Write).
DCR_byte_enable	Bits 4 through 7, specify the conditions to generate the byte enable signals $\overline{\text{DBE}}$[3:0] according to the address and mode (Read/Write).
DCR_ programmable_output_1_enable	Bit 3, when asserted the *DCR_programmable_1_value* is used as $\overline{\text{DSELx}}$[1] signal active value.
DCR_ programmable_output_0_enable	Bit 2, when asserted the *DCR_programmable_0_value* is used as $\overline{\text{DSELx}}$[0] signal active value.
DCR_ programmable_output_1_value	Bit 1, active value of $\overline{\text{DSELx}}$[1] signal.
DCR_ programmable_output_0_value	Bit 1, active value of $\overline{\text{DSELx}}$[0] signal.

Device Timing Control Register (DTCR)

30	29	28	27	26	25-24	23-22
_timing_lock	*_addr_to _select_wr*	*_select_to _addr_wr*	*_addr_to_byte _enable_wr*	*_byte_enable _to_addr_wr*	*_select_to_addr*	*_select_to_muxed*

21-20	19-18	17-16	15-13	12-8	7-5	4-0
_addr_to_select	*_addr_to _byte_enable*	*_byte_enable _to_addr*	*_select _precharge*	*_select_width*	*_byte_enable _precharge*	*_byte_enable _width*

Device Timing Control Register (DTCR)

DTCR_timing_lock — Bit 30, when asserted timing are locked.

DTCR_addr_to_select_wr — Bit 29, when asserted the field *DTCR_addr_to_select* is used as a timing for write access only.

DTCR_select_to_addr_wr — Bit 28, when asserted the field *DTCR_ select_to_addr* is used as a timing for write access only.

DTCR_addr_to_byte_enable_wr — Bit 27, when asserted the field *DTCR_addr_to_byte_enable* is used as a timing for write access only.

DTCR_byte_enable_to_addr_wr — Bit 26, when asserted the field *DTCR_ byte_enable_to_addr* is used as a timing for write access only.

DTCR_select_to_addr — Bits 24 through 25, number of cycles from $\overline{\text{DSELx}}$[1:0] to A[24:2], XA[1:0]

DTCR_select_to_muxed — Bits 22 through 23, number of cycles from $\overline{\text{DSELx}}$[1:0] to MA[11:0]

DTCR_addr_to_select — Bits 20 through 21, number of cycles from A[24:2], XA[1:0] to $\overline{\text{DSEL}}$x[1:0]

DTCR_addr_to_byte_enable — Bits 18 through 19, minimum number of cycles from A[24:2], XA[1:0] to $\overline{\text{DBE}}$[3:0].

DTCR_byte_enable_to_addr — Bits 16 through 17, minimum number of cycles from $\overline{\text{DBEx}}$[3:0] to A[24:2], XA[1:0].

DTCR_select_precharge — Bits 13 through 15, minimum number of necessary deasserted $\overline{\text{DSELx}}$[1:0] between two accesses.

DTCR_select_width — Bits 8 through 12, minimum number of cycles for $\overline{\text{DSELx}}$[1:0] to be asserted.

DTCR_byte_enable_precharge — Bits 5 through 7, number of cycles of $\overline{\text{DBE}}$[3:0] to be not asserted.

DTCR_byte_enable_width Bits 0 through 4, number of cycles of $\overline{\text{DBE}}$[3:0] to be asserted.

Refresh register (RR)

This register is a general Bus Interface Controller register. The refresh when active is dispatched to all device controllers for which the *DCR_refresh* bit is asserted. The refresh mode proposed is the CAS before RAS refresh. This register is 28 bit wide and composed as follows:

27	26	25	24-20	19-18	17-16	15-0
_refresh_overflow	*_refresh_lock*	*_refresh_enable*	*_refresh _select_width*	*_refresh_select _to_byte_enable*	*_refresh_byte _enable_to_selec t*	*_refresh_period*

Table 33.Refresh Register (RR)

RR_refresh_overflow Bit 27, when this bit is active more than four refresh cycles have not been satisfied.

RR_refresh_lock Bit 26, when asserted the refresh configuration is locked.

RR_refresh_enable Bit 25, when assserted the refresh is enabled.

RR_refresh_select_width Bits 20 through 24, $\overline{\text{DSELx}}$[1:0] signals will be active during the number of cycles contained in this field.

RR_refresh_select_to_byte_enable Bit 18 through 19, number of cycles between $\overline{\text{DSELx}}$[1:0] and $\overline{\text{DBE}}$[3:0].

RR_refresh_byte_enable_to select Bit 16 through 17, number of cycles between $\overline{\text{DBE}}$[3:0 and $\overline{\text{DSELx}}$[1:0].

RR_refresh_period Bit 0 through 15, this field indicates the period used for the refresh.

SpaceMap register (SMR)

This register is used in order to split the memory space handled by the bus interface controller in supervisor and user area. If accesses are done in user mode to a supervisor area, an error is returned to the CPU and the access is not performed. The BIC checks automatically the access using the following rule: *A[25:11] & SMR_map_mask = SMR_map_value*.

This register is 32bit wide and composed as follows:

31	30	29-15	14-0
_check_false	*_check_true*	*_map_value*	*_map_mask*

Table 34.SpaceMap Register (SMR)

SMR_check_false — Bit 31. When set, indicates that the space is protected if the condition is false.

SMR_check_true — Bit 30. When set, indicates that the space is protected if the condition is true.

SMR_map_value — Bit 15 through 29, This field is used in the check equation in order to define the tagged page.

SMR_map_mask — Bit 0 through 14 This field is used in the check rule in order to define the page size

Mode 2: Access to external decoded MIOS devices

In this mode the BIC is working as a traditional bus controller and generate all signals associated to a transaction on the external IoBus. *The detail of the bus protocol will be described in a further document*. However, the goal of this bus is to minimize external glue logic needed to interconnect any kind of devices. Here are the listed features of the Iobus.

Clock synchronous
Multimaster capability
Demultiplexed operation
8/16/32 bit data accesses

5.3 PCM/USART

The transmitter and the Receiver are independent entities (except id "Loop Back" is used) composed of a baud rate generator , a Fifo,a Sync Register which contains up to two synchronizing characters and a Command Register which controls the sequencer. The receiver has an additional Status register indication errors on received data streams. The Usart can generate interrupts on errors or fifo information via the Command Register. The so-called "Interface" registered have been implemented to allow the highest flexibility to the serial link parameters.

Table 35. PCM/USART registers.

Register	Bit Width	Description
Transmitter FIFO	12 8-bits words	Contains data to be transmitted
Transmitter Baud Rate Count	20	Generate 50 % duty cycle baud rate
Transmitter Baud Rate Reload Value	20	Contains half baud rate value
Transmitter Sync	16	Contains up to 2 sync characters
Transmitter Command	32	Controls operating modes, handshaking and interrupt level
Transmitter Status	17	Contains the transmitter status bits
Transmitter Interface	13	Additional command bits
Receiver FIFO	12 8-bits words	Contains received data
Receiver Baud Rate Count	20	Generates 50 % duty cycle baud rate
Receiver Baud Rate Reload Value	20	Contains half baud rate value
Receiver Sync	16	Contains up to 2 sync characters
Receiver Command	32	Controls operating modes, handshaking and interrupt level
Receiver Status	24	Contains the transmitter status bits
Receiver Interface	5	Additional command bits

5.3.1 Register mapping

The registers are mapped as follow :

Table 36. PCM/USART register mapping

Register	A20-A15	Access
Transmitter FIFO	0 0 x x x x	Store only (double/word/byte)
Transmitter Baud Rate Count	0 1 0 0 0 0	Load / Store
Transmitter Baud Rate Reload Value	0 1 0 0 0 1	Load / Store
Transmitter Sync	0 1 1 0 0 0	Load / Store
Transmitter Command	0 1 0 1 0 1	Load / Store
Transmitter Status	0 1 0 0 1 0	Load / Store
Transmitter Status Reset	0 1 0 0 1 1	Load/Store
Transmitter Interface	0 1 0 1 0 0	Load / Store
Transmitter Internal Reset	0 1 0 1 1 0	Store only
Transmitter Re-Sync	0 1 0 1 1 1	Store only

Register	A20-A15	Access
Receiver FIFO	1 0 x x x x	Load only (double/word/byte)
Receiver Baud Rate Count	1 1 0 0 0 0	Load / Store
Receiver Baud Rate Reload Value	1 1 0 0 0 1	Load / Store
Receiver Sync	1 1 1 0 0 0	Load / Store
Receiver Command	1 1 0 1 0 1	Load / Store
Receiver Status	1 1 0 0 1 0	Load / Store
Receiver Status Reset	1 1 0 0 1 1	Load/Store
Receiver Interface	1 1 0 1 0 0	Load / Store
Receiver Internal Reset	1 1 0 1 1 0	Store only
Receiver Re-Sync	1 1 0 1 1 1	Store only

The registers can be divided in 4 categories :

- the real registers already mentionned on PCM/USART Registers table
- the virtual registers on which Stores only intend to send a command to the module controller (Internal Reset, Re-Sync)
- the two Fifo registers which are located at the same address, and target the Transmitter Fifo on a Store and the Receiver Fifo on a Load.
- the Receiver and Transmitter Status Reset registers : they actually target the Status registers.When loaded, they target the Status content and clear all the errors.When Stored, they act as for a regular store, except that if an error bit is set by the module at the same time, it is written in priority.

5.3.2 Transmitter section

The transmitter is mainly ruled by the Transmitter Command register, which defines the main parameters of the serial link. 13 other control bits have been added in the Transmitter Interface Register to enhance the flexibility of the physical interface parameters.

Transmitter Command Register (TCR)

The Transmitter Command Register contains the operating modes, controls the transmission sequencer and handles Modem handshaking signals. When "Mode" bit is set (Synchronous mode) and both "Sync" bits are set, the module is configured as a PCM Transmitter.

Table 37. Transmitter Command Register (TCR)

31	30	29	28	27	26	25-24	23	22-18	17-13
_Clock _Gen	*_Frame SyncGen*	*_CtsEn*	*_CtsValue*	*_ClkEn*	*_ClkValue*	*_Sync*	*_Mode*	*_CharNum*	*_IrlChar _Level*

12-9	8	7	6-5	4	3-2	1	0
_Irl Char _Num	*_Loop _Back*	*_Hole/stop*	*_PValue*	*_PEn*	*_Dbl*	*_TdValue*	*_Ten*

TCR_Ten	Transmitter Enable
TCR_TdValue	Td Signal Value
TCR_Dbl	Data bit length (00= 5bits,01= 6 bits, 10= 7 bits, 11= 8 bits)
TCR_PEn	Parity Enable
TCR_PValue	Parity Value, selects the parity (00= zero, 01= one, 10= odd, 11= even)
TCR_Hole/Stop	Allows the generation of the Hole (PCM) or Stop (USART) bits
TCR_Loop_Back	Enables the loop back feature
TCR_IrlChar_Num	Low limit of free slots in the Transmitter Fifo.When this limit is reached, the module's controller generates an Interrupt to the processor
TCR_IrlChar_Level	Level of the generated Interrupt
TCR_CharNum	Number of free characters in the Transmitter Fifo.
TCR_Mode	Selects the operationg Mode (0= Asynchronous, 1= Synchronous)
TCR_Sync	Number synchronizing characters (00= reserved, 01 = one, 10= two, 11=PCM)
TCR_ClkValue	Value of the TClk signal (provides a way to use TClk as a regular I/O when it is not used for the serial link)
TCR_ClkEn	Selects what will be output on the TClk pin : if ClkEnthe internal baud rate is sent out, otherwise the level contained in Clk_Value is output.
TCR_CTSValue	Self explanatory : contains the value present on the Cts lead.
TCR_CTSEn	When set, forces the use of Cts in the serial protocol."0" allows to ignore Cts.
TCR_FrameSyncGen	When set, Frame Synchronisation is generated
TCR_ClockGen	When set, the TClk signal is generated and output.

Transmitter Sync Register (TSR)

The Transmitter Sync Register contains two 8-bits characters to be emitted to synchronize the Receiver.

Table 38.Transmitter Sync Register (TSR)

31-17	16	15-8	7-0
Reserved	*_Z*	*_SyncChar2*	*_SyncChar1*

The TSR_Z bit, when set, replaces the Synch Characters by a tristate level.

Transmitter Interface Register (TIR)

The Transmitter Interface register provides additional control bits, which ensure the full flexibility on the serial link parameters :

Table 39.Transmitter Interface Register (TIR)

12	11	10	9	8	7	6	5	4	3	2	1	0
_Lock	*_Clk_ Invert*	*_Fs/CTS _Invert*	*_HoleEn*	*_Hole Value*	*_Fs1En*	*_Fs1Val*	*_Fs0En*	*_Fs0val*	*_TdEn*	*_Td1Val*	*_Td0En*	*_Td0Val*

TIR_Td0Val allows to translate all the transmitted "0" datas to "1" (When Td0Val= 1)

TIR_Td0En tristates all the transmitted "0" datas.

TIR_Td1Val allows to translate all the transmitted "1" datas to "0" (When Td1Val= 0)

TIR_Td1En tristates all the transmitted "1" datas (Td1En= 1)

TIR_Fs0Val,_Fs0En same functionality as Td0Val and Td0En applied on Frame Synchronisation signals Fs0

TIR_Fs1Val,_Fs1En same functionality as Td1Val and Td1En applied on Frame Synchronisation signals Fs1

TIR_HoleValue,_HoleEn same functionality as Td0Val and Td0En applied on Hole signal

TIR_Fs/Cts_Invert allows level inversion of Fs/Cts signal

TIR_Clock_Invert allows level inversion of the transmission clock

TIR_Lock locks the content of the current configuration

Transmitter Status Register (TSTR)

The Transmitter Status provides the status of the module, and is able to generate an Interrupt Request to the processor (software programmable level) for various reasons. All those possible origins are individually maskable.

Table 40.Transmitter Status Register (TSTR)

16	15	14	13	12-5	8-4	3	2	1	0
_Sync _Mask	*_Fifo_Empty_ Mask*	*_Underrun_ Mask*	*_Framing_Error_Mask*	*_Irl_ Level*	*_Char Free*	*_Sync*	*_Fifo_ Empty*	*_Underrun*	*_Framing_ Error*

TSTR_Framing_Error,_Framing_Error_Mask	The Framing_Error bit flags the occurence of a framing problem. If the Framing_Err_Mask bit is set, this occurence will generate a Interrupt request to the CPU
TSTR_Underrun,_Underrun_Mask	Same behaviour with an underrun detection
TSTR_Fifo_Empty,_Fifo_Empty_Mask	Same behaviour with a Fifo empty state detection.
TSTR_Sync, _Sync_Mask	Same behaviour with a Transmitter Synchronisation detection.
TSTR_Char Free	Provides the number of free characters in the Transmitter Fifo
TSTR_Irl_Level	Allows to assign the Interrupt Request Level that will be generated for all the previous detected reasons.

5.3.3 Receiver section

The receiver section follows the same global organization as the transmitter from a register content standpoint. We will mainly focus on the field exclusively implemented in the Receiver section. For those which are similar to the Transmitter section ones, please refer to the Transmitter section chapter.

The Receiver Command Register (RCR)

Table 41.Receiver Command Register (RCR)

31	30	29	28	27	26	25-24
Reserved	*_Ignore P*	*_RtsEn*	*_RtsValue*	*_Int/Ext_Sync*	*_ClkValue*	*_Sync*

23	22-18	17-14	13-9	8	7	6-5	4	3-2	1	0
_Mode	*_CharNum*	*_IrlChar Level*	*_IrlChar Num*	*_Loop_ Back*	*Reserved*	*_PValue*	*_PEn*	*_Dbl*	*_Rd Value*	*_Ren*

RCR_In/Ext_Sync	chooses between Internal and External Synchronisation clock
RCR_RtsValue	displays the current logical state of the CTS lead

The Receiver Synchronisation Register (RSR)

Table 42.Receiver Synchronisation Register (RSR)

31-17	16	15-8	7-0
Reserved	*_Z*	*_SyncChar2*	*_SyncChar1*

The Receiver Interface Register (RIR)

Table 43.Receiver Interface Register (RIR)

4	3	2	1	0
_Lock	*_RtsEn*	*_Clock_invert*	*_Rts/Fs_invert*	*_Rd_invert*

The Receiver Status Register (RSTR)

Table 44.Receiver Status Register (RSTR)

24	23	22	21	20	19	18	17
Break State_ Mask	*_Sync _Mask*	*_Fifo_Full _ Mask*	*_Spurious_ Char_Mask*	*_Break_ Mask*	*_Overrun _Mask*	*_Parity_Erro r_Mask*	*_Framing_ Error_Mask*

16-13	12-8	7	6	5	4	3	2	1	0
Irl Err_Level	*_Fifo_Char_ Num*	*_Break_ State*	*_Sync*	*_Fifo_Full*	*_Spurious _Char*	*_Break*	*_Overrun*	*_Parity_Erro r*	*_Framing _ Error*

5.4 Real-Time and General Purpose Peripherals

The timers and PIAs are managed through dedicated registers which are programmable through memory-like transactions :

Table 45. Peripherals programming instructions

Write	Store (ST) instructions
Read	Load (LD) instructions

For some locations, the Load instruction also resets the flag values. This feature is called "Load and Reset".

5.4.1 Timers

Operating System Timer

This is a 32 bit decremental timer which generates an interrupt upon zero detection. It is managed by 3 registers of 32 bit:

- the Reload Value Register (OSRVR) which contains the interrupt period
- the Counter Register which contains the timer value (OSCTR)
- the Command Register, decribed below. (OSTCR)

Table 46.Operating System Timer

31-11	10	9	8	7	6	5-2	1	0
Reserved	*_Reload_ Lock*	*_Counter _Lock*	*_Comman d _Lock*	*_HWZ*	*_CE*	*_Irl*	*_IrlEn*	*_IrlActive*

OSTCR_Reload_Lock	When set, Store Instruction into RVR disabled
OSTCR_Counter_Lock	Store Instruction into Counter Register disabled (one time programmable after reset)
OSTCR_Command_Lock	Store Instruction into Command Register disabled
OSTCR_HW	Halt when zero
OSTCR_CE	Count Enable (active high)
OSTCR_Irl	Interrupt Request Level
OSTCR_IrlEn	Irl Enable (active high)
OSTCR_IrlActive	When set, means that Counter Register reached zero

Watchdog

The watchdog is an additional instanciation of the OsTimer, with a reduced functionality:

Load and Reset commands performed on the Counter Register will reload the watchdog while the Command Register content is discarded. If the watchdog is enabled, the software has to reload periodically the watchdog timer, otherwise when this one reaches the zero state, the watchdog output signal will go low.The user has different options to connect this output. It can be wired externally to the $\overline{\text{RESET}}$ input (possibly through an external circuitry) or also to the reset inputs of some external peripherals.

V8e Compliant Timer

This is a general purpose timer based on two 16 bit counters. It can generate interrupts and external waveforms. The timer is triggered by external events or system clock. The timer is controlled by 6 registers: the Input Handler, the Scaler, the Scaler Reload Register, the Counter, the Counter Reload Register and the Shaper.

The Timer Input Handler Register (TIHR) contains the attributes of the external counting events.

Table 47. Timer Input Handler Register (TIHR)

31- 6	5	4	3:2	1	0
Reserved	*_IhPuls*	*_IhInv*	*_IhWidth*	*_IhEn*	*_IhM*

TIHR_IhPuls Input Handler Pulse Command (0= edge, 1= level)

TIHR_IhInv Input Handler Invert Command.When set, input is active high

TIHR_IhWidth Input Handler Width Command defines the sampling size. Out of those n samples, a majority vote defines the value taken in account.

- 00: 1 sample
- 01: 3 samples (2 identical values set the definitive value)
- 10: 5 samples (3 identical values set the definitive value)
- 11: 7 samples (5 identical values set the definitive value)

TIHR_IhEn Input Handler Enable Command

TIHR_IhM Input Handler Mask

The Scaler (16 bit decrementer) is the less significant half of the timer. The Counter (16 bit decrementer) is the most significant half of the timer. The Scaler Reload Register contains the data to be loaded into the Scaler, upon specified conditions. The Counter Reload Register contains the data to be loaded into the Counter, upon specificied conditions.

The Shaper (TSHR) determines the waveform of the output generated by the timer and the level of the possibly generated interrupt.

Table 48. Shaper Register (TSHR)

31-14	13-10	9	8	7	6	5	4	3	2	1	0
Reserved	*_Irl_Level*	*_IrlEn*	*_IrlActive*	*_Pwm*	*_StartBit*	*_Czar*	*_Szacz*	*Reserved*	*_Sync*	*_Cz*	*_SzCz*

TSHR_Irl Interrupt Request Level

TSHR_IrlEn Interrupt Enable (active high)

TSHR_IrlActive Interrupt Active (indicates that the counter reached zero. Software resetable)

TSHR_Pwm	Pulse Width Modulation (When set, the Counter and the Scaler are reloaded when they reach zero.When Scaler reaches zero, it stops until Counter reaches zero.The value of the output bit is Czar when Scaler is not zero and Sczar when Scaler is zero.
TSHR_StartBit	TOUT (Timer Out) output value when the Timer is synchronized (Scaler and Counter reloaded) by asserting the Sync bit of the Shaper Register
TSHR_Czar	Counter zero after Restart.If Pwm bit is zero, Czar is the value of the of the "Timer Out" output when the Counter reaches zero for the first time after a restart. If Pwm is set, Czar is the value of the "Timer Out" signal when Scaler is not zero.
TSHR_Szacz	Scaler zero after counter zero .If Pwm bit is zero, Czar is the value of the of the "Timer Out" output when the Scaler reaches zero for the first time after counter has reached zero. If Pwm is set, Czar is the value of the "Timer Out" signal when Scaler is zero.
TSHR_Sync	Synchronisation (When set by a Store instruction, forces the reload of both Scaler and Counter). Active during one cycle after the Store.
TSHR_Cz	Counter zero (When set, counter stops when ir reaches zero.Otherwise, it is relaoded and restarts decrementing)
TSHR_SzCz	Scaler zero and Counter zero (If set, the Scaler stops when both Scaler and Counter reach zero.Otherwise, the Scaler is reloaded)

The Shaper allows the 90C701 to support the PWM mode.

5.4.2 Peripheral Interface adapter (PIA)

This cell allows the attributes of a single port pin to be programmed. This is done by using the PIA Command Register (PCR) which content determines :

- if the port is input or output
- any filtering functions on the port (polarity, noise reduction, level or edge detection and masking)
- the interrupt level associated to the port if any

Table 49. PIA command register (PCR)

31-13	12-11	10	9-6	5	4	3-2	1	0
Reserved	*_I/OConf*	*_Out*	*_Irl*	*_IhPuls*	*_IhInv*	*_IhWidth*	*_IhEn*	*_IhOut*

PCR_I/OConf I/O Configuration
- 00: Input reprogrammable
- 01: Output reprogrammable
- 10: Input not reprogrammable (until a hard reset occurs)
- 11: Output not reprogrammable (until a hard reset occurs)

PCR_Out Output value

PCR_Irl Interrupt request level

PCR_IhPuls Input Handler Pulse Command (0= level, 1= Edge)

PCR_IhInv Input Handler Invert Command

PCR_IhWidth Input Handler Width command defines the sampling process of the input:
- 00: 1 sample
- 01: 3 samples (2 identical values set the definitive value)
- 10: 5samples (3 identical values set the definitive value)
- 11: 7samples (5 identical values set the definitive value)

PCR_IhEn Input Handler Enable Command (When set, Input is enqble)

PCR_IhOut Input Handler Output Value

6 90C701 Pin Out

Pin	Name	Type	Description
	Clock,Power and Reset Management		
	VCCO	Power	5V(3.3V) +/-10% Output buffers power supply
	VSSO	Power	Output buffers ground
	VCCI	Power	5V(3.3V) +/-10% core and input buffers power supply
	VSSI	Power	Core and input buffers ground
	CLK	I	CLocK
	$\overline{\text{RESET}}$	I	Hardware RESET
	$\overline{\text{HALT}}$	I	HALTs the Fetch and Decode Unit at a logical boundary
	DEBUG[2:0]	O	program flow indication, for DEBUG purpose
	Memory controller interface signals (IoBus)		
	D[31:0]	I/O	Data bus
	A[25:2]	I/O	Address bus : 256 Megabytes of address space.
	XA[1:0]	I/O	eXtended Address, to be matched with AP[1:0]
	$\overline{\text{REQ}}$	O	IoBus REQuest for full address space (from master to arbiter)
	$\overline{\text{XREQ}}$	O	IoBus eXternal REQuest for multimaster configuration (from master to arbiter)
	$\overline{\text{GNT}}$	I	IoBus GraNT to all request types (from arbiter to master)
	$\overline{\text{XGNT}}$	I	IoBus eXternal GraNT for multimaster configuration. (from arbiter to master)
	$\overline{\text{AS}}$	I/O	IoBus Access Start
	AP[1:0]	I	IoBus Address Prefix (defines the processor's address Identifier)
	$\overline{\text{DS}}$	I/O	IoBus Data Strobe (from master to slave)
	$\overline{\text{WR}}$	I/O	IoBus Write (from master to slave)
	$\overline{\text{BE}}$[0]	I/O	IoBus Byte Enable: WordAddress[31:24]
	$\overline{\text{BE}}$[1]	I/O	IoBus Byte Enable: WordAddress[23:16]
	$\overline{\text{BE}}$[2]	I/O	IoBus Byte Enable: WordAddress[15:8]
	$\overline{\text{BE}}$[3]	I/O	IoBus Byte Enable: WordAddress[7:0]
	SA	I/O	IoBus Supervisor Access (from master to slave)
	SD	I/O	IoBus Supervisor -Data in Supervisor space (from slave to master)
	$\overline{\text{DBE}}$[3:0]	O	IoBus Device Byte Enable
	$\overline{\text{DRD}}$	O	IoBus Device Read

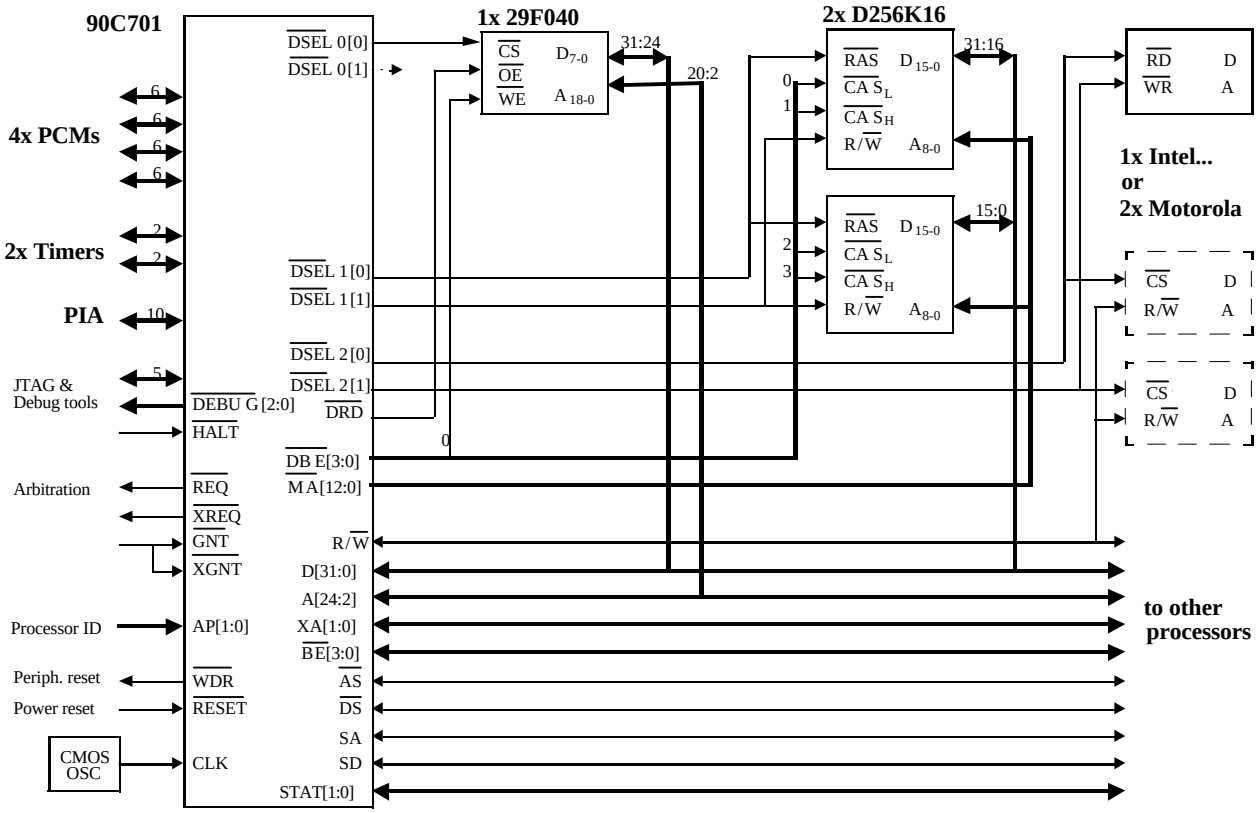
Pin	Name	Type	Description
	STAT[1:0]	I/O	IoBus STatus - Bus transaction status (from slave to master) 11: wait (transaction on going) 01: OK (transaction successfully terminated) 10: FullOK (transaction successfully terminated and success prediction for dual transaction to enable copy back) 00: error (transaction terminated with error)
	$\overline{\text{DSEL0}}$[1:0]	O	IoBus Device Enable 0- Control lines for address space:0-16M
	$\overline{\text{DSEL1}}$[1:0]	O	IoBus Device Enable 1- Control lines for address space:16-32M
	$\overline{\text{DSEL2}}$[1:0]	O	IoBus Device Enable 2- Control lines for address space:32-48M
	MA[11:0]	O	IoBus Multiplexed Address.(DRAM multiplexed address : 24 bit max)
	PCM/USART's signals		
	RXD[3:0]	I	PCM/USART Receive Data
	RXClkx[3:0]	I	PCM/USART Receive bit Clock
	TFS/CTS[3:0]	I/O	PCM Transmit Frame Synchronization /USART CTS (Clear To Send)
	TXD[3:0]	O	PCM/USART Transmit Data
	TXClkx[3:0]	I/O	PCM/USART Transmit bit Clock
	RFS/RTSx[3:0]	I/O	PCM Receive Frame Synchronization/USART RTS (Ready To Send)
	JTAG signals		
	TDI	I	jTag Data In
	TDO	O	jTag Data Out
	TCLK	I	jTag CLocK
	TMS	I	jTag Mode
	TRST	I	jTag ReSeT
	PIA signals		
	PIA[9:0]	I/O	Parallel Interface Adapter
	TIMER's signals		
	TIN [1:0]	I	V8e CounTeR/timer INputs
	TOUT [1:0]	O	V8e CouNTeR/timer OUTputs
	$\overline{\text{WDR}}$	O	WatchDog Reset

Note : all the active low signals are overlined.

7 90C701 Basic Configuration

The following figure outlines a typical glueless implementation using one 90C701 processor.

Figure 20. 90C701 Basic Board Configuration



The left side of the drawing displays all the internal peripherals:

- 4 PCM lines,
- 2 general purpose timers
- 1 PIA of 10 bits

It displays also the support pins:

- the JTAG and the DEBUG[2:0] outputs may be use for debug support,
- the AP[1:0] inputs set the processor self-controlled address space,
- the $\overline{WDR}$ output may be used to trigger the resetting of the external peripherals,
- the $\overline{RESET}$ inputs receive the power-on low level reset pulse,
- the CLK input is connected to a CMOS oscillator output.

On the right side are shown the implementation dependent features:

- a boot PROM,
- a main memory,
- additional external devices, either "Intel"- or "Motorola"-like.

The boot PROM is controlled by the first device controller. $\overline{DSEL0}$[0] is connected to the chip select input of the device.

In fact it is possible to control a Flash ROM, using $\overline{DRD}$ as an output enable and $\overline{DBE}$[0] as a write enable. One 8 bit device is enough to store up to 512 KBytes of program, however 4 devices may be used as well to store up to 2 MBytes of program in 32 bits words. Then, the four $\overline{DBE}$[3:0] lines allows to write into each device. It is further possible to implement similarly another bank of standard SRAM/FlashROM memory using $\overline{DSEL0}$[1] as a chip select, provided the timings are the same.

The main memory is a bank of DRAM fully supported by the second device controller. $\overline{DSEL1}$[0] is connected to the raw address strobe inputs of the two 16 bits devices, while the four $\overline{DBE}$[3:0] lines are connected to the column address strobe inputs and the $\overline{DSEL1}$[1] line is connected to the read/write control inputs of the two devices.

Applications requiring a larger capacity may use 4 bits devices instead, the dedicated multiplexed address bus is designed to support their heavier overall load capacitance.

The last device controller may be used to control any kind of additional external peripheral. Either the two $\overline{DSEL2}$[1:0] are connected to the read and write inputs of an "Intel"-like (or FIFO...) device, or to the chip select inputs of two "Motorola"-like devices with $\overline{WR}$ tied to their read/write control input.

The remaining signals may be used for arbitration and direct connexion to up to 4 other processors in the case of a companion implementation. If there is no other possible master, all the AP[1:0], $\overline{GNT}$ and $\overline{XGNT}$ inputs must be tied to a low level.