FUJITSU

930 Series
Embedded Processor
User's Manual

MB86933H–20

JUNE 1995, EDITION 1.0

FUJITSU MICROELECTRONICS, INC.

This document is published by the marketing department of Fujitsu Microelectronics, Inc., Semiconductor Division, 3545 North First Street, San Jose, California, U.S.A. 95134–1804.

Table Of Contents

CHAPTER

F1

Overview of the MB86933H–20

The MB86933H–20 is functionally and architecturally similar to the MB86930 SPARClite RISC processor. The MB86933H–20 has the same integer unit as the MB86930, supports the same instruction set as the MB86930, and is system bus compatible with the MB86930.

Several MB86930 features and signals are not available on the MB86933H–20, however, to reduce processor cost and package size. The MB86933H–20 has six register windows rather than eight. It has twenty-six Address Bus signals (ADR<27:2>) rather than thirty, has four Address Space Identifier signals (ASI<3:0>) rather than eight, and has no emulator-support signals. The MB86932 can be used for MB86933H–20 in-circuit emulation.

The MB86933H–20 does support 8, 16, and 32–bit bus based on the configuration register or (–BMODE8 and –BMODE16). MB86933H–20 also has an internal DRAM controller.

F1.1 Organization and Content

This section is organized in the same way as section 1 of this manual which describes the MB86930 processor. In general, this section contains descriptions of the MB86933H–20 processor that differ from the MB86930 processor. Descriptions that are the same for both processors are generally not repeated in this section, and the reader is referred to the main section of the manual for these identical descriptions.

These MB86933H–20 differences with respect to the MB86930 processor are summarized as follows:

- No data cache
- 1K–byte instruction cache instead of 2K–byte
- complete DRAM controller
- Supports 8/16–bit bus mode for any chip select
- Six register windows rather than eight
- ADR<31:28> not present
- ASI<7:4> not present
- –EMU_SD<3:0> not present
- –EMU_D<3:0> not present
- –EMU_BRK not present
- –EMU_ENB not present
- –BMODE8 and –BMODE16 inputs added to support 8– and16–bit ROMs, as well as 32–bit ROMs.

F1.2 General Description

The MB86933H–20 is a high-performance processor that is suitable for use in embedded control applications such as printers, scanners, robotic machinery, telecom switches and monitors, and I/O subsystems. It operates at clock speeds up to 20 MHz, executes SPARC instructions at a maximum rate of 18 MIPs, and is available in a 160-pin QFP package.

The processor consists of a Harvard (Aiken) architecture Integer Unit (IU) core and a Bus Interface Unit (BIU). These units are connected internally with separate instruction and data buses, and to external memory and I/O with separate 26–bit address and 32–bit data buses.

A register file in the IU is accessed through 6 register windows. An integer multiply unit (MU) within the IU speeds applications that require integer multiplication. The processor uses software to emulate floating-point instructions. The data path and other arrayed blocks are full-custom designs to optimize die area and speed. Random control blocks are standard-cell designs. All circuits are fully static.

The MB86933H–20 provides a mechanism for code and data protection, but is optimized for embedded applications that do not require virtual-to-physical address

translation. The MB86933H–20 processor can be designed into in a virtual-memory system, however, by using external memory management logic for address translation.

F1.3 Special Features

The following MB86933H–20 features make the processor an ideal choice for a wide variety of low cost, high-performance embedded systems:

- **Fast Instruction Execution:** The instruction set is streamlined and hardwired for fast execution, with most instructions executing in a single cycle. At 20 MHz the MB86933H–20 executes instructions at a peak rate of 20 MIPs and at a sustained rate of 18 MIPs. The Integer Unit (IU) features a 5-stage pipeline that has been designed to handle data interlocks, and an optimized branch handler for efficient control transfers.
- **Large Register Set:** An internal register file, consisting of eight global registers and 96 registers organized into six overlapping windows, speeds interrupt response time and context switches. The register file windows minimize accesses to memory during procedure linkages, and facilitate passing of parameters and assignment of variables.
- **Instruction Cache:** A 1Kbyte direct–mapped instruction cache is included on chip. The cache is organized into sixty–four 16–byte lines. Individual lines or the entire cache can be locked.
- **System Support Functions:** Glue logic between the MB86933H–20 and the system is minimized by programmable chip selects, programmable wait-state circuitry, and support for connection to fast page-mode DRAM. Multiple bus masters are supported through a simple handshake protocol.
- **Clock Generator:** A crystal can be connected directly to the on-chip oscillator, or an external clock source can be used. A phase-locked loop minimizes the skew between on- and off-chip clocks.
- **Enhanced Instruction Set:** The MB86933H–20 incorporates a fast integer multiply instruction that executes in 5, 3 or 2 cycles for 32–bit, 16–bit and 8–bit operands, respectively. An integer divide-step instruction cuts divide times by a factor of 5 to 10 over previous SPARC implementations. A scan instruction supports a single-cycle search for the most significant non-sign bit in a word.
- **Fully Static Circuit Design:** Its static design gives the MB86933H–20 superior noise immunity. Other members of the SPARClite family support a low-power mode in which the processor clock can be slowed or stopped for arbitrary periods of time to reduce operating current.
- **ROM Size Option Support:** Two external signals allow the processor to identify whether 8–, 16–, or 32–bit ROMs are in use. This feature allows use of smaller ROMs for a reduction in cost and in board space.

- **DRAM Controller:** A complete DRAM controller is integrated on the MB86933H–20. This controller provides address multiplexing and control signal generation for page–mode DRAMS.
- **Interrupt Controller:** The MB86933H–20 has an interrupt controller on board.

F1.4 Programmer's Model

This section briefly introduces those aspects of the MB86933H–20 processor architecture that are visible to software: the user and supervisor modes of program execution, the organization of the address space, the register set, the supported data types, the instruction set, and interrupts and traps. Each of these topics are discussed in more detail in following chapters.

F1.4.1 Program Modes

The MB86933H–20 architecture supports protection in multitasking environments by providing two mutually exclusive modes of program execution, *user mode* and *supervisor mode*. Certain instructions are privileged, and can only be executed when the processor is in supervisor mode. Any attempt to execute a privileged instruction in user mode causes a trap.

Typically, application programs run in user mode, while operating systems run in supervisor mode. Following reset, the processor is in supervisor mode. To enter user mode, software must clear a bit in the Processor State Register. The processor enters supervisor mode from user mode only when a hardware reset, an interrupt, or a trap occurs.

F1.4.2 Memory Organization

The processor can directly address up to 4 Gigabytes of memory, organized into 16 address spaces of 256 Megabytes each. Every external access involves an 4–bit Address Space Identifier (ASI), as well as a 26–bit word address. The ASI selects one of the address spaces, and the 26–bit address selects a 32–bit word within that space.

Four of the address spaces are defined in the SPARC architecture: the User Instruction, Supervisor Instruction, User Data, and Supervisor Data spaces. The other address spaces are application-defined or reserved. The application-defined address spaces can be used for either data memory or for I/O. All I/O is memory-mapped.

The organization of the entire addressable range is illustrated in Figure F1-1.

Loads and stores are the only instructions that cause external accesses. Versions of these instructions exist for transferring bytes, half-words, words and double words

between external memory (or I/O) and processor registers. The user instruction and data spaces are accessible in both user and supervisor modes. The remaining address spaces are accessible only in supervisor mode.

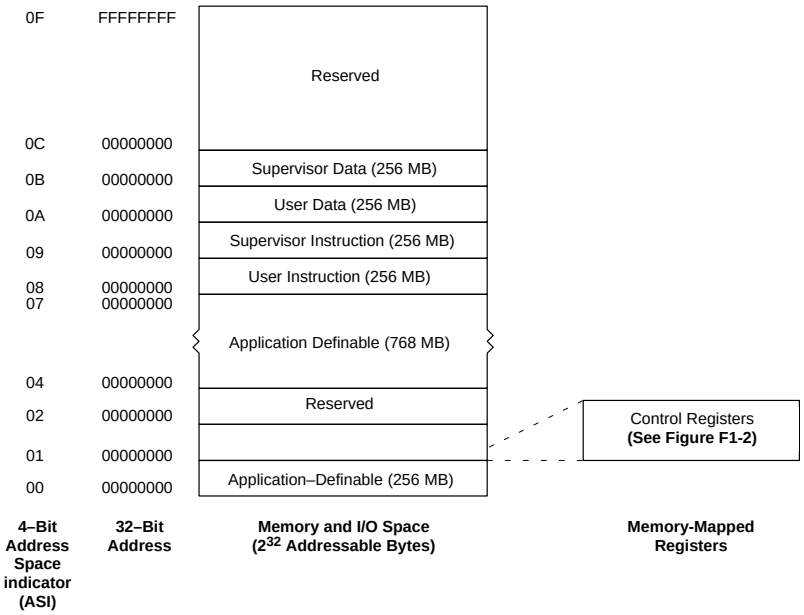


Figure F1-1. Address Space Organization

The MB86933H–20 processor does not contain memory-management hardware. Virtual-addresses can be translated by software, or by an external memory-management unit.

Note that the MB86933H–20 has six register windows rather than eight. It has twenty-six Address Bus signals (ADR<27:2>) rather than thirty, four Address Space Identifier signals (ASI<3:0>) rather than eight, no emulator-support signals, and no memory management unit. These and other differences between the MB86933H–20 and other SPARClite processors should be considered when porting code to the MB86933H–20 from another SPARClite processor, and when porting code from the MB86933H–20 to another SPARClite processor. Documentation for other SPARClite should be referenced to identify differences with the MB86933H–20 that may affect ported code.

F1.4.3 Registers

All registers are 32 bits wide. There are *general-purpose registers*, whose contents have no pre-assigned meaning, and *special-purpose registers* that contain control and status

information or special data values. Some of the special-purpose registers are defined in the SPARC architecture; the rest are MB86933H–20– specific registers. The non-SPARC special-purpose registers are memory-mapped. The general-purpose registers and the special-purpose Y Register are the only registers that can be accessed in user mode. The register set is illustrated in Figure F1-2.

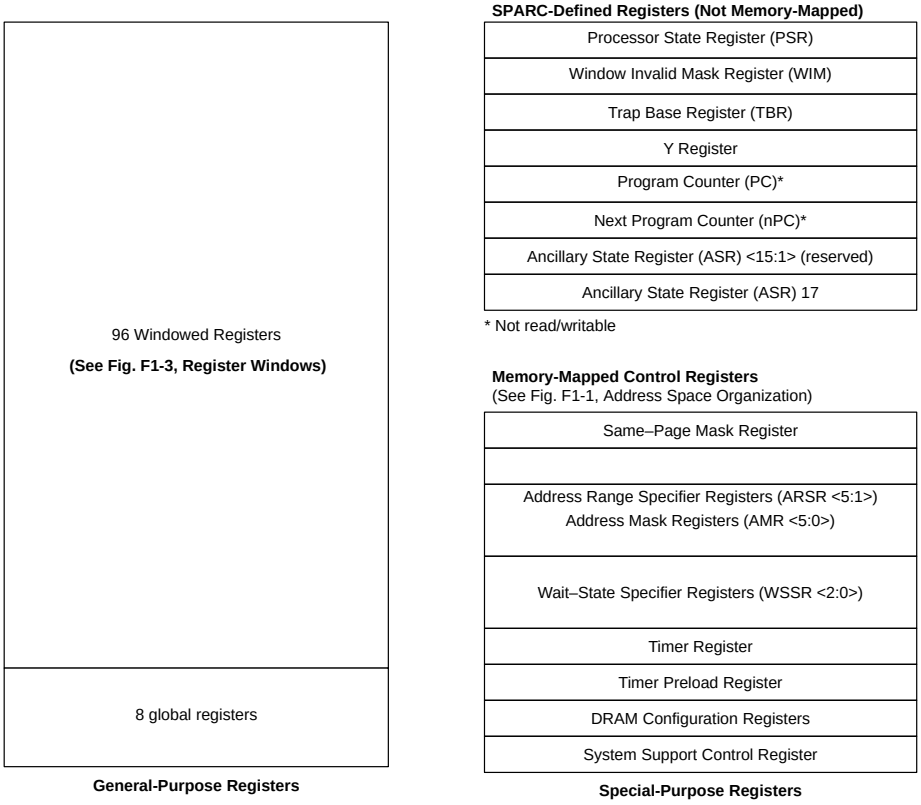


Figure F1-2. Register Set

General-Purpose Registers

The MB86933H–20 contains 104 general-purpose registers; 8 of these are *global registers*; the other 96 registers are divided into 6 overlapping blocks, or *windows*. Each window contains 24 registers. Of these, 8 are *local* to the window, 8 are *"out"* registers shared with the adjacent window below, and 8 are *"in"* registers shared with the adjacent window above. This organization is illustrated in Figure F1-3.

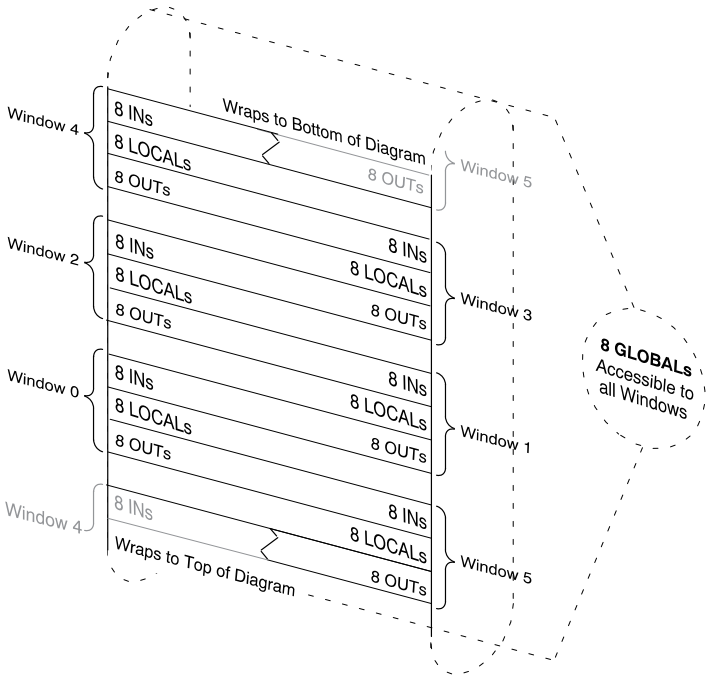


Figure F1-3. Register Windows

At any given time, 32 general-purpose registers can be accessed directly: the 8 global registers, and the 24 registers of the currently active window. The value in the Current Window Pointer (CWP) field of the Processor State Register (PSR) determines which window is active.

The overlap between adjacent windows makes it easy to pass parameters to a subroutine. Values to be passed are written to the “out” registers of the current window, which are the same as the “in” registers of the adjacent window. A SAVE instruction can then be used to decrement the Current Window Pointer, making the parameter values available to the subroutine without moving any data. A RESTORE instruction can be used to increment the CWP upon return from the subroutine. In effect, the general-purpose registers cache the top portion of the run-time stack.

The window overlap also speeds interrupt handling because interrupts automatically decrement the CWP, giving the interrupt routine its own window. The SPARC architecture requires a free window to be available to handle these traps.

Special-Purpose Registers

The special-purpose registers include the control and status registers defined by the SPARC architecture, and a collection of memory-mapped registers that control peripheral functions.

Special instructions exist for reading and writing each of the SPARC control and status registers except the Program Counter and the Next Program Counter. The Y Register can be read and written in user mode; the instructions that access the other SPARC-defined registers are privileged.

The memory-mapped registers can be read and written with the alternate-space load and alternate-space store instructions, which are also privileged.

The SPARC-defined registers, shown in Figure F1-2, are as follows:

- Processor State Register (PSR)—The primary processor control and status register. It contains *mode* fields that are set by the operating system to configure the processor, and *status* fields that are set by the processor to indicate the effects of instruction execution.
- Window Invalid Mask Register (WIM)—Used by software to detect the occurrence of register file underflows and overflows. It contains one mask bit for each register window. If an operation that normally increments or decrements the Current Window Pointer would cause the CWP to point to a window whose corresponding WIM bit equals 1, a trap occurs.
- Trap Base Register (TBR)—Contains three fields used by the processor to generate the address of the service routine when an interrupt or trap occurs.
- Y Register—Used in stepwise multiplication and division routines based on the MULScc and DIVScc instructions. Also used for integer multiply operations.
- Program Counter (PC)—Contains the word address of the instruction currently being executed by the Integer Unit. The PC cannot be directly read or written.
- Next Program Counter (nPC)—Contains the word address of the next instruction to be executed, assuming that no trap occurs. The nPC cannot be directly read or written.
- Ancillary State Registers (ASR[31:1])—The SPARC definition includes 31 Ancillary State Registers, 15 of which (ASR[15:1]) are reserved for future use. The remaining ASR's can be defined and used in any way by SPARC implementations. SPARClite defines the following ASR:

  ASR17— Used to enable and disable single-vector trapping. (When this feature is enabled, all traps vector to a single location.) Single vector trapping provides a small memory alternative to the standard 1K word trap table.

The memory-mapped MB86933H–20–specific registers, shown in Figure F1-2, are as follows:

- Same-Page Mask Register—Controls the operation of the same-page detection logic by specifying which bits of the current ASI and address are to be compared with those of the previous ASI and address.
- Address Range Specifier Registers (ARSR[5:1])—Control the assertion of the Chip-Select outputs (–CS[5:1]). –CSn is asserted when the value on the address bus falls in the address range specified by ARSRn. –CS0 is asserted during accesses to the lowest address range in Supervisor Instruction Space.
- Address Mask Registers (AMR[5:0])—AMRn controls the comparison of the current address with ARSRn by specifying which bits are to be compared and which are “don’t cares.”
- Wait-State Specifier Registers (WSSR[2:0])—Determines for each address range the number of clock cycles between assertion of an address in that range on the address bus, and assertion of –READY signal by the processor. This makes it possible for memory and I/O devices with different access times to be connected to the processor without additional logic.
- Timer Register—Contains the current timer count.
- Timer Pre-Load Register—Contains the value that is loaded into the timer when the timer overflows.
- System Support Control Register—Allows selective enabling and disabling of same-page detection, chip-select, programmable wait-states, and the timer.
- DRAM Configuration Registers—

F1.4.4 Data Types

The MB86933H–20 supports the same data types as the MB86930 processor. Please refer to Section 1.3.4 of the main section of this manual for a description of the data types.

F1.4.5 Instructions

The MB86933H–20 supports the same instructions as the MB86930 processor. Please refer to Section 1.3.5 of the main section of this manual for a description of the instructions.

F1.4.6 Interrupts and Traps

The MB86933H–20 supports the same interrupts and traps as the MB86930 processor. Please refer to Section 1.3.7 of the main section of this manual for a description of the interrupts and traps.

F1.5 Internal Architecture

The internal architecture of the MB86933H–20 is illustrated in Figure F1-4. The processor core consists of an Integer Unit that supports a superset of the SPARC integer instruction set. A 1K–byte direct–mapped instruction cache is provided on chip. The Bus Interface Unit handles the interface between the processor and the system. A Clock Generator with built-in phase-locked loop simplifies system clock design.

Internally, the various functional units are connected by separate instruction and data buses. For connection with external memory and I/O, a unified address bus and a unified data bus are extended off-chip. The main functional units are discussed briefly in the following sections, and more fully in the *Internal Architecture* chapter.

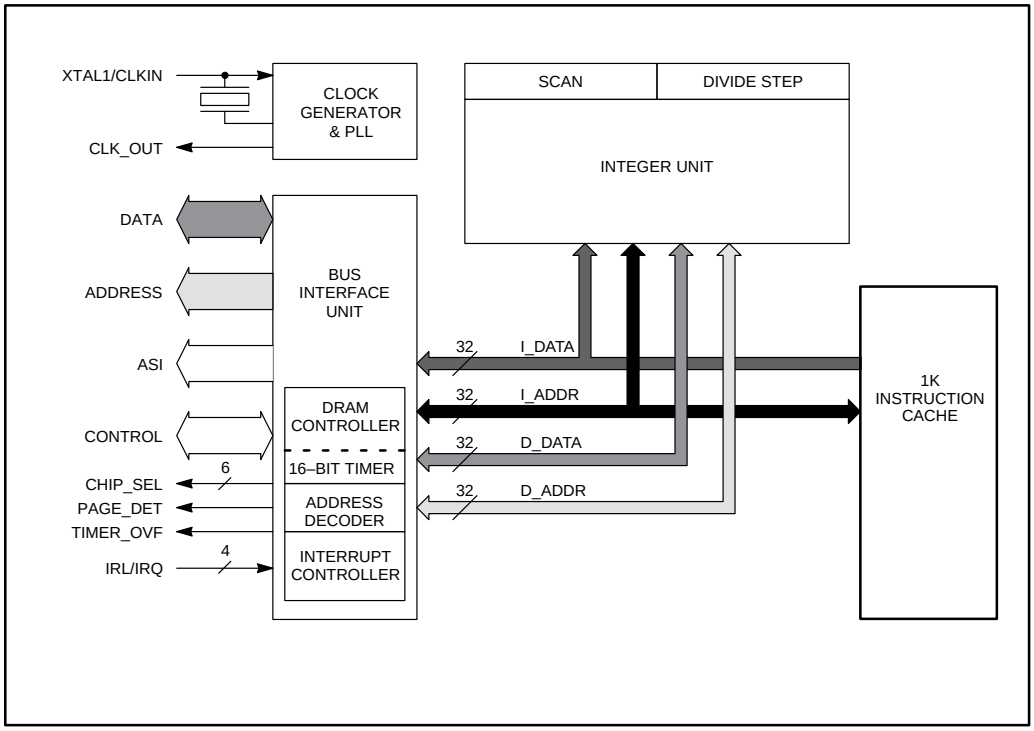


Figure F1-4. Internal Architecture (Block Diagram)

F1.5.1 Integer Unit

The Integer Unit (IU) is a compact, fully custom implementation of the SPARC architecture. The IU is hard-wired for high performance. Its internal functional units are

designed around a modular architecture and can be customized to meet different application requirements. In the MB86933H–20, for example, this flexibility was used to provide direct hardware support for integer multiplication, and to extend the SPARC instruction set by supporting divide-step and scan instructions.

The IU implements a five-stage instruction pipeline to allow a sustained execution rate of nearly one instruction per cycle. The operation of the pipeline under ideal conditions is illustrated in Figure F1-5.

The pipeline consists of the following stages:

- Fetch (F)—One of the instruction memory spaces is addressed and returns an instruction.
- Decode (D)—The instruction is decoded; the register file is addressed and returns operands.
- Execute (E)—The ALU computes a result.
- Memory (M)—External memory is addressed (for load and store instructions only; this stage is idle for other instructions).
- Writeback (W)—The result (or loaded memory datum) is written into the register file.

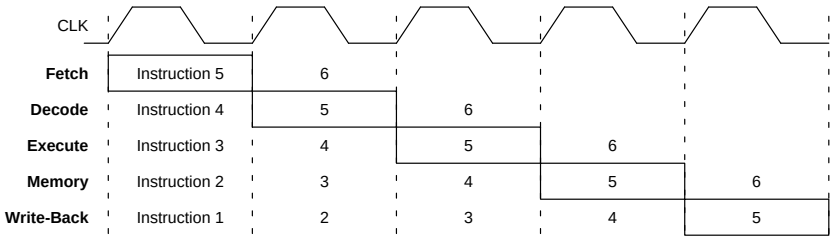


Figure F1-5. Instruction Pipeline

No instructions execute out-of-order; that is, if instruction A enters the pipeline before instruction B, then instruction A necessarily reaches the writeback stage before instruction B. Conditions that hold up the pipeline, and the effect of traps on pipeline operations, are discussed in the *Internal Architecture* chapter.

F1.5.2 Instruction Cache

An on–chip instruction cache allows the building of a high–performance system without incurring the cost of fast external memory and the associated control logic. The icache

in the MB86933H–20 processor is 1Kbyte in size and is organized into one bank of sixty four 16–byte lines. Cache lines are refilled in 4–byte increments to avoid the interrupt latency incurred by long uninterruptible cache line replacement.

F1.5.3 Bus Interface Unit

The Bus Interface Unit (BIU) contains the logic that allows the processor to communicate with the system.

F1.6 External Interface

The processor's external interface consists of signals, bus operations, and system support functions. This section gives an overview; details are discussed more fully in the *External Interface* chapter. The *System Design Considerations* chapter discusses issues that are likely to arise in the design of MB86933H–20–based system.

F1.6.1 Signals

The processor's external signals, illustrated in Figure F1-6, can be grouped by function as follows:

- Processor Control and Status—Reset, error, and clock signals.
- Memory Interface—Data and address buses, ASI and byte-enables, chip-selects, and other control signals used to access external memory and memory-mapped devices.
- Bus Arbitration—Signals used by external devices in requesting, and by the processor in granting, control of the bus.
- Peripheral Functions—Interrupt-requests and timer overflow.
- Boundary-Scan—Test signals used for hardware verification.
- ROM Size—Used to identify ROM size.

F1.6.2 Bus Operation

At any given time the Bus Interface Unit is handling requests for external memory and I/O operations, is arbitrating for bus access, or is idle. From the point of view of the external system, bus transactions are handled in fairly standard ways:

- Memory and I/O Operations—Read and write transactions are initiated with the BIU asserting the –AS signal. The RD/–WR output indicates the transaction type. The

–BE[3:0] outputs indicate the transaction width. The BIU drives the address and ASI signals, and either drives (during stores) or reads (during loads) the signals on the data bus. The transaction ends when the external system or programmable wait-state generator asserts –READY.

An atomic load-store is executed as a load followed immediately by a store, with no operation allowed between. The –LOCK output is asserted to indicate that the bus is being used for more than one consecutive memory operation.

- Arbitration—Any external device can request ownership of the bus by asserting the –BREQ signal. The BIU three-states its bus drivers and asserts –BGRNT to indicate that it is relinquishing control of the bus. Upon completion of its transaction the external device de-asserts –BREQ, and the BIU responds by de-asserting –BGRNT during the following cycle.

Chapter 4 of this addendum contains bus timing diagrams and a bus state diagram, further describes bus operations, and describes transactions that are interrupted by exceptions.

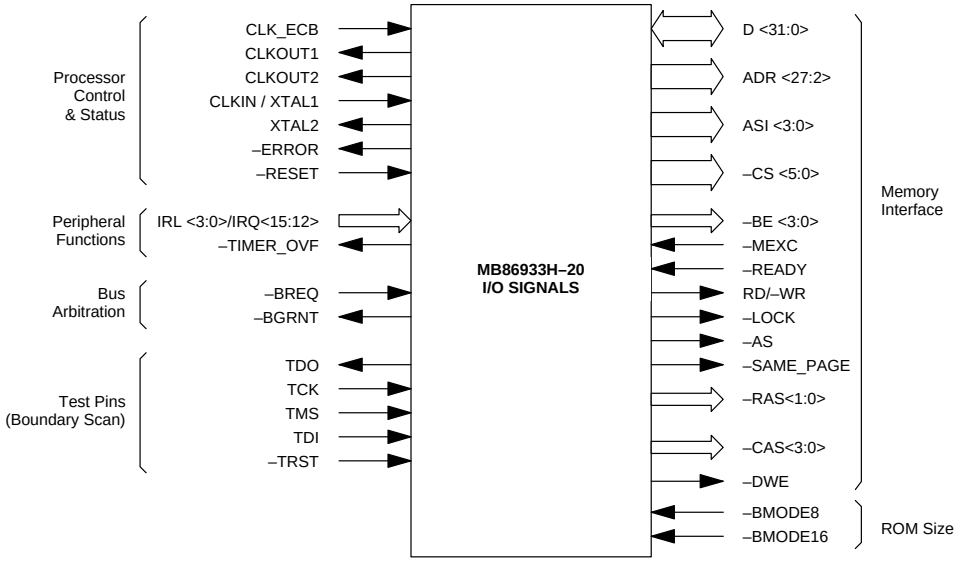


Figure F1-6. Input and Output Signals

F1.6.3 System Support Functions

MB86933H–20 system support is the same as MB86930 system support. Please refer to Section 1.5.3 of the main section of this manual for a description of the system support functions.

F1.7 Development-Support Tools

The MB86933H–20 development-support tools are the same as the MB86930 development-support tools. Please refer to section 1.6 of the main section of this manual for a description of the development-support tools.

CHAPTER F2

Programmer's Model

This chapter describes the MB86933H–20 processor resources that are available to software. It discusses the user and supervisor modes, the organization of the address space, the processor registers, the supported data types, the instruction set, the on–chip cache ,and interrupts and traps. A separate section describes the internal state of the processor after reset.

The *Programming Considerations* chapter contains information about how to use these processor resources to best advantage.

2.1 Program Modes

The SPARC architecture provides two mutually exclusive modes of program execution, *user mode* and *supervisor mode*. The processor is in supervisor mode when the S bit of the Processor State Register (PSR) is 1, and in user mode when this bit is 0. Instructions which access either special-purpose registers or alternate memory spaces are privileged. The use of *privileged* instructions is restricted to supervisor mode.

Separate user and supervisor modes provides system protection in multitasking environments. System code runs in supervisor mode and has full access to processor resources, while application code runs in user mode and is prevented from having unwanted side effects. Embedded systems connected to a network can use a protection scheme based on the distinction between user and supervisor modes. In such a scheme, network service routines intended to have system-wide effects run in supervisor mode. Routines intended to have only local effects, on the other hand, run in user mode.

In many embedded systems, however, this hierarchy is not required, and the processor can operate exclusively in supervisor mode. In this way, application code can directly manipulate the Current Window Pointer (in the PSR) and other processor control fields.

On reset, the processor is in supervisor mode. To enter user mode, software must clear the S bit in the PSR. The processor enters supervisor mode from user mode only when a hardware reset, an interrupt, or a trap occurs. A return from trap (RETT) instruction restores the value the S bit had before the trap was taken.

2.2 Memory Organization

The processor can directly address up to 4 Gb of memory, organized into 16 address spaces of 256 Mb each. These address spaces may or may not overlap in physical memory, depending on the system design. Every external access involves a 4–bit Address Space Identifier (ASI) as well as a 26-bit word address. The ASI selects one of the address spaces, and the address selects a word within that space (see Table F2-1).

Only the user instruction and data spaces are accessible in user mode. The other 14 address spaces can be accessed only in supervisor mode.

Table F2–1. ASI Address Space Map

ASI <3:0>	Address Space
0x0	Application Definable
0x1	Control Registers
0x2	Instruction Cache Lock
0x3	Reserved
0x4 – 0x7	Application Definable
0x8	User Instruction Space
0x9	Supervisor Instruction Space
0xA	User Data Space
0xB	Supervisor Data Space
0xC	Instruction Cache Tag Ram
0xD	Instruction Cache Data Ram
0xE – 0xF	Reserved

Note that the MB86933H–20 has no data cache and has six register windows rather than eight. It has twenty-six Address Bus signals (ADR<27:2>) rather than thirty, four Address Space Identifier signals (ASI<3:0>) rather than eight, no emulator-support signals, and no memory management unit. These and other differences between the MB86933H–20 and other SPARClite processors should be considered when porting code to the MB86933H–20 from another SPARClite processor, and when porting code from the MB86933H–20 to another SPARClite processor. Documentation for other SPARClite should be referenced to identify differences with the MB86933H–20 that may affect ported code.

Loads and stores are the only instructions that cause external accesses. Versions of these instructions exist for transferring bytes, half-words, words and double words between memory (or I/O) and processor registers. Addressing conventions for external accesses are “big-endian”:

- *Bytes*—Increasing the address decreases the significance of a byte within the word. That is, the most significant byte of a word—the “big end” of the word—is accessed when bits [1:0] of the address are both 0. The least significant byte is accessed when address bits [1:0] are both 1.
- *Halfwords*—The most significant halfword of a word is accessed when bit 1 of the address is 0, and the least significant halfword when address bit 1 is 1.
- *Doublewords*—The most significant word of a doubleword is accessed when bit 2 of the address is 0, and the least significant word is accessed when address bit 2 is 1.

The address of a halfword, word, or doubleword is the address of its most significant byte. The addressing conventions are illustrated Figure F2-1.

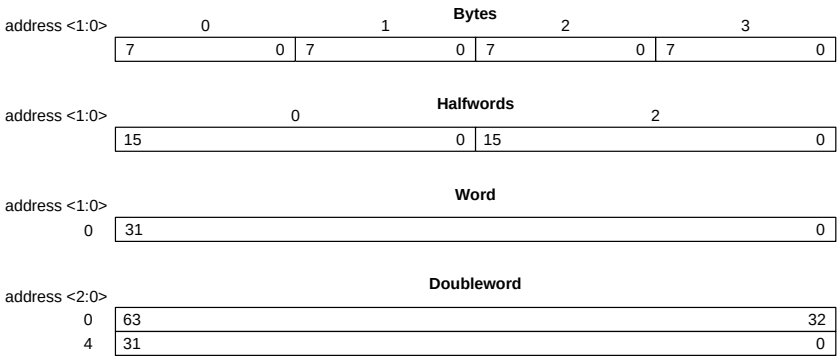


Figure F2–1. Addressing Conventions

Load and store operations require proper alignment of data in memory. An aligned doubleword address is divisible by 8, an aligned word address is divisible by 4, and an aligned half-word address is divisible by 2. If a load or store instruction generates an improperly aligned address, a memory_address_not_aligned trap occurs, and the access must be performed piecemeal under software control.

The processor does not contain memory-management hardware. Virtual-address translation can be handled by software or by an external memory-management unit.

2.3 Registers

There are two types of registers: the *general-purpose* or *r registers* whose contents have no pre-assigned meaning, and the *special-purpose registers* that contain control and status information, or special-purpose data. All registers are 32 bits wide. The register set is illustrated in Figure F2-2.

The general-purpose (r) registers can be accessed in user mode. There are 104 r registers. Eight are *global registers*; the other 96 registers are divided into six overlapping blocks called *windows.*

There are of two kinds of special-purpose registers: (1) registers that are defined by the SPARC architecture, and (2) memory-mapped registers that control peripheral functions. Special instructions exist for reading and writing each SPARC register except the Program Counter and the Next Program Counter. The memory-mapped registers can be read and written with the alternate-space load and store instructions. All instructions that access special-purpose registers are privileged except reads and writes to the SPARC-defined Y register.

2.3.1 Register Windows

The general-purpose register set is organized into a set of 8 global registers and a set of overlapping windows, as specified by the SPARC architecture. There are 6 windows in the MB86933H–20. Each window contains 24 registers. Of these, 8 are *local* to the window, 8 are *"out"* registers shared with the adjacent window below, and 8 are *"in"* registers shared with the adjacent window above. This organization is illustrated in Figure F2-2.

Thirty-two general-purpose registers can be accessed directly at any time: the 8 global registers, and the 24 registers of the currently active window. The value in the Current Window Pointer (CWP) field of the Processor State Register (PSR) determines which window is active. (See Section F5.3 for register addressing conventions.)

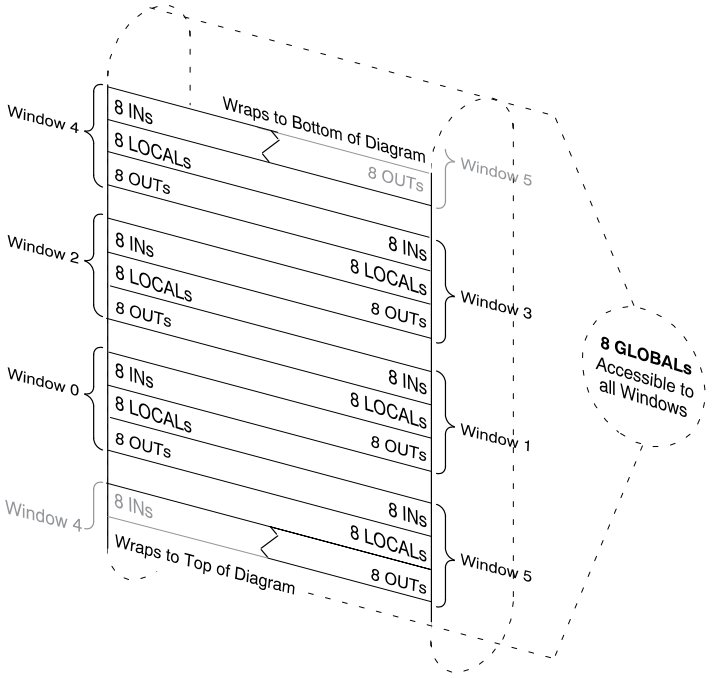


Figure F2–2. Register Windows

Register Addressing

Please refer to Section F2.3.1 of the main section of this manual for a description of MB86933H–20 register addressing.

Performance Features

Please refer to Section F2.3.1 of the main section of this manual for a description of the MB86933H–20 performance features.

2.3.2 Special Uses of the r Registers

Please refer to Section F2.3.2 of the main section of this manual for a description of MB86933H–20 r register use.

2.3.3 SPARC-Defined Special-Purpose Registers

The registers discussed in this section are defined as part of the SPARC architecture.

Processor State Register (PSR)

The Processor State Register is the primary processor control and status register. It contains 11 mode and status fields that configure the processor and report processor status and exception results. The *mode* fields, shown in upper case in Figure F2-3, are set by the operating system to configure the processor. The *status* fields, shown in lower case, are set by the processor to indicate the effects of instruction execution.

Except for several fields described below, the PSR can be written and read directly with the privileged instructions WRPSR and RDPSR. The PSR can also be modified by the SAVE, RESTORE, Ticc, and RETT instructions, and by any instruction that modifies the condition codes.

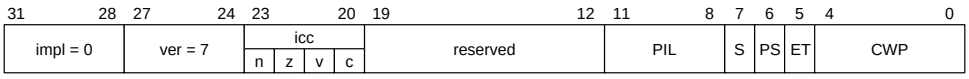


Figure F2–3. Processor State Register

Bits 31-28: Implementation (impl)—Identifies the implementation number of the processor as 0. The value in this field cannot be changed by a WRPSR instruction.

Bits 27-24: Version (ver)—Identifies the processor version as 7, and is intended for factory use. It can be read, but not written.

Bits 23-20: Integer Condition Codes (icc)—Contains the negative (n), zero (z), overflow (v), and carry (c) integer condition-code flags. These bits are modified by the WRPSR instruction, and by arithmetic and logical instructions whose names end with the letters *cc* (for example, ANDcc). The Bicc (Branch on integer condition codes) and Ticc (Trap on integer condition codes) instructions transfer program control based on the values of these bits. The integer condition code flags are defined as follows:

- n (Bit 23) Set to 1 if the ALU result was negative for the last instruction that modified the icc field; equal to 0 otherwise.
- z (Bit 22) Set to 1 if the ALU result was zero for the last instruction that modified the icc field; equal to 0 otherwise.
- v (Bit 21) If this bit equals 1, an arithmetic overflow occurred on the last instruction that modified the icc field; it equals 0 otherwise. Logical instructions that modify the icc field always reset the overflow bit to 0.
- c (Bit 20) If this bit equals 1, either an arithmetic carry out of bit 31 occurred on the last addition that modified the icc, or a borrow out of bit 31 occurred as the result of the last subtraction that modified the icc. The carry bit equals 0 otherwise. Logical instructions that modify the icc field always reset the carry bit to 0.

Bits 19-12: Reserved —This field is reserved. When using the WRPSR instruction, this field should always be written with 0s.

Bits 11-8: Processor Interrupt Level (PIL)—Specifies the levels of interrupt that the processor will accept. The processor accepts only interrupts with level 15 (non-maskable interrupts), or with levels higher than the value in the PIL field (maskable interrupts). Bit 11 is the most significant bit, and bit 8 is the least significant.

Bit 7: Supervisor Mode (S)—Determines whether the processor is in supervisor mode (S=1) or user mode (S=0). Since instructions that write the PSR are available only in supervisor mode, the processor enters supervisor mode from user mode only when a reset, trap, or interrupt occurs.

Bit 6: Prior S State (PS)—Records the value of the S bit when a trap is taken, so that the processor can return to the proper operating mode (user or supervisor) on return from the trap. Processor hardware changes the PS bit to the state of the S bit when entering a trap, and changes the S bit to the state of the PS bit when returning from the trap.

Bit 5: Enable Traps (ET)—Enables traps (ET=1). When ET=0, traps are disabled and all interrupts are ignored.

Bits 4-0: Current Window Pointer (CWP)—Points to the register window that is currently active. The CWP is written and read with the WRPSR and RDPSR instructions, is decremented by traps and the SAVE instruction, and is incremented by the RESTORE and RETT instructions. The MB96933H processor implements 6 of the 32 windows allowed in the SPARC definition, so only the 3 least significant bits of the CWP field are used. Arithmetic on the CWP is always performed modulo 6. Attempting to write a value to the CWP field that points to an unimplemented window results in an “illegal instruction” error.

Window Invalid Mask Register (WIM)

The Window Invalid Mask Register contains 6 register-window mask bits, each of which corresponds to an implemented register window. If an operation that normally increments or decrements the Current Window Pointer would cause the CWP to point to a window whose corresponding WIM bit equals 1, a Window Overflow or Window Underflow trap occurs.

The WIM can be written with the WRWIM instruction, and read with the RDWIM instruction. Both of these instructions are privileged. Bits corresponding to unimplemented windows are read as 0s; values written to these bits are ignored.

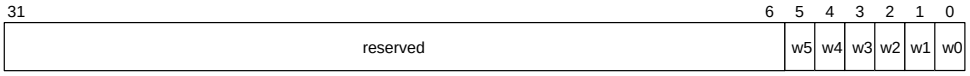


Figure F2–4. Window Invalid Mask Register

Bits 31-6: Reserved Field—This field is reserved for potential future expansion to additional windows.

Bits 5-0: Window Masks (W5-W0)—Window mask bits, with W5 the mask bit for window 5, etc.

Trap Base Register (TBR), Y Register, Program Counter, Next Program Counter, Ancillary State Registers,

Please refer to Section F2.3.3 of the main section of this manual for a description of these registers.

2.3.4 Memory-Mapped Control Registers

In addition to the registers defined by the SPARC architecture, the MB86933H–20 provides a collection of memory-mapped registers that control peripheral functions. Figure F2-3 shows these registers and their locations in memory. The memory-mapped registers can be read and written with the alternate-space load and store instructions, which are privileged.

0x00000000	ASI=0x1	Cache/Bus interface Unit Control Register
0x00000004	ASI=0x1	Lock Control Register
0x00000008	ASI=0x1	Lock Control Save Register
0x0000000C	ASI=0x1	Cache Status Register
0x00000010	ASI=0x1	Restore Lock Control Register
0x00000080	ASI=0x1	System Support Control Register
0x00000120	ASI=0x1	Same-Page Mask Register
0x00000124	ASI=0x1	Address Range Specifier Registers (ARSR <5:1>)
0x00000140	ASI=0x1	Address Mask Register (AMR <5:0>)
0x00000160	ASI=0x1	Wait-State Specifier Registers (WSSR <2:0>)
0x00000174	ASI=0x1	Timer Register
0x00000178	ASI=0x1	Timer Preload Register
0x000007D0 – 7D4	ASI=0x1	DRAM Configuration Registers

Figure F2–5. Locations of Memory-Mapped Control Registers

Cache/Bus Interface Unit Control Register

The Cache/BIU Control Register controls the operation of the instruction cache, and the write and prefetch buffers of the Bus Interface Unit. This register is located at address 0x00000000 with an ASI of 0x1.

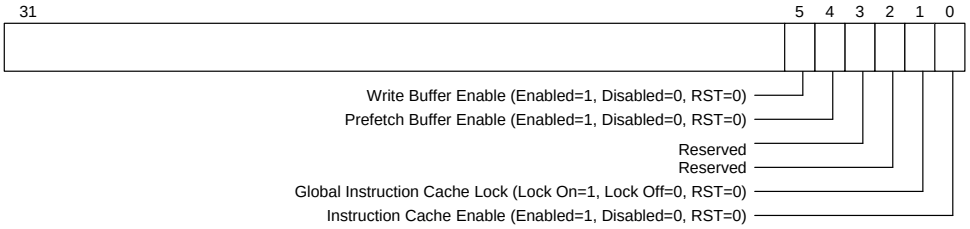


Figure F2–6. Cache/Bus Interface Unit Control Register

Bits 31:6 Reserved

Bit 5: Write Buffer Enabled—When set to 1, enables the write buffer of the BIU only if both the instruction and data caches are enabled. At reset, this bit is 0. This bit should be changed only when the instruction and data caches are off.

Bit 4: Prefetch Buffer Enabled—When set to 1, enables the prefetch buffer of the BIU only if both the instruction and data caches are enabled. At reset, this bit is 0. This bit should be changed only when the instruction and data caches are off.

Bits 3–2: Reserved

Bit 1: Global Instruction Cache Lock—Locks the current entries into the on-chip instruction cache; with this bit set to 1, no valid entry in the instruction cache will be replaced. To insure the best performance with the cache locked, invalid words in allocated cache locations will be updated. When this bit is 0, the cache operates normally. Writes to the Instruction Cache Lock bit do not affect cache operation for the following three instructions. At reset, this bit is 0.

Bit 0: Instruction Cache Enable—Turns the on-chip instruction cache on (1) and off (0). Writes to the Instruction Cache Enable bit do not affect cache operation for the following three instructions. At reset, this bit is 0.

Lock Control Register

The cache does not have to be enabled for the Lock Control Save Register to be updated, and the register is both readable and writable.

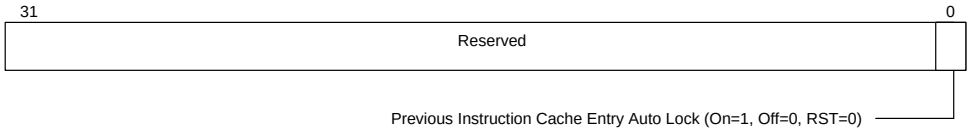


Figure F2–7. Lock Control Save Register

Bit 1: Reserved

Bit 0: Instruction Cache Entry Auto Lock—Enables (1) and disables (0) auto-locking for entries in the on-chip instruction cache. All instructions fetched while this bit is 1 have the lock bits in their cache tags set to 1. Writes to this bit do not affect cache operation for the following three instructions. At reset, this bit is 0.

Lock Control Save Register

When an external interrupt or hardware trap occurs, the auto-locking of entries in the on-chip cache is disabled. The Lock Control Save Register is used to re-enable auto-locking after the interrupt has been serviced. The register is updated with the contents of the Lock Control Register when there is a hardware interrupt, an exception condition (illegal instruction, memory data alignment error), or a DSU hardware breakpoint. The updated Lock Control Save Register is then used to restore the Lock Control Register after the interrupt or trap. This "autosave" feature allows restoration of the Lock Control Register following interrupts and traps that cannot be anticipated by software. In other cases, the program can save the Lock Control Register directly for later restoration.

The value of the Lock Control Register before the interrupt or trap is automatically saved in the Lock Control Save Register, located at address 0x00000008 with an ASI of 0x1. The correct auto-lock value is restored in the Lock Control Register by setting bit <0> in the Restore Lock Control Register to 1. This causes the value that is saved in the Lock Control Save Register to be moved to the Lock Control Register when a RETT is executed (See Section F2.6.2).

The cache does not have to be enabled for the Lock Control Save Register to be updated, and the register is both readable and writable.

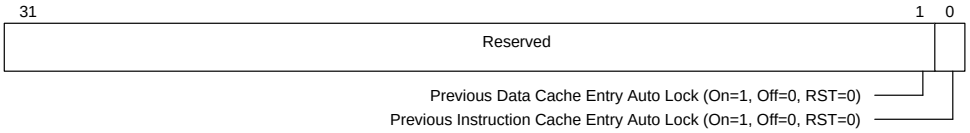


Figure F2–8. Lock Control Save Register

Restore Lock Control Register

On return from an external interrupt or hardware trap service routine, the Lock Control Register can have its previous value restored from the Lock Control Save Register. The Restore Lock Control Register, located at address 0x00000010 with an ASI of 0x1, controls this feature. When bit 0 of this register is set to 1 and a RETT instruction is executed, the value in the Lock Control Save Register is placed into the Lock Control Register.

There should be no traps between writing a 1 to bit 0 of the Restore Lock Control Register and the corresponding RETT instruction. This bit is cleared to 0 on reset, and also when a return from external interrupt or hardware trap is executed.

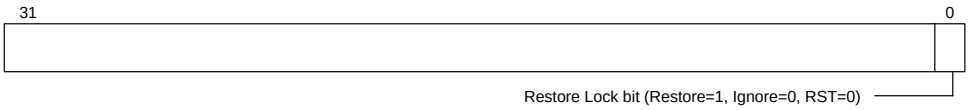


Figure F2–9. Restore Lock Control Register

Cache Status Register

If an attempt is made to lock a cache entry which is already locked, bit 0 in the Cache Status Register is set to 1. This bit can be cleared by software. The Cache Status Register is located at address 0x0000000C with an ASI of 0x1.

The Cache Status Register is meaningful only when auto-locking is utilized. In the case of writing the cache tags manually to lock cache lines (either by writing the Tag Lock

Bit address or the Cache Tag address directly), an attempt to lock a line which is already locked will not be indicated by the Cache Status Register.

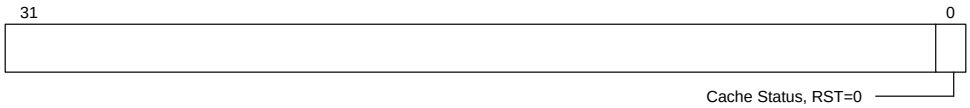


Figure F2–10. Cache Status Register

Same-Page Mask Register

The Same-Page Mask Register controls the operation of the same-page detection logic by specifying which bits of the current ASI and address are to be compared with those of the previous ASI and address. If the specified (i.e., unmasked) bits all match, then the processor recognizes the two accesses as being "in the same page," and asserts the –SAME_PAGE signal. These registers should not be written if the bus interface unit will handle addresses that are affected by the change in the next 3 processor cycles. The Same-Page Mask Register is located at address 0x00000120 with an ASI of 0x1.

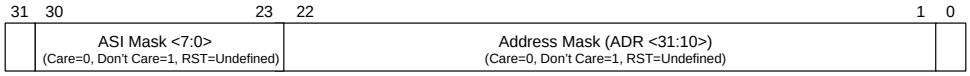


Figure F2–11. Same-Page Mask Register

Bit 31: Reserved

Bits 30-23: ASI Mask—Specifies which bits in the ASI of the current external access are to be compared with the corresponding bits in the ASI of the previous access. Only those bits are compared for which the mask bit is 0. Mismatches in any other bits do not prevent the two accesses from being recognized as "on the same page." The bits of this field are cleared to 0 on reset.

Bits 22-1: Address Mask—Specifies which of the 22 most significant bits in the address of the current external access are to be compared with the corresponding bits in the address of the previous access. Only those bits are compared for which the mask bit is 0. Mismatches in any other bits do not prevent the two accesses from being recognized as "on the same page." The bits of this field are cleared to 0 on reset.

Bit 0: Reserved

Note: Since DRAM uses SAMEPAGE pin to determine the wait–state, 933H supports the same page only in the range of chip–select 4. The BIU will save the last address if it is in the range of CS4. If the current address is in the range of CS4 and same page with the last address then the SAMEPAGE pin will be asserted. Thus if the user wants to use SAMEPAGE_pin, memory must be set in the range of CS4.

Address Range Specifier Registers (ARSR[5:1])

Values in the Address Range Specifier Registers define up to five different address ranges, which are used for various system-support functions. The ARSRs are located in a contiguous block beginning at address 0x00000124 with ASI 0x1 (see Table F2-2).

The ARSRs, together with the Address Mask Registers, can be used to control the assertion of the Chip-Select outputs (–CS[5:1]). –CSn is asserted when the value on the address bus falls in the address range specified by ARSRn and AMRn. See the discussion of the Address Mask Registers, below. –CS0 is asserted when the value on the address bus, as masked by AMR0, falls into the lowest range of Supervisor Instruction Space. The range of –CS0 (as masked by AMR0) is 8K words.

These registers should not be written if the bus interface unit will handle addresses that are affected by the change in the next 3 processor cycles. The user should be careful that two chip selects are never selected at the same time. A programmable wait-state generator is also associated with each address range. See the discussion of the Wait-State Specifier Registers, below.

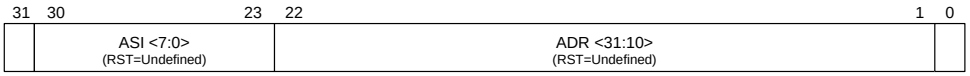


Figure F2–12. Address Range Specifier Registers

Bit 31: Reserved

Bits 30-23: ASI[7:0]—Specifies the ASI of a target address range. The value of this field is undefined on reset.

Bits 22-1: ADR[31:10]—Specifies the 22 most significant bits of a target address range. The value of this field is undefined on reset.

Bit 0: Reserved

Address Mask Registers (AMR[5:0])

AMRn works with ARSRn to define an address range. AMRn specifies which bits of the currently driven ASI and address are to be compared with the contents of ARSRn, and which bits are “don’t cares.” Except for AMR0, reset leaves the values in the AMR registers undefined (see Table F2-2). These registers should not be written if the bus interface unit will handle addresses that are affected by the change in the next 3 processor cycles. The AMRs are located in a contiguous block beginning at address 0x00000140 with ASI 0x1.

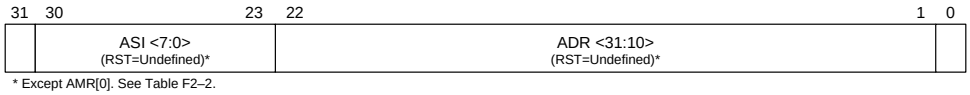


Figure F2–13. Address Mask Registers

Bit 31: Reserved

Bits 30-1: Mask—Specifies which bits in the ASI and address of the current external access are to be compared with the corresponding bits in the address-range specifier. Only those bits are compared for which the mask bit is 0. See Table F2-2 for reset value.

Bit 0: Reserved

Wait-State Specifier Registers (WSSR[2:0])

The wait-state specifiers determine, for each of the address ranges defined by the ARSR and AMR registers, the number of clock cycles between the time an address in a given range appears on the address bus and the time the processor generates an internal –READY signal. This makes it possible for memory and I/O devices with different access times to be connected to the processor without additional logic.

The wait-state specifiers for the six address ranges are kept in three Wait-State Specifier Registers. These registers are located in a contiguous block beginning at address 0x00000160 with ASI 0x1 (see Table F2-2). Each register contains the wait-state specifiers for two address ranges. When the address currently being driven by the processor matches the unmasked bits in one of the Address Range Specifiers, the corresponding wait-state specifier is selected. These registers should not be written if the bus interface unit will handle addresses that are affected by the change in the next 3 processor cycles.

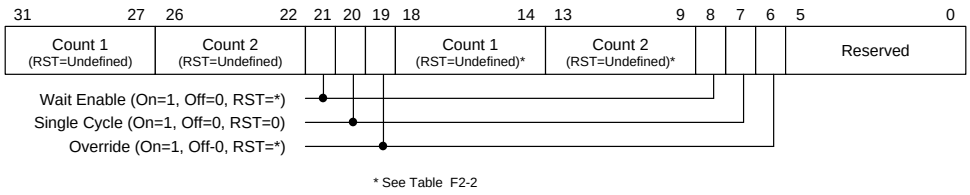


Figure F2–14. Wait-State Specifier Registers

Bits 31-19: Wait-State Specifier—When an external access falls within an address range defined by an ARSR and AMR, the corresponding wait-state specifier determines when, and whether, the processor generates an internal –READY signal to terminate the access.

Count1 (Bits 31-27): The number of wait-states inserted before the internal –READY, under the following conditions: the Single Cycle bit equals 0 and the current access is not on the same page as the previous access. The number of wait-states is the value of this field +1 (i.e., 0=1 wait-state, 1=2 wait-states, etc.) The value of Count1 is undefined on reset.

Count2 (Bits 26-22): The number of wait-states inserted before the internal –READY, under the following conditions: the Single Cycle bit equals 0 and the current access is on the same page as the previous access. The number of wait-states is the value of this field +1 (i.e., 0=1 wait-state, 1=2 wait-states, etc.) The value of Count2 is undefined on reset.

Wait Enable (Bit 21): Enables and disables the wait-state generator for an individual address range. If the Wait Enable bit of a wait-state specifier equals 0, the internal –READY is not asserted when addresses in the corresponding range are accessed by the processor. If Wait Enable is 1, the single cycle bit must be 0. See Table F2–2 for reset value.

Single Cycle (Bit 20): Specifies the timing of the internal –READY signal. If the Single Cycle bit equals 1 when an address in the appropriate range is accessed, the internal –READY is asserted in the same cycle. If the Single Cycle bit equals 0, and the current transaction is in the same page as the previous transaction, then Count2 is used as the number of cycles after which –READY is asserted internally. If the transaction is not in the same page, Count1 is used instead. If Single Cycle is enabled, the Wait Enable bit must be 0. See Table F2-2 for reset value.

Override (Bit 19): Allows the system to terminate a memory transaction before the internally specified time. If the Override bit equals 1, and external hardware asserts the external –READY signal, then the wait-state generator will stop counting and will wait for the next transaction. This bit is cleared to 0 on reset.

Bits 18-6: Wait-State Specifier—The wait-state specifier for a second address range. This field is organized just like bits 31-19.

Bits 5-0: Reserved

System Support Control Register

The System Support Control Register enables or disables the various system-support features, independently of one another. However, the chip-select logic for address range 0 is always enabled, regardless of the value in the System Support Control Register. This register is located at address 0x00000080 with ASI 0x1 (see Table F2-2).

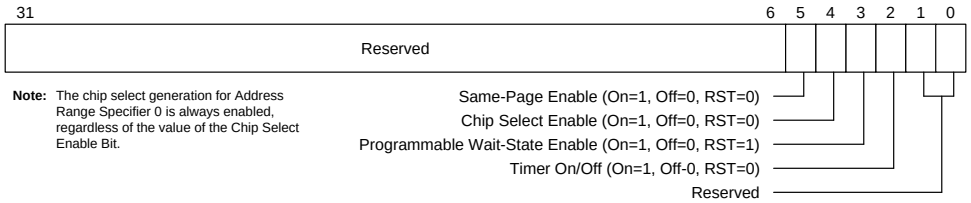


Figure F2–15. System Support Control Register

Bits 31-6: Reserved

Bit 5: Same-Page Enable—Enables (1) and disables (0) the same-page detection logic. When this bit is 1, the –SAME_PAGE signal is asserted whenever the address of an external access is on the same page as the previous access. The page size is controlled by the Same-Page Mask Register (see above). When this bit is 0, –SAME_PAGE is never asserted. The Same-Page Enable bit is cleared to 0 on reset.

Bit 4: Chip Select Enable—Enables (1) and disables (0) the generation of chip-select signals for external accesses in address ranges 1 through 5. Regardless of the state of this bit, however, –CS0 is always asserted when the current address lies in address range 0. The Chip Select Enable bit is cleared to 0 on reset.

Note: Before enabling chip selects all chip select Address Mask and Address Range registers should be initialized so that two chip selects are never selected at the same time.

Bit 3: Programmable Wait-State Enable—Enables (1) and disables (0) the programmable wait-state generators for all address ranges. The Programmable Wait-State Enable bit is set to 1 on processor reset.

Bit 2: Timer On/Off—Enables (1) and disables (0) the timer. This bit is cleared to 0 on reset.

Bits 1-0: Reserved

Table F2–2. System Support Register Summary

Chip Selects	Affected by Chip-Select Enable?	Address Range Specifier		Address Mask		Wait-State Specifier	
		Address (ASI=0x01)	Value at Reset	Address (ASI=0x01)	Value at Reset	Address (ASI=0x01)	Value at Reset
0	No	N/A	ASI=0x09 ADR<31:10>=0	0x0000 0140	All mask bits 0 except ADR<14:10> = 1	0x0000 0160 (low halfword)	Count 1,2 = 31 Wait Enable=1 Single Cycle =0 Override=1
1	Yes	0x0000 0124	Undefined	0x0000 0144	Undefined	0x0000 0160 (high halfword)	Count 1,2 = Undefined Wait Enable =0 Single Cycle =0 Override=0
2		0x0000 1280		0x0000 0148		0x0000 0164 (low halfword)	
3		0x0000 012C		0x0000 014C		0x0000 0164 (high halfword)	
4		0x0000 0130		0x0000 0150		0x0000 0168 (low halfword)	
5		0x0000 0134		0x0000 0154		0x0000 0168 (high halfword)	

Timer Register

The Timer Register contains the current count of the internal 16-bit timer. When the timer overflows, the processor asserts the –TIMER_OVF signal and reloads the Timer Register with the contents of the Timer Preload Register. The Timer Register can also be loaded directly by writing to the address 0x00000174 with ASI 0x1. The timer is clocked at the processor clock frequency.

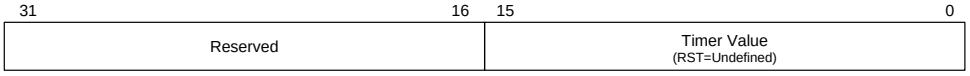


Figure F2–16. Timer Register

Timer Preload Register

The Timer Preload Register contains the value which is loaded into the timer when the timer overflows. In effect, this register specifies the number of clock cycles between assertions of the –TIMER_OVF signal. The Timer Preload Register is located at address 0x00000178 with ASI 0x1.

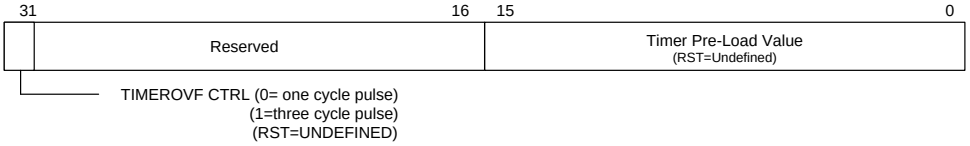


Figure F2–17. Timer Pre-Load Register

2.4 Data Types

Please refer to Section F2.4 of the main section of this manual for a description of the data types.

2.5 Instructions

Please refer to Section F2.5 of the main section of this manual for a description of the instructions. Note that *modulo 8* in the description becomes *modulo 6* for the MB86933H–20 processor.

2.6 Interrupts and Traps

Please refer to Section F2.7 of the main section of this manual for a description of the interrupts and traps. Note that *modulo 8* in the description becomes *modulo 6* for the MB86933H–20 processor.

F2.6 Instruction Cache

The MB86933H–20 has a 1K–byte instruction cache on chip which is designed for maximum flexibility of operation. Under software control, individual entries or the entire cache can be locked. This section discusses the structure and operation of the cache as seen from the programmer's point of view.

F2.6.1 Structure

In the MB86933H–20 processor, the instruction cache is 1 Kbytes in size, divided into 64 *lines* of 4 words (16 bytes) each. The contents of the cache data memory and tag memory is undefined at reset.

The cache organization, illustrated in Figure F2–16, is direct–mapped; that is, each address in memory is cached in a specific location. On a cache access, the address bits ADR[9:4] are used to select a line.

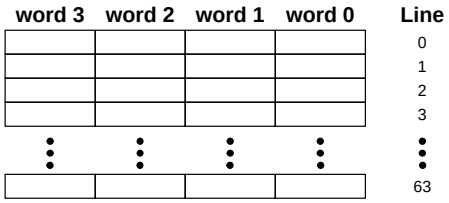


Figure F2–18. Cache Organization

Associated with each cache line is a *tag*, that indicates the memory location to which the line is currently mapped, and contains status information for the cached instructions. Cache tags are located in the address space indicated by ASI 0xC (see Table 2-2). A cache *entry* consists of a cache line together with the corresponding tag. The structure of a cache tag is illustrated in Figure F2–17.

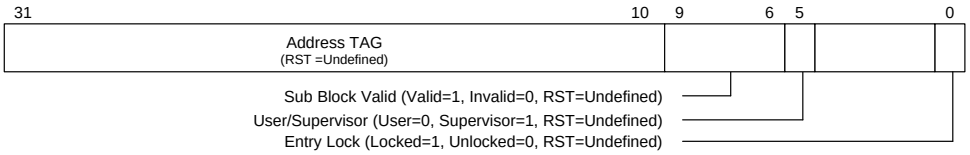


Figure F2–19. Cache Tag

Bits 31-10: Address Tag—Contains the 22 most significant bits of the memory address of the data or instructions cached in the corresponding line. Undefined on reset.

Bits 9-6: Sub-Block Valid—Contains one Valid bit for each of the 4 words in the corresponding line. When a Valid bit is 1, it indicates that the corresponding cache word contains a current data or instruction value for the address indicated by the tag. Undefined on reset.

Bit 5: User/Supervisor—Indicates whether the data or instructions cached in the corresponding line come from user space (User/Supervisor bit = 0) or from supervisor space (User/Supervisor bit = 1). Undefined on reset.

Bits 4-1: Reserved

Bit 0: Entry Lock—Locks the current address into the cache tag entry. An access which competes with currently locked entries in both banks of the cache is treated as non-cacheable. Undefined on reset.

A faster way to set and clear the tag entry-lock bits is to write the Tag Lock Bit addresses as shown in Table 2-2. Writes to these locations map to the same entry lock

bits in the cache tags described in Figure F2–17 above. The advantage of writing the entry lock bit using these alternate memory locations is that only the lock-bit is affected on a write, the reset of the associated tag is not affected. The same operation using the cache tag address would require a read-modify-write so as not to change the rest of the tag value.

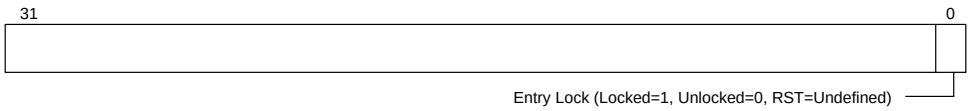


Figure F2–20. Tag Lock Bit

Bit 0: Entry Lock- Locks the current address into the cache tag entry. An access which competes with a currently locked entry in the cache is treated as non-cacheable. Writing this bit has the same effect as writing the corresponding bit in the cache tags except that the rest of the tag remains unaffected by a write to this location.

Table F2–3. Cache Tag Addresses

	Line	Cache Tag Address ASI=0xC	Tag Lock Bit ASI=0x2
Instruction Cache	0	0x 80000 0000	0x 80000 0000
	1	0x 80000 0010	0x 80000 0010
	2	0x 80000 0020	0x 80000 0020
	3	0x 80000 0030	0x 80000 0030
	4	0x 80000 0040	0x 80000 0040
	.	.	.
	.	.	.
	.	.	.
	63	0x 80000 003F0	0x 80000 03F0

F2.6.2 Operation

This section discusses software initialization of the caches and the various cache operating modes.

Initialization

On reset, the cache is turned off, and all memory requests are sent to the Bus Interface Unit. In order to use the caches, software must initialize the Valid and Entry Lock bits

by writing 0's to the appropriate alternate address spaces. After initializing the cache, a program can write 1's to the Cache Enable bit of the Cache/BIU control register to turn the cache on. Detailed code to accomplish this initialization is provided in Section F2.6.3. Due to the pipeline in the IU, all writes are delayed by three instruction cycles.

Normal Operation

Fetches from the user and supervisor instruction spaces, are generally cacheable. Stores to the instruction address space are not supported.

On any cacheable access, the address bits ADR[9:4] are used to select a line in the cache. Address bits ADR[3:2] are used to select a word from the line; the Valid bits corresponding to that word are checked. The address bits ADR[31:10] are compared with the address tag. The User/Supervisor bit is tested against the ASI indicated by the IU.

A *cache hit* occurs if all of the following are true:

- ADR[31:10] matches the address tag in either set.
- The User/Supervisor bit corresponds to the ASI indicated by the IU.
- The Valid bit corresponding to the word being accessed is 1.

In the case of a hit, the requested instruction is in the cache. The instruction is returned to the IU, and the pipeline is not held up. The lock bit may be updated based on the value of the Cache Entry Auto Lock bit in the Lock Control Register (see *Locking Modes*, below).

A *read miss* freezes the IU pipeline, and sends the request on to external memory. Though each cache line is four words long, only a single word is fetched on a miss. Assuming neither global nor local locking is in force, the fetched word will overwrite the appropriate word in one of the entries in the set. (Under global or local locking, a different policy is followed; see *Locking Modes*, below).

Sometimes a read miss occurs *only* because the Valid bit for the requested word is not set. In this case, a cache line has already been allocated for a 4-word memory block which includes the requested address. The fetched word simply overwrites the appropriate word in this line; the Valid bit for the word is then set.

Otherwise, a new line needs to be allocated on a read miss. The fetched word overwrites the appropriate word in this line; its Valid bit is then set, and the Valid bits for the other words in the line are cleared.

Locking Modes

Without locking, read misses can cause cache lines to be re-allocated. The entire cache, or selected entries corresponding to time-critical routines, however, can be locked into cache. Locked entries cannot be re-allocated.

The two modes of cache locking are:

- Global Locking — Affects the entire cache. When the cache is locked in this way, valid entries are not replaced; invalid words in allocated cache locations will be updated. Bits in the cache/Bus Interface Unit Control Register enable or disable the global locking mode independently for each cache. Enabling global locking does not affect the Entry Lock bits of individual Cache lines; when global locking is subsequently disabled, lines with clear Entry Lock bits are once again subject to re-allocation.
- Local Locking — Affects individual cache lines.

Bits in the Lock Control Register enable or disable an auto lock mode in which all subsequent cache accesses automatically set the Entry Lock bit of the accessed entry. Software can also lock and unlock an individual entry by writing the lock bit in that entry's tag.

With auto-locking enabled for the instruction cache, any lines accessed have their entry-lock bit set. This makes it easy to lock a routine into the cache by setting the auto lock bit in the Lock Control Register at the beginning of the routine and then executing the routine to lock the entries. The auto lock bit is cleared in one of two ways. Normally, software clears the auto lock bit at the end of the routine being locked. If a trap or interrupt occurs the auto lock bit will be cleared by hardware. This disables the locking mechanism so that the service routine is not locked into cache by mistake.

Two registers are provided to make it easy to re-enable the auto locking when the processor returns from the interrupt. The value of the Lock Control Register before the interrupt is automatically saved in the Lock Control Save Register when an interrupt or trap occurs. To restore the correct auto-lock value on return from the service routine, software sets a bit in the Restore Lock Control Register. This will cause the value saved

in the Lock Control Save Register to be moved to the Lock Control Register when a RETT is executed (see Figure F2–19).

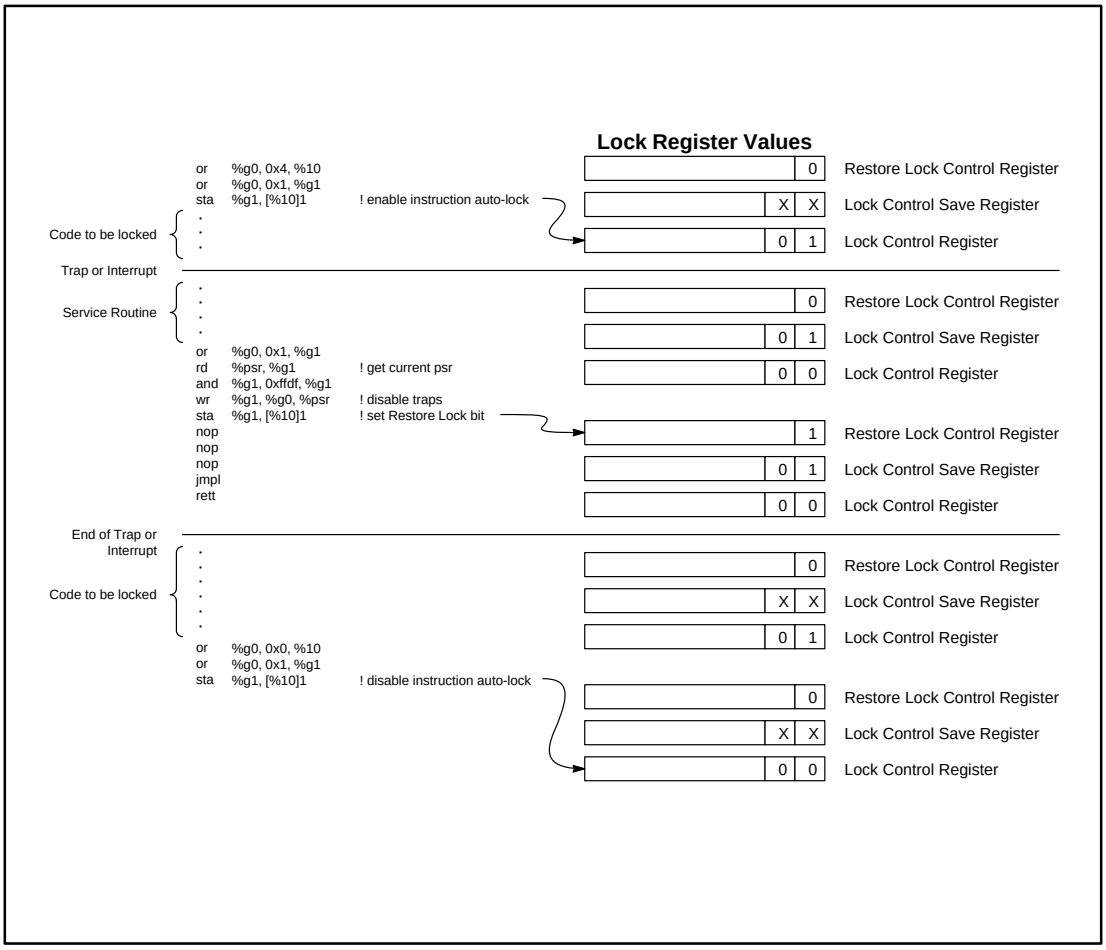


Figure F2–21. I–CACHE

2.6.1 Code for Initializing the On–Chip Cache

The following code initializes the instruction cache, then enables caching and BIU buffering as described in the *Initialization* item of Section F2.6.2.

```
#define ram_size         64
#define ini_tag          0
#define adr             0x80000000
#define CTL_BITS         0x31    /* turn on i–cache,  prefetch buf, write buf.*/

.seg "text"
     set    ram_size, %l7        /* RAM size */
     set    adr, %o2            /* start address */
     set    ini_tag, %l0         /* initial tag value */

loopinit:
     sta    %l0, [%o2] 0xc       ! write itag
     subcc  %l7, 1, %l7
     bne    loopinit
     add    %o2, 16, %o2

     set    0, %l1
     set    CTL_BITS,%i7         ! turn on caches.
     sta    %i7,[%l1]1
     nop
     nop
     nop
     nop
```

CHAPTER

F3

Internal Architecture

The MB86933H–20 internal architecture is illustrated in Figure F3-1. The processor consists of a Clock Generator, an Integer Unit, a 1k–byte instruction cache and a Bus Interface Unit. Internally, the various functional units are connected by separate instruction and data buses. A unified address bus and a unified data bus extend off-chip for connecting external memory and I/O.

This chapter discusses the individual functional units and gives an overview of the flow of data and control signals through the processor.

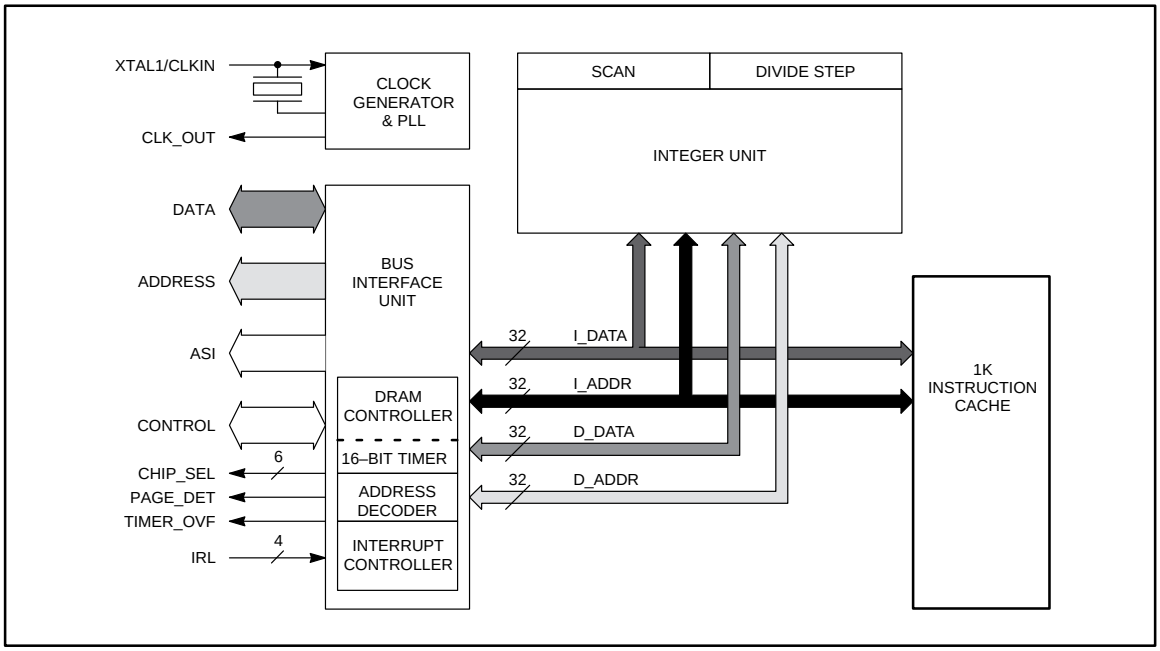


Figure F3–1. Internal Architecture (Block Diagram)

F3.1 Integer Unit

The Integer Unit (IU) is a compact, full-custom implementation of the SPARC architecture. It is hard-wired for maximum performance; that is, it uses no microcode. It contains three functional units:

- *Instruction Block*—Contains the instruction pipeline and decodes instructions into control signals for the other blocks.
- *Address Block*—Performs all instruction-address manipulations.
- *Execute Block*—Performs all data manipulations, and generates operand addresses for load and store instructions and effective addresses for some of the control transfer instructions.

The IU is based on a Harvard (Aiken) architecture, as shown in Figure F3-2. There are separate address buses for instructions and data. There are also two 32-bit data interfaces: the instruction data bus, and the data bus.

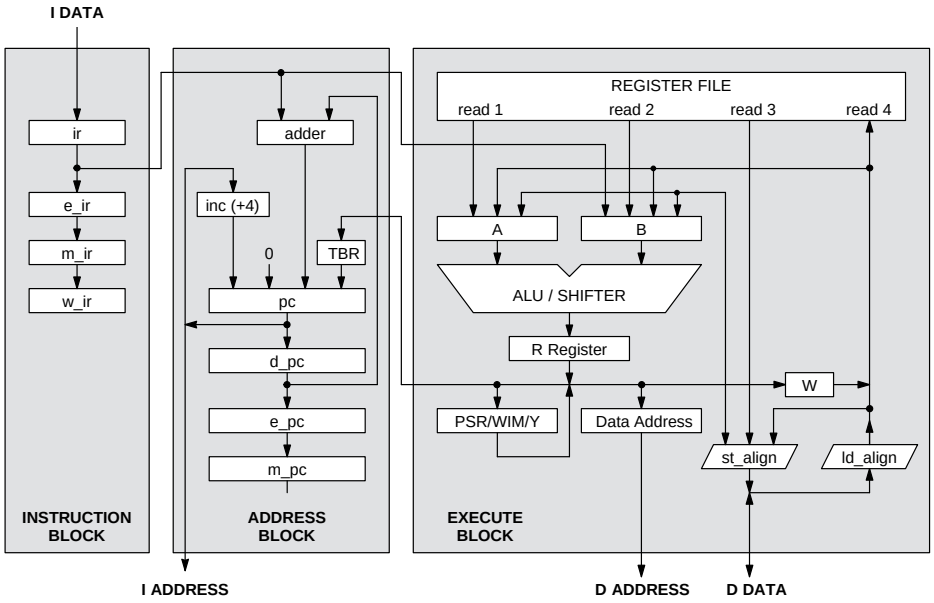


Figure F3–2. Integer Unit Data Path

F3.1.1 I Block

The instruction block (I Block) contains the five-stage instruction pipeline and the logic that decodes instructions into control signals for the rest of the IU. The I block detects all bypass and interlock conditions.

The main interfaces to the I block are:

- The Instruction data bus from the instruction cache or main memory.
- The Immediate data field that goes to the A block for computing PC relative control transfers and to the E block to be used as immediate data.
- Control signals to the A block and E block including the register file read and write addresses, register enable signals, multiplexer controls, and partly or fully decoded operation codes for the ALU/Shifter.
- Status signals back from the E block including possible trap conditions such as memory_address_not_aligned and tag_overflow.

Instruction Pipeline

The IU implements a five-stage instruction pipeline to allow a sustained execution rate of nearly one instruction per cycle. The operation of the pipeline under ideal conditions is illustrated in Figure F3-3. The pipeline consists of the following stages:

1. Fetch (F)—One of the instruction memory spaces is addressed and returns an instruction. (The figure below assumes a hit in the instruction cache.)
2. Decode (D)—The instruction is decoded; the register file is addressed and returns operands.
3. Execute (E)—The ALU computes a result.
4. Memory (M)—External memory is addressed (for load and store instructions only; this stage is idle for other instructions).
5. Writeback (W)—The result (or loaded memory datum) is written into the register file.

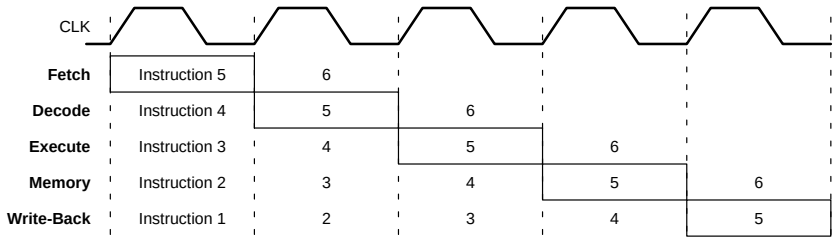


Figure F3–3. Instruction Pipeline

No instructions execute out-of order; that is, if instruction A enters the pipeline before instruction B, then instruction A necessarily reaches the writeback stage before instruction B.

The control logic for the instruction pipeline is illustrated in Figure F3-4. At each cycle a horizontal control word is available that is wider than 32 bits and controls every multiplexer, latch-enable, and unit op-code in the chip. The horizontal control word is composed of control signals that are active during the decode stage of instruction N, the execute stage of instruction N-1, the memory stage of instruction N-2 and the writeback stage of instruction N-3. Some control bits require no decoding and are simply hardwired from the appropriate bits in the instruction register. Because the SPARC instruction set is not completely orthogonal (not every instruction field has the same meaning in every instruction) most bits require some decoding based on a single instruction in the pipeline. Some control bits require decoding using logic that looks at two instructions in the pipeline - when controlling multiplexers to select data bypass paths, for example.

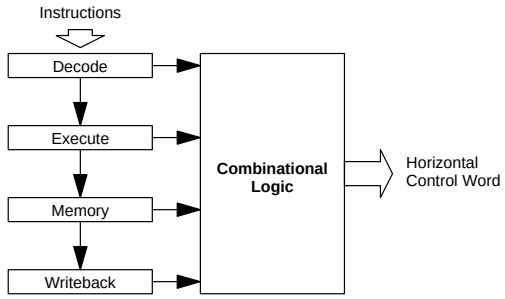


Figure F3–4. Instruction Pipeline Control Logic

Pipeline Hold

The IU does not complete one instruction on absolutely every cycle. During a load instruction, for example, external memory may be slow in returning the requested data. Because the IU does not execute or complete instructions out of order, the pipeline must be stopped until the requested data is returned. Only then can the instruction complete, and only then can the following instructions be executed.

There are also some hazards built into the IU data path that require interrupting the one-cycle-per-instruction sequence of the pipeline. For example, a doubleword load cannot be performed in one cycle because there is not enough memory or register-file bandwidth to move the data through the datapath. Another example is a load to a

register that is followed by an instruction that uses that register. Because the operand of the second instruction is required in the decode stage but is not available, this instruction must be delayed until the operand is available.

Conditions that hold up the processor pipeline are handled uniformly by the I Block control logic and are referred to as *hold conditions*. A complete list of possible hold conditions is given in Table F3-1.

The *interlock conditions* are:

- Load/Use Instruction Pairs—If a load instruction that has rd=N as its destination register is followed by an instruction that uses rs=N as one of its source operands, then the load must proceed through Writeback before the following instruction can enter the Execute stage.
- CALL/Use %r15 Instruction Pairs—Similarly, since the CALL instruction implicitly writes the current value of the PC into r15, it must proceed to Writeback before any following instruction that uses r15 can enter the Execute stage.

Any time an interlock is detected, a NOP is inserted into the pipeline. The address block is signaled, so that the address of the instruction that causes the interlock is replicated in the address pipe. The NOP itself cannot cause a trap.

Table F3–1. Conditions That Cause a Pipeline Hold

Name	Description	Pipeline Stage	Instruction Affected
ihold	Processor is attempting to fetch an instruction that is not yet available.	Fetch	Any instruction
dhold	Data is not yet available	Memory	Loads and Stores
mhold	Multiplication in progress	Execute	Integer Multiplication
Interlock	An instruction in the pipeline must wait for some prior instruction to be completed (through Writeback).		Load/Use and CALL/Use r15 Instruction Pairs
Multicycle Instruction	An instruction which inherently requires more than one cycle is in the pipeline	Execute	Load and Store Double-word, Atomic Load/Store

The multicycle instructions are LDD, LDDA, STD, STDA, LDSTUB, LDSTUBA, SWAP, and SWAPA. When a multicycle instruction enters the Execute stage, it and the instruction in the d_ir register are frozen for an additional cycle. Although it is possible to detect a multicycle instruction while it is in the Decode stage (unlike interlocks, which cannot be detected without looking at two instructions, those in the d_ir and e_ir registers), the I Block allows it to progress to the Execute stage before a hold is generated and inserted. This simplifies control somewhat because there are fewer points at which the pipeline must be held.

Note that the maximum number of internally generated hold cycles an instruction can cause is two, as in the following case:

```
LDD [%r1+%r2],%0r4
ADD %r5,%r5,%r6
```

The LDD takes two cycles, and it generates an interlock because the next instruction uses the data loaded in the second data memory cycle of the LDD instruction.

When a hold condition occurs, combinational logic generates one or more *freeze signals* that prevent latches from being updated, and hence keep the pipeline from advancing. For some holds—dhold, for example—the entire pipeline is frozen, with freeze signals being generated for all stages in the pipeline. For other holds—interlock conditions, for example—later stages in the pipeline must advance for the hold condition to be resolved. Thus only the earlier stages of the pipeline are frozen.

Trap Logic

The MB86933H–20 supports precise traps. That is, when a trap occurs, the saved programmer-visible state of the processor reflects the completion of all instructions prior to the trapped instruction, and no following instructions including the trapped instruction. Thus, when an instruction causes a trap, one of two statements is true:

- No results from that instruction have been written into the programmer-visible registers (the register file or the PSR, TBR, WIM, or Y registers).
- Or, if data has been written into a programmer-visible register, the data contained in that register prior to being written by the trapped instruction is saved by the processor and can be restored when the trap is taken.

Table F3-2 shows the pipeline stages in which the various trap conditions are detected.

Table F3–2. Detection of Trap Conditions

Priority	Trap Type	Stage Detected	Trap
1			reset (hardware reset)
1	—	D	reset
2	1	F	instruction_access_exception
3	3	D	priv_instruction
4	2	D	illegal_instruction
5	4	D	fp_disabled
5	36	D	cp_disabled
6	5	D	window_overflow
7	6	D	window_underflow
8	7	E	mem_address_not_aligned
10	9	M	data_access_exception
11	10	E	tag_overflow
12	128-254	D	trap_instruction (Ticc)
13	255	F	instruction_breakpoint
13	255	M	data_breakpoint
14	31		interrupt_level_15
15	30		interrupt_level_14
.	.		.
.	.		.
.	.		.
28	17		interrupt_level_1

As shown in Table F3-2, the last stage in which a trap can be detected is the Memory stage (a data memory exception for a load or store). If a programmer-visible register is updated prior to this stage, its original contents must be restored when and if the trap is taken.

Due to the pipelined operation of the IU, a trap condition for one instruction may actually be detected before a trap condition for a prior instruction. Thus, it is necessary to align the detected trap conditions so that all trap conditions for instruction N are considered together before any trap conditions resulting from instruction N+1 are considered.

The trap coder is illustrated in Figure F3-5. Its purpose is to align in time the (possibly several) trap sources for a single instruction to determine if a trap is to be taken or not and, if taken, to determine the highest priority trap and code its trap type.

.

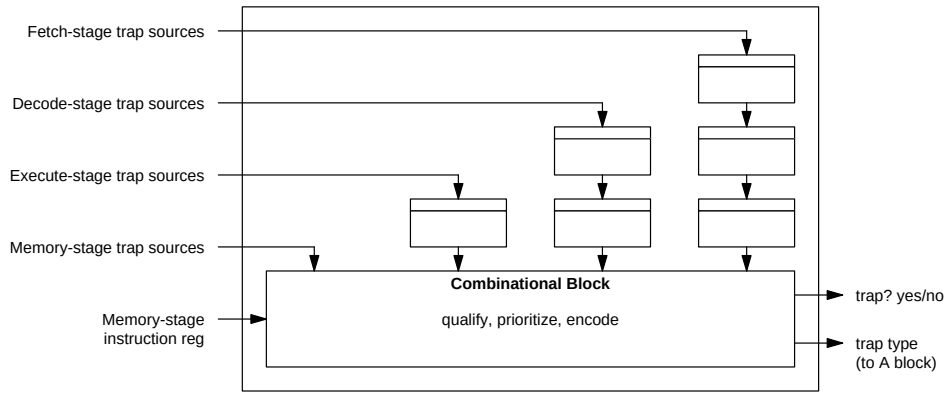


Figure F3–5. Trap Coder

When a trap is taken, the trap type field goes to the A Block where it is used immediately as a trap target address (when concatenated with the Trap Base Address) and is latched into the Trap Base Register.

F3.1.2 A Block

The A Block contains the address pipeline. Along with the E Block, it is responsible for all instruction-address manipulations. The A Block executes the CALL and Bicc instructions. The A Block and E Block are used together to execute the JMPL, Ticc, and RETT instructions. In these cases, the A Block controls the update of the Program Counter. The A Block's main interface to the rest of the chip outside the IU is the instruction address bus.

The address pipeline is illustrated in Figure F3-6. The fetch-stage program counter (PC) addresses instruction memory via the instruction address bus. Because a CALL, JMPL, or trap may require that the address of an instruction be written back to the register file, the address of every instruction tracks the instruction itself in the instruction pipeline so that it is available in the memory stage if it must to be written back to the register file. These address pipeline registers are the decode, execute, and memory program counters. Each of these registers contains the address from which the instruction register was fetched.

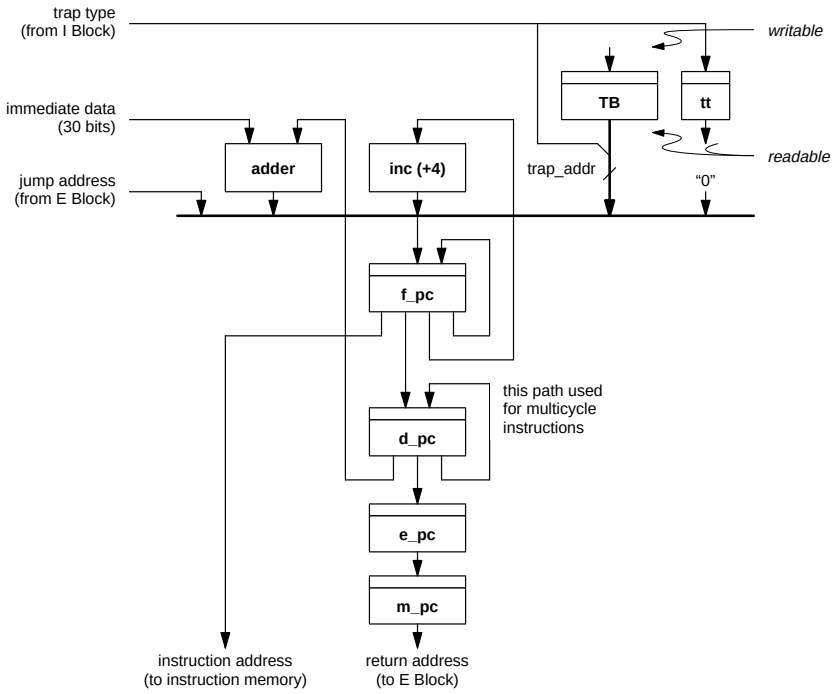


Figure F3–6. Address Pipeline

The PC has five possible sources:

1. +4 incrementer, for normal, sequential instruction fetch.
2. The address adder, for PC-relative control transfer (Bicc or CALL instruction). The immediate data field contains offset information and comes from the I Block.
3. The jump address for a JMPL or RETT instruction. The jump address bus contains jump target information and comes from the E block by way of the register file and ALU.
4. The TBR, concatenated with the trap type (tt) or with zeroes (when Single-Vector Trapping is enabled), during a Ticc instruction execution or an interrupt or trap. The trap type comes from the trap priority encoder, part of the I Block; when concatenated with TBR[31:12], it gives the target address for a trap.
5. Zeroes, concatenated with the trap type, for reset.

Note that "+4" is used to indicate that the (byte) address is incremented by 4 to fetch the next instruction. In reality, the two least significant bits of the address are not

implemented in hardware because they are never used. Word alignment, for the case of a jump address coming from the E Block is verified in the E Block (and to some extent, the I Block).

The return address bus is written back to the register file in the case of a CALL, JMPL or Trap.

Several control signals come from the I block. These include:

- PC input-select signals that control the PC input multiplexer.
- The address adder control signal, which determines whether a 30-bit or a 22-bit immediate address field is added to the previous value of the PC (now found in the decode-stage PC).
- Pipeline freeze signals that can prevent the updating of registers in the pipeline when a hold condition is detected.

F3.1.3 E Block

The E Block is responsible for all IU data manipulations. It generates operand addresses for load and store instructions, and effective addresses for some of the control transfer instructions.

As shown in Figure F3-7, the E Block contains the Store Align Unit (SAU), the Load Align Unit (LAU), the Register File (RF), and the Adder, Shift, and Logic Unit (ASLU). The E Block also contains the result bypass logic that determines which operands are driven into the ASLU, and the store bypass logic that determines what data is latched for stores.

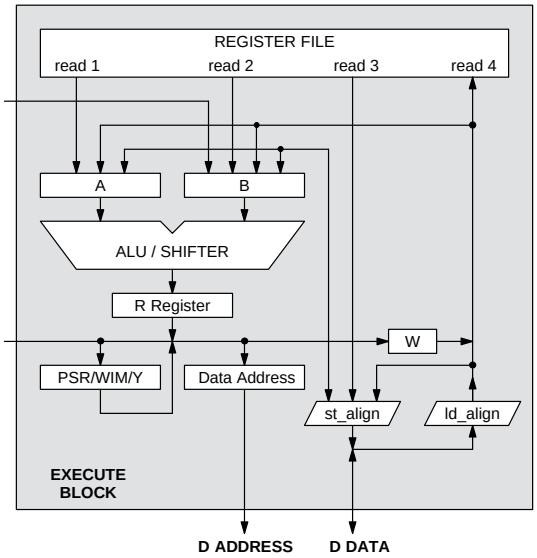


Figure F3–7. Execute Block

Adder, Shift, and Logic Unit (ASLU)

The ASLU incorporates an integer adder, a barrel shifter, a logic unit, and a scan unit. The integer adder calculates the results of the addition, subtraction, multiply-step, and divide-step instructions, and generates the carry, overflow, negative, and zero condition code values. It is used in load and store operations to calculate effective data addresses, and in register-indirect control transfers to calculate the new address to be placed in the PC register of the A Block. The integer adder also serves the multiplication unit by adding the “sum” and “carry” vectors during integer multiplications. The barrel shifter/logic unit executes the logic and shift instructions. The scan unit exists solely to support the scan instruction.

Results from the integer adder, the barrel shifter, the logic unit, and the scan unit are multiplexed into the R (Result) Register. Results from the integer adder are also made available to the Y Register.

Register File

The register file contains 104 registers of 32 bits each. The organization of these registers into windows is discussed in the *Programmer’s Model* chapter. The register file has one write port and three read ports. The write port is used for the instruction destination register (denoted *rd* in instruction descriptions). Two of the read ports are

used for the two instruction source registers (*rs1* and *rs2*). The remaining port is used for the data to be stored when a store or swap instruction is executed. In this way, even store instructions can be executed in a single cycle.

The register file also contains the address decoders for all four ports. Each address presented to the decoders consists of 8 bits derived from an instruction field, and the Current Window Pointer. These are physical addresses into the register file memory array.

Bypass Logic

As shown in Figure F3-7, the A and B operand registers have inputs that come from sources other than the register file or the immediate data bus. These inputs are results from previous instructions that have not yet written back to the register file. There are two such *bypass paths* in the E Block:

- *Result Bypass*—The result of an ALU operation in the R register is written back to the A or B operand register in the Memory stage of the following ALU operation.
- *Write Bypass*—The data in the W register is written to the A or B operand register, in the Writeback stage.

The result bypass path is selected when one instruction generates a result that can be used by the immediately following instruction. More precisely, if an instruction in the Decode stage of the pipeline has *rs1* = N, and the instruction in the Execute stage has *rd* = N, the *rs1* operand will not come from the register file, but directly from the R register in the ALU through the result bypass. Since an intervening SAVE or RESTORE instruction may have changed the Current Word Pointer, it is the *physical addresses* of the register source and destination that are compared, not the logical addresses (which depend on the CWP).

As an example, consider the instruction sequence:

```
add  %r1,%r2,%r3          ; r1 + r2 –> r3
add  %r3,%r4,%r5          ; r3 + r4 –> r5
```

The second add instruction takes its A source operand not from the register file, but directly from the result of the ALU through the result bypass.

The write bypass is selected when an instruction in the Decode stage has *rs1* = N, and the instruction in the Memory stage has *rd* = N. In this case, the *rs1* operand will not come from the register file, but from the W register through the write bypass. In the following instruction sequence, the third instruction uses the write bypass as its A source operand:

```
add %r1,%r2,%r3          ; r1 + r2 –> r3
add %r4,%r5,%r6          ; r4 + r5 –> r6
add %r3,%r7,%r8          ; r3 + r7 –> r8
```

If both bypass conditions apply, the result bypass takes precedence.

There is a third bypass path, called the *store bypass*, that is shown in Figure F3-7. The register file has a dedicated store port that is used for reading the rd register of a store instruction, which contains the data to be stored. The store port is read in the Execute stage of the store. When a store and the immediately preceding instruction access the same rd register, a bypass from the Writeback stage of the preceding instruction to the Memory stage of the store is needed. In the code sample below, the result of the first instruction becomes available to the Memory stage of the store by means of the store bypass path.

```
add%r1,%r2,%r3           ; r1 + r2 –> r3
st  %r3[%r4 + %r5]       ; r3 –> mem[r4 + r5]
```

Branch Evaluation Logic

The branch evaluation logic, which forms part of the E Block, evaluates branch conditions based on the current values of the integer condition codes of the PSR register. The icc bits n (negative), z (zero), c (carry) and v (overflow) form part of the branch evaluation block. The interpretation of these bits is discussed in the *Programmer's Model* chapter.

There are several ways that the icc bits can be modified. First, they can be written and read via the jump address bus by the instructions WRPSR and RDPSR.

Certain arithmetic instructions modify the icc bits as a side effect. When one of these instructions is executing, the new icc values are generated in the E Block during the Execute stage, latched at the end of this stage, and loaded into the PSR during the Memory stage.

Another path leads to the icc bits from the Writeback-stage copy of the PSR. When a trap occurs on an instruction that alters the icc bits, this path allows the pre-trap icc values to be restored to the PSR.

The combinational logic that performs the branch evaluation for the IU condition codes has as inputs:

- *Integer Condition Codes*—Directly from the ALU if the instruction in the Execute stage is one that can modify the icc, from the multiplication unit, or from the icc bits of the PSR if the instruction in the Execute stage is not one that can modify the icc.

- *The cond Field*—From the branch instruction in the Execute stage. (See the discussion of the Bicc instruction in the *Programmer's Model* chapter.)
- *Bicc Indicator*—A control signal that indicates whether the instruction in the Decode stage is a Bicc instruction. This signal remains valid into the Execute stage.

The output of the combinational logic is a single signal that, when active, causes the branch target address to be loaded into the PC during the Execute stage. Otherwise, PC+4 is loaded into the PC.

Load Align Unit (LAU) and Store Align Unit (SAU)

The LAU and SAU align data for loads and stores, respectively. Bytes and halfwords to be loaded are right-justified in a 32-bit word, and either sign-extended or zero-extended on the left, depending on whether the load instruction specified signed or unsigned operation. The LAU performs the alignment and extension during Writeback.

Byte and halfword stores take their data from the least significant byte or halfword of the register specified in the instruction's rd field. The SAU performs the necessary alignment for writing the data to the byte or halfword memory address specified in the instruction.

Multiply Unit

The E Block contains hardware to perform integer multiplications. The Multiply Unit (MU) multiplies two 32-bit signed or unsigned integers to produce a 64-bit product. Some multiplication instructions modify the integer condition codes as a side effect; others do not. The multiplication instructions are discussed in the *Programmer's Model* chapter.

The multiply hardware implements a version of *Booth's algorithm*. Booth's algorithm is similar to a "shift and add" multiply algorithm in that it scans the multiplier from the least significant to the most significant bit and, based on the bit string encountered, iteratively adds the multiplicand to produce partial products. It is also similar in that the resulting partial product is right shifted to ready it for the following iteration of the algorithm.

Booth's algorithm differs from a "shift and add" algorithm in that it can also be used directly with a negative multiplier (whereas "shift and add" requires a positive multiplier). It also differs in that the hardware must provide for both addition and subtraction of the multiplicand. In particular, a 1-bit Booth's algorithm examines two multiplier bits per iteration, looks for a bit transition, and either adds the multiplicand, subtracts the multiplicand, or adds zero to the existing partial product to produce the new partial product. It "retires" one bit of the multiplier per iteration.

Table F3-3 shows the possible bit transitions encountered in the multiplier for a 1-bit Booth, and the value that is added to the multiplicand for each transition.

Table F3–3. Booth's Algorithm

Multiplier Bits		
Current	**Previous**	**Add to Shifted Partial Product**
0	0	+0
0	1	+multiplicand
1	0	–multiplicand
1	1	+0

This technique can be extended so that more than one bit is examined during a given iteration. In particular, the MU performs an 8-bit Booth's algorithm. It examines 9 bits of the multiplier at a time and, based on the eight transitions of these nine bits, determines what multiple of the multiplicand to add to the old partial product to produce the new partial product. The addition is performed in the ALSU.

The MU produces 8 bits of the final product and "retires" 8 bits of the multiplier per cycle, and therefore requires only 5 cycles to do a 32x32 bit multiply (producing a 64-bit result).

The execution of the instruction is controlled by a synchronous state machine that generates control signals for the multiply hardware. Since instructions do not execute out of order, the Integer Unit (IU) must be frozen during the multiply instructions that require more than 1 cycle. Conceptually, the multiply instruction goes through all of the pipeline stages (F,D,E,M,W), but its Execute stage is from 1 to 5 machine cycles long. During the Fetch and Decode stages, the multiply instruction progresses like other instructions.

F3.1.4 Programmer-Visible State and Processor State

The SPARC Architecture defines the *programmer-visible state* of the processor as a collection of registers, and specifies the effects of instructions in terms of these registers. These definitions implicitly assume that every instruction completes before the next one begins. The MB86933H–20 processor, however, is pipelined, so that normally four instructions begin execution before the first one completes. The actual *processor state* (excluding the register file) therefore encompasses more than the programmer-visible state. For most of the programmer-visible registers, there is a corresponding register in the processor associated with the Writeback stage of the pipeline. That is, instructions normally update the register file and programmer-visible state registers in the Writeback stage.

An instruction may update staged copies of the PSR before Writeback, making the new values available to following instructions sooner; but these staged copies are not user visible. The PSR associated with the Writeback stage can never be updated early; if an instruction traps, it will not have altered any state that can not be restored.

F3.2 Instruction Cache

The MB86933H–20 includes an instruction cache, which allows designers to build high-performance systems without incurring the cost of fast external memory and its associated control logic. The software-visible features of the cache are discussed in detail in the *Programmer's Model* chapter, above. The instruction cache is 1Kbyte in size and is organized into one bank of sixty-four 16-byte lines. Cache lines are refilled in 4-byte increments to avoid the interrupt latency incurred by long, uninterruptible cache line replacements. The instruction cache is read-only and has two RAM arrays. There is one array for instruction memory and one for tags. In addition to the tag memory, the tag array also contains the logic to compare the address tag with the address that is being accessed. It also checks the VALID bits in the tag. The hit-detection logic is illustrated in Figure F3–8.

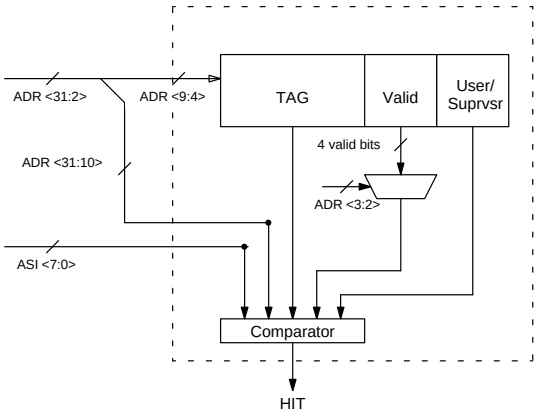


Figure F3–8. Cache Hit Detection Logic

F3.3 Bus Interface Unit

The Bus Interface Unit (BIU) contains the logic that allows the processor to communicate with the system. When the BIU performs a read, it returns the data to the IU.

The BIU also handles external requests for control of the bus. The external signals of the BIU and the relative timing of events in typical bus operations are discussed in the *External Interface* chapter that follows. That chapter also treats the various system-support features of the processor in detail.

Generally speaking, the main difference between the MB86933 and the MB86933H–20 is icache and on–chip DRAM controller. Because of the icache, the 933H can have prefetch buffer and write buffer to improve the performance of the CPU. The following will briefly describe some of the new features of 933H compared to the 933.

Prefetch buffer:

Same as the 932 chip, the prefetch buffer is used to fetch the next sequential instruction after an instruction cache miss. This buffer can be enabled by setting bit 4 of the CACHE/BIU Control Register. The prefetch buffer operates only when the instruction cache is on.

F3.3.1 Write Buffer

The purpose of the write buffer is to save write cycles to external memory with more than zero wait–state while IU can continue to fetch instructions from ICACHE. This buffer can be enabled by setting bit 5 of the CACHE/BIU Control Register. The write buffer operates only when the instruction cache is on.

F3.3.2 DRAM Control, Support

Since the 933H has the on–chip DRAM controller, BIU dedicates the range of CS4 for DRAM. If a user wants to use the on–chip DRAM controller, bit 6 of System Support Control register must be set.

Since DRAM uses SAMEPAGE pin to determine the wait–state, 933H supports the same page only in the range of chip–select 4. The BIU will save the last address if it is in the range of CS4. If the current address is in the range of CS4 and same page with the last address then the SAMEPAGE pin will be asserted. Thus if the user wants to use SAMEPAGE_pin, memory must be set in the range of CS4.

Bus Width Support

The bus mode of ROM bus (CS0) is determined by the BMODE16_ and BMODE8_ pins (Same as the 933). For other chip selects the bus mode is controlled by the Bus

Width Register. This register resides at ASI=0x01 and ADR=0x16C. It can also be read and written. The following is the format of this register.

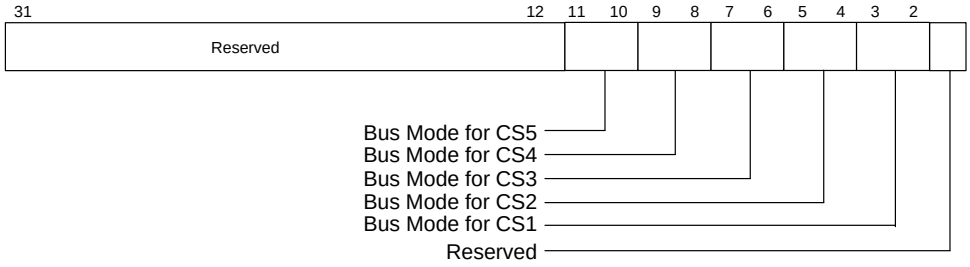


Figure F3–9. Buswidth Register

Table F3–4. Buswidth

Bus width <1:0>	Bus Mode
11	Invalid
10	16 Bit bus
01	8 Bit bus
00	32 Bit bus

Note: This register should not be written if the bus interface unit will handle addresses that are affected by the change in the next 3 processor cycles.

1. READ: Same as 933
2. WRITE: The 933H can support a write to non 32 bit bus memory without restriction. For 8–bit bus memory: the {ADR<27:2>, BE_<2>, BE_<3>} is the stored address. Writing a byte, halfword or word will take 1, 2 or 4 accessed cycles, respectively. Each access is commenced by an AS_ and terminated by a READY_; It is always started with the least significant byte first and so on. The D<7:0> is the data bus.

For 16–bit bus memory: the {ADR<27:2>, BE_<2>} is the stored address and BE_<1:0> are the byte enable. Writing a byte or halfword will take 1 accessed cycle, a word for 2 accessed cycles. For store word, each access is commenced by an AS_ and terminated by a READY_. It is always started with the least significant halfword first and so on. The D<15:0> is the data bus;

F3.3.3 Exception Handling

The external memory system can indicate an exception during a memory operation by asserting the –MEXC input. If –MEXC is asserted during an instruction fetch, the BIU indicates an instruction memory exception to the IU. If –MEXC is asserted during a data fetch, the BIU indicates a data access exception to the IU.

Any system that wants to recover from this error should store the address and data for the write causing the exception into a register. It should also have a status bit to indicate that the exception was caused during a write operation. It is the responsibility of the data access exception service routine to determine the cause of the exception, and to recover accordingly.

F3.3.4 Effect on the Pipeline

The pipeline hold signals, ihold and dhold, are asserted if an instruction or data cannot be made available in the cycle that it is required by the pipeline. In general the following hierarchy rules apply to the bus interface unit:

- The bus cycle currently in progress will complete
- If there is a pending request for a load or store operation, it will be serviced
- If there is a pending request for an instruction, it will be fetched.

The pipeline is stalled during every external memory access if the external –Ready signal or the internal Ready signal is not asserted. (See the Wait-State Specifier Registers description in Section F2.3.4 of the main section of this manual for a description of the internal Ready signal).

CHAPTER

F4

MB86933H–20 Interrupt Request Controller

The Interrupt Request Controller (IRC) is a 15-channel, programmable-trigger interrupt controller that arbitrates pending unmasked interrupt requests, encodes the highest-priority interrupt, and interrupts the processor. The system processor responds by servicing the interrupt and clearing the latched interrupt request in the IRC.

The SPARC V8 architecture, and the MB86933H–20 in particular, provides for up to 15 separate external interrupt sources. The MB86933H–20 has four external interrupt pins and an on–chip interrupt controller (IRC) which can support two modes of operation.

Mode 0 (IRL mode): In mode 0 the input on the four external pins is interpreted as an encoded interrupt vector. This mode allows for external logic to generate any one of the 15 possible interrupts (”0” represents ”no interrupt request”). In this mode of operation it is assumed that the external interrupt source maintains the interrupt vector on the pins until it is explicitly cleared by writing to an external memory mapped location. Note that this mode is the same as that on the MB86930/932/933 and is compatible with the MB86940 companion chip.

Mode 1 (IRQ mode): In the mode 1 the four pins are considered to be four separate interrupt sources mapping to interrupts 12 through 15. Note this mode is the same as that on the MB86931

Figure F4-1 shows a block diagram of the IRC.

The Trigger Mode Control logic selects one of four trigger modes for each channel: high level, low level, rising edge, or falling edge. The processor controls the triggers by writing to the Trigger Mode registers.

The IRQ Latch captures each interrupt request. The system processor reads the latch via the Request Sense register, and clears the latch by writing to the Request Clear register.

The IRQ Mask logic allows selective masking of the interrupts. The processor controls masking by writing to the Mask register.

The Priority Encoder prioritizes the interrupt requests and encodes the highest-priority pending interrupt that is not masked. IRQ15 has the highest priority, and IRQ12 the lowest.

The IRL Latch captures the coded interrupt level number that is generated by the Priority Encoder.

The IRL Mask logic allows masking of all interrupt requests by forcing the interrupt level asserted on IRL<3:0> to 0. The processor can still poll for pending interrupts by reading the Request Sense register even if the interrupt level is masked. The processor controls interrupt level masking by writing to the Mask register.

4.1 IRC Registers

The IRC features six internal registers that allow the processor to control IRC operation and to monitor system interrupt requests that may be pending. Register addressing is shown in Table F4-1.

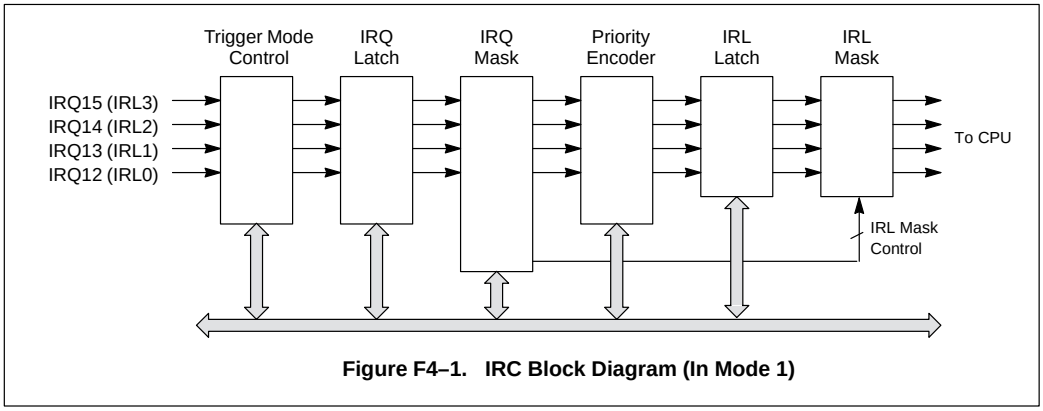


Figure F4–1. IRC Block Diagram (In Mode 1)

Table F4–1. IRC Register Map

Address	Register	Access
0x00000200	Trigger Mode 0	R/W
0x00000204	Trigger Mode 1	R/W
0x00000208	Request Sense	R
0x0000020C	Request Clear	W
0x00000210	Mask	R/W
0x00000214	IRL Latch/Clear	R/W
0x00000218	IRC Mode	R/W

4.1.1 Trigger Mode Registers

The Trigger Mode registers control the trigger mode for each interrupt channel. Trigger Mode Register 0 controls trigger modes for interrupt channels 8-15; Trigger Mode Register 1 controls trigger modes for interrupt channels 1-7.

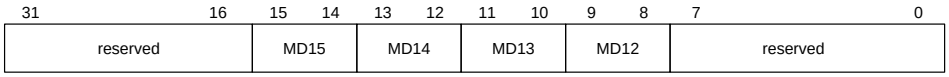


Figure F4–2. Trigger Mode Register

Bits 15-8: Trigger Mode Selects - Select trigger modes for channels 8-15.

Bits 7-0: Reserved.

Two-bit fields in the registers select one of four trigger modes for each channel as follows:

Table F4–2. Four Trigger Modes

MDx Value*	Trigger Mode
0	High Level
1	Low Level
2	Rising Edge
3	Falling Edge

Reset clears the Trigger Mode registers, resulting in high level triggering for each interrupt channel.

Note: An interrupt channel should be masked before its trigger mode is changed, or a false interrupt may occur.

4.1.2 Request Sense Register

The processor reads the state of the IRQ Latch through the Request Sense register to identify pending interrupts.

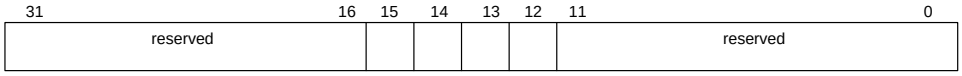


Figure F4–3. Request Sense Register

Bits 15-12: Sense IRQ Latch - Correspond to interrupt channels 15-1 and indicate, when high, that the corresponding interrupts are latched and pending.

Bit 11–0: Reserved.

Reset clears the Request Sense Register.

4.1.3 Request Clear Register

The processor writes to the Request Clear register to clear the IRQ Latch. The processor typically uses this register to clear the latch associated with an interrupt when it services the interrupt.

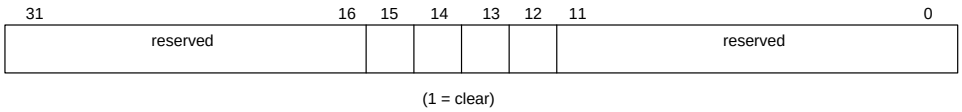


Figure F4–4. Request Clear Register

Bits 15-12: Clear IRQ Latch - Correspond to interrupt channels 15-1, and writing the bits to 1 clears the corresponding interrupt latches.

Bit 11–0 Reserved.

Reset clears the Request Clear Register.

Note: The processor should clear the latch associated with an interrupt following a change in its trigger mode, or a false interrupt may occur.

4.1.4 Mask Register

The Mask register is used to mask the outputs of the IRQ Latch from the Priority Encoder, and the output of the IRL latch from the IRL<3:0> bus. The processor uses the Mask register to mask unused interrupt channels, to temporarily mask individual interrupt requests, and to mask all interrupt requests.

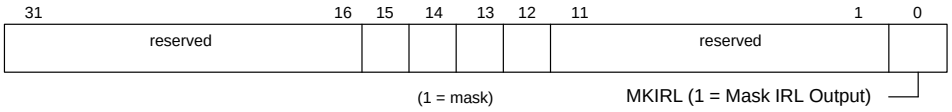


Figure F4–5. Mask Register

Bits 15-12: Interrupt Request Mask - Correspond to interrupt channels 15-12, and writing them to 1 masks the corresponding interrupt requests.

Bits 11-1: Reserved.

Bit 0: Mask IRL - Masks the output of the IRL Latch. When MKIRL is set to 1, the IRL Latch output is masked, and the IRL<3:0> bus is forced to 0. When MKIRL is 0, the encoded interrupt level number in the IRL latch is asserted on the IRL<3:0> bus to interrupt the processor. MKIRL is typically set to 1 (mask enabled) in systems that poll interrupt requests.

Reset clears the Mask register.

4.1.5 IRL Latch/Clear Register

The processor uses the IRL Latch/Clear register to clear and read the IRL Latch.

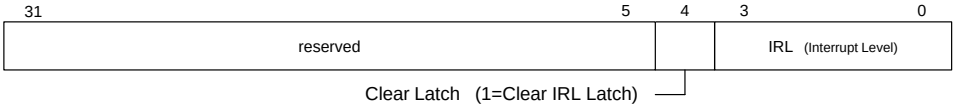


Figure F4–6. IRL Latch/Clear Register

Bit 4: Clear IRL Latch - Clears the IRL Latch when written to 1.

Bits 3-0: Interrupt Level - Holds the value of the IRL Latch. The processor typically reads IRL to identify the highest-priority interrupt level in systems that poll the interrupts.

Reset clears the IRL Latch/Clear Register. The “Clear Latch” bit is only writable while the interrupt level bits are only readable. Writes do not affect them.

NOTE:

The IRQ latch captures each of the four interrupt requests. The system processor reads the latch via the Request Sense register and clears the latch by writing to the Request Clear register. The example assembly language program below shows the code sequence for writing to the Request Clear register of channel 12.

```
------------------------------------------------------
     . . . .
     . . . .
! define Request Clear Register #define rqc 0x20c
! define a valid memory location #define rqs_loc 0x1000
! define control register ASI address space #define casi 0x1

    . . . .
    . . . .   ! Request Clear;
    set         rqc, %10
    set         0x1000,  %17
    set         rqs_loc,  %16     !memory location defined in main prog
    st          %g0,  [%16]
    sta         %17,  [%10]  casi   !write to Request Clear register
     . . . .
     . . . .
```

A write to Request Clear register must be preceeded by the store of 0x0 to any valid memory location to prevent the previous high value bits on the data bus from unintentionally setting other bits of the Clear Request register.

4.1.6 IRC Mode Register

The interrupt mode of IRC isselected by programming the IRC Mode Register. Reset sets the IRC to Mode 0.

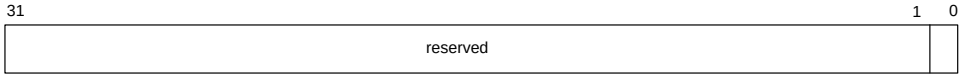


Figure F4–7. IRC Mode Register

Bits 31-1: Reserved.

Bits 0: '0' means Mode 0; '1' means Mode 1

4.2 IRC Operation

The IRC latches interrupt requests into the IRQ Latch according to the trigger mode option selected for each interrupt channel. The Priority Encoder prioritizes the

unmasked interrupts and generates an encoded interrupt level number for the highest-priority interrupt. The IRL Latch latches the encoded interrupt level number, which is then transferred through the IRL Mask logic to the IRL<3:0> bus to interrupt the processor. The processor responds by servicing the interrupt identified on IRL<3:0>, clearing the latched interrupt from the IRQ Latch through the IRL Latch/Clear register and clearing the IRL latch. The IRC then generates a new level number for the highest-priority interrupt that may be latched in the IRQ Latch.

The interrupt request latency is ten system clock cycles. That is, the corresponding interrupt level is asserted on IRL<3:0> ten clock cycles after an interrupt request is recognized by the IRC.

4.2.1 Polling

The processor can poll interrupts by reading either the IRQ Latch via the Request Sense register, or the IRL Latch via the IRL Latch/Clear register.

The processor may mask interrupts that it polls via the Request Sense register by masking either the IRQ Latch or the IRL Latch. The processor then periodically reads the IRQ Latch and clears interrupts from the latch when they are serviced. The IRL Latch may remain unmasked to allow interrupt-driven servicing of some interrupts if the polled interrupts are masked with the IRQ Latch mask.

The processor may mask all interrupts when it polls interrupts via the IRL Latch/Clear register by masking the IRL Latch. The processor then periodically reads the IRL Latch for the highest-level pending interrupt and clears both the IRL Latch and the interrupt from the IRQ Latch once the interrupt is serviced.

4.2.2 Initialization

All IRC registers are cleared to 0 by Reset. This results in high-level trigger mode for all interrupts, and all masks disabled.

After reset, the interrupt trigger modes should be changed after the interrupts are masked with the IRQ mask to eliminate false interrupts. The masks can then be disabled.

4.2.3 Noise Immunity

The IRQ pins are sampled at the rising edge of the IRC internal clock. The pin value must be verified by three successive samples for recognition by the IRC. For example, a level trigger must be asserted for at least 3 internal clock periods (6 system clock periods) for recognition.

Figure F4–8 shows the IRQ pin sample timing.

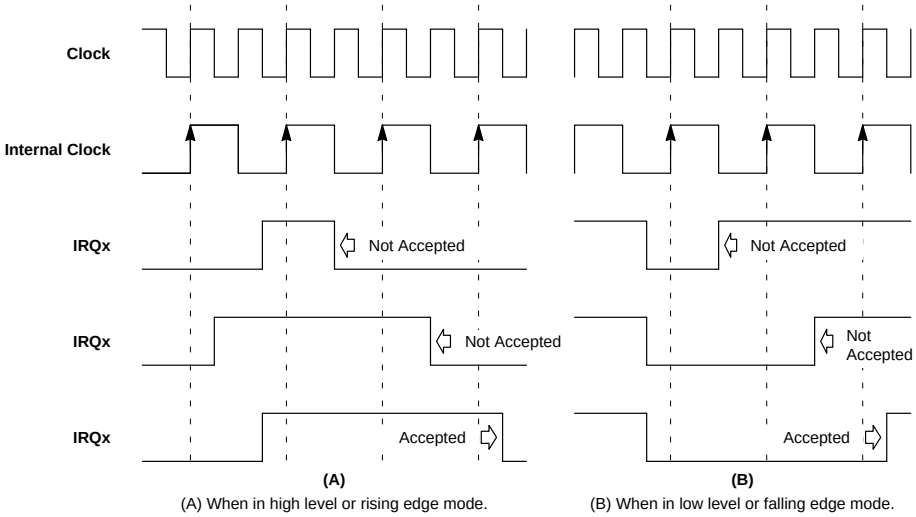


Figure F4–8. -IRQ Pin Sample Timing

CHAPTER

F5

External Interface

The processor external interface consists of signals for bus operations and for system control. This chapter details the MB86933H–20 signal set, describes basic bus timing, and describes the programmable wait-state generator, on-chip timer, and same-page detection logic. See the MB86933H–20 Data Sheet for specific electrical and timing information.

The System Design Considerations chapter of this document discusses issues that are likely to arise in the design of SPARClite systems.

F5.1 Signals

The processor's external signals are illustrated in Figure F1-6 of the *Overview* chapter, and are listed in Table F5-1. A dash at the beginning of a signal name, as in –RESET, indicates that the signal is active-low.

Table F5–1. Input and Output Signals

Symbol	Type	Symbol	Type	Symbol	Type	Symbol	Type
ADR <27:2>	O S(L) G(Z) I (1)	–BMODE16	I	–LOCK	O S(L) G(Z) I (1)	TDI	I
–AS	O S(L) G(Z) I (1)	CLKOUT1 CLKOUT2	O G(Q) I (Q)	–MEXC	I S(L)	TDO	O
ASI <3:0>	O S(L) G(Z) I (1)	CLK_ECB	I	–SAME_PAGE	O S(L) G(1) I (1)	–TIMER_OVF	O S(L) G(Q) I (Q)
–BE 3-0	O S(L) G(Z) I (0)	–CS0, –CS1 –CS2, –CS3 –CS4, –CS5	O S(L) G(1) I (1)	RD/–WR	O S(L) G(Z) I (1)	TMS	I
–BGRNT	O S(L) G(0) I (Q)	D <31:0>	I/O S(L) G(Z) I (Z)	–READY	I S(L)	–TRST	I
–BREQ	I S(L)	–ERROR	O S(L) G(Q) I (Q)	–RESET	I A(L)	XTAL1 (CLKIN) XTAL2	I O G(Q) I (Q)
–BMODE8	I	IRL <3:0>	I A(L)	TCK	I	RAS <1:0>	O
CAS <3:0>	O	–DWE	O				

NOTE:

I = Input Only Pin
O = Output Only Pin
I/O = Either Input or Output Pin
\- = Pins "must be" connected as described
S(L)= Synchronous: Inputs must meet setup and hold times relative to CLKIN. Outputs are Synchronous to CLKIN
A(L) = Asynchronous: Inputs may be asynchronous to CLKOUT.
G(…) = While the bus is granted to another bus master (–BGRNT=asserted), the pin is
G(1) is driven to V_{CC}
G(0) is driven to V_{SS}
G(Z) floats
G(Q) is a valid output
I(…)= While the bus is between bus cycles (or being reset) and is not granted to another bus master, the pin is
I (1) is driven to V_{CC}
I (0) is driven to V_{SS}
I (Z) floats
I (Q) is a valid output

The following sections describe the signal set in detail, arranged by functional group as follows:

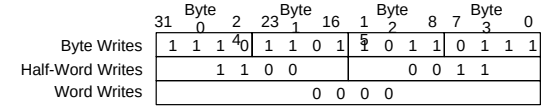
- Processor Control and Status—Reset, error, and clock signals.
- Memory Interface—Data and address buses, ASI and byte-enables, chip selects, and other control signals used to access external memory and memory-mapped devices.
- Bus Arbitration—Signals used by external devices in requesting, and by the processor in granting, control of the bus.
- Peripheral Functions—Interrupt-requests and timer overflow.
- Boot ROM Size—Input signals used to identify the boot ROM size.
- Boundary-Scan—JTAG-compatible test signals used for board verification.

F5.1.1 Processor Control and Status

Signal	Function
CLKOUT1 CLKOUT2	**CLOCK OUTPUTS (O):** MB86933H–20 bus transactions can be referenced against these outputs. CLKOUT1 has the same frequency and phase as the internal oscillator, or the signal applied to CLKIN. CLKOUT2 is the same as CLKOUT1, but phase-shifted 180 degrees.
–ERROR	**ERROR SIGNAL (O):** Asserted by the CPU to indicate that it has halted in an error state as a result of encountering a synchronous trap while traps are disabled. In this situation, the CPU saves the Trap Type (tt) value in the Trap Base Register, enters into an error state and asserts the –ERROR signal. The system can monitor the –ERROR pin and initiate a reset to recover from the error condition.
–RESET	**SYSTEM RESET (I):** Resets the processor to a known internal state. –RESET should be asserted for at least 4 processor cycles after the clock has stabilized. The internal state of the processor immediately after reset is described in the *Programmer's Model* chapter.
XTAL1 (CLKIN) XTAL2	**EXTERNAL OSCILLATOR (XTAL1, XTAL2):** Determines the execution rate and timing of the processor. Connecting a crystal across these pins forms a complete crystal oscillator circuit. The processor operating frequency is the same as the crystal oscillator frequency. The processor can also be driven by an external clock. In this case, the clock signal is applied to XTAL1 (CLKIN); XTAL2 should be left unconnected. The processor operating frequency is the same as the external clock frequency.

F5.1.2 Memory Interface

<table>
<tr><th>Signal</th><th>Function</th></tr>
<tr><td>ADR[27:2]</td><td>ADDRESS BUS (O): Specifies the data or instruction address of a 32-bit word. Reads are always one word in size while byte, half-word, or word transaction sizes for writes are identified by separate byte-enable signals (–BE3-0). The value on the address bus is valid for the duration of the bus transaction.</td></tr>
<tr><td>–AS</td><td>ADDRESS STROBE (O): Asserted by the MB86933H–20 or other bus master to indicate the start of a new bus transaction. A bus transaction begins with the assertion of –AS and ends with the assertion of –READY. During cycles in which neither the processor nor another bus master is driving the bus, the bus is idle, and –AS remains de-asserted. See Table F5-1 for signal values while the bus is idle. The MB86933H–20 asserts –AS for 1 clock cycle.</td></tr>
<tr><td>ASI[3:0]</td><td>ADDRESS SPACE IDENTIFIERS (O): Indicates which of the 16 available address spaces the current bus transaction is accessing. The ASI values are defined as follows:
<table>
<tr><th>ASI <3:0></th><th>ADDRESS SPACE</th></tr>
<tr><td>0x0</td><td>Application Definable</td></tr>
<tr><td>0x1</td><td>Control Registers</td></tr>
<tr><td>0x2 - 0x3</td><td>Reserved</td></tr>
<tr><td>0x4 - 0x7</td><td>Application Definable</td></tr>
<tr><td>0x8</td><td>User Instruction Space</td></tr>
<tr><td>0x9</td><td>Supervisor Instruction Space</td></tr>
<tr><td>0xA</td><td>User Data Space</td></tr>
<tr><td>0xB</td><td>Supervisor Data Space</td></tr>
<tr><td>0xC - 0xF</td><td>Reserved</td></tr>
</table>
The ASI values specified as "application definable" can be used by privileged (supervisor mode) instructions such as load and store alternate. The ASI value is available in the same cycle in which the corresponding address value is asserted on the address bus. The values on the ASI pins are valid for the duration of the bus transaction.</td></tr>
</table>

Signal	Function
–BE3-0	**BYTE ENABLES (O):** These pins indicate whether the current store transaction is a byte, half-word or word transaction. –BE3-0 signals are available in the same cycle in which the corresponding address value is asserted on the address bus and is valid for the duration of the bus transaction. This bus should be used only to qualify store transactions. For load transactions all sub-word requests are read (and replaced in the cache) as words and then the appropriate byte or half-word is extracted by the integer unit. Possible values for –BE3-0 are as follows:  BE<2:3> are also used in 8 and 16-bit ROM accesses as follows: Bus Mode / Byte / BE<2:3> 8-bit: 0 / 00; 1 / 01; 2 / 10; 3 / 11 16-bit: 0 & 1 / 00; 2 & 3 / 10
–BMODE8	**8-BIT BOOT MODE:** This signal is sampled during reset and causes read accesses memory mapped to –CS0 to assume 8-bit ROM memory. The MB86933H–20 generates four sequential fetches to assemble a complete instruction or data word before continuing. Bytes are fetched in sequence (0,1,2,3) as encoded by –BE[2] and –BE[3] (00, 01, 02, 03). Writes to –CS0 are unaffected by boot mode selection. If left unconnected, a weak pull-up on this pin (and –BMODE16 pin) causes the processor to default to 32-bit mode. *Note*: At reset, –BMODE8 must not be asserted while –BMODE16 is asserted, or undefined operation may result.
–BMODE16	**16-BIT BOOT MODE:** This signal is sampled during reset and causes read accesses memory mapped to –CS0 to assume 16-bit ROM memory. The MB86933H–20 generates two sequential fetches to assemble a complete instruction or data word before continuing. Half words are fetched in sequence (0,1) as encoded by –BE[2]. Writes to –CS0 are unaffected by boot mode selection. If left unconnected, a weak pull-up on this pin (and –BMODE8 pin) causes the processor to default to 32-bit mode. *Note*: At reset, –BMODE16 must not be asserted while –BMODE8 is asserted, or undefined operation may result.
–CS[5-0]	**CHIP SELECTS (O):** One of these signals is asserted when the value on the address bus lies in the range specified by the corresponding Address Range Specifier Register. The –CS signals are used to decode the current address into one of eight address ranges. Address ranges should not overlap. Each address range has a corresponding wait-state specifier which is used to generate an internal –READY signal after a user-defined number of processor clock cycles. This allows a variety of memory and I/O devices with different access times to be connected to the MB86933H–20 without the need for additional logic. CS0 is enabled at reset (see Chapter 2).

Signal	Function
D[31:0]	**DATA BUS (I/O):** D31 corresponds to the most significant bit of Byte 0. D0 corresponds to the least significant bit of byte 3. A double word is aligned on an 8-byte boundary, a word is aligned on a 4-byte boundary, and a half-word is aligned on a 2-byte boundary. If a load or store of any of these quantities is not properly aligned, a mem_address_not_aligned Trap will occur in the processor. During write cycles, the point at which data is driven onto the bus depends on the type of the preceding cycle. If the preceding cycle was a write, data is driven in the cycle immediately following the cycle in which –READY was asserted. If the preceding cycle was a read, data is driven one cycle after the cycle in which –READY was asserted, in order to minimize bus contention between the processor and the system.
–LOCK	**BUS LOCK (O):** Asserted by the processor to indicate that the current bus transaction requires more than one transfer on the bus. The Atomic Load Store instruction, for example, requires contiguous bus transactions and so causes the BUS LOCK signal to be asserted. The bus will not be granted to another bus master as long as –LOCK is active. –LOCK is asserted with the assertion of –AS and remains active until –READY is asserted at the end of the locked transaction
–MEXC	**MEMORY EXCEPTION (I):** Asserted by the memory system to indicate a memory error on either a data or instruction access. Assertion of this signal initiates either a Data or Instruction Access Exception trap in the IU. The current bus access is invalidated by asserting the –MEXC in the same cycle as the –READY signal. The IU ignores the value on the data bus in cycles where –MEXC is asserted.
RD/–WR	**READ/WRITE BUS TRANSACTION (O):** Specifies whether the current bus transaction is a read or a write operation. When –AS is asserted and RD/–WR is high, then the current transaction is a read. With –AS asserted and RD/–WR low, the current transaction is a write. RD/–WR remains active for the duration of the bus transaction and is de-asserted with the assertion of –READY.
–READY	**READY (I):** Asserted by the external memory system to indicate that the current bus transaction is being completed and that it is ready to start with the next bus transaction in the following cycle. In case of a fetch from memory, the processor will strobe the value on the data bus at the rising edge of CLKIN following the assertion of –READY. In the case of a write, the memory system will assert –READY when the appropriate access time has been met. In most cases, no external logic is required to generate the –READY signal. On-chip circuitry can be programmed to assert –READY internally, based on the address of the current transaction. The external system can override the internal ready generator to terminate the current bus cycle early. Up to 6 address ranges each with different transaction times can be programmed. (See the *System Support Functions* section, below.)
–SAME_PAGE	**SAME-PAGE DETECT (O):** Asserted when the address of the current memory access is within the same page as the previous memory access. –SAME_PAGE can be used to take advantage of fast consecutive accesses within page-mode DRAM page boundaries. –SAME_PAGE is asserted with –AS and remains active for one processor cycle. –SAME_PAGE is never asserted in the first transaction following a transaction by another device on the bus. The page size is specified by writing the Same-Page Mask Register. (See the *System Support Functions* section, below.)

F5.1.3 Bus Arbitration

Signal	Function
–RAS[1:0]	**ROW ADDRESS STROBES (O):** Asserted after a valid row address appears on ADR[27:16]. –RAS0 is the address strobe for bank 0. –RAS1 is the address strobe for bank1.
–CAS[3:0]	**COLUMN ADDRESS STROBES (O):** Asserted after a valid column address appears on ADR[27:16]. –CAS[3:0] controls bytes 3:0 respectively. During word accesses all –CAS[3:0] are active simultaneously. During halfword accesses either –CAS[3:2] or –CAS[1:0] are simultaneously active. During byte accesses only one –CAS signal is active.
–DWE	**DRAM WRITE ENABLE (O):** This signal is asserted low during DRAM write accesses. It is deasserted otherwise.

F5.1.4 Bus Arbitration

Signal	Function
–BGRNT	**BUS GRANT (O):** Asserted by the CPU in response to a request from a device wanting ownership of the bus. The CPU grants the bus to other devices only after all transfers for the current transaction are completed. All bus drivers are three-stated with the assertion of the BUS GRANT signal.
–BREQ	**BUS REQUEST (I):** Asserted by another device on the bus to indicate that it wants ownership of the bus. The request must be answered with a bus grant (–BGRNT) from the MB86933H–20 before the device can proceed by driving the bus. Once the bus has been granted, the device has ownership of the bus until it de-asserts –BREQ. The user should ensure that devices on the bus do not monopolize the bus to the exclusion of the CPU. The assertion of –BREQ is recognized by the processor even when –RESET is being asserted.

F5.1.5 Peripheral Functions

Signal	Function
IRL[3:0] /IRQ [15:12]	**INTERRUPT REQUEST BUS:** Based on the mode selected in the on–chip interrupt controller, these pins are defined in two ways. In one mode (IRL) the value on these pins defines an external interrupt vector . IRL < 3:0 >=1111 forces a non–maskable interrupt. IRL value of 0000 indicates no pending interrupts. All other values indicate maskable interrupts as enabled in the PIL field of the processor status register (PSR). In this mode, interrupts should be latched and prioritized by external logic and should be held pending until acknowledged by the processor. In the other mode (IRQ), each pin represents a decoded interrupt source. When active, the values on pins IRQ<15:12> will cause the processor to vector to interrupts 15 through 12, respectively. The trigger for each IRQ pin can be set for high–level, low–level, rising edge, or falling edge.
–TIMER_OVF	**TIMER OVERFLOW (O):** Indicates that the processor's internal 16-bit timer has overflowed. This signal can be used to initiate a DRAM refresh cycle or a one-cycle periodic waveform. On reset, the timer is turned off and –TIMER_OVF is high.

F5.1.6 Test and Boundary-Scan

Signal	Function
CLK_ECB	**EXTERNAL CLOCK BYPASS (I):** When tied high, causes the CLKIN signal to bypass the on-chip phase-locked loop. This signal is intended primarily for testing the chip.
TCK	**TEST CLOCK (I):** JTAG compatible test clock input.
TDI	**TEST DATA IN (I):** JTAG compatible test data input.
TDO†	**TEST DATA OUT (O):** JTAG compatible test data output.
TMS†	**TEST MODE (I):** JTAG compatible test mode select pin.
–TRST†	**TEST RESET (I):** Asynchronous reset for JTAG logic. If not using JTAG, this signal must be pulled low.

†. See appendix for more information

F5.2 Bus Operation

The Bus Interface Unit handles requests for external memory and I/O operations, arbitrates for bus access, or is idle. Bus transactions are handled as follows:

- Memory and I/O Operations—Read and write transactions are initiated with the processor asserting the –AS signal. The RD/–WR output indicates the transaction type. The –BE[3:0] outputs indicate the transaction width. The processor drives the address and ASI signals and either drives (during stores) or reads (during loads) the signals on the data bus. The transaction ends when –READY is asserted.

  An atomic load-store is a load followed immediately by a store, with no operation between. The –LOCK output is asserted during atomic operations to indicate that the bus is being used for more than one consecutive memory operation.

- Arbitration—Any external device can request ownership of the bus by asserting the –BREQ signal. The processor three-states its bus drivers and asserts –BGRNT to indicate that it is relinquishing control of the bus. Upon completion of its transaction, the external device de-asserts –BREQ, and the processor responds by de-asserting –BGRNT the following cycle.

In any cycle the BIU can receive a request for accesses to instruction memory, to data memory, or to both. If it receives a request for both in the same cycle, it completes the data memory transaction first.

F5.2.1 Exception Handling

The external memory system can indicate an exception during a memory operation. The BIU signals the appropriate data or instruction exception to the IU, which will trap accordingly.

Any system that must recover from this error should store the address and data of the write operation in hardware. If the system can generate both read and write exceptions, the system must also provide a status bit that indicates whether the exception was generated during a read or during a write operation. With access to this information, the data access exception service routine can determine the cause of the exception and recover accordingly.

F5.2.2 Bus Cycles

This section describes the relative timing of events in representative bus transactions.

Load

A read transaction begins with the BIU asserting –AS to indicate a new bus transaction. The –AS signal is de-asserted after one cycle. At the same time, ADR<27:2> and ASI<3:0> bits are asserted with the location to be read. The BIU drives the RD/–WR signal high to indicate a read transaction.

Note that the –BE lines indicate byte, halfword or word operations during load operations, although their use is optional. The processor loads a word regardless of the size of the data requested (byte, halfword, word).

The external memory system responds with the read data on pins D<31:0>. It also asserts the –READY signal when the data is ready (unless internal ready generation is selected). For slow memory, the –READY signal is delayed until data is valid.

A load double operation is treated as back-to-back reads.

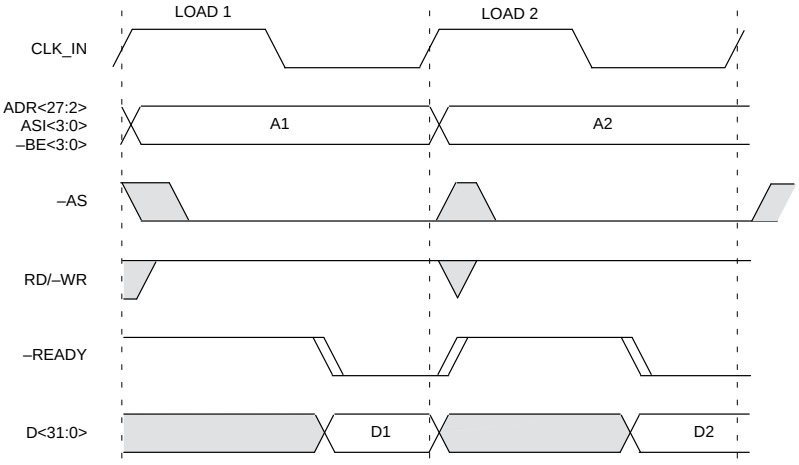


Figure F5–1. Load Timing

Load with Exception

If the external memory system sees a memory exception, it can terminate the current memory transaction by asserting the –MEXC and –READY signals. The data on the data bus is ignored by the MB86933H–20.

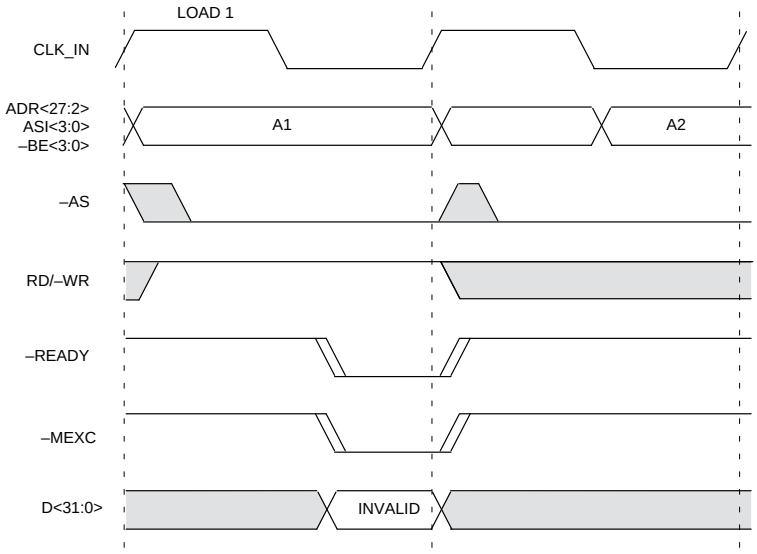


Figure F5–2. Load with Exception Timing

Store

A write transaction begins with the BIU asserting –AS, to indicate a new bus transaction. The –AS signal is de-asserted after one cycle. At the same time the ADR<27:2> and ASI<3:0> pins are driven with the location to be written, and the write data is asserted on D<31:0>. The –BE3-0 pins indicate byte, half-word or word transaction width. The BIU drives the RD/–WR signal low to indicate a write transaction.

The external memory system responds by asserting the –READY signal when it has stored the data. There is always one idle bus cycle between the termination of a read cycle and the beginning of a write cycle to provide time for switching of the data bus drivers.

A store double operation is treated as back-to-back writes.

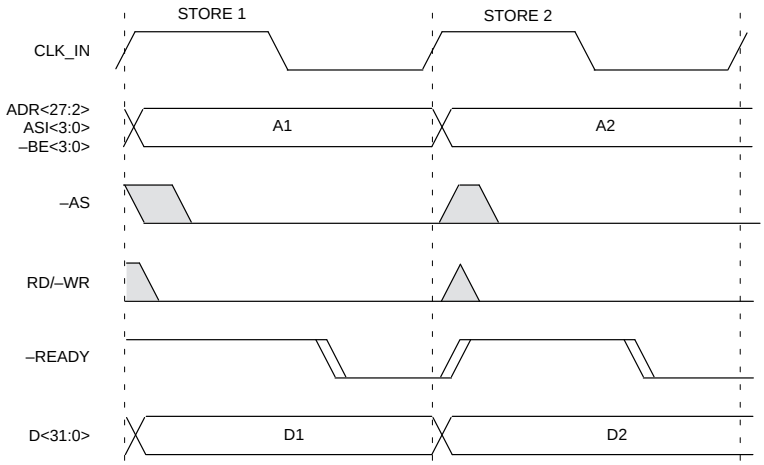


Figure F5–3. Store Timing

Store with Exception

If an access exception occurs during a write, the external memory system can terminate the current memory transaction by asserting the –MEXC and –READY signals. The external memory system is expected to ignore the data on the data bus in this situation.

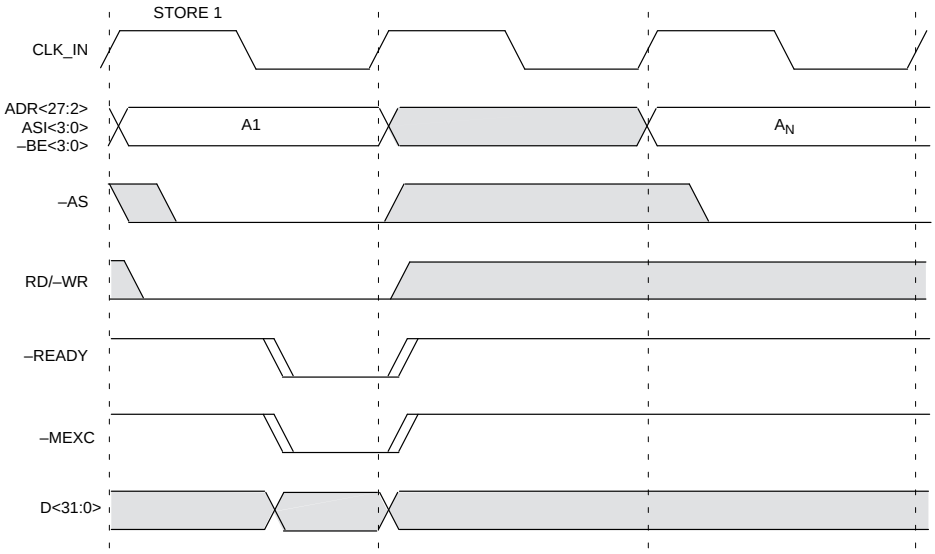


Figure F5–4. Store with Exception Timing

Atomic Load Store

An atomic load store executes as a load followed by a store, with no operation between. The –LOCK signal is asserted to indicate that the bus is being used for more than one external memory operation.

There is one cycle between the termination of the read and the beginning of the write to provide time for the switching of the data bus drivers.

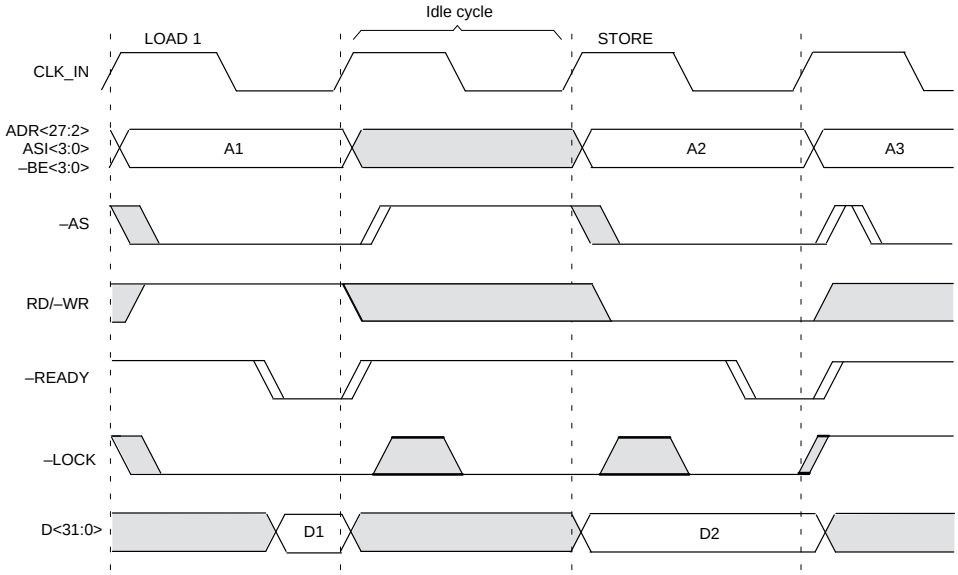


Figure F5–5. Atomic Load Store Timing

External Bus Request and Grant

Any external device can request ownership of the bus by asserting the –BREQ signal. The BIU asserts the –BGRNT signal to indicate that it is relinquishing control of the bus, and three-states all of its bus drivers. The external device can complete its transaction during the following cycle. Upon completion of its transaction, the external device de-asserts the –BREQ signal. The BIU responds by de-asserting the –BGRNT signal during the following cycle.

The MB86933H–20 is the default owner of the bus.

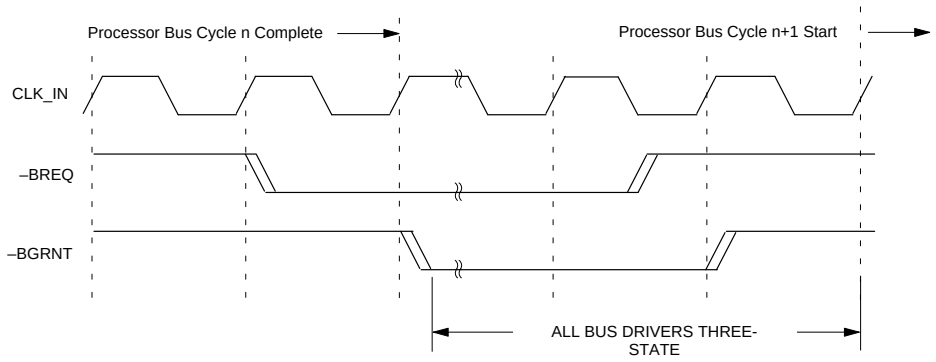


Figure F5–6. External Bus Request and Grant Timing

Processor Reset

The MB86933H–20 is reset by asserting the –RESET signal for a minimum of 4 clock cycles (see Figure 4-7). Systems using an external crystal to clock the processor should assert –RESET for at least 4 cycles after the crystal has stabilized.

If the processor is reset following a halt in Error Mode and if power to the processor is not removed, after reset the *tt* field will contain the value of the Trap that caused the processor to halt.

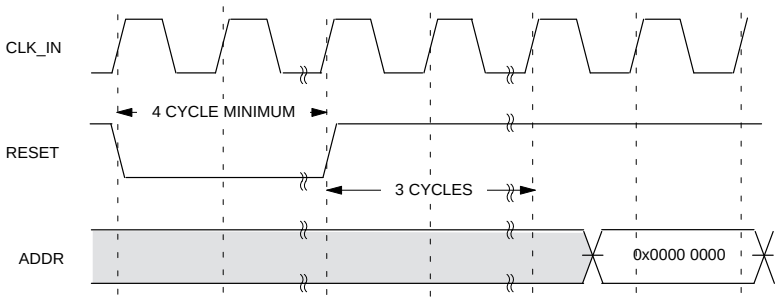


Figure F5–7. Reset Timing

F5.3 System Support Functions

Built-in system support functions help to minimize the amount of glue logic required in the external system. The support includes programmable chip select logic, programmable wait-state generation, same-page detection logic and a timer for generating refresh requests. For a more detailed description of the programming of these registers refer to Chapter 2.

The System Support Control Register turns the various system support features on and off.

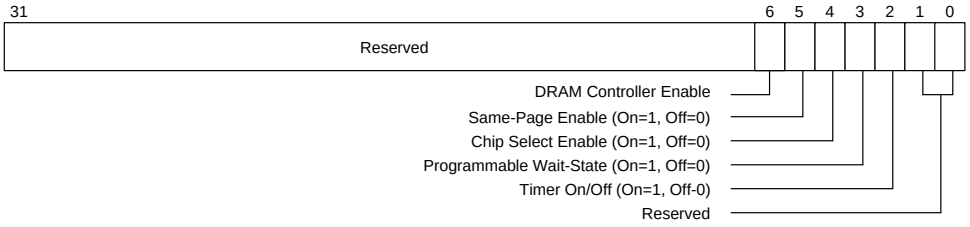


Figure F5–8. System Support Control Register

F5.3.1 System-Configuration Registers

The system-configuration registers (Address Range Specifiers, Address Masks, and Programmable Wait-State Specifiers) allow software to define six different address ranges. When an address driven by the processor is in one of these ranges, the corresponding Chip-Select (–CS) pin is asserted. After a number of clock cycles determined by the corresponding Programmable Wait-State Specifier, the processor

automatically generates an internal –READY signal. This makes it possible for memory and I/O devices with different access times to be connected to the processor without additional logic.

The contents of the Address Range Specifier Registers 1-5 (ARSR[5:0]) define five of the six address ranges. An additional address range is available, corresponding to –CS0. For this address range, ADR is hardwired to 0, and ASI is hardwired to 0x9 (Supervisor Instruction Space). With Mask Register AMR0, –CS0 ranges 8K words. –CS0 is enabled at reset. –CS1, –CS2, –CS3, –CS4 and –CS5 are disabled at reset.

Note that the MB86933H–20 has six register windows rather than eight. It has twenty-six Address Bus signals (ADR<27:2>) rather than thirty, four Address Space Identifier signals (ASI<3:0>) rather than eight, no emulator-support signals, and no memory management unit. These and other differences between the MB86933H–20 and other SPARClite processors should be considered when porting code to the MB86933H–20 from another SPARClite processor, and when porting code from the MB86933H–20 to another SPARClite processor. Documentation for other SPARClite should be referenced to identify differences with the MB86933H–20 that may affect ported code.

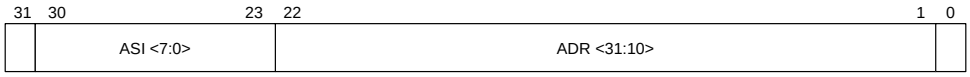


Figure F5–9. Address Range Specifier Register Format

An Address Mask Register is associated with each address range. Any address driven by the chip is compared with the value in all address range specifiers. Only those bits of the register are compared for which the corresponding mask bits are 0. If the specified bits of the current address match one of the address range specifiers, the corresponding chip-select (–CS) pins are asserted. When no bus transaction is being performed, all the –CS pins are high (inactive). The Address Mask Register corresponding to –CS0 is initialized to compare all bits except ADR<14:10>.

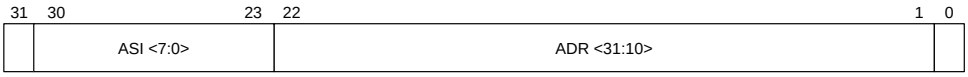
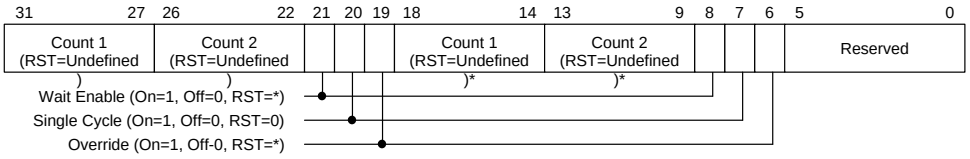


Figure F5–10. Address Mask Register Format

A Programmable Wait-State Specifier is associated with each address range. Three registers are used to specify the wait states for the six address ranges. Each register contains the wait-state specifiers for two address ranges.

When the address currently being driven by the processor matches the unmasked bits in one of the Address Range Specifiers, the corresponding wait-state specifier is selected. The format of Wait-State Specifier Registers is shown in Figure F5-11.



* See Table 2-3 in *MB86930 Chapter 2 "Programmer's Model"*

Figure F5–11. Wait-State Specifier Registers

Bits 31-19: Wait-State Specifier—When an external access falls within an address range defined by an ARSR and AMR, the corresponding wait-state specifier determines when, and whether, the processor generates an internal –READY signal to terminate the access.

Count1 (Bits 31-27): The number of wait-states inserted before the internal –READY, under the following conditions: the Single Cycle bit equals 0 and the current access is not on the same page as the previous access. The number of wait-states is the value of this field +1 (i.e., 0=1 wait-state, 1=2 wait-states, etc.) The value of Count1 is undefined on reset.

Count2 (Bits 26-22): The number of wait-states inserted before the internal –READY, under the following conditions: the Single Cycle bit equals 0 and the current access is on the same page as the previous access. The number of wait-states is the value of this field +1 (i.e., 0=1 wait-state, 1=2 wait-states, etc.) The value of Count2 is undefined on reset.

Wait Enable (Bit 21): Enables and disables the wait-state generator for an individual address range. If the Wait Enable bit of a wait-state specifier equals 0, the internal –READY is not asserted when addresses in the corresponding range are accessed by the processor. If Wait Enable is 1, the single cycle bit must be 0. See Table 2-3 in *MB86930 Chapter 2 "Programmer's Model"* for reset value.

Single Cycle (Bit 20): Specifies the timing of the internal –READY signal. If the Single Cycle bit equals 1 when an address in the appropriate range is accessed, the internal –READY is asserted in the same cycle. If the Single Cycle bit equals 0, and the current transaction is in the same page as the previous transaction, then Count2 is used as the number of cycles after which –READY is asserted internally. If the transaction is not in the same page, Count1 is used instead. If Single Cycle is enabled, the Wait Enable bit must be 0. See Table 2-3 in *MB86930 Chapter 2 "Programmer's Model"* for reset value.

Override (Bit 19): Allows the system to terminate a memory transaction before the internally specified time. If the Override bit equals 1, and external hardware asserts the external –READY signal, then the wait-state generator will stop counting and will wait for the next transaction. This bit is cleared to 0 on reset.

Bits 18-6: Wait-State Specifier—The wait-state specifier for a second address range. This field is organized just like bits 31-19.

Bits 5-0: Reserved

The Count1 and Count2 fields of the Wait-State Specifier corresponding to –CS0 have all their bits set to 1 following reset. In this way, 32 wait-state cycles (the maximum number) are inserted into the processor's first instruction accesses. The override bit for –CS0 is enabled as well.

F5.3.2 Same-Page Detection

The MB86933H–20 supports same memory page operation **only in the chip select 4 (–CS[4]) address range**.

The same-page detection logic determines whether the address of the current memory transaction is on the same page as the previous transaction. If it is, the processor asserts the –SAME_PAGE signal. The system can then take advantage of the fast consecutive accesses possible within fast-page mode DRAM page boundaries. The same-page detection logic consists of a mask register, a register to store the address and ASI bits of the previous transaction, and a comparator.

The Same-Page Mask Register specifies which bits of the current address and ASI must be compared with the previous address and ASI. Only those bits are compared for which the mask bit is 1.

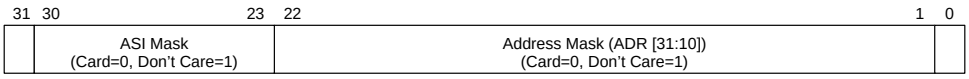


Figure F5–12. Same-Page Mask Register

The –SAME_PAGE signal is never asserted for the first transaction following a transaction by another device on the bus. When using the internal wait-state generator, DRAM control logic should issue a bus request when initiating a refresh cycle so that the –SAME_PAGE logic is reset appropriately. The –SAME_PAGE feature is disabled at reset.

F5.3.3 Programmable Timer

The 16-bit programmable timer causes the –TIMER_OVF output signal to be asserted at software-defined intervals. This signal can be used to initiate DRAM refresh cycles, or to control other periodic events in the external system.

The current timer count is stored in the Timer Register. When the timer overflows, it is loaded with the value in the Timer Preload Register. The contents of both of these registers are undefined following reset.

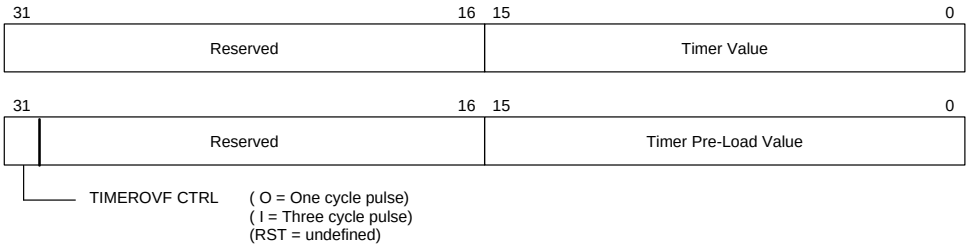


Figure F5–13. Timer and Timer Preload Registers

The timer can also be loaded by writing directly to the Timer Register. The timer can be turned off by writing a 0 to the Timer On/Off bit in the System Support Control register. The timer is clocked at the processor clock frequency.

F5.4 8/16–bit BUS MODE

F5.4.1 Purpose

The data bus of the MB86933H–20 can be configured to 8- and 16-bit bus modes as well as the standard 32-bit mode. This flexibility accommodates those cases in which boot code resides in memories organized as blocks of bytes or halfwords.

F5.4.2 Features

Bus Configuration: the data bus configurations are fixed to specific segments of the bus:

- 8-bit mode: D[7:0]
- 16-bit mode: D[15:0]
- 32-bit mode: D[31:0]

F5.4.3 Bus Configuration

Chip Select 0 (–CS[0])

Two external pins, –BMODE16 and –BMODE8 are used to determine the bus configuration in **the chip select 0 (–CS[0])**. The two bus configuration pins have weak pull-ups, so that if unconnected, the bus configuration will default to a 32-bit bus.

Table F5–2. –CS0 Bus Width Configuration

–BMODE16	–BMODE8	–CS0 Bus Width
0	0	Illegal
0	1	16-bit Memory Bus
1	0	8-bit Memory Bus
1	1	32-bit Memory Bus

Chip Select [5:1] (–CS[5:1])

In –CS[5:1], the bus configuration is determined by the Bus Width Control Register (ASI = 0x1, ADR = 0x16C). Bus width for memory space referred by –CS[5:1] can be programmed by writing to the Bus Width Control Register (ASI = 0x1, ADR = 0x16C).

Bus Width Control Register

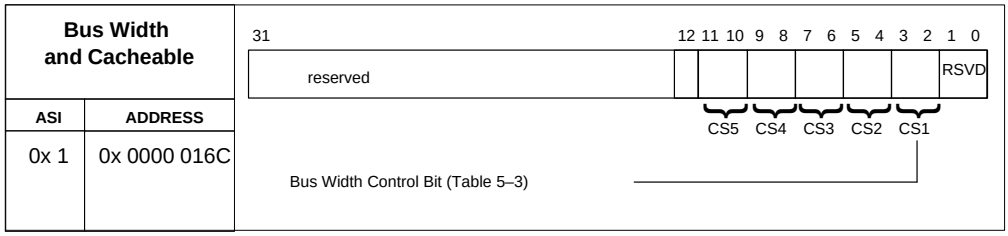


Table F5–3. Bus Width Control Bits of –CS1 to –CS5

BW1	BW0	Bus Width
0	0	32-bit Memory Bus
0	1	8-bit Memory Bus
1	0	16-bit Memory Bus
1	1	Illegal

Memory space referred by each chip select (CS[5:1]) can be individually programmed for 8/16/32 bit bus mode. Upon reset, 32-bit bus mode is the default.

F5.4.4 System Interface

In order to minimize external “glue logic” required for interfacing to the 8- or 16-bit bus, the BE bits are encoded to reflect the two LSBs of a byte address or the LSB of a halfword address. Therefore, the ADR[27:2] and selected –BE bits can be concatenated to form a complete address for a non-32 bit bus mode.

Table F5–4. System Interface BE Bits

Bus Mode	**Byte**	**BE[0:3]**
8-bit bus	0	0000
	1	0001
	2	0010
	3	0011
16-bit bus	0 & 1	0000
	2 & 3	0010

8-bit bus mode address= {ADR[27:2], –BE[2], –BE[3]}
16-bit bus mode address={ADR[27:2], –BE[2]}

–CS[0], which is enabled on reset, and the internal –READY generation logic, can be used to define as the boot memory address space. On reset, the wait state generator, corresponding to –CS[0] for internal –READY generation, is set to 32 cycles. Later on in the boot code, the wait state generator can be changed to a more appropriate value.

F5.4.5 Load

One of the functions of the boot code is to set the processor and system configuration. This might involve loading system parameters from the boot memory, loading data from memory mapped I/O, and storing data to non-boot memory address space. All loads from the boot address space or from any 8/16–bit memory peripherals behave the same way as instruction fetches, in that, for a non-32 bit bus mode –BE , bit encoding and word assembly are done. In order to meet the –BE AC timing, the –BE bits on the MB86933H–20 need to be all 0’s for all types of loads—word, halfword, and byte—from the non 8/16–bit memory space. This requires a functional change from the current specification of the MB86930’s –BE bits, which reflect the byte information for loads. This change does not cause a problem, since the processor fetches a full 32-bit word on a load, and the IU selects the byte appropriately. As on the MB86930 –BE bits should be ignored for 32-bit loads.

A summary of the –BE[0:3] bit behavior for loads from the 8/16–bit bus is shown below. For all load instructions (byte, halfword, word), a full 32-bit fetch occurs. For example, in the 8-bit bus mode, four bytes will be fetched for all loads, and the BE bits will sequence with the proper 2 LSBs of the byte address.

Table F5–5. Load –BE[0:3] Bit Behavior

Bus Mode	Operation	BE[0:3] in PROM space
8-bit bus	Loads (all)	0000=>0001=>0010=>0011
16-bit bus	Loads (all)	0000=>0010
32-bit bus	Loads (all)	0000

F5.4.6 Store

The MB86933H–20 allows user to write to 8/16-bit memory as follows:

(1) In 8-bit Bus Mode, {ADR[27:2], –BE[2], –BE[3]} is the store address. BIU stores only as many access cycles as required. For example, a store byte requires only one access cycle, store halfword requires two access cycles and store word requires four access cycles on a 8-bit Bus. 8–bit DRAM access is not supported by the internal DRAM Controller. For store half word or word, it is always started at the least significant byte first and so on.

(2) In 16-bit Bus Mode, {ADR[27:2], –BE[2]} is the store address, and –BE[1:0] are the byte enables. –BE[1] enables the upper byte (D[15:8]), and –BE[0] enables the lower byte (D[7:0]). For a store byte or store halfword, BIU executes one access cycle. MB86933H–20 takes two access cycles to store word in 16-bit Bus Mode. For store word, it is always started at the least significant half word first and so on.

Notes: Since the MB86933H–20 does not have extra pins to indicate size of the store, to support store in 8/16–bit bus each store access cycle is commenced by an –AS and ended by a –READY (or internal READY). This is different from load in 8/16–bit bus. Section 5.4.9 shows timing diagrams for all cases of load/store.

F5.4.7 Memory Exception

Any memory exception that occurs during a fetch from the non–32 bit address space will be held off until the entire word is fetched.

Any memory exception that occurs during a store access cycle to the non–32 bit address space will be held off until the entire size of the store is completed.

F5.4.8 Bus Request

Any bus request happening during the non-32 bit bus mode fetch will not be recognized until the end of the complete 32-bit fetch operation.

Any bus request happening during a store access cycle to the non–32 bit address space will not be recognized until the entire size of the store is completed.

F5.4.9 Timing

Timing examples for the 8- and 16-bit bus modes with 1 wait-state memory are shown below. Note that –AS is asserted at the beginning for one cycle.

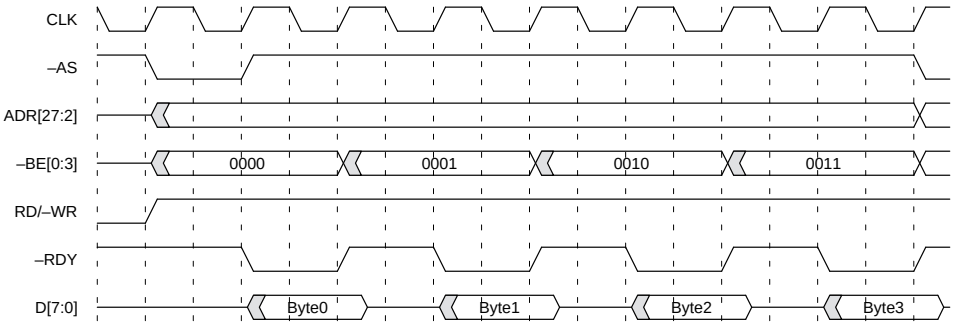


Figure F5–14. 8-bit Bus Mode Read (1 Wait State)

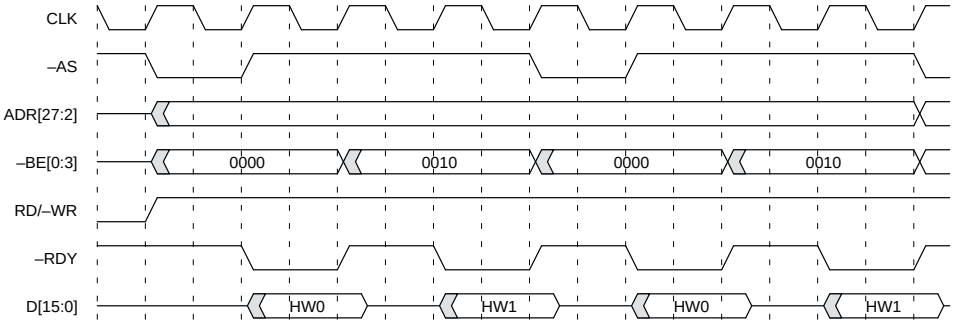


Figure F5–15. 16-bit Bus Mode Read (1 Wait State)

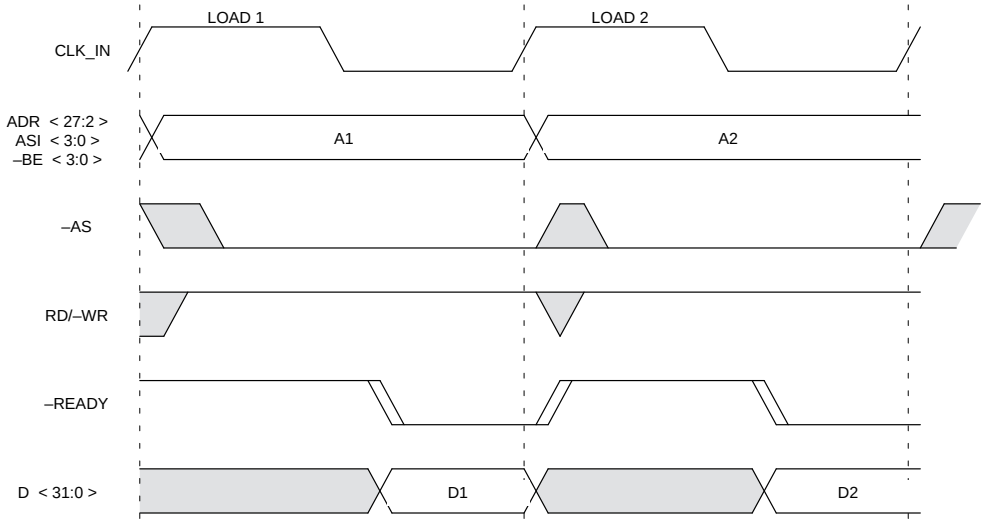


Figure F5–16. Timing 1 – Typical Back–to–Back Loads with 32–Bit Wide Bus (Same as Load Double)

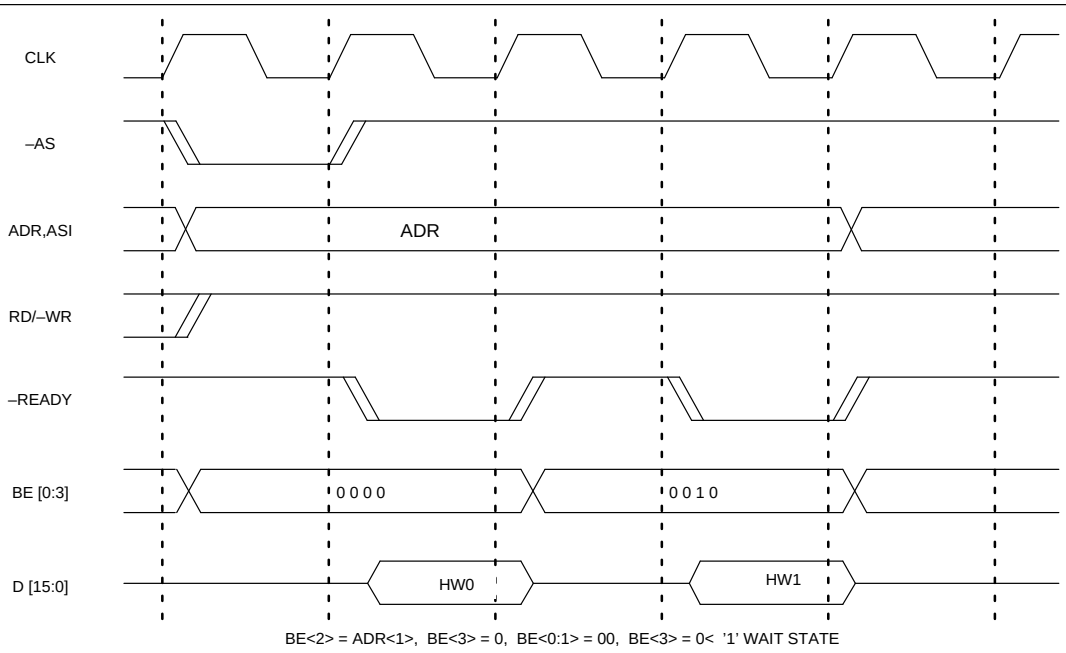


Figure F5–17. Timing 2 – Load with 16–Bit Wide Bus

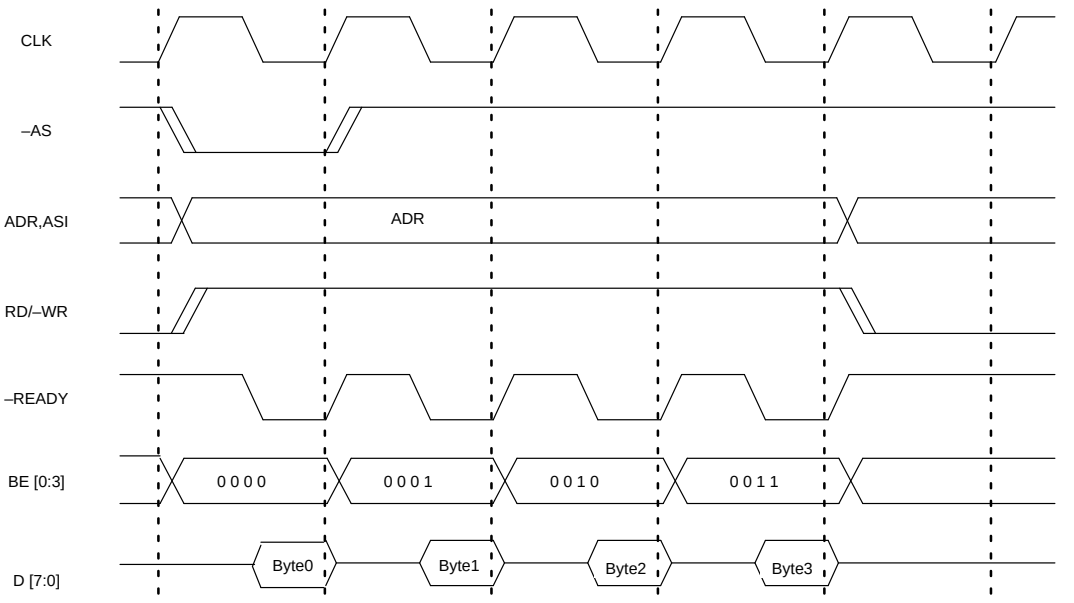


Figure F5–18. Timing 3 – Load with 8–Bit Wide Bus

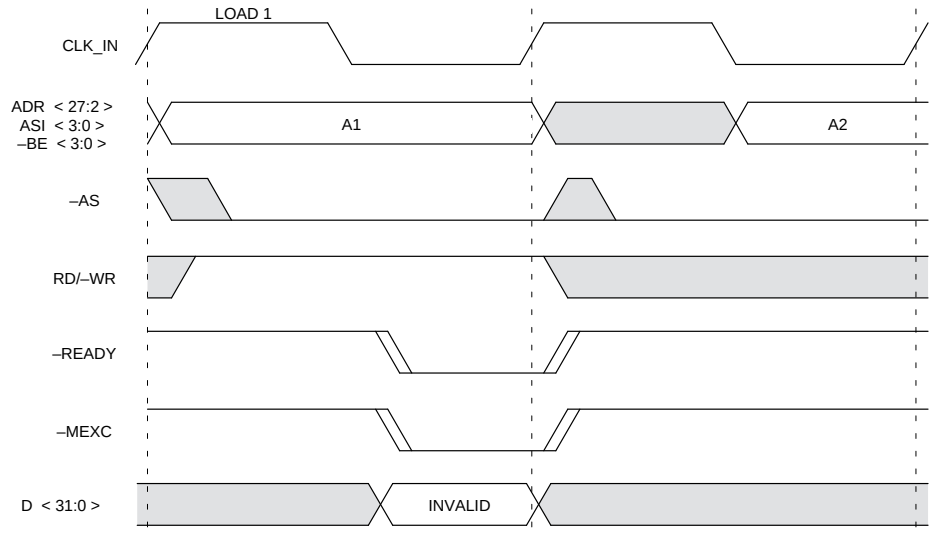
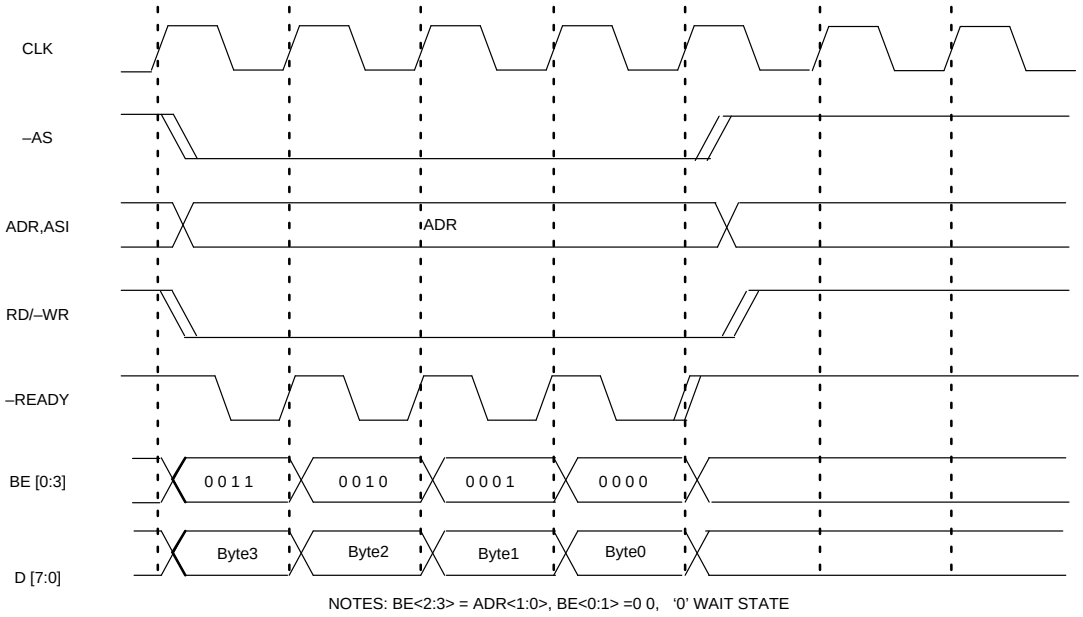
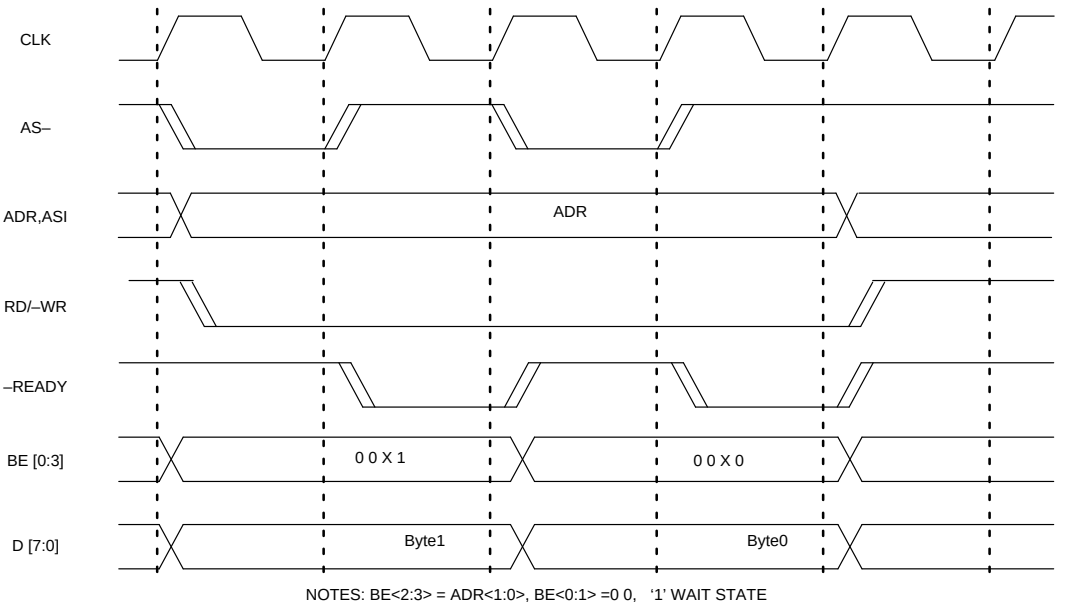


Figure F5–19. Timing 4 – Load with Exception – 32–Bit Wide Bus



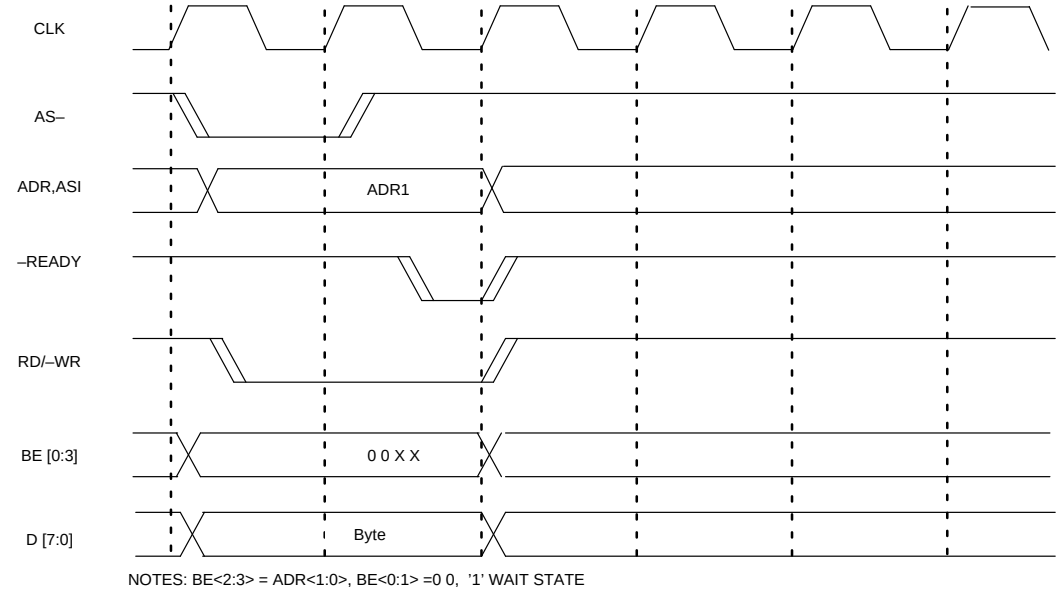
NOTES: BE<2:3> = ADR<1:0>, BE<0:1> =0 0, '0' WAIT STATE

Figure F5–20. Timing 9 – Store Word with 8–Bit Bus



NOTES: BE<2:3> = ADR<1:0>, BE<0:1> =0 0, '1' WAIT STATE

Figure F5–21. Timing 10 – Store Half–Word with 8–Bit Bus



NOTES: BE<2:3> = ADR<1:0>, BE<0:1> =0 0, '1' WAIT STATE

Figure F5–22. Timing 11 – Store Byte with 8–Bit Bus

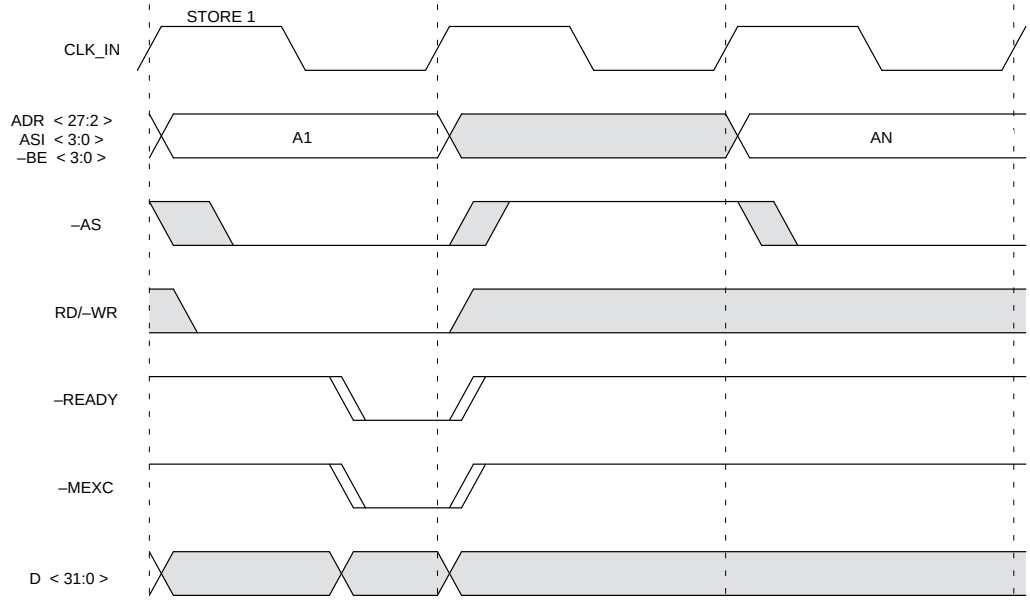
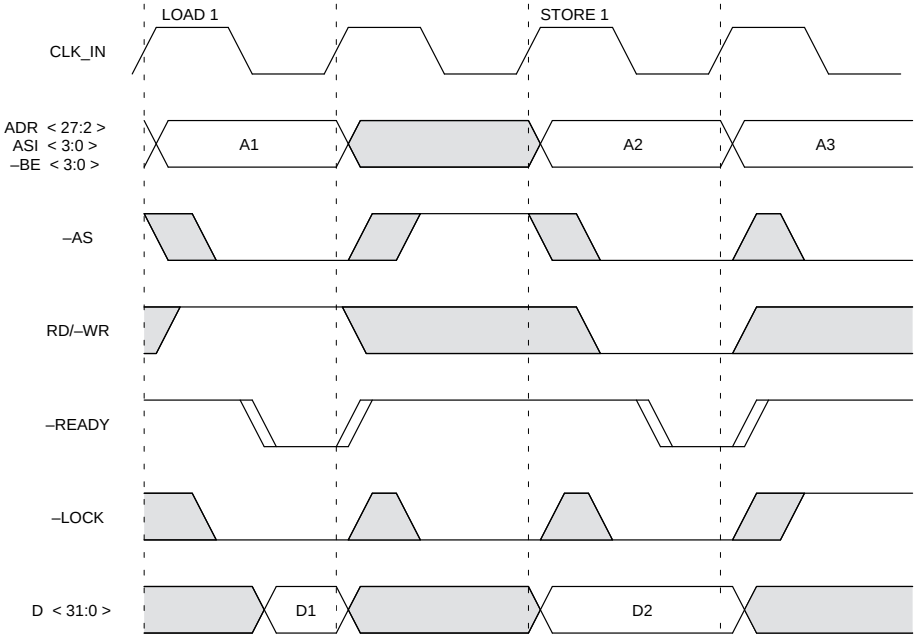


Figure F5–23. Timing 12 – Store with Exception



Note: A load followed by a store requires an intervening clock cycle on the bus while a store followed by a load can occur in consecutive clock cycles.

Figure F5–24. Timing 13 – Atomic Operation

F5.4.10 Store in 32–bit bus mode

For all stores in the 32–bit bus mode the –BE bits will reflect the byte information of the store. The following note may be useful for system designers.

Store Byte: All 4 bytes are the same in the whole word data (i.e., D[31:24] = D[23:16] = D[15:8] = D[7:0]).

Byte	–BE[3:0]
0	1110
1	1101
2	1011
3	0111

Store halfword: 2 half words are the same in the whole word data (i.e., D[31:16] = D[15:0]).

Half Word	–BE[3:0]
0	1100
1	0011

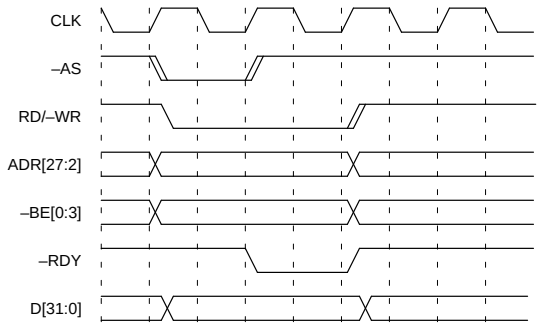


Figure F5–25. Store to 32–bit Address Space

CHAPTER

F6

MB86933H–20 DRAM Controller

F6.1 Overview

The MB86933H–20 integrates an address multiplexer and a state machine with the Same Page Detection logic and the Programmable Timer to form a complete page–mode DRAM controller. The state machine governs the timing relationships of the multiplexed row and column address and the DRAM control signals (–RAS, –CAS, –DWE).

The DRAM controller was designed to support the most common type of DRAM: the page–mode DRAM with CAS before RAS refresh. CAS before RAS refresh uses the DRAM's internal address counter to specify the row to be refreshed and avoids the added cost of an external counter. The DRAM controller does not support interleaving of memory banks.

F6.2 Registers

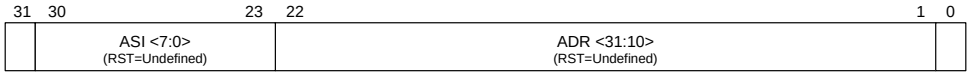
The registers used in configuring and activating the DRAM controller are:

- **Address Range Specifier Register 4 and Address Mask Register 4** – used together to set the DRAM address space boundaries. Accesses to this address space are considered accesses to DRAM and will activate –CS4.
- **DRAM Bank Configuration Registers** – determines the bank address space boundary and address multiplexing.
- **Timer Register and Timer Preload Register** – used together to set the refresh interval.

- **Same Page Mask Register** – sets the page size.
- **Bus Width Register** – used to select the DRAM data bus width. Either 16–bit or 32–bit data width may be used. 8–bit bus DRAM data bus width is not supported.
- **System Support Control Register** – enables the DRAM controller, the Same Page Detection logic and the Refresh Timer

F6.5.1 Address Range Specifier Register 4 and Address Mask Register 4

Address Range Specifier Register 4 is used in conjunction with Address Mask Register 4 to select which portion of a 4GB address space is to be assigned as the DRAM address space. Chip Select 4 (CS4) will be asserted whenever an access is made to an address which falls in the range specified by this register pair. The DRAM address range must not be accessed during the three cycles after this register pair is written.



Address: 0x00000130
ASI 0x01

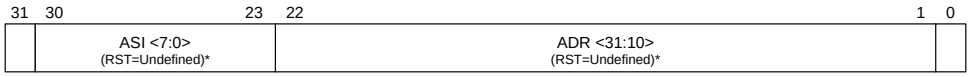
Figure F6–1. Address Range Specifier Registers

Bit 31: Reserved

Bits 30-23: ASI[7:0]—Specifies the ASI of a target address range. The value of this field is undefined on reset.

Bits 22-1: ADR[31:10]—Specifies the 22 most significant bits of a target address range. The value of this field is undefined on reset.

Bit 0: Reserved



* Except AMR[0].

Address: 0x00000150
ASI 0x01

Figure F6–2. Address Mask Registers

Bit 31: Reserved

Bits 30-1: Mask—Specifies which bits in the ASI and address of the current external access are to be compared with the corresponding bits in the address-range specifier. Only those bits are compared for which the mask bit is 0. See Table 2-3 for reset value.

Bit 0: Reserved

Table F6–1. Programming the Address Mask Register

DRAM address space	Value in Address Mask Register 4
1MB	0x000007fe
2MB	0x00000ffe
4MB	0x00001ffe
8MB	0x00003ffe
16MB	0x00007ffe
32MB	0x0000fffe
64MB	0x0001fffe

F6.5.2 DRAM Bank Configuration Register

There is one configuration register for each of the two banks supported by the DRAM Controller. The DRAM type and the starting address of the bank is written to these registers. Together, they determine the bank's size and boundaries.

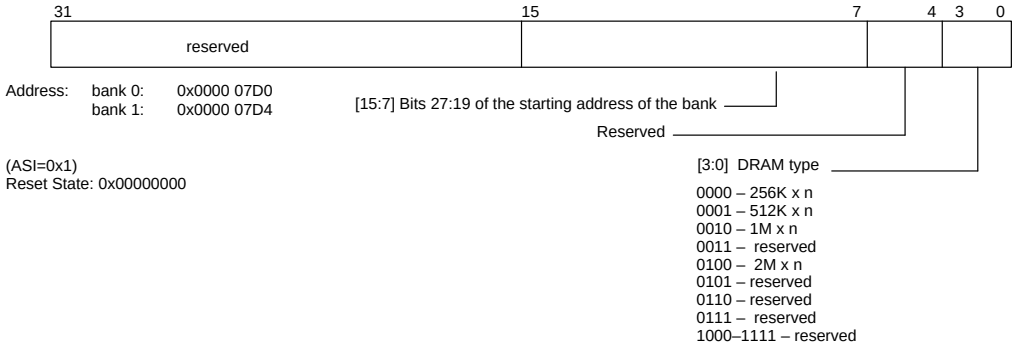


Figure F6–3. DRAM Bank Configuration Registers

DRAMs are specified by bit density, ie. the total number of bits in the chip. For a given bit density, a DRAM is available in different depth and data width combinations. For example, a 16Mbit DRAM can be found in a 2Mx8, 4Mx4, or 16Mx1 arrangement. To form a 32–bit memory bank, the number of chips required are four, eight and thirty two, respectively. The corresponding bank sizes will be 8MB, 16MB, and 64MB. The number of bits used for the row and column addresses are given in the DRAM specification.

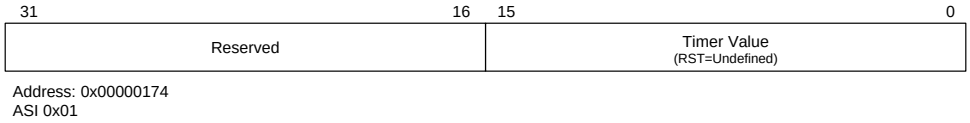
In the MB86933H–20, the DRAM address space is defined by the Address Range Specifier Register and Address Mask Register for CS4. This address space may be further subdivided into banks of addresses. The starting address of a bank and the DRAM type will determine the address range of a particular bank. Addresses falling in the range of addresses defined for a bank will activate the corresponding –RAS. Each –RAS corresponds to a particular bank.

For example, a 32 bit bank of memory using 1M x n DRAMs will yield 4 megabytes. If it is desired that this bank start at address 0x0000000, then bits 15:7 of the DRAM Bank Configuration Register will be set to '000000000', bits 6:4 set to '000' and are reserved, and bits 3:0 set to '0010' to indicate a 1M x n DRAM type .

The DRAM address space may be divided into the different banks in any order. The largest bank does not have to be bank 0. However it is important to observe the restriction that the largest bank occupy the lowest addresses in the DRAM space with progressively smaller banks occupying the higher addresses. The starting address of a bank must be on a bank size boundary (eg. a 4MB bank can start at 0x00000000, 0x00400000, etc.) and the largest bank must occupy the lowest address followed by the next largest bank and so on.

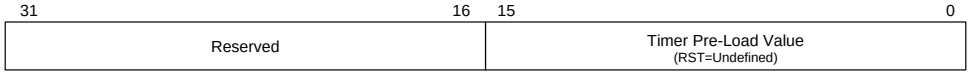
F6.5.3 Timer Register and Timer Preload Register

These registers are used to set the DRAM refresh interval. The Timer Register contains the current count of a 16–bit timer. The Timer Preload Register contains a value which is loaded into the timer when the timer overflows. The timer overflows when its count decrements to zero. The processor will then assert the –TIMER_OVF signal externally. The DRAM controller will detect this and begin a refresh cycle. The minimum value written to the Timer Register is 0x01. Typically, the same value is written to both registers.



Address: 0x00000174
ASI 0x01

Figure F6–4. Timer Register



Address: 0x00000178
ASI 0x01

Figure F6–5. Timer Pre-Load Register

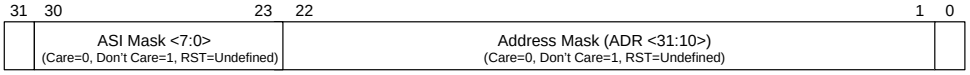
F6.5.4 Same Page Mask Register

If the current DRAM access and the previous DRAM access share the same DRAM row address, then these accesses are said to be in the 'same' page . These accesses need not be sequential. Accesses to the same page in DRAM are faster than accesses to different pages since –RAS remains asserted and only the column address needs to be changed ie. –CAS will be reasserted.

The Same Page Mask Register sets the size of a DRAM page. A page refers to the number of column locations in a given row. The number of column address bits in the DRAM determines the page size. For example, a 1Mx4 DRAM has 10 column address bits and its page size is 1K. If more than one bank is used and the page sizes differ between banks, then the Same Page Mask Register should be programmed for the smallest page size.

This register controls which bits of the current address and ASI will be compared with the previous address and ASI. If the unmasked bits in the current address and ASI match the bits in the previous address and ASI then the current access is in the same 'page' as the previous access. Only addresses mapped to CS4 will be compared. The DRAM address range must not be accessed during the three cycles after this register is written.

The external output signal, –SAME_PAGE, is active when the DRAM Controller is enabled but is not needed and may be left unconnected.



Address: 0x00000120
ASI 0x01

Figure F6–6. Same-Page Mask Register

Bit 31:	Reserved
Bits 30-23:	ASI Mask—Specifies which bits in the ASI of the current external access are to be compared with the corresponding bits in the ASI of the previous access. Only those bits are compared for which the mask bit is 0. Mismatches in any other bits do not prevent the two accesses from being recognized as "on the same page." The bits of this field are cleared to 0 on reset.
Bits 22-1:	Address Mask—Specifies which of the 22 most significant bits in the address of the current external access are to be compared with the corresponding bits in the address of the previous access. Only those bits are compared for which the mask bit is 0. Mismatches in any other bits do not prevent the two accesses from being recognized as "on the same page." The bits of this field are cleared to 0 on reset.
Bit 0:	Reserved

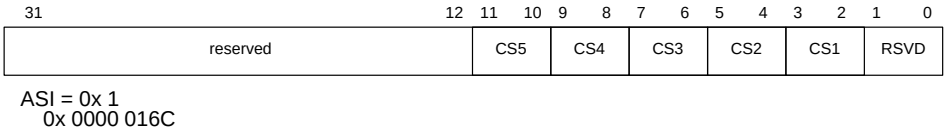
Table F6–2. Same Page Mask Register Values with ASI not masked

# Column bits	16–bit	32–bit
8	not used	0x00000000
9	0x00000000	0x00000002
10	0x00000002	0x00000006
11	0x00000006	0x0000000e
12	0x0000000e	0x0000001e

F6.5.5 Bus Width Register

The DRAM controller supports 32–bit and 16–bit wide memory. It does not support 8–bit wide memory. The data bus width is specified in this register.

Bits 9:8 specifies the bus width for CS4, as defined in table F6–6.



ASI = 0x 1
0x 0000 016C

Figure F6–7. Bus Width Register

Table F6–3. Bus Width Settings

Bus Width	bits [9:8]
32–bit	00
16–bit	10

F6.5.6 System Support Control Register

This register is used to enable the DRAM controller. It also enables the refresh timer. DRAM wait states are set by the DRAM Timing Registers. If the Programmable Wait–State Enable bit is set then the Wait Enable bit of the Wait State Specifier Register for CS4 must be cleared.

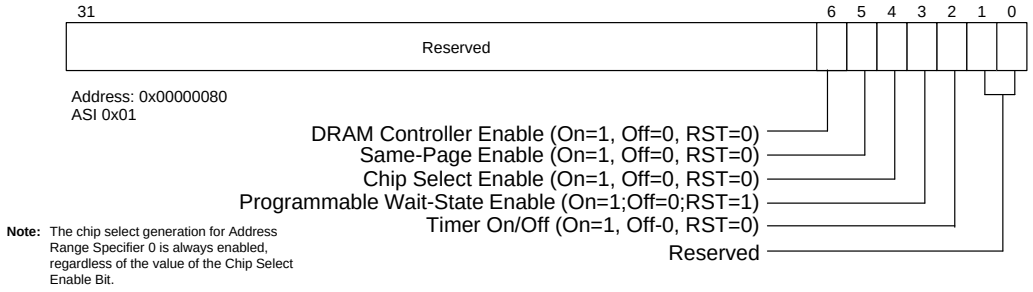


Figure F6–8. System Support Control Register

Bits 31-7: Reserved

Bits 6: DRAM Controller Enable —- Enables (1) and disables (0) the internal DRAM controller.

Bit 5: Same-Page Enable—Enables (1) and disables (0) the same-page detection logic. When this bit is 1, the –SAME_PAGE signal is asserted whenever the address of an external access is on the same page as the previous access. The page size is controlled by the Same-Page Mask Register (see above). When this bit is 0, –SAME_PAGE is never asserted. The Same-Page Enable bit is cleared to 0 on reset.

Bit 4: Chip Select Enable—Enables (1) and disables (0) the generation of chip-select signals for external accesses in address ranges 1 through 5. Regardless of the state of this bit, however, –CS0 is always asserted when the current address lies in address range 0. The Chip Select Enable bit is cleared to 0 on reset.

Note: Before enabling chip selects all chip select Address Mask and Address Range registers should be initialized so that two chip selects are never selected at the same time.

Bit 3: Programmable Wait-State Enable—Enables (1) and disables (0) the programmable wait-state generators for all address ranges. The Programmable Wait-State Enable bit is set to 1 on processor reset. The DRAM Controller does not use this bit to set wait states.

Bit 2: Timer On/Off—Enables (1) and disables (0) the timer. This bit is cleared to 0 on reset.

Bits 1-0: Reserved

Table F6–4. System Support Register Summary

Chip Selects	Affected by Chip-Select Enable?	Address Range Specifier		Address Mask		Wait-State Specifier	
		Address (ASI=0x01)	Value at Reset	Address (ASI=0x01)	Value at Reset	Address (ASI=0x01)	Value at Reset
0	No	N/A	ASI=0x09 ADR<31:10>=0	0x0000 0140	All mask bits 0 except ADR<14:10> = 1	0x0000 0160 (low halfword)	Count 1,2 = 31 Wait Enable=1 Single Cycle =0 Override=1

F6.3 –CAS Behavior during Word, Halfword, and Byte Accesses

The DRAM controller has four –CAS output signals. Each –CAS controls a byte in a 32 bit memory system. –CAS0 controls byte0 (the most significant byte in a Big Endian architecture such as SPARC) and –CAS<1:3> control the least significant three bytes, respectively. In a 16 bit system, –CAS2 controls the most significant byte (byte 2) and –CAS3 controls the least significant byte (byte 3). –CAS0 and –CAS1 are not used in a 16–bit system.

In a 32 bit system, a word access will cause all four –CAS signals to be asserted simultaneously during the –CAS active phase. An access to the most significant halfword will cause –CAS<0:1> to be asserted. An access to the least significant halfword will cause –CAS<2:3> to be asserted.

F6.4 Address Multiplexing

The multiplexed DRAM row and column address appears on ADR[27:16]. These pins should be connected to the DRAM address pins (ADR16 should be connected to the least significant bit of the DRAMs). If ADR27, ADR26, ADR25 or ADR24 are unused they should be left unconnected. ADR[15:2] are unaffected by the DRAM controller and will reflect bits 15:2 of the address.

For 32–bit systems, the least significant address bit is A2. It is also the least significant column address bit. For 16–bit systems, the least significant address bit and the least significant column address bit is A1.

During the row address phase, the row address appears on ADR[27:16] with the least significant bit at ADR[16]. The row address' least significant bit will change according to the number of column address bits in the DRAM. The column bits will appear on ADR[27:16] during the column address phase with the least significant bit (A2/A1 for 32/16 bit systems) going through ADR[16].

For example, given a DRAM with 11 row address bits and 10 column address bits to be used in a 32–bit memory system. The DRAM has 11 address pins and will be connected to ADR[27:16]. During the row address phase, A[22:12] will appear on ADR[26:16] and will be latched by –RAS. A[23] will appear on ADR[27], but since it is not connected to the DRAM, it is ignored.

During the column address phase, A[11:2] will appear on ADR[25: 16]. A[13:12] will appear on ADR[27:26]. Both will be ignored since ADR[13] is unconnected and A[12] is not needed by the DRAM during the column address phase.

In general, all DRAM address pins should be connected to ADR[27:16] with ADR[16] connected to A[0] of the DRAM. If there are fewer than 12 address pins on the DRAM, then some of the ADR pins will be unconnected, as in the example above. The proper row and column addresses will appear on ADR[13:2] and only the relevant address bits will be latched by the DRAM.

Table F6–5. Address Multiplexing in a 32–bit System

# column addr bits	Row address	Column address	Output pins
8	A[21:10]	A[13:2]	ADR[27:16]
9	A[22:11]	A[13:2]	ADR[27:16]
10	A[23:12]	A[13:2]	ADR[27:16]
11	A[24:13]	A[13:2]	ADR[27:16]
12	A[25:14]	A[13:2]	ADR[27:16]

Table F6–6. Address Multiplexing in a 16–bit System

# column addr bits	Row address	Column address	Output pins
8	A[20:9]	A[12:1]	ADR[27:16]
9	A[21:10]	A[12:1]	ADR[27:16]
10	A[22:11]	A[12:1]	ADR[27:16]
11	A[23:12]	A[12:1]	ADR[27:16]
12	A[24:13]	A[12:1]	ADR[27:16]

F6.5 16–bit Operation

The smallest memory data bus width supported by the DRAM controller is 16 bits. The DRAM controller does not support 8 bit wide memory. When using a 16–bit data bus the BIU will make two accesses to load or store a word and one access to load or store a halfword. Instruction fetches involve two accesses.

F6.6 Refresh

–CAS before –RAS refresh is used by the internal DRAM controller. All four –CAS signals are asserted while –RAS is deasserted. After appropriate setup and hold times, both –RAS signals are asserted. –DWE is deasserted during refresh. Care must be taken to insure that sufficient power and ground are supplied to the DRAMs.

F6.7 Programming the DRAM Controller

The internal DRAM Controller is disabled after reset. The user completes the following initialization sequence before making accesses to DRAM through the internal DRAM controller.

- Allocate the CS4 address space by writing the Address Range Specifier Registers and Address Mask Registers for CS4.
- Further subdivide the CS4 address space into bank address spaces by writing the appropriate values in the DRAM Bank Configuration Registers. Insure that the largest bank occupies the lowest address range and that all banks are aligned on bank boundaries, eg. a 4MB bank can start at address 0x00000000, 0x00400000(4MB), 0x008000000(8MB), etc. The starting address of the bank and the DRAM type are programmed into the DRAM Bank Configuration Register.
- Program the refresh interval in Timer Register and the Timer Preload Register. The same value may be used for both registers. Refresh cycles will not occur until the DRAM Controller is enabled. Most DRAMs require eight –CAS before –RAS cycles to occur before this mode of refresh is recognized by the DRAMs. The DRAM space should not be accessed until eight refresh cycles has occurred.
- Program the page size in the Same Page Mask Register.
- If Programmable Wait state is enabled in the System Support Control Register, disable Wait State Generation for –CS4 by setting the WE bit to '0' in the Wait State Specifier Register for –CS4.
- Program the DRAM data width, either 16–bit or 32–bit into the Bus Width Register.
- Enable the DRAM controller, the Refresh Timer, the Chip Selects, and the Same–Page logic by writing the System Support Control Register.

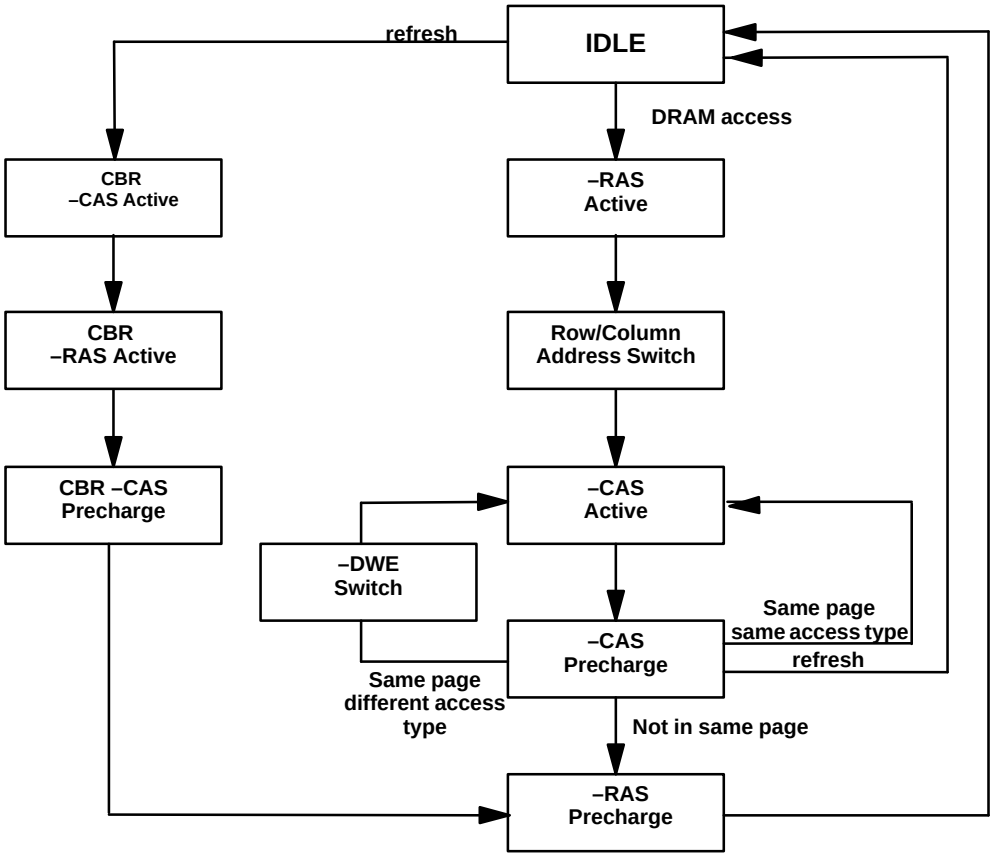


Figure F6–9. DRAM Controller State Diagram

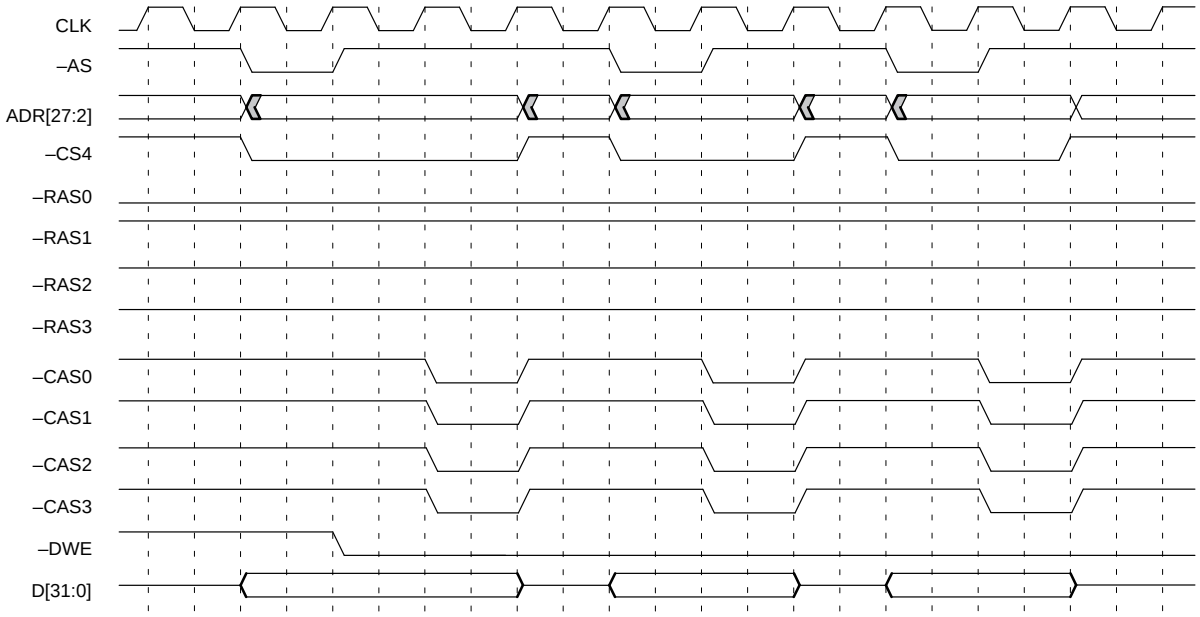


Figure F6–10. Back to Back Page_Mode Writes

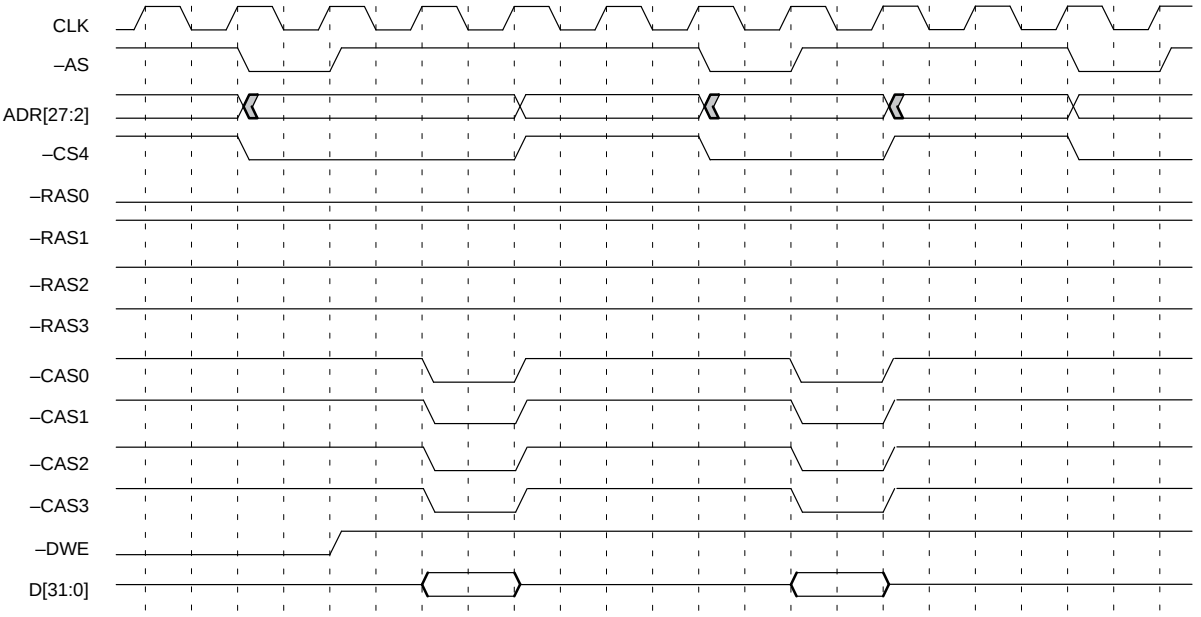


Figure F6–11. Back to Back Page_Mode Reads

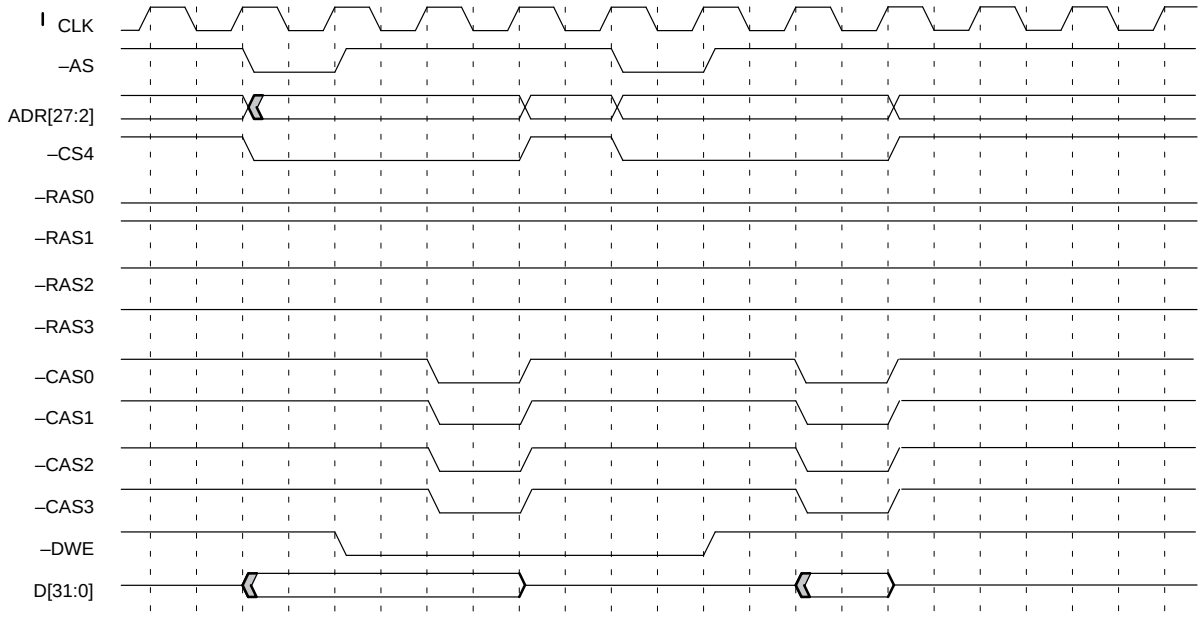


Figure F6–12. Page_Mode Write followed by a Page_Mode Read

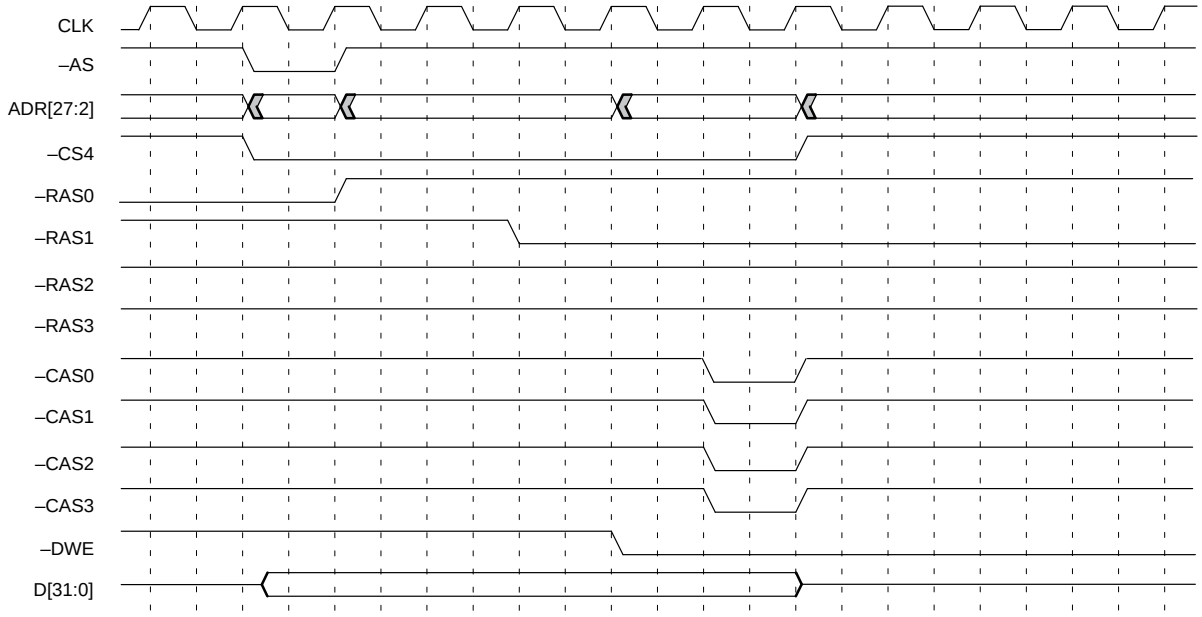


Figure F6–13. Non Page_Mode Write on Bank1 following an access to Bank0

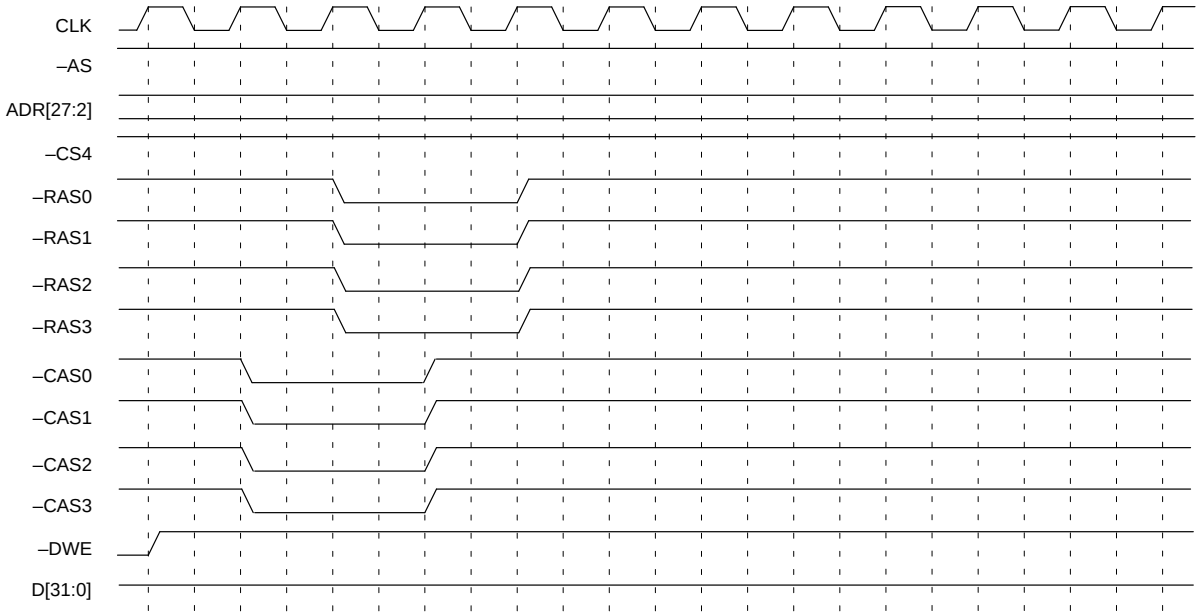


Figure F6–14. CAS before RAS refresh

CHAPTER

F7

System Design Considerations

This chapter describes SRAM and page-mode DRAM interfacing to the MB86933H–20 processor, and MB86933H–20 in-circuit emulation. Chapter 6 of this manual describes system design considerations for SPARClite processors in more detail.

F7.1 Interfacing SRAM

The address bus, data bus, and chip select signals of the SRAM can be connected directly to the address bus, data bus, and a chip select of the processor. The output enable signal can be generated by gating RD/–WR high and Chip select low to produce output enable low. Write enable for the SRAMs requires more consideration.

The processor data hold time for a write is specified as zero hold after the rising edge of the clock. RD/–WR hold time at the end of a write operation can be 0 after the rising edge of the clock, or can be held low if the next cycle is also a write. Thus an implementation cannot use RD/–WR directly as –WE for the SRAMs.

Figure F7-1 shows timing for an typical system using 2 cycle access SRAM operating at 20 MHz. Individual –WE signals are generated for each of the 4 bytes in the data word.

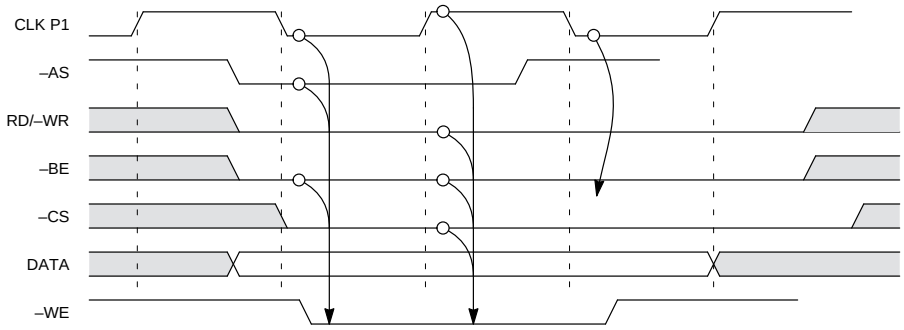


Figure F7–1. SRAM Interfacing Example

The SRAM is controlled with a PAL using the following equations:

```
!clkd = !clkp1;
!soe_ = rw & !scs_;
!swe3_= !rw & !as_ & !be3_ & !clkp1
   # !rw & !as_ & !be3_ & !clkd
   # !rw & !scs_ & !swe3_ & clkp1
   # !rw & !scs_ & !swe3_ & clkd;

!swe2_= !rw & !as_ & !be2_ & !clkp1
   # !rw & !as_ & !be2_ & !clkd
   # !rw & !scs_ & !swe2_ & clkp1
   # !rw & !scs_ & !swe2_ & clkd;

!swe1_= !rw & !as_ & !be1_ & !clkp1
   # !rw & !as_ & !be1_ & !clkd
   # !rw & !scs_ & !swe1_ & clkp1
   # !rw & !scs_ & !swe1_ & clkd;

!swe0_= !rw & !as_ & !be0_ & !clkp1
   # !rw & !as_ & !be0_ & !clkd
   # !rw & !scs_ & !swe0_ & clkp1
   # !rw & !scs_ & !swe0_ & clkd;
```

Clock low, –AS lo, –BE low, and RD/–WR low cause –WE to be asserted. Clock high, –CS low, –BE low and RD/–WR low cause –WE to stay low. When clock goes low again, –WE is negated. This way there is sufficient data hold time.

For this system, CLKOUT1 from the processor was used because it has better duty cycle control than an oscillator clock.

F7.2 Interfacing Page-Mode DRAM using an External DRAM Controller

Interfacing Dynamic RAM requires a DRAM controller for generating RAS and CAS (Row Address Strobe and Column Address Strobe), and for handling refresh. The DRAM controller is typically implemented as a state machine. The DRAM controller and signal interfaces should be designed carefully to accommodate refresh operations and fast page mode access.

The programmable 16-bit timer provided in the MB86933H–20 processor core can be used for timing the refresh interval. The timer output signal, –TIMER_OVF (Timer Overflow), goes low for a single clock cycle at the end of each timer interval. The timer interval is programmed in software, with the correct time interval depending on how the refresh operation is implemented.

The correct number of wait states can be generated by either the processor's internal wait-state generator, or the DRAM controller.

The processor supports fast "page mode" access to DRAM. When the current DRAM address is within the same page as the previous DRAM access, the –SAME_PAGE (Same-Page Detect) signal is asserted. This tells the DRAM controller that DRAM can be accessed using CAS only without selecting a new row of the DRAM, saving time. Page-mode accesses thus provide timing advantages comparable to the burst-mode accesses of some other processors.

To take advantage of page hits, RAS is asserted and left asserted to continuously select a row. CAS is asserted one access at a time to select a memory location in that row. Accesses need not be in consecutive locations. RAS can remain asserted as long as each access is in the same row, and CAS can be asserted once to access each memory location. RAS remains asserted between accesses.

The wait-state generator can be programmed to use a different (smaller) number of clock cycles for a "page hit" (when the current address is within the same page as the previous DRAM access).

When using the internal wait-state generator instead of the external –READY signal, the processor has no way of detecting a refresh operation that occurs during an access. One solution is to have the DRAM controller take control of the bus during refresh using –BREQ (Bus Request), thereby preventing the processor from requesting a memory access for the duration of the refresh operation. The disadvantage of this solution is that the processor is forced to remain idle. An alternative solution is to disable the internal wait-state generator and let the DRAM controller generate the –READY signal for all DRAM accesses.

Figure F7-2 is a simplified state diagram for a DRAM memory controller. Upon reset, the state machine starts in the RAS Precharge and Idle state, and remains in that state until a memory access or refresh request occurs.

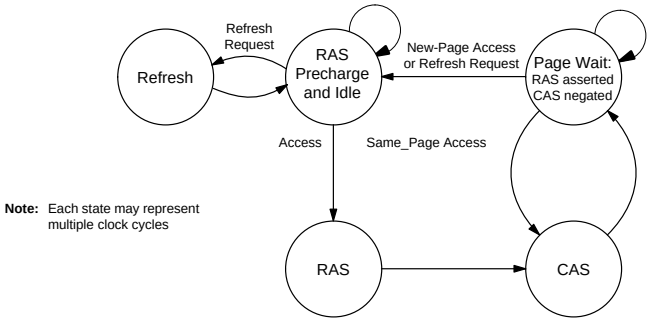


Figure F7–2. Simplified State Diagram for DRAM Controller

If a refresh request occurs, the state machine goes into the Refresh state. (In practice, this will actually be a number of sequential states.) When the refresh operation is complete, the state machine returns to the RAS Precharge and Idle state.

When the processor requests a DRAM memory access, the state machine enters the RAS state, in which the RAS signal is asserted to select the row. From there it goes to the CAS state, in which the CAS signal is asserted to select the column. At this point, data is clocked into the appropriate part, and the bus cycle ends.

From there the state machine enters the Page Wait state, in which the state machine waits for either another memory access, or a refresh request. In this state, RAS is asserted and CAS is negated. If there is a memory access to the same page of DRAM (as indicated by the –SAME_PAGE signal), the state machine goes directly to the CAS state, and CAS is asserted to select the memory location. If there is a memory access to a different page of DRAM or if a refresh request occurs, the state machine goes to the RAS Precharge and Idle state, then to the requested operation. The state machine waits with RAS asserted until one of these events occurs.

For more information, refer to SPARClite Application Note #1, which describes DRAM interfacing.

CHAPTER

F8

Instruction Set

F8.1 MB86933H–20 Instruction Set

The MB86933H–20 processor supports the same instruction set as the MB86930 processor. Chapter 7 of the main section of this manual therefore fully describes the MB86933H–20 instruction set.

Note that the MB86933H–20 has six register windows rather than eight. Therefore, references to eight register windows in the description should be changed to six register windows for the MB86933H–20, and *modulo 8* in the description should be changed to *modulo 6*.

CHAPTER

F9

Programming Considerations

F9.1 MB86933H–20 Programming Information

Chapter 5 of the main section of this manual contains programming information for the SPARClite processors that applies specifically to the MB86930 processor.

The MB86933H–20, however, has no caches, has six register windows rather than eight, and differs from the MB86930 processor in other ways (see the Overview section of this addendum). Therefore, information given in Chapter 5 relating to features that are not supported by the MB86933H–20 should be disregarded. The chapter should be referenced only for programming information that is appropriate for the MB86933H–20.

CHAPTER
F10

MB86933H–20 JTAG

F10.1 MB86933H–20 JTAG Pin List

The MB86933H–20 JTAG cells are arranged in a shift register configuration (see Figure F10-1). When shifting in a JTAG pattern through TDI, the LSB should correspond to the JTAG cell value for IRL<3> pin, and the MSB of the pattern should correspond to the –BMODE16 pin's JTAG cell. As far as JTAG output through TDO is concerned, the first bit out corresponds to IRL<3> JTAG cell value, and the last output bit corresponds to the –BMODE16 cell value. Table F10–1 lists the order of all of the JTAG cells.

Table F10–1. JTAG Pin Order

Order	JTAG Cell	JTAG Cell Type	Function
1	IRL<3>	input	MSB of Interrupt request pin
⋮	⋮	⋮	⋮
4	IRL<0>	input	LSB of interrupt request pin
5	ADR<2>	output	LSB of Address output pins
⋮	⋮	⋮	⋮
19	ADR<16>	output	Address output pins
20	–CAS0	output	DRAM Column address strobe to byte 0

Table F10–1. JTAG Pin Order (Continued)

21	ADR<17>	output	Address output pins
22	ADR<18>	output	Address output pins
23	–CAS1	output	DRAM Column address strobe to byte 1
24	ADR<19>	output	Address output pins
25	ADR<20>	output	Address output pins
26	–CAS2	output	DRAM Column address strobe to byte 2
27	ADR<21>	output	Address output pins
28	ADR<22>	output	Address output pins
29	–CAS3	output	DRAM Column address strobe to byte 3
30	ADR<23>	output	Address output pins
⋮	⋮	⋮	⋮
34	ADR<27>	output	MSB of Address output pins
35	D_i<31>	input	Input bit 31 of D<31:0> bus
36	D_o<31>	output	Output bit 31 of D<31:0> bus
⋮	⋮	⋮	⋮
73	D_i<12>	input	Input bit 12 of D<31:0> bus
74	D_o<31>	output	Output bit 12 of D<31:0> bus
75	–RAS0	output	DRAM Row address strobe for bank 0
⋮	⋮	⋮	⋮
98	D_i<0>	input	Input bit 0 of <31:0> bus
99	D_o<0>	output	Output bit 0 of <31:0> bus
100	dbusiojo	output	D<31:0> bus bidirectional control signal dbusiojo = 1: D<31:0> bus is an input dbusiojo = 0: D<31:0> bus is an output
101	tstatejo	output	Three–state control signal If tstatejo=1 then the following pins are three–stated. ADR<27:2>, ASI<3:0>, –BE<3:0>, –AS, –RD/WR, –LOCK
102	–MEXC	input	Memory exception input
103	–READY	input	External memory transaction complete signal
104	–BREQ	input	Bus request input
105	–AS	output	Start of memory transaction output signal
106	–RD/WR	output	Memory Read/Write output signal
107	–LOCK	output	Bus lock output signal
108	–BGRNT	output	Bus grant output signal
109	–DWE	output	DRAM Write Enable
110	–ERROR	output	Error output signal

Table F10–1. JTAG Pin Order (Continued)

111	–SAME_PAGE	output	Same–Page output signal
112	–CS<0>	output	LSB of chip select output signal
⋮	⋮	⋮	⋮
117	–CS<5>	output	MSB of chip select output signal
118	CLK_ECB	input	PLL control pin. CLK_ECB=1: PLL on CLK_ECB=0: PLL off
119	XTAL1	input	Crystal input
120	–TIMER_OVF	output	Timer Overflow pin
121	–BE<0>	output	Byte 0 enable output signal
⋮	⋮	⋮	⋮
124	–BE<3>	output	Byte 3 enable output signal
125	–ASI<0>	output	LSB of ASI output pins
⋮	⋮	⋮	⋮
128	–ASI<3>	output	MSB of ASI output pins
129	–RAS1	output	DRAM Row address strobe for bank 1
130	–RESET	input	Chip reset pin
131	–BMODE8	input	8–bit Boot Mode
132	–BMODE16	input	16–bit Boot Mode

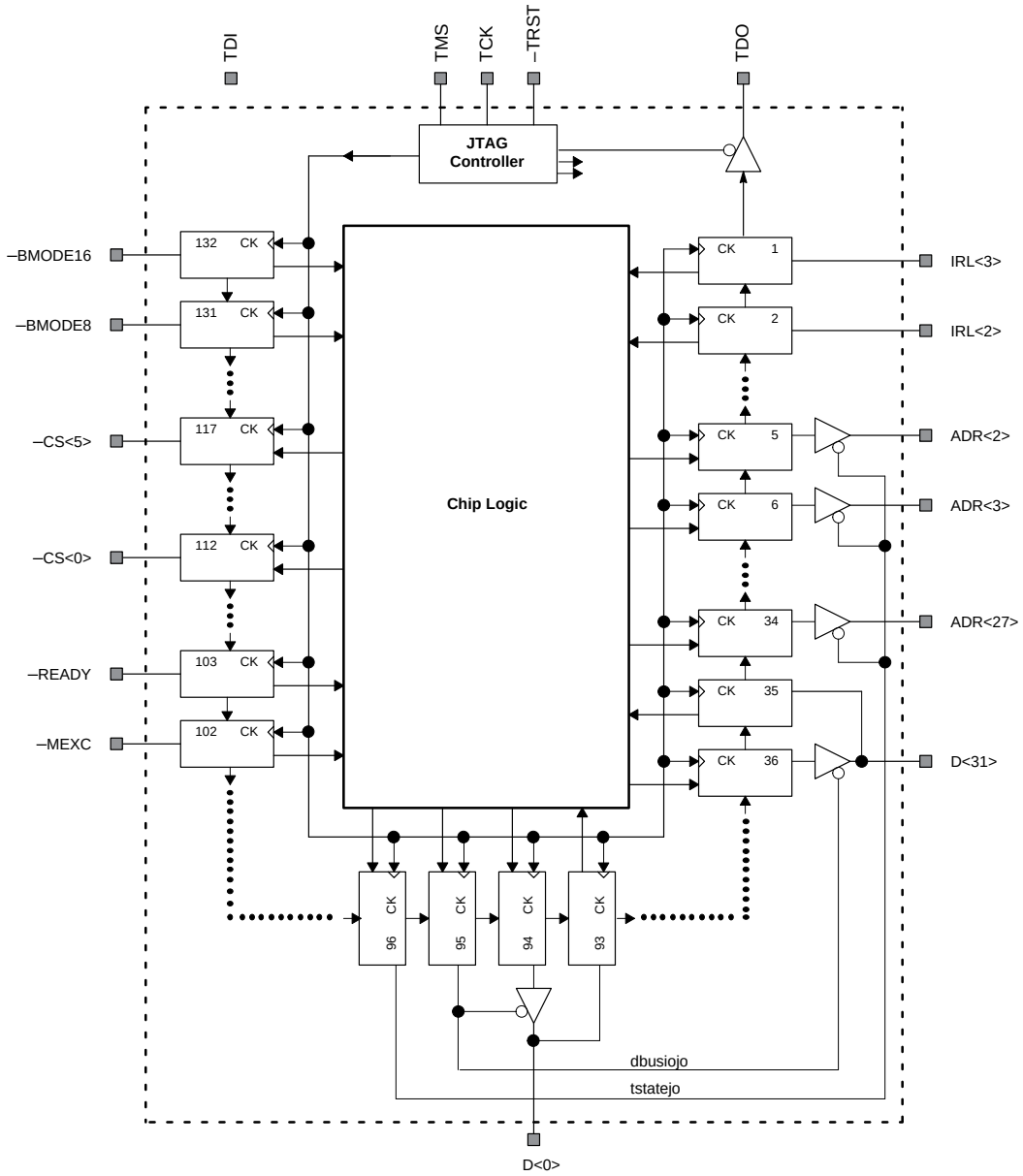


Figure F10–1. AG Cell Organization