Shihman

Service Processor Module

General:

The Service Processor Module (SPM) attaches to the system bus and provides various service funtions for the system. These functions include system console interfaces, front panel interface, real time clock, system clock interrupt source, power supply controls, floppy disk controller, system wide interrupt dispatcher, various system monitor functions (power, temperature), and system diagnostic capabilities.

Hardware:

The Service Processor Module is a special purpose computer consisting of two boards, the main board, which plugs into the CSS bus backplane, and the real world interface board, which contains all connectors for the serial ports, the power supply control connectors, and the temperature sensor connectors. The real world interface board is connected to the main board via a third 96pin din connector at the bottom of the main board. The real world interface board resides behind the CSS backplane in the interface slot behind the Service Processor Main board.
The SPM contains the following elements:

* 68020 processor operating at 10Mhz (1/2 the system clock rate)

* 32 or 64 or 128 kbytes of EPROM organized bytewise.

* 256 kbytes of static RAM organized as 64 k x 32 bits (8 32k x 8 RAM chips)

  floppy disk controller (WD37C65)

* (RWI) Z8030 SCCs (2) for 4 console ports

* Z8036 CIO for internal timing

* (RWI) additional Z8036 CIOs (2) for various input/output bits

* Real time clock and calendar chip (MK48T12) (also contains 2K bytes of RAM for nonvolatile storage) and battery

* (RWI) RS232 drivers and receivers

* A/D converter chip for voltage and a temperature sensor

* (RWI) a second A/D converter for system temperature sensors

* (RWI) relays for power supply controls

  CSS bus interface

state machine and queuing registers for interrupt dispatcher

* Clock controller for switching between a local CPU clock and the CSS bus clock

Mapping RAM (16 x 8 RAM for mapping the entire systems address space in 1/4 Gbyte segments into the 68020s address space).

all sections marked with a (RWI) are located on the real world interface

Power:

Part of the SPM is powered separately from the rest of the system (all sections marked with an *).
This separate power supply must be capable of supplying +5V at 3.5 A and +12V at .25 A and -12V at .25 A. This power supply is ored with the main power supply, so that, after the main power is on, the SPM can continue to function, even in the event of a failure of the separate SPM power supply.

Physical:

The Service Processor consists of two physical PC boards. The main board is a standard size CSS board which resides in the CSS cardcage. The real world interface board contains the serial port drivers and receivers, the serial controller chips (SCCs), an Analog to Digital converter chip for sensing temperatures, a parallel I/O chip (CIO) and relays and optoisolated input receivers for controlling power supplies and all necessary connectors to support these funcions.
The real world interface board receives power from the main board and is powered whenever the SPMs CPU is powered, i.e. even when the system's main power is turned off (see above).

68020 Processor:

The SPMs CPU is a 12.5 MHz 68020 which operates either from the CSS bus system clock divided by two (10 MHz) or its own 10 MHz clock source. (See below for clock select information).

Address Decoding:

The address bits of the 68020 are numbered Add.00 to Add.31 with Add.00 as the least significant bit and Add.31 the most significant bit. Add.31 is used to differentiate between CSS bus accesses and local board acesses. Add.31 = 0 means the CPU is accessing a local resource. The local resources are further decoded as follows:

Add.31 30 29 28 27 26 25 24

0	x	x	x	0	0	0	0	0	EPROM (or SRAM)
0	x	x	x	0	0	0	1	1	SRAM (or EPROM)
0	x	x	x	0	0	1	0	2	Z-bus Peripherals
0	x	x	x	0	0	1	1	3	Real Time Clock & NV RAM
0	x	x	x	0	1	0	0	4	reserved
0	x	x	x	0	1	0	1	5	Clock control
0	x	x	x	0	1	1	0	6	reserved
0	x	x	x	0	1	1	1	7	Status and Control Registers
0	y	y	y	1	0	0	0	8	Map RAM location (yyy)
0	x	x	x	1	0	0	1	9	Floppy disc
0	x	x	x	1	0	1	0	A	reserved
0	x	x	x	1	0	1	1	B	Interrupt Dispatcher Idle Que
0	x	x	x	1	1	0	0	C	Interrupt Dispatcher Queue
0	x	x	x	1	1	0	1	D	Interrupt Dispatcher Pointer
0	x	x	x	1	1	1	0	E	Interrupt Dispatcher Misc.
0	x	x	x	1	1	1	1	F	reserved

x are not decoded and are don't cares.

EPROM:

The EPROM is 8 bits wide and can be read by the processor on data bits 24 to 31. There are 2 sockets for the EPROM, one or both can be populated with either 27256s (32k x 8 each), or 27512s (64k x 8 each) or 271001s (128k x 8 each). Address bits Add.00 to Add.15 or Add.16 or Add.17 are used to address the EPROM. The decoding of the high bit is controlled by the Promsize field in control register 2. This allows the two EPROM chips to be contiguous regardless of their size. (See Control register 2 desciption for details).

SRAM:

The SRAM is 32 bits wide and consists of 8 or 4 32k x 8 static RAM chips for 64k x 32 or 32k x 32 bits of memory. It is addressed with address bits Add.02 to Add.17.

Z - Bus Peripherals

There are 7 peripherals attached to the Z - bus. 2 of them are located on the main board, and 5 are on the auxiliary board. When the Z-bus is selected (Add.31 = 0 and Add.27 to Add.24 = 2), Add.11 to Add.08 are further decoded as follows:

Add.11	10	9	8		
0	0	0	0	0	reserved
0	0	0	1	1	Local Analog to Digital Converter
0	0	1	0	2	Local CIO
				3 - 7	reserved for local devices
1	0	0	0	8	Auxiliary SCC 0
1	0	0	1	9	Auxiliary SCC 1
1	0	1	0	A	Auxiliary CIO 0

1	0	1	1	B	Auxiliary Analog to Digital Converter
1	1	0	0	C	Auxiliary CIO 1
1	1	0	1	D	Auxiliary Temperature mux
				E - F	reserved for auxiliary devices

Local Analog to Digital Converter:

The local A/D converter is an MC14442 and a REF02 voltage reference. It is used to measure the temperature on the Service Processor (i.e. the cardcage) and the 6 voltages present on the SPM (i.e. +5V main, +5V aux., +12V main, +12V aux., -12V main, and -12V aux.).

Local CIO:

The local CIO is a Zilog 8036, it is used for various timing functions including the CSS bus timeout and the main System clock interrupt.

Auxiliary Z-Bus Peripherals

The Auxiliary Z-bus peripherals are covered as part of the Real World Interface board (see separate spec)

Real Time Clock:

The real time clock is a MK48T12 chip. This chip has a built in crystal oscillator and a lithium battery. In addition to a real time clock, it has 2K bytes of nonvolatile RAM. This RAM can be used to keep configuration information. The service processor does not have any switches.

Clock Control:

The Clock control port is a single bit port which is written with data bit 0. Data.00 = 0 means the CPU is using its on-board 10 MHz clock, Data.00 = 1 means the CPU is using the CSS bus clock divided by two as its clock source. When switching from one clock source to the other, approximately 1 us after the write to the clock control port, the hardware will issue a reset to the CPU, hold the reset for approximately 500 us, and switch clocks during the reset. This is required to meet the timing specifications of the 68020. In the process, the entire board will be reset, so any VLSI chips will have to be reinitialized. The memory, however, will keep its data valid. If the main power is turned on, the state of the clock bit (i.e. which clock is being used) can be read in the status register. If the CSS bus clock is not running (e.g. the CSS bus is powered down), the hardware will not select the CSS clock. If the CSS bus clock is selected and fails, the clock select mechanism will switch to the local clock.

Status and Control Registers:

There are 3 Status registers and 3 control registers. The status registers are read by the CPU, and the control registers are written by the CPU. When the Status/Control is selected (Add.31 = 0 and Add.27 to Add.24 = 7), Add.07 to Add.05 are further decoded as follows:

Add.07	06	05			
0	0	0	0	Read	Write control register 0 readback
0	0	1	2	Read	Write control register 1 readback
0	1	0	4	Read	Write control register 2 readback
0	1	1	6	Read	reserved
1	0	0	8	Read	CSS command register
1	0	1	A	Read	Status register
1	1	0	C	Read	CSS Error information register
1	1	1	E	Read	Dispatcher Error Register
0	0	0	0	Write	Write control register 0
0	0	1	2	Write	Write control register 1
0	1	0	4	Write	Write control register 2
			6-E	Write	reserved for other control bits

CSS Command Register:

The CSS command register is a 32 bit register which allows the local CPU to read the value written to the SPM via a CSS write to address 0xXXXX AAYY by any other CSS module. X are don"t care and AA will be captured by the SPM and can be read in the Status register. YY must be of the form 100X XXXX. The SPM only decodes the top three bits of the least significant byte of CSS address. When another module writes to an address of the form described above on the SPM, the 68020 receives an interrupt informing it that CSS command has been received. The 68020 will read the CSS command register and the status register to determine the data of the command.

Status Register:

The Status register is a 32 bit register, data bits 00 to 31 can be read by the 68020. The bits mean the following:

data.00 = CSS bus nak at last CSS error (active high)
data.01 = CSS bus ack at last CSS error (active high)
data.02 = CSS bus granterror at last CSS error (active low)
data.03 = Protocol violation (if = 1, there is a violation, e. g. bad parity or illegal type, if = 0, there is no violation, but another error, e. g. missing ack.)
data.04 = reserved
data.05 = CSS bus active at last CSS error
data.06 to 07 = reserved
data.08 to 15 = CSS data byte 6 at last CSS command write

data.16 to 19 = Source slot of the originator of the last CSS command write
data.20 to 23 = reserved
data.24 to 27 = The SPM's own slot id as read from the backplane
data.28 = Idle full. If 0, indicates that the dispatcher idle queue is full.
data.29 = status of clock (if = 1, clock is local, if = 0, clock is CSS clock divided by 2)
data.30 = Floppy ready (active low)
data.31 = CPU dispatcher access allowed (if = 0, the interrupt dispatcher has suspended its dispatching function and the CPU is allowed to access the interrupt dispatcher RAMs. If = 1, the dispatcher is running normally and cannot be accessed by the CPU.)

CSS Error information register:

When the SPM detects an error on the CSS bus, the hardware will interrupt the 68020. The 68020 will read the CSS error information register to determine the nature of the error. The conditions which cause such an error are: bad data parity, bad type parity, destination error, source error, invalid type, bus nack, or granterror. All error conditions will latch in valid data for the CSS error information register as follows:

data.00 to 07 = bus data parity 0 to 7 (bit x = 0 means parity error on byte x)
data.08 to 13 = bus type 0 to 5
data.14 = source error (bit = 1 means error)
data.15 = destination error (bit = 1 means error)
data.16 to 19 = bus source 0 to 3
data.20 to 23 = bus destination 0 to 3
data.24 = bus type parity (bit = 0 is error)
data.25 to 26 = reserved
data.27 to 31 = CSS data bits 57 and 40 to 43.

Additional error information is captured in the Status register (see above). When a bus error condition has been latched in the error register, no new error conditions can be latched until after the bus error register has been read by the 68020.

Dispatcher Error Register:

When the Interrupt Dispatcher detects an error, it will suspend its operation and latch error information in this register. This register is a 32 bit register with the meaning of the bits as follows:

data.00 to 06 I/O bus source slot (only valid if

data.07 = 1, don't care otherwise)

data.07	I/O bus interrupt. When this bit is a 1, the interrupt came from an I/O bus module attached to the CSS bus module. If this bit is a 0, the interrupt came from a CSS bus module.
data.08 to 11	CSS bus source slot
data.12 to 15	CSS bus destination slot
data.16 to 17	Interrupt Priority
data.18 to 19	reserved
data.20 to 22	error code:

22	21	20	
0	0	0	reserved for hardware failure
0	0	1	Interrupt acknowledge received without pending interrupt
0	1	0	Request error (timeout or error on attempted Interrupt level request)
0	1	1	Receive error (new interrupt arrived, but queue is full)
1	0	0	reserved for hardware failure
1	0	1	Interrupt acknowlegde response error (timeout or error on attempted acknowledge response)
1	1	0	same as 010.
1	1	1	same as 011.

data.23	Directed Interrupt.
data.24 to 31	Interrupt Vector number

When the Dispatcher detects an error, it will assert a level 1 autovector interrupt at the 68020. The interrupt service routine must assert the cpu.disp.req* bit to gain access to the dispatcher, remove the cause of the error, and reenable the dispacher by deasserting the cpu.disp.req* bit.
Removing the cause is recommended as follows:

error code	recommended action
001	assert, then deassert forc.res.ack*.
010 or 110	identify the intended target of the error interrupt and remove this slot number from the CFIG RAM. Examine the pointers for directed interrupts pending for this slot number and compute their difference. Decrement the counters for these directed interrupts by these differences. Set the pointers to be equal. This has the effect of flushing all directed interrupts pending for the target on which the attmpted request failed.

011 or 111 assert, then deassert forc.res.recv*.

Write Control Register 0:

The Control register 0 is a 32 bit register with the meaning of the bits as follows:

data.00 to 05 = Forced type (this is used instead of the hardware generated type, if Use.forced.type is active (see control register 1)).

data.06 to 07 = forced address bits 00 and 01. When 'use.forced.type' is active, these two bits are used as the low address bits (CSS data bits 70 and 71).

data.08 to 11 = slot id for use of the interrupt dispatcher.

data.12 to 15 = slot id for use of the CSS backplane .(This is the value which is used as the SPMs slot id for comparison with the destination of a CSS bus command from another module. if the destination address on the bus matches this field, the SPM will assume that it is being adressed).

data.16 to 19 = out.src 0 to 3. This is the source id which the SPM will use as the source when it is issueing a command on the CSS bus.

data.20 to 23 = out.src 0 to 3*. This is the complement of the source id. (It is separately settable in order to facilitate the forcing of errors).

data.24 = forc.ack.nak*, when asserted (=0), forces a single (50ns) occurrence of busack, if diag.any.type is false (=1) or a single occurrence of busnack, if diag.any.type is true

data.25 = forc.grant.err*, when asserted (=0), forces a single occurrence of granterr. This and the preceeding signal are edge sensitive; i.e. the true going (transition form 1 to 0) edge causes the descibed action.

data.26 = forc.bad.dest*, when asserted, forces a mismatch between dest(0...3) and dest(0...3)* on any CSS bus access.

data.27 = freeze = 1 and css.res.ready* = 1 together cause the bus.freeze* to be true (=0).

data.28 = inhibit*, when asserted (=0), causes the CSS bus error capturing to be inhibited.

data.29 to 31 = reserved

Write Control Register 1:

Write Control Register 1 is a 32 bit register with the meaning of the bits as follows:

data.00 to 07 = out data.parity 0 to 7. These bits set the polarity of the parity bits for each byte of the CSS data bytes. (0 = parity even, 1 = parity odd. This is used to force data parity errors).

data.08 to 11 = fake.arb.dest 0 to 3. These bits are presented to the arbiter as the intended destination of a requested command if the use.fake.dest* bit

is on (= 0).

data.12 = terminal count - used by the floppy controller.

data.13 = force.response*. this bit is used to force the response signal at the arbiter when arbiter request is asserted. It is used for memory emulation.

data.14 = fdack*. used by the floppy controller.

data.15 = floppy.motor*. Used by the floppy controller.

data.16 to 17 = These two bits are used to indicate to the hardware what type of reads are to be done when the CSS bus is selected for a read.

data.17	16	CSS reads performed:
0	0	4 byte
0	1	8 byte
1	0	16 byte
1	1	32 byte

data.18 = int.disp.res*. This signal, when asserted, resets the interrupt dispatcher, if the CSS bus clock is active.

data.19 = Use.forced.type*. When this bit is a 0, the forced type from control register 0, bits 0 to 5 is used as the bus type. If the bit is 1, the hardware will generate the bus type according to the access being performed. (This bit must be 1 for proper operation of the interrupt dispatcher).

data.20 = Type.polarity. This bit sets the polarity of the type field (normally = 1).

data.21 = force.reset.receive*. This bit resets the received flag to the interrupt dispatcher. It must be used when the dispatcher has caused an error during the reception of an interrupt.

data.22 = Idle.reset*. This is the reset for the Idle FIFO. It must be written with a 0, then a 1, for proper operation of the interrupt dispatcher. It can be used during operation of the dispatcher, if the CPU wishes to clear the idle FIFO.

data.23 = CSS.reset*. This bit, when 0, holds the reset line on the CSS backplane at 0. It must be asserted (= 0) during main power up, and = 1 during normal operation.

data.24 = cpu.disp.req*. When 0, this bit indicates to the interrupt dispatcher statemachine that the 68020 wishes to access the dispatcher RAMs. The state machine will suspend normal dispatcher operation and turn on (set = 0) bit 31 in the status register. When 1, the interrupt dispatcher resumes its dispatching function.

data.25 = diag.frc.int.rec*. When 0, this bit indicates to the hardware that it is to decode a CSS CPU interrupt request level as a CSS command and treat it as such. This bit is used for diagnosing the interrupt dispatcher only, it is normally = 1.

data.26 = diag.any.type*. When 0, this bit indicates to

the hardware that it is to accept any CSS bus action as a valid response to a SPM initiated read request. It is only used for diagnostic purposes and is normally = 1. If there is a response, the response data will be latched in the CSS command register and the CSS command interrupt will be asserted.

data.27 = use.fake.dest*. When this bit is 0, the destination at the arbiter is forced to be bits 08 to 11 of control register 1. If this bit is a 1, the destination at the arbiter is the same as the destination which will be used on the CSS bus with the requested CSS bus transaction. This bit is normally 1.

data.28 = force.modify*. When 0, this bit forces a modify cycle on the CSS bus as long as it is asserted (= 0). This bit is normally 1.

data.29 = cpu.res.ready*. This bit resets all pals associated with CSS activity. It also resets the ready count. If asserted (=0), all CSS interrupt sources are disabled. It is recommended, that this bit be = 0, when switching to the CSS clock.

data.30 = cpu.inc.ready*. This bit increments the ready count in the arbiter for the SPM. It must be used after the SPM receives a CSS command addressed to it which caused a 68020 interrupt to increment the ready count for the SPM. The SPM only has a single register for receiving commands, so the ready count is either 0 or 1. This bit and the cpu.res.ready* are egde sensitive, i.e. a transition from a 1 to a 0 causes the assertion of the corresponding bit on the arbiter for one CSS bus cycle. A transition from a 0 to a 1 has no effect.

data.31 = forc.res.ack*. This bit is used with dispatcher error handling, see above.

Write Control Register 2

This is a 8 bit register writeable with data bits 24 to 31. Their meaning is as follows:

data.24 = prom.at.1. When 0, the EPROM is located at starting address 0x0000 0000. When this bit is 1, the EPROM starting address is 0x0100 0000.

data.25 to 26 = promsiz.0 and promsiz.1. These two bits are used to indicate the size of the EPROM chips to be used:
0 0 = EPROMs are 27256s (32k x 8)
0 1 = EPROMs are 27512s (64k x 8)
1 0 = EPROMs are 27010s (128k x 8)
1 1 = reserved - do not use.

data.27 to 30 = LED.0 to LED.3. When a bit is 0, the LED is on, when the bit is 1, the LED is off. LED.0, 2, and 3 are red, LED.1 is green.

data.31 = bd.reset*. When 0 this bit holds the hardware of the SPM, with the exception of the 68020

in reset. On power up or when the 68020 has been reset, this bit will be 0. To allow operation of the board, the CPU must write this bit with a 1.

Map RAM:

The map RAM consists of two 16 x 4 RAM chips which are used as an 8 x 8 RAM. It is addressed with address bits add.28 to 30. When address bit add.31 = 1, the SPM is performing a CSS access, and the map RAM is read. CPU Address bits 28 to 30 select one of eight locations in the map RAM. The top four bits of the map RAM output give the destination (i.e. the slot id of the board being accessed) and the bottom four bits are mapped to address bits bus_data_44 to bus_data_47 for the CSS bus. The lower 28 bits of the 68020s address are directly mapped to address bits bus_data_40 to 43 and bus_data_50 to 77 in the conventional manner.
The map RAM is written when add.31 = 0 and add.24 to 27 = 8. It is written with data.24 to 31 at the location selected by add.28, 29, and 30. Before accessing the CSS bus, the 68020 must initialize the map RAM.

Floppy Disc Controller:

The floppy disc controller is a WD37C65 single chip controller. It controls a single 5 1/4 in floppy drive.

Interrupt Dispatcher CPU Access:

The CPU has four decoded address ranges for accessing the various RAMs present in the interrupt dispatcher. This allows the CPU to initialize the interrupt dispatcher and full diagnostic capabitily for analyzing any dispatcher malfunctions.

Interrupt Dispatcher Idle Queue:

This is a 16 deep by 4 wide FIFO. It is written with data bits 24 to 27. If there are valid data in the Idle FIFO, the interrupt dispatcher will consider the data slot ids of CPUs which are preferred to receive the next interrupt request. When an interrupt is requested from the CPU corresponding to the first entry on the FIFO, that entry is taken off the FIFO. The SPMs 68020 will normally use this FIFO to indicate to the interrupt dispatcher, which CSS CPUs are idle.

Interrupt Dispatcher Queue:

The main queue is a 4k x 32 bit RAM which contains all the information attached to an interrupt as it is defined on the CSS bus (see the interrupt dispatcher description for details). It is adressed with add.02 to 13. It does not need to be intialized, however it is recommended that it is initialized with a known value for diagnostic reasons.

Interrupt Dispatcher Pointer:

There are two pointer RAMs, each 256 x 7 bits, called the incoming pointer RAM and the outgoing pointer RAM. These pointer RAMs allow the main queue to function as multiple FIFOs. The pointer RAMs are acessed together with add.02 to 09. Data.24 to 30 are for the incoming pointer, data.16 to 22 for the outgoing pointer. The ponters must be initialized in such a way that at all address locations, the incoming pointer equals the outgoing pointer (e.g. all 0).

Interrupt Dispatcher Miscellaneous RAMs:

There are three other RAMs in the interrupt dispatcher, they are the counter RAM, the service pending RAM, and the CPU configuration RAM.
They are all simultaneously addressed as a 32 bit port, with different data bits going to the different RAMs as follows:

name	size	address	data
counter	8 x 10	02 to 04	00 to 09
serv.pend	128 x 1	02 to 08	15
config.	16 x 5	02 to 05	24 to 28

The counter must be initialized to all 1s, the service pending must be initialized to all 0s, and the configuration RAM must be initialized with all slot ids of CPUs present in the CSS backplane which are available for receiving interrupts as follows:

address	data (24 to 27)	data.28
CPU slot id n	CPU slot id 1	X
CPU slot id 1	CPU slot id 2	X
.	.	
CPU slot id n-1	Cpu slot id n	X
n+1	CPU slot id 1	X
.	.	
16	CPU slot id 1	X

X = 1 means the CPU slot number at this address can be the target of any interrupt, X = 0 means the CPU slot number at this address must be the Service Module itself.

Interrupt Dispatcher:

The action of dispatching an interrupt consists basically of three parts:
1.) Receive an interrupt request. The interrupting device issues a CSS write to address FFFF FFA0 on the SPM.

2.) Request a CPU interrupt service. The SPM issues a CSS write to a CSS CPU at address FFFF FFFC with data = xyxx xxxx where x = don't care and y = 1ppp where ppp is the requested CPU interrupt priority level.

3.) Acknowledge an interrupt. The requested CSS CPU issues a read to the SPM at address FFFF FFYY where YY = 00pp p100 where ppp is the CPU interrupt priority level which is acknowledged. The SPM responds to the read with the interrupt data corresponding to the interrupt pending at the priority level which is being acknowledged.

General Description:

The Interrupt dispatcher hardware consists of the following:

Incoming interrupt register (32 bits)

Pointer RAM (two 256 x 8 RAMs with latches and +1 adders for pointing into the main queue RAM and over- and underflow comparators (full and empty indicators)).

Main queue RAM (2 2k x 32 RAMs, one for up to 22 queues of length 64 for 2 levels of directed interrupts for up to 11 CPUs, one for 4 queues of length 128 for 4 levels of nondirected interrupts.

Interrupts present counter RAM (8 x 10 RAM keeps a count of the number of interrupts present at each interrupt level). Attached to it is a latch with a priority encoder which indicates the level of the highest priority interrupt present.

Interrupt service request state machine

CPU configuration RAM (16 x 5 RAM initialized by 68020 with all CPUs physically present at initial power up or reset)

Service pending RAM (128 x 1 RAM. Only up to 66 bits are actually used, one for each of 6 CPU levels one for each of up to 11 CPUs).

Interrupt information register (32 bits to be read by CPU in response to an Interrupt acknowlege read request).

Functional Description:

1.) Receive an Interrupt Request.

When an Interrupt request(CSS write to address FFFF FFA0) is received by the SPM, the interrupt information is written to the incoming register. If the incoming register is full, the SPM will be not ready and therefore will not be issued any CSS writes.
The receive logic will look at the CSS priority level, the directed/non-directed bit, and if directed, the CPU number, select the appropriate queue in the main queue RAM, and if the pointer RAM indicates not full, write the interrupt information into the main queue RAM into the proper queue. It will increment the incoming pointers in the pointer RAM and the count in the proper location in the counter RAM. The proper queue in

the main queue RAM is selected by the CSS interrupt priority level (2 bits) and the CPU number (4 bits) and the Directed/Nondirected bit.

2.) Request a CPU Interrupt Service.

The Interrupt service request state machine will cycle thru the CPU numbers given by the CPU configuration RAM. At each CPU number, it will look at the output of the interrupt present indicator. If the highest interrupt present is a CPU level 5 or 6 (a directed interrupt) it will access the pointer RAM to determine if this level 5 or 6 is pending for this CPU number. If yes, it will attempt to issue the service request (see below). If this level 5 or 6 is pending for a different CPU the state machine will disable CPU levels 5 and 6 from the interrupt present indicator priority encoder and determine which is the highest CPU level interrupt present from CPU levels 4 and below. If there is one present, it will attempt to issue the service request.
Attempting the service request means the state machine will access the Service pending RAM with the CPU interrupt level and the CPU number and test the service pending bit. if the bit is already set, an interrupt service is already pending at that CPU level on that CPU and no further service requests can be made. In this case, no service request will be made and the state machine will look at the next CPU number. If the bit is reset, no service request is pending, the bit will be set and the service request will be issued to the given CPU (write to FFFF FFFC with the level as data). The appropriate location in the counter RAM will be written with the decremented number of interrupts present at that level, and the level latch will be written.
After issueing the service request, the state machine will look at the next CPU number.

3.) Acknowledge an interrupt.

When the SPM receives an interrupt acknowledge read request, the acknowledge logic will access the service pending RAM with the acknowledging CPU number and CPU level and reset the service pending bit. Using the CPU number and level information it accesses the main queue RAM, and latches the interrupt information into the interrupt information register ready to send to the acknowledging CPU. It will also increment the outgoing pointers in the pointer RAM.

Arbitration

All three processes will access most of the elements of the dispatch

logic. It is therefore necessary to arbitrate among the three functions of the interrupt dispatcher. At any time, there can only be one of acknowledge response or interrupt request pending, because both of these actions are caused by a CSS bus transaction from another CSS bus board. The SPM will be not ready until the condition has been taken care of. Normally, the CPU service request state machine will always be running, attempting to request service for any interrupts which are present in the SPM. When one of the above functions is requested, the state machine will complete its current CPU service request attempt and then allow the requested function (ack response or interrupt request).

Timing

All timings assume a 50ns bus clock.

To receive an interrupt request into the incoming register takes one bus cycle (50ns). To receive an interrupt request into the main queue RAM takes 5 bus cycles (250ns).

The Service request state machine runs at 1 x bus cycle (50ns/tick). The following are projections based on a preliminary version of the state machine code and are subject to change:
If there are interrupts present at all CPU levels in tne main queue RAM, an interrupt service can be requested every 6 ticks (300ns). This is the maximum rate at which interrupt service can be requested. Since interrupt requests need to be received, and service requests need to be acknowledged, this rate can only be sustained for short bursts. The maximum sustained rate is one interrupt dispatched every 800ns (250ns to receive the request, 300ns to request service, 250ns to respond to the acknowledge)
The longest delay occurs when there is only one directed interrupt present in the main queue RAM and no other interrupts present. It takes 300ns x (n-1) + 300ns max., where n is the number of CPUs in the system, that is 3.3us for an 11 CPU system. It will take 300ns x (n-1/2) + 300ns average, or 1.8us for an 11 CPU system.

The acknowledge response takes 250ns (5 bus cycles).

All these times, if they are taken for a single transaction, must have the typical CSS bus latency added to it (150ns when there is no other bus traffic).

Clock Interrupts

The SPM is the source for system wide clock interrupts. Every 20ms, an interrupt will be requested from each CPU. This will be done by the SPM with a directed interrupt request for the given CPU via a interrupt request write to the SPM itself.

CSS Bus Access:

CSS Bus Commands by other Modules Addressed to the SPM:

For the purpose of recognizing and responding to CSS bus commands, the SPM decodes only three CSS bus address bits, bus.data75, 76, and 77. They are decoded as follows:

bus.data.77	76	75	read/write	recognized
0	0	0	write	Command Register
0	0	1	-	-
0	1	0	-	-
0	1	1	write	Interrupt Request (Dispatcher)
1	0	0	read	Interrupt Acknowledge (Dispatcher)
1	0	1	-	-
1	1	0	-	-
1	1	1	write	Interrupt Vector
1	1	1	read	Status Register (Board ID only)

The Command Register is a general purpose 32 bit register which can be written by any other module. When this register is written by another module, the SPMs 68020 is interrupted. In response to such an interrupt, the 68020 will read the Command register, and force an increment ready count to the CSS bus arbiter.

The Interrupt Request Register is a 32 bit register. It is the input register for the interrupt dispatcher. Any other module may write to this register. When another module writes to the Interrupt Request Register, the hardware sets a bit to indicate to the dispatcher statemachine that an interrupt has been requested. The dispatcher will read the register into the appropriate queue and increment ready for the CSS bus arbiter. The meaning of the bits in the Interrupt Request Register is as follows:

bus.data.00 to 07	Interrupt Vector number
bus.data.10 to 11	Interrupt Priority
bus.data.12 to 16	reserved
bus.data.17	Directed Interrupt. When this bit is on, the interrupt must be sent to the specified destination slot. If this bit is off, the interrupt may be sent to any CSS bus CPU.
bus.data.20 to 23	CSS bus source slot
bus.data.24 to 27	CSS bus destination slot (only used if bus.data.17 = 1, ignored otherwise)
bus.data.30 to 36	I/O bus source slot (only used if bus.data.37 = 1, ignored otherwise)
bus.data.37	I/O bus interrupt. When this bit is a 1,

the interrupt came from an I/O bus module attached to the CSS bus module. If this bit is a 0, the interrupt came from a CSS bus module.

The Interrupt Acknowledge location is a range of addresses which can be read by any other CSS bus module. It is normally used by a CSS bus processing module after it has reveived an interrupt level request to acknowledge this request. Address bits bus.data.72 to 74 are further decoded to determine the interrupt level which is being acknowledged:

bus.data.74	73	72	acknowledge level
0	0	0	-
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	-

When a read to the Interrupt Acknowledge location is received, a bit is set for the dispatcher state machine and it will read the interrupt information from the appropriate queue and respond to the read with this information in the format given above. It will also take other actions to indicate to itself that this interrupt has been acknowledged. If a CSS bus module reads an interrupt acknowledge location without having first received an interrupt level request, this will disrupt the normal function of the interrupt dispatcher. This should be done for diagnostic purposes only.

The Interrupt Vector Register is not a physical register. In normal mode, a CSS write to this location causes a level 3 auto vector. For diagnostic purposes only, this is a feature which allows the SPMs 68020 to test the function of the interrupt dispatcher. (See the description of the diag.frc.int.rec* bit in the Control register 1 for more).

The Status Register allows any other CSS bus module to read the SPMs board id.

CSS Bus Commands Initiated by the SPM:

There are two sources of SPM initiated CSS bus commands, the interrupt dispatcher and the SPMs 68020.

The interrupt dispatcher can initiate CSS writes and CSS read responses. When the dispatcher wishes to request an interrupt level, it will request the CSS bus for a write. The format of this write is as outlined above.

The SPMs CPU can initiate CSS read requests and CSS writes as well as any other CSS command including invalid commands. Normal reads and

writes, e.g. to a memory module, are transparent to the SPMs CPU. It needs only to select the system bus (address bit 31 = 1) and the map RAM will map the high order address bits as descibed above. The low order address bits are used directly. All CSS bus writes are one or 2 or 3 or 4 byte writes. The type field is automatically encoded from the size field of the 68020. All CSS bus reads are 4 or 8 or 16 or 32 byte reads. If none of the 8.byte, 16.byte, or 32.byte read bits are set in the control register, the normal default CSS bus read is a 4 byte read. If one of the control bits is set, the all CSS bus reads intitiated while the bit is set will be of the length given by the bit. For example if the 32.byte.read bit is set, a read by the SPMs 68020 - of necessity 4 bytes - with address bits 2, 3, and 4 = 0, will cause a 32 byte CSS bus read. The next 7 reads by the 68020, if not all of address bits 2, 3, and 4 are 0, are assumed to be part of the 32 byte CSS bus read and will give the data obtained during the 32 byte read response back to the 68020 in the order given by the address bits 2, 3, and 4. The 16.byte and 8.byte CSS reads operate in a similar manner. To do CSS bus accesses other than reads or writes, the SPM can set the CSS type field directly. See the forced type field in control register 0 and the Use.forced.type bit in control register 1. While the Use.forced.type bit is asserted (=0), any CSS bus access will use the forced type as the CSS bus type.

68020 Interrupt Levels

There are 7 interrupt levels on the 68020. The processor on the Service Module uses 6 of them as follows:

Level 1	=	Interrupt Dispatcher Error	Auto vector
Level 2	=	CSS Bus Command Write	Auto vector
Level 3	=	CSS Interrupt received	Auto vector
Level 4	=	Local Timer (CIO)	Programmable vector
Level 5	=	Real World Interface Interrupt	Programmable vector
Level 6	=	Floppy Controller	Auto vector
Level 7	=	CSS Bus Error	Auto vector

LEDs

There are 10 LEDs on the front panel of the Service Module The meaning of the LEDs is as follows:

LED 1) (red) Software LED 0. Udes as fault indicator

LED 2) (green) Software LED 1. Used as go indicator.

LED 3) (red) Software LED 2. Uncomitted.

LED 4) (red) Software LED 3. Uncomitted.

LED 5) (red) 68020 Address Strobe.

LED 6) (red) CSS Bus Active.

LED 7) (red) CSS Clock present.

LED 8) (red) Interrupt present in Dispatcher not yet dispatched

LED 9) (red) Interrupt present in Dispatcher not yet dispatched and interrupt pending on same level for the same Interrupt target (PM).

LED 10) (red) CSS bus Ready.

APPENDIX 1:

Translation table for CSS data bits to local address and data bits:

CSS byte	CSS bit	local byte	local bit
	00		data 24
	01		data 25
	02		data 26
0	03	0	data 27
	04		data 28
	05		data 29
	06		data 30
	07		data 31
	10		data 16
	11		data 17
	12		data 18
1	13	1	data 19
	14		data 20
	15		data 21
	16		data 22
	17		data 23
	20		data 08
	21		data 09
	22		data 10
2	23	2	data 11
	24		data 22
	25		data 23
	26		data 24
	27		data 25
	30		data 00
	31		data 01
	32		data 02
3	33	3	data 03
	34		data 04
	35		data 05
	36		data 06
	37		data 07
	40		addr 24
	41		addr 25

```
        42                      addr 26
4       43              0       addr 27
        44                      addr 28
        45                      addr 29
        46                      addr 30
        47                      addr 31

        50                      addr 16
        51                      addr 17
        52                      addr 18
5       53              1       addr 19
        54                      addr 20
        55                      addr 21
        56                      addr 22
        57                      addr 23

        60                      addr 08
        61                      addr 09
        62                      addr 10
6       63              2       addr 11
        64                      addr 12
        65                      addr 13
        66                      addr 14
        67                      addr 15

        70                      addr 00
        71                      addr 01
        72                      addr 02
7       73              3       addr 03
        74                      addr 04
        75                      addr 05
        76                      addr 06
        77                      addr 07
```

```
/*****************************************************************************
 *
 *    Service Processor Module
 *
 *    The Service Processor Module (SPM) attaches to the system bus and
 *    provides various service functions for the system. These functions
 *    include system console interfaces, front panel interface, real time
 *    clock, system clock interrupt service, power supply controls, floppy
 *    disk controller, system wide interrupt dispatcher, various system
 *    monitor functions (power, temperature), and system diagnostic capabil-
 *    ities.
 *    The SPM consists of two boards, the main board which plugs into the CSS
 *    bus backplane, and the real world interface board which contains all
 *    connectors for the serial ports, the power supply control connectors,
 *    and the temperature sensor connectors. The real world interface board
 *    is connected to the main board via a third 96pin din connector at the
 *    bottom of the main board. The real world interface board resides behind
 *    the CSS backplane in the interface slot behind the Service Processor
 *    Main board.
 *
 *****************************************************************************/

/*****************************************************************************
*                          Service Processor Memory Map
*****************************************************************************
*
* 31302928 27262524 23222120 19181716 15141312 111098 7654 3210 - address bits
*
*-----------------------------------------------------------------------------
* 0 x x x  0 0 0 0  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0000 0000 Eprom
* 0 x x x  0 0 0 0  0 0 0 0  0 0 0 0  1 1 1 1  1 1 11 1111 1111 - 512k <----
* 0 x x x  0 0 0 0  0 0 0 0  0 0 0 1  1 1 1 1  1 1 11 1111 1111 - 1M
*-----------------------------------------------------------------------------
* 0 x x x  0 0 0 0  0 0 0 0  0 0 1 0  0 0 0 0  0 0 00 0000 0000 Spare
* 0 x x x  0 0 0 0  1 1 1 1  1 1 1 1  1 1 1 1  1 1 11 1111 1111
*-----------------------------------------------------------------------------
*    0x01000000
*    0x0103ffff
* 0 x x x  0 0 0 1  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0000 0000 Sram (256k)
* 0 x x x  0 0 0 1  0 0 0 0  0 0 1 1  1 1 1 1  1 1 11 1111 1111
*-----------------------------------------------------------------------------
* 0 x x x  0 0 0 1  0 0 0 0  0 1 0 0  0 0 0 0  0 0 00 0000 0000 Spare
* 0 x x x  0 0 0 1  1 1 1 1  1 1 1 1  1 1 1 1  1 1 11 1111 1111
*-----------------------------------------------------------------------------
* 0 x x x  0 0 1 0  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0000 0000 reserved
* 0 x x x  0 0 1 0  0 0 0 0  0 0 0 0  0 0 0 0  0 0 01 0000 0000 local A/D conv
* 0 x x x  0 0 1 0  0 0 0 0  0 0 0 0  0 0 0 0  0 0 10 0000 0000 local CIO
*-----------------------------------------------------------------------------
* 0 x x x  0 0 1 0  0 0 0 0  0 0 0 0  0 0 0 0  0 0 11 0000 0000 reserved for
* 0 x x x  0 0 1 0  0 0 0 0  0 0 0 0  0 0 0 0  0 1 11 0000 0000 local devices
*-----------------------------------------------------------------------------
* 0 x x x  0 0 1 0  0 0 0 0  0 0 0 0  0 0 0 0  1 0 00 0000 0000 aux. SCC 0
* 0 x x x  0 0 1 0  0 0 0 0  0 0 0 0  0 0 0 0  1 0 01 0000 0000 aux. SCC 1
* 0 x x x  0 0 1 0  0 0 0 0  0 0 0 0  0 0 0 0  1 0 10 0000 0000 aux. CIO
* 0 x x x  0 0 1 0  0 0 0 0  0 0 0 0  0 0 0 0  1 0 11 0000 0000 aux. A/D conv
*-----------------------------------------------------------------------------
```

```
* 0 x x x  0 0 1 0  0 0 0 0  0 0 0 0  0 0 0 0  1 1 00 0000 0000 reserved for
* 0 x x x  0 0 1 0  0 0 0 0  0 0 0 0  0 0 0 0  1 1 11 0000 0000 aux. devices
*-------------------------------------------------------------------------------
* 0 x x x  0 0 1 1  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0000 0000 real time clock
* 0 x x x  0 0 1 1  1 1 1 1  1 1 1 1  1 1 1 1  1 1 11 1111 1111 and NV ram
*-------------------------------------------------------------------------------
* 0 x x x  0 1 0 0  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0000 0000 reserved
* 0 x x x  0 1 0 0  1 1 1 1  1 1 1 1  1 1 1 1  1 1 11 1111 1111
*-------------------------------------------------------------------------------
* 0 x x x  0 1 0 1  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0000 0000 clock control
*-------------------------------------------------------------------------------
* 0 x x x  0 1 0 1  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0000 0001 reserved
* 0 x x x  0 1 0 1  1 1 1 1  1 1 1 1  1 1 1 1  1 1 11 1111 1111
*-------------------------------------------------------------------------------
* 0 x x x  0 1 1 0  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0000 0000 reserved
* 0 x x x  0 1 1 0  1 1 1 1  1 1 1 1  1 1 1 1  1 1 11 1111 1111
*-------------------------------------------------------------------------------
* 0 x x x  0 1 1 1  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0000 0000 Wr cntl 0
* 0 x x x  0 1 1 1  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0000 0000 Wr cntl 0 (read)
* 0 x x x  0 1 1 1  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0010 0000 Wr cntl 1
* 0 x x x  0 1 1 1  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0010 0000 Wr cntl 1 (read)
* 0 x x x  0 1 1 1  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0100 0000 Wr cntl 2
* 0 x x x  0 1 1 1  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0100 0000 Wr cntl 2 (read)
* 0 x x x  0 1 1 1  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0110 0000 Reserved
* 0 x x x  0 1 1 1  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 1000 0000 CSS command reg
* 0 x x x  0 1 1 1  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 1010 0000 Status reg
* 0 x x x  0 1 1 1  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 1100 0000 CSS error reg
* 0 x x x  0 1 1 1  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 1110 0000 Dispatcher error
*                                                                register
*-------------------------------------------------------------------------------
* 0 x x x  0 1 1 1  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 1110 0001 Spare
* 0 x x x  0 1 1 1  1 1 1 1  1 1 1 1  1 1 1 1  1 1 11 1111 1111
*-------------------------------------------------------------------------------
* 0 0 0 0  1 0 0 0  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0000 0000 Map RAM loc 0
* 0 0 0 1  1 0 0 0  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0000 0000  "   "   "  1
* 0 0 1 0  1 0 0 0  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0000 0000  "   "   "  2
* 0 0 1 1  1 0 0 0  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0000 0000  "   "   "  3
* 0 1 0 0  1 0 0 0  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0000 0000  "   "   "  4
* 0 1 0 1  1 0 0 0  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0000 0000  "   "   "  5
* 0 1 1 0  1 0 0 0  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0000 0000  "   "   "  6
* 0 1 1 1  1 0 0 0  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0000 0000  "   "   "  7
*-------------------------------------------------------------------------------
* 0 x x x  1 0 0 1  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0000 0000 Floppy Disk
* 0 x x x  1 0 0 0  1 1 1 1  1 1 1 1  1 1 1 1  1 1 11 1111 1111
*-------------------------------------------------------------------------------
* 0 x x x  1 0 1 0  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0000 0000 reserved
* 0 x x x  1 0 1 0  1 1 1 1  1 1 1 1  1 1 1 1  1 1 11 1111 1111
*-------------------------------------------------------------------------------
* 0 x x x  1 0 1 1  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0000 0000 I.D. Idle Queue
* 0 x x x  1 0 1 1  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0000 1111
*-------------------------------------------------------------------------------
* 0 x x x  1 0 1 1  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0001 0000 unused
* 0 x x x  1 0 1 1  1 1 1 1  1 1 1 1  1 1 1 1  1 1 11 1111 1111
*-------------------------------------------------------------------------------
* 0 x x x  1 1 0 0  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0000 0000 I.D. Queue Ram
* 0 x x x  1 1 0 0  0 0 0 0  0 0 0 0  0 0 1 1  1 1 11 1111 1111
```

```
*-------------------------------------------------------------------------------
* 0 x x x  1 1 0 0  0 0 0 0  0 0 0 0  0 1 0 0  0 0 00 0000 0000 unused
* 0 x x x  1 1 0 0  1 1 1 1  1 1 1 1  1 1 1 1  1 1 11 1111 1111
*-------------------------------------------------------------------------------
* 0 x x x  1 1 0 1  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0000 0000 I.D. Pointer Ram
* 0 x x x  1 1 0 1  1 1 1 1  1 1 1 1  1 1 1 1  1 1 11 1111 1111
*-------------------------------------------------------------------------------
* 0 x x x  1 1 1 0  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0000 0000 Int Dispatcher
* 0 x x x  1 1 1 0  1 1 1 1  1 1 1 1  1 1 1 1  1 1 11 1111 1111 Misc. Rams
*-------------------------------------------------------------------------------
* 0 x x x  1 1 1 1  0 0 0 0  0 0 0 0  0 0 0 0  0 0 00 0000 0000 reserved
* 0 x x x  1 1 1 1  1 1 1 1  1 1 1 1  1 1 1 1  1 1 11 1111 1111
*-------------------------------------------------------------------------------

/*****************************************************************************
 *
 *    EPROM
 *
 *    The EPROM is 8 bits wide and can be read by the processor on data bits
 *    24 to 31. There are 2 sockets for the EPROM, one or both can be populat-
 *    ed with either 27256s (32k x 8 each), or 27512s (64k x 8 each) or
 *    271001s (128k x 8 each). Address bits Add.00 to Add.15 or Add.16 or
 *    Add.17 are used to address the EPROM. The decoding of the high bit is
 *    controlled by the Promsize field in control register 2. This allows the
 *    two EPROM chips to be contiguous regardless of their size. (See Control
 *    register 2 description for details).
 *
 *****************************************************************************/
#define PROMSIZ    0x10000          /* cpu monitor prom*/
#define PROMSTART  0x00000000       /* start of prom area */

/*****************************************************************************
 *
 *    Static Ram
 *
 *    The SRAM is 32 bits wide and consists of 8 or 4 32k x 8 static RAM chips
 *    for 64k x 32 or 32k x 32 bits of memory. It is addressed with address
 *    bits Add.02 to Add.17.
 *
 *****************************************************************************/
#define SRAMSTART 0x01000000
#define SRAMSIZE  0x40000

/* NOVRAM...
*/

#define NOVRAM  ((struct novram *)(0x03000000))

/*****************************************************************************
 *
 *   Real Time Clock
 *
 *   The real time clock is a MK48T12 chip. This chip has a built in crystal
 *   oscillator and a lithium battery. In addition to a real time clock, it
 *   has 2K bytes of nonvolatile RAM. This RAM can be used to keep config-
 *   uration information. The service processor does not have any switches.
```

```
 *
 ******************************************************************************/

#define RTC ((struct rtc *)(0x030007f8))

/*****************************************************************************
 *
 *    Local A/D converter
 *
 *    The local A/D converter is an MC14442 and a TL431A voltage reference.
 *    It is used to measure the temperature on the Service Processor (i.e. the
 *    cardcage) and the 6 voltages present on the SPM(i.e. +5V main, +5V aux.,
 *    +12V main, +12V aux., -12V main, and -12V aux.)
 *
 ******************************************************************************/

#define ADC_CNTL ((unsigned short*)(0x02000102))
#define ADC_ADATA ((unsigned short*)(0x02000100))
#define ADC_SC    0x0100
#define ADC_CH0   0x0000        /* measures +5v for master system */
#define ADC_CH1   0x0001        /* reference voltage 4.5v must read FF */
#define ADC_CH2   0x0002        /* Measure +5v. aux */
#define ADC_CH3   0x0003        /* Measure +12v */
#define ADC_CH4   0x0004        /* Measure +12v aux */
#define ADC_CH5   0x0005        /* Measure -12v */
#define ADC_CH6   0x000e        /* Measure -12v aux */
#define ADC_CH7   0x000f        /* measure on board temperature. */
#define ADC_EOC   0x8000
#define ADC_MASK  0x00ff        /* mask for valid data. */


/*****************************************************************************
 *
 *    Local CIO
 *
 *    The local CIO is a Zilog 8036, used for various timing functions includ-
 *    ing the CSS bus timeout and the main system clock interrupt.
 *
 ******************************************************************************/

#define LOCCIO ((struct cio *)(0x02000200))

/*****************************************************************************
 *
 *    SCC
 *
 ******************************************************************************/
#define ASCC ((struct ascc *)(0x02000800))
#define AUXASCC0B ((struct ascc *)(0x02000800))
#define AUXASCC0A ((struct ascc *)(0x02000820))
#define AUXASCC1B ((struct ascc *)(0x02000900))
#define AUXASCC1A ((struct ascc *)(0x02000920))

/*****************************************************************************
 *
 *    Clock Control (write only)
```

```
 *
 *    The Clock control port is a single bit port which is written with data
 *    bit 0. Data.00 = 0 means the CPU is using its on-board 10Mhz clock,
 *    Data.00 = 1 means the CPU is using the CSS bus clock divided by two as
 *    its clock source. When switching from one clock source to the other,
 *    approximately 1us after the write to the clock control port, the hard-
 *    ware will issue a reset to the CPU, hold the reset for approximately
 *    500us, and switch clocks during the reset. This is required to meet the
 *    timing specifications of the 68020. In the process, the entire board
 *    will be reset, so any VLSI chips will have to be reinitialized. The
 *    memory, however, will keep its data valid. The state of the clock bit
 *    (i.e. which clock is being used) can be read in the status register. If
 *    the CSS bus clock is not running(e.g. the CSS bus is powered down), the
 *    hardware will not select the CSS clock.
 *
 *      31 ---------- 1 0
 *     +---------------+-+
 *     |       x       |c|
 *     +---------------+-+
 *             |         |
 *             |         +-------------- 0 - on board 10Mhz clock
 *             |                         1 - CSS bus clock
 *             +------------------------ unused
 ************************************************************************/

#define CLKCNTL  ((unsigned *)(0x05000000))

/************************************************************************
 *
 *   CSS Command Register (read only)
 *
 *   The CSS command register is a 32 bit register which allows the local CPU
 *   to read the value written to the SPM via a CSS write to addr 0xXXXXAAYY
 *   by any other CSS module. X's are don't cares and AA will be captured by
 *   the SPM and can be read in the Status register. YY must be of the form
 *   000X XXXX. The SPM only decodes the top three bits of the least signif-
 *   icant byte of CSS address. When another module writes to an address of
 *   the form described above on the SPM, the 68020 recieves an interrupt
 *   informing it that a CSS command has been received. The 68020 will read
 *   the CSS command register and the status register to determine the data
 *   of the command.
 *
 *      31 ---------- 0
 *     +---------------+
 *     |   command     |
 *     +---------------+
 *             |
 *             +----------------------- command to SPM via CSS write
 *
 ************************************************************************/

#define CSSCMD ((unsigned *)(0x07000080))

/************************************************************************
 *
```

```
*     Status Register (read only)
*
*     The status register is a 24 bit register, data bits 00 to 07 and 16 to
*     23 and 24 to 31 can be read by the 68020. Bits 24 to 27 are the bus slot
*     number given by the position of the SPM in the backplane. For diagnostic
*     purposes, the SPM does not have its slot id hardwired, so for the board
*     to function properly, the slot id must be read in the status register
*     and written in write control registers 0 and 1. Bit 31 in the Status
*     register is a status bit from the interrupt dispatcher state machine.
*     When this bit is 0, the interrupt dispatcher has suspended its dispatch-
*     ing function and the CPU is allowed to access the various RAMs in the
*     interrupt dispatcher. This is needed for initializing the interrupt
*     dispatcher and for diagnostic purposes. When the bit is a 1, the
*     interrupt dispatcher is in its normal operating mode and the CPU must
*     not access any of the dispatcher RAMs. Bit 30 is an indicator from the
*     floppy disk, floppy ready. Bit 29 is a status bit from the clock select
*     mechanism. When this bit is a 1, the 68020 is using the local clock,
*     when 0, the 68020 is using the CSS clock divided by 2. Bit 28 is reserv-
*     ed. Bits 00 to 07 are part of the CSS error register as follows:
*     Bit 00 has the state of the CSS bus nack signal at the time of the last
*     CSS bus error, bit 01 has the state of the CSS bus ack, bit 02 has the
*     state of the CSS arbiter grant error, bit 03 is the bad command bit.
*     This bit is a 0 if the SPM has detected a CSS bus protocol violation
*     (e.g. data parity is bad), it is a 1 if the SPM has not detected such
*     a violation but it has nevertheless detected a CSS bus error. Bit 04
*     is the state of the CSS arbiter burst signal. Bits 05 to 07 are reserv-
*     ed. Bits 16 to 23 are the CSS data byte 6 at the time of the most recent
*     CSS command write.
*
*       31  30  29 28 27 - 24 23 - 16
*      +--+----+--+--+-------+-------+
*      |ID|FRDY| c| x|slot # | byte 6|
*      +--+----+--+--+-------+-------+
*        |    |   |  |       |
*        |    |   |  |       +-----  CSS data byte 6 at the time of the most
*        |    |   |  |               recent CSS command write.
*        |    |   |  +-------------  bus slot # determined by position of
*        |    |   |                  SPM in backplane
*        |    |   +----------------  reserved
*        |    +--------------------  1 = local clock
*        |                           0 = CSS clock
*        +------------------------  floppy ready
*         +---------------------------  from Interrupt Dispatcher state machine
*                                       0 - dispatcher has been suspended and
*                                           the cpu is allowed access to ID rams
*                                       1 - Int Dispatcher normal mode, cpu
*                                         should not access dispatcher RAM's
*
*       15-8 7-6     5       4      3      2    1    0
*      +----+---+--------+------+------+-----+----+----+
*      |  x |   | active |burst*|bad cmd|gerr*|back|nack|
*      +----+---+--------+------+------+-----+----+----+
*        |    |     |       |      |      |     |   |
*        |    |     |       |      |      |     |   +-state of CSS bus nack
*        |    |     |       |      |      |     |     signal at the last
*        |    |     |       |      |      |     |     CSS bus error
```

```
 *        |    |      |        |       |      +-- state of CSS bus ack
 *        |    |      |        |       |     +--------- state of CSS arbiter
 *        |    |      |        |       |                  grant error
 *        |    |      |        |      +---------------- bad command bit
 *        |    |      |        |                    0 - SPM has detected a CSS bus
 *        |    |      |        |                        protocol violation(data
 *        |    |      |        |                        parity is bad)
 *        |    |      |        |                    1 - SPM has not detected a
 *        |    |      |        |                        violation, but detected a
 *        |    |      |        |                        CSS bus error
 *        |    |      |       +----------------------- state of CSS arbiter
 *        |    |      |                                  burst signal
 *        |    |     +------------------------------- always a 1 except during
 *        |    |                                       force ack,nack,grant errors
 *        |    +----------------------------------- reserved
 *       +-------------------------------------- unused
 *
 ***************************************************************************/

#define STATUSREG ((unsigned *)(0x070000a0))
#define STAT_SLOTMASK  0x0f000000
#define STAT_SLOTSH    24
#define STAT_REG_IDRAM 0x80000000
#define STAT_IDLE_FULL 0x10000000    /* if set, idle-que is not full. */

/***************************************************************************
 *
 *   CSS Error Register (read only)
 *
 *   When the SPM detects an error on the CSS bus the hardware will interrupt
 *   the 68020. The 68020 will read the CSS error information register to
 *   determine the nature of the error. The conditions which cause such an
 *   error are: bad data parity, bad type parity, destination error, source
 *   error, invalid type, bus nack, or grant error. All error conditions
 *   except the last two will latch in valid data for the CSS error informa-
 *   tion register as shown below. Additional error information is captured
 *   in the status register. When a bus error condition has been latched in
 *   the error register no new error conditions can be latched until after
 *   the bus error register has been read by the 68020.
 *
 *   31 - 27 26-25  24  23-20 19-16  15   14   13-8  7-0
 * +--------+-----+----+-----+-----+----+-----+----+----+
 * |CSS data|  x  |bpar|dest | src |derr|serr|btype|bpar|
 * +--------+-----+----+-----+-----+----+-----+----+----+
 *     |        |    |     |     |    |     |    |
 *     |        |    |     |     |    |     |    +- bus data parity
 *     |        |    |     |     |    |     |       0 - 7
 *     |        |    |     |     |    |     +------ bus type 0 - 5
 *     |        |    |     |     |    +------------ source error
 *     |        |    |     |     +----------------- destination error
 *     |        |    |     +----------------------- bus source 0 to 3
 *     |        |    +------------------------------ bus dest 0 to 3
 *     |        +----------------------------------- bus type parity
 *     +-------------------------------------------- reserved
 *                                                  CSS data bits 57,40-43
 *
```

```
 ***************************************************************************/

#define CSSERROR    ((unsigned *)(0x070000c0))
#define DERR        15
#define SERR        14
#define BTYPE_SH    8
#define SRC_SH      16
#define DEST_SH     20
#define BPAR_SH     24
#define BPAR_MASK   0x01000000
#define DEST_MASK   0x00f00000
#define SRC_MASK    0x000f0000
#define BTYPE_MASK  0x00003f00
#define TYPE_SH     0x0b
#define TYPE_MASK   0x3800
#define SIZE_MASK   0x0700
#define SIZE_SH     0x08

/***************************************************************************
 *  Dispatcher Error Register
 *
 *  When the Interrupt Dispatcher detects an error it will suspend its
 *  operation and latch error information in this register.
 *
 *   31 - 24  23  22-20 19-18 17-16 15-12 11-8   7  6-0
 * +--------+----+-----+-----+-----+-----+-----+---+---+
 * | Int Vec|dint| err |  x  | pri |dest | src |int|src|
 * +--------+----+-----+-----+-----+-----+-----+---+---+
 *      |        |     |     |     |     |     |   |
 *      |        |     |     |     |     |     |   +-- i/o bus source slot
 *      |        |     |     |     |     |     |       (only valid if bit 7=1)
 *      |        |     |     |     |     |     +------ i/o bus int
 *      |        |     |     |     |     |             1 - int came from an
 *      |        |     |     |     |     |                 i/o module
 *      |        |     |     |     |     |             0 - int came from a CSS
 *      |        |     |     |     |     |                 bus module
 *      |        |     |     |     |     +------------ CSS bus source slot
 *      |        |     |     |     +------------------ CSS bus dest slot
 *      |        |     |     +------------------------ Interrupt priority
 *      |        |     +---------------------------- reserved
 *      |        +-------------------------------- error code
 *      |                                          (see below for codes)
 *      +---------------------------------------- Directed interrupt
 *  +------------------------------------------- Int vector number
 *
 *
 *      error information bits 22 21 20
 *                             -- -- --
 *                              x  x  0 - request error
 *                              1  0  x - ack error, bus error on ack response
 *                              0  0  x - ack error, no previous requests
 *                              x  1  1 - recieve error, fix pointers or reset
 *                                        recieve
 *
 ***************************************************************************/
```

```
#define DISPERROR     ((unsigned *)(0x070000e0))
#define DISP_ACK_ERR        1         /* insane ack, an ack for no good reason */
#define DISP_REQ_ERR        2         /* request with no int. pending */
/*      unused              3         /* recieve err  */
/*      unused              4         /* */
#define DISP_ACK_TOUT       5         /* timeout on ack response */
#define DISP_REQ_TOUT       6         /* timeout on request */
#define DISP_RCV_ERR        7
#define DISP_ERR_SH         20

/*****************************************************************************
 *
 *        Write Control Register 0 (write only)
 *
 *   31-29  28    27    26    25    24   23-20  19-16
 *  +-----+-----+-----+-----+-----+-----+------+-----+
 *  |     |inh7 |freez| dest |grant| ack |srcID*|srcID|
 *  +-----+-----+-----+------+-----+-----+------+-----+
 *     |     |     |     |     |     |      |     |
 *     |     |     |     |     |     |      |     +------ out.src 0-3 source id
 *     |     |     |     |     |     |      |             used by SPM when it is
 *     |     |     |     |     |     |      |             issuing a command
 *     |     |     |     |     |     |      +------------ out.src 0 to 3*
 *     |     |     |     |     |     |                   complement of source id
 *     |     |     |     |     |     +-------------------- force ack/nak error.
 *     |     |     |     |     |
 *     |     |     |     |     +-------------------------- force grant error.
 *     |     |     |     |
 *     |     |     |     +-------------------------------- force bad dest.
 *     |     |     |
 *     |     |     +---------------------------------------- Freeze CPU bit:
 *     |     |                                               active 1 (Must have
 *     |     |                                               CSS Reset rdy 1)
 *     |     +-------------------------------------------- Inhibit level 7
 *     |                                                     CSS int's.
 *     |
 *     +-----------------------------------------
 *
 *
 *
 *     15-12  11-8  7-6  5-0
 *    +------+------+---+-----+
 *    |slotID|slotID|f10|Ftype|
 *    +------+------+---+-----+
 *       |      |     |    |
 *       |      |     |    +---- forced type
 *       |      |     |          (used if "use.forced.type" bit is active)
 *       |      |     +--------- forced address bits 00 & 01
 *       |      +--------------- slot id used by int dispatcher
 *       |                       (tells SPM where to place pointers for
 *       |                       non-dir ints 0-3)
 *       +--------------------- slot id used with CSS backplane
 *                               (to pretend to be a CPU)
 *
 *****************************************************************************/
```

```
#define WRCNTL0 ((unsigned *)(0x07000000))
#define WR0_FTYPE_MASK    0x0000003f
#define WR0_FADD01_MASK   0x000000c0
#define WR0_FADD01_SH     0x06
#define WR0_PRETEND_MASK  0x0000f000
#define WR0_PRETEND_SH    12
#define WR0_FRC_ACK       0x01000000
#define WR0_FRC_GRANT     0x02000000
#define WR0_FRC_DEST      0x04000000
#define WR0_FRC_FREE      0x08000000
#define WR0_FRC_INH7      0x10000000

/*****************************************************************************
 *
 *        Write Control Register 1 (write only)
 *
 *   31  30  29   28    27   26   25   24
 * +---+---+----+----+-----+----+----+----+
 * |ack|rdy|rrdy|fmod|fdest|diag|diag|dreq|
 * +---+---+----+----+-----+----+----+----+
 *   |   |   |    |     |    |    |    |
 *   |   |   |    |     |    |    |    +-- cpu.disp.req*
 *   |   |   |    |     |    |    |        0 - indicates to int dispatcher that
 *   |   |   |    |     |    |    |        68020 wants to access its rams.
 *   |   |   |    |     |    |    |        1 - int dispatcher resumes normally.
 *   |   |   |    |     |    |    +------- diag.frc.int.rec*
 *   |   |   |    |     |    |             0 - decode a CSS cpu int request
 *   |   |   |    |     |    |                 level as a CSS command.
 *   |   |   |    |     |    |             1 - normal mode.
 *   |   |   |    |     |    +------------ diag.any.type*
 *   |   |   |    |     |                  0 - accept any CSS bus action as a
 *   |   |   |    |     |                      valid response to a SPM
 *   |   |   |    |     |                      initiated read request.
 *   |   |   |    |     |                  1 - normal mode.
 *   |   |   |    |     +------------------ use.fake.dest*
 *   |   |   |    |                        0 - destination at arbiter is forced
 *   |   |   |    |                            to be bits 8 to 11 of cntl reg 1
 *   |   |   |    |                        1 - normal - dest at arbiter is the
 *   |   |   |    |                            same as the dest which will be
 *   |   |   |    |                            used on the CSS bus with the
 *   |   |   |    |                            requested CSS bus transaction.
 *   |   |   |    +---------------------- force.modify*
 *   |   |   |                             0 - forces a modify cycle on the CSS
 *   |   |   |                                 bus as long as the bit is 0
 *   |   |   |                             1 - normal mode.
 *   |   |   +------------------------------ cpu.res.ready*
 *   |   |                                 1 to 0 resets the ready count in the
 *   |   |                                   arbiter for the SPM.
 *   |   |                                   Normally a 1.
 *   |   +-------------------------------- cpu.inc.ready*
 *   |                                     1 to 0 increments the ready count
 *   |                                       in the arbiter for the SPM.
 *   |                                       Normally a 1.
 *   +------------------------------------ frc.reset.ack*
 *                                         use to reset ACK request during ACK
 *                                         error interrupt service.
```

```
*                                                Normally a 1.
*
*   23  22  21  20  19   18   17  16
* +---+---+---+---+----+-----+----+----+
* |rst|rst|rst|pol|type|idrst| m1 | m0 |
* +---+---+---+---+----+-----+----+----+
*   |   |   |   |    |     |     |    |
*   |   |   |   |    |     |     |    +------------ m_byte0.
*   |   |   |   |    |     |     +-------- m_byte1.
*   |   |   |   |    |     |                  0        1    8 byte read
*   |   |   |   |    |     |                  1        0   16 byte read
*   |   |   |   |    |     |                  1        1   32 byte read
*   |   |   |   |    |     |                  0        0    4 byte read (default)
*   |   |   |   |    |     +------------ int.disp.res*
*   |   |   |   |    |                   interrupt dispatcher reset(active low)
*   |   |   |   |    |                   if above 3 bits are 0 CSS bus reads are
*   |   |   |   |    |                   4 bytes. Normally a 1.
*   |   |   |   |    +----------------- Use.forced.type*
*   |   |   |   |                        0 - forced type from cntl reg 0 (0-5)
*   |   |   |   |                            is used as the bus type.
*   |   |   |   |                        1 - hardware will generate the type.
*   |   |   |   |                            Normally a 1.
*   |   |   |   +---------------------- Type.polarity
*   |   |   |                            sets the polarity of the type field
*   |   |   |                            for forcing errors on the type.
*   |   |   |                            Normally a 1.
*   |   |   +------------------------- force.reset.recieve*
*   |   |                                resets the recieved flag to the int
*   |   |                                dispatcher (active low)
*   |   |                                Normally a 1.
*   |   +------------------------------ Idle.reset*
*   |                                    reset for the Idle FIFO (0 then 1)
*   |                                    Normally a 1.
*   +--------------------------------- CSS.reset
*                                        0 - normal.
*                                        1 - holds reset line to 0 on
*                                            CSS backpllane.
*
*    15    14     13   12 11-8  7-0
* +-----+-----+-------+--+----+-----+
* |motor|fdack|frespon|tp|dest| pol |
* +-----+-----+-------+--+----+-----+
*    |     |       |    |    |    |
*    |     |       |    |    |    +---- out.data.parity 0 to 7
*    |     |       |    |    |          sets the polarity of the parity bits
*    |     |       |    |    |          for each byte of the CSS data bytes.
*    |     |       |    |    |          0-even, 1-odd
*    |     |       |    |    +--------- fake.arb.dest 0 to 3
*    |     |       |    |               these bits are presented to the arbiter
*    |     |       |    |               as the intended destination of a req'd
*    |     |       |    |               command if the fake dest bit is 0.
*    |     |       |    |               Default to our own slot id.
*    |     |       |    +------------- terminal count - used by the floppy
*    |     |       |                    controller.
*    |     |       +------------------ force.response*
*    |     |                            force the response signal at the arbiter
```

```
 *     |      |                              (active low)
 *     |      +---------------------------- fdack* used by floppy controller
 *     +------------------------------------ floppy.motor*
 *                                           motor on or off *
 *
 ********************************************************************************/

#define WRCNTL1 ((unsigned *)(0x07000020))
#define WR1_DIAG_ANY_TYPE    0x04000000

#define WR1_FRC_RST_ACK      0x80000000
#define WR1_IRDY             0x40000000
#define WR1_RRDY             0x20000000
#define WR1_FMOD             0x10000000
#define WR1_FDEST            0x08000000
#define WR1_DATYPE           0x04000000
#define WR1_FINTREC          0x02000000
#define WR1_DREQ             0x01000000
#define WR1_CSSRST           0x00800000
#define WR1_IDLERST          0x00400000
#define WR1_FRC_RST_RCV      0x00200000
#define WR1_RSTREC           0x00200000
#define WR1_TPOL             0x00100000
#define WR1_UFTYPE           0x00080000
#define WR1_ID_RST           0x00040000
#define WR1_CSSRD_MASK       0x00030000
#define WR1_32BYTE_RD        0x00030000
#define WR1_RD32             0x00030000
#define WR1_16BYTE_RD        0x00020000
#define WR1_RD16             0x00020000
#define WR1_8BYTE_RD         0x00010000
#define WR1_RD8              0x00010000
#define WR1_4BYTE_RD         0x00000000
#define WR1_MOTOR            0x00008000
#define WR1_FDACK            0x00004000
#define WR1_FRES             0x00002000
#define WR1_TP               0x00001000
#define WR1_DEST_SH          8
#define WR1_POL              0x000000ff

/********************************************************************************
 *
 *        Write Control Register 2 (write only)
 *
 *   31  30   29   28   27  26-25 24
 * +---+----+----+----+----+----+----+
 * |rst|led0|led1| grn| red|size|prom|
 * +---+----+----+----+----+----+----+
 *   |    |    |    |    |    |    |
 *   |    |    |    |    |    |    +------ prom.at.1
 *   |    |    |    |    |    |             0 - eprom at 0x00000000
 *   |    |    |    |    |    |             1 - eprom at 0x01000000
 *   |    |    |    |    |    +----------- promsiz.0 and promsiz.1
 *   |    |    |    |    |                 prom size 00 - 27256 (32k x 8)
 *   |    |    |    |    |                           01 - 27512 (64k x 8)
 *   |    |    |    |    |                           10 - 27010 (128k x 8)
```

```
 *    |     |     |     |     |                              11 - reserved
 *    |     |     |     |     |
 *    |     |     |     |     +----------------- red fault* led (top of board)
 *    |     |     |     +----------------------- green led* (second from top)
 *    |     |     +------------------------------ red led 1* (third from top)
 *    |     +-------------------------------------- red led 0* (fourth from top)
 *    +-------------------------------------------- bd.reset*
 *                                                 board reset when 0, cpu must set to
 *                                                 1 on power up
 *
 ***************************************************************************/

#define WRCNTL2 ((unsigned char *)(0x07000040))

#define WR2_BDRST 0x80
#define WR2_LED0 0x40
#define WR2_LED1 0x20
#define WR2_GRNLED 0x10
#define WR2_REDLED 0x08
#define WR2_LED01_SH 0x05

/***************************************************************************
 *
 *     Map Ram
 *
 *     When the SPM is performing a CSS access, add.31 = 1, the map ram is
 *     read. CPU address bits 28 to 30 select one of eight locations in the
 *     map ram. The top 4 bits of the map ram output give the destination
 *     (i.e. the slot id of the board being accessed) and the bottom four bits
 *     are mapped to address bits bus_data_44 to bus_data_47 for the CSS bus.
 *     The lower 28 bits of the 68020s address are directly mapped to address
 *     bits bus_data_40 to 43 and bus_data_50 to 77. Before accessing the CSS
 *     bus, the 68020 must initialize the map ram.
 *
 *   31  30   29   28   27   26   25   24
 * +---+----+----+----+----+----+----+----+
 * | destination slot |  CSS add 44 - 47  |
 * +---+----+----+----+----+----+----+----+
 *
 ***************************************************************************/

#define MAPBASE (unsigned char *)0x08000000
#define MAP0    (unsigned char *)0x08000000
#define MAP1    (unsigned char *)0x18000000
#define MAP2    (unsigned char *)0x28000000
#define MAP3    (unsigned char *)0x38000000
#define MAP4    (unsigned char *)0x48000000
#define MAP5    (unsigned char *)0x58000000
#define MAP6    (unsigned char *)0x68000000
#define MAP7    (unsigned char *)0x78000000

/***************************************************************************
 *
 *    Interrupt Idle Queue
 *
 *    The queue is a 16 character fifo, where the programmed character is an
```

```
 *    "idle" cpu board.  This should cuase the next non-directed interrupt to
 *    be passed off to this cpu, rather than the next cpu in the main queue.
 *
 *   31  30   29   28   27   26   25   24    23 - 0
 * +---+----+----+----+----+----+----+----+----------+
 * |  Not used       |     idle cpu      |  Not used |
 * +---+----+----+----+----+----+----+----+----------+
 *
 ************************************************************************/

#define IDLEREG     (unsigned int *)0x0b000000
#define IDLESHFT    24

/************************************************************************
 *
 *      Interrupt Dispatcher Queue
 *
 *      The main queue is a 4k x 32 bit RAM which contains all the information
 *      attached to an interrupt as it is defined on the CSS bus. It is
 *      recommended that it is initialized with a known value for diagnostic
 *      reasons. Below is a map describing where interrupt data is placed in
 *      the Queue Ram.
 *
 *        loc                                  interrupt type
 *        ---                                  --------------
 *
 *      +------------+-------------------------------------------------+ ----
 *      | 0x0c000000 |                  non-directed level 0           | 128
 *      | 0x0c0001fc |                                                 | ints
 *      +------------+-------------------------------------------------+ ----
 *      | 0x0c000200 |                  reserved                       |
 *      | 0x0c0007fc |                                                 |
 *      +------------+-------------------------------------------------+
 *      | 0x0c000800 |                  non-directed level 1           |
 *      | 0x0c0009fc |                                                 |
 *      +------------+-------------------------------------------------+
 *      | 0x0c000a00 |                  reserved                       |
 *      | 0x0c000ffc |                                                 |
 *      +------------+-------------------------------------------------+
 *      | 0x0c001000 |                  non-directed level 2           |
 *      | 0x0c0011fc |                                                 |
 *      +------------+-------------------------------------------------+
 *      | 0x0c001200 |                  reserved                       |
 *      | 0x0c0017fc |                                                 |
 *      +------------+-------------------------------------------------+
 *      | 0x0c001800 |                  non-directed level 3           |
 *      | 0x0c0019fc |                                                 |
 *      +------------+-------------------------------------------------+
 *      | 0x0c001a00 |                  reserved                       |
 *      | 0x0c001ffc |                                                 |
 *      +------------+-------------------------------------------------+ ----
 *      | 0x0c002000 |                  directed level 4, CPU 0        | 64
 *      | 0x0c0020fc |                                                 | ints
 *      +------------+-------------------------------------------------+ ----
 *      | 0x0c002100 |                  directed level 4, CPU 1        |
 *      | 0x0c0021fc |                                                 |
```

```
*       +------------+-------------------------------------------------+
*       | 0x0c002200 |                          directed level 4, CPU 2 |
*       | 0x0c0022fc |                                                 |
*       +------------+-------------------------------------------------+
*       | 0x0c002300 |                          directed level 4, CPU 3 |
*       | 0x0c0023fc |                                                 |
*       +------------+-------------------------------------------------+
*       | 0x0c002400 |                          directed level 4, CPU 4 |
*       | 0x0c0024fc |                                                 |
*       +------------+-------------------------------------------------+
*       | 0x0c002500 |                          directed level 4, CPU 5 |
*       | 0x0c0025fc |                                                 |
*       +------------+-------------------------------------------------+
*       | 0x0c002600 |                          directed level 4, CPU 6 |
*       | 0x0c0026fc |                                                 |
*       +------------+-------------------------------------------------+
*       | 0x0c002700 |                          directed level 4, CPU 7 |
*       | 0x0c0027fc |                                                 |
*       +------------+-------------------------------------------------+
*       | 0x0c002800 |                          directed level 4, CPU 8 |
*       | 0x0c0028fc |                                                 |
*       +------------+-------------------------------------------------+
*       | 0x0c002900 |                          directed level 4, CPU 9 |
*       | 0x0c0029fc |                                                 |
*       +------------+-------------------------------------------------+
*       | 0x0c002a00 |                          directed level 4, CPU A |
*       | 0x0c002afc |                                                 |
*       +------------+-------------------------------------------------+
*       | 0x0c002b00 |                          directed level 4, CPU B |
*       | 0x0c002bfc |                                                 |
*       +------------+-------------------------------------------------+
*       | 0x0c002c00 |                          directed level 4, CPU C |
*       | 0x0c002cfc |                                                 |
*       +------------+-------------------------------------------------+
*       | 0x0c002d00 |                          directed level 4, CPU D |
*       | 0x0c002dfc |                                                 |
*       +------------+-------------------------------------------------+
*       | 0x0c002e00 |                          directed level 4, CPU E |
*       | 0x0c002efc |                                                 |
*       +------------+-------------------------------------------------+
*       | 0x0c002f00 |                          directed level 4, CPU f |
*       | 0x0c002ffc |                                                 |
*       +------------+-------------------------------------------------+
*       | 0x0c003000 |                          directed level 5, CPU 0 |
*       | 0x0c0030fc |                                                 |
*       +------------+-------------------------------------------------+
*       | 0x0c003100 |                          directed level 5, CPU 1 |
*       | 0x0c0031fc |                                                 |
*       +------------+-------------------------------------------------+
*       | 0x0c003200 |                          directed level 5, CPU 2 |
*       | 0x0c0032fc |                                                 |
*       +------------+-------------------------------------------------+
*       | 0x0c003300 |                          directed level 5, CPU 3 |
*       | 0x0c0033fc |                                                 |
*       +------------+-------------------------------------------------+
*       | 0x0c003400 |                          directed level 5, CPU 4 |
```

```
 *      | 0x0c0034fc |                                                  |
 *      +------------+--------------------------------------------------+
 *      | 0x0c003500 |                          directed level 5, CPU 5 |
 *      | 0x0c0035fc |                                                  |
 *      +------------+--------------------------------------------------+
 *      | 0x0c003600 |                          directed level 5, CPU 6 |
 *      | 0x0c0036fc |                                                  |
 *      +------------+--------------------------------------------------+
 *      | 0x0c003700 |                          directed level 5, CPU 7 |
 *      | 0x0c0037fc |                                                  |
 *      +------------+--------------------------------------------------+
 *      | 0x0c003800 |                          directed level 5, CPU 8 |
 *      | 0x0c0038fc |                                                  |
 *      +------------+--------------------------------------------------+
 *      | 0x0c003900 |                          directed level 5, CPU 9 |
 *      | 0x0c0039fc |                                                  |
 *      +------------+--------------------------------------------------+
 *      | 0x0c003a00 |                          directed level 5, CPU A |
 *      | 0x0c003afc |                                                  |
 *      +------------+--------------------------------------------------+
 *      | 0x0c003b00 |                          directed level 5, CPU B |
 *      | 0x0c003bfc |                                                  |
 *      +------------+--------------------------------------------------+
 *      | 0x0c003c00 |                          directed level 5, CPU C |
 *      | 0x0c003cfc |                                                  |
 *      +------------+--------------------------------------------------+
 *      | 0x0c003d00 |                          directed level 5, CPU D |
 *      | 0x0c003dfc |                                                  |
 *      +------------+--------------------------------------------------+
 *      | 0x0c003e00 |                          directed level 5, CPU E |
 *      | 0x0c003efc |                                                  |
 *      +------------+--------------------------------------------------+
 *      | 0x0c003f00 |                          directed level 5, CPU f |
 *      | 0x0c003ffc |                                                  |
 *      +------------+--------------------------------------------------+
 *
 *
 ************************************************************************/

#define QUEUE_RAM (unsigned *)(0x0c000000)
#define QUEUE_LEN      0x4000
#define QUEUE_LWORDS   0x1000
#define VECTOR_MASK    0xff000000
#define VECTOR_SH      24
#define Q_DEST_MASK    0x0000f000
#define Q_DEST_SH      12
#define Q_SRC_MASK     0x00000f00
#define Q_SRC_SH       8

/************************************************************************
 *
 *      Interrupt Dispatcher Pointer
 *
 *      There are two pointer RAM's, each 256 x 7 bits, called the incoming
 *      pointer RAM and the outgoing pointer RAM. These pointer RAM's allow the
 *      main queue to function as multiple FIFOs. The incoming and outgoing
```

```
*       pointer associated with a particular long word in the Pointer RAM will
*       either point to a queue for directed level 4 or 5 interrupts for a CPU
*       in slot x, or if a SPM is in slot x, the pointers are used to maintain
*       interrupt info for non-directed level 0 to 3 interrupts.
*
*          31  30 29 28 27 26 25 24 23  22 21 20 19 18 17 16  15 - 0
*         +---+---------------------+---+---------------------+------+
*         | x |  incoming pointer   | x |   outgoing pointer  |   x  |
*         +---+---------------------+---+---------------------+------+
*
*
*             loc                          pointer type
*             ---                          ------------
*         +------------+-------------------------------------------------+
*         | 0x0d000000 |                 directed level 4, CPU 0         |
*         |            |                 or non-directed level 0         |
*         +------------+-------------------------------------------------+
*         | 0x0d000004 |                 directed level 4, CPU 1         |
*         |            |                 or non-directed level 0         |
*         +------------+-------------------------------------------------+
*         | 0x0d000008 |                 directed level 4, CPU 2         |
*         |            |                 or non-directed level 0         |
*         +------------+-------------------------------------------------+
*         | 0x0d00000c |                 directed level 4, CPU 3         |
*         |            |                 or non-directed level 0         |
*         +------------+-------------------------------------------------+
*         | 0x0d000010 |                 directed level 4, CPU 4         |
*         |            |                 or non-directed level 0         |
*         +------------+-------------------------------------------------+
*         | 0x0d000014 |                 directed level 4, CPU 5         |
*         |            |                 or non-directed level 0         |
*         +------------+-------------------------------------------------+
*         | 0x0d000018 |                 directed level 4, CPU 6         |
*         |            |                 or non-directed level 0         |
*         +------------+-------------------------------------------------+
*         | 0x0d00001c |                 directed level 4, CPU 7         |
*         |            |                 or non-directed level 0         |
*         +------------+-------------------------------------------------+
*         | 0x0d000020 |                 directed level 4, CPU 8         |
*         |            |                 or non-directed level 0         |
*         +------------+-------------------------------------------------+
*         | 0x0d000024 |                 directed level 4, CPU 9         |
*         |            |                 or non-directed level 0         |
*         +------------+-------------------------------------------------+
*         | 0x0d000028 |                 directed level 4, CPU A         |
*         |            |                 or non-directed level 0         |
*         +------------+-------------------------------------------------+
*         | 0x0d00002c |                 directed level 4, CPU B         |
*         |            |                 or non-directed level 0         |
*         +------------+-------------------------------------------------+
*         | 0x0d000030 |                 directed level 4, CPU C         |
*         |            |                 or non-directed level 0         |
*         +------------+-------------------------------------------------+
*         | 0x0d000034 |                 directed level 4, CPU D         |
*         |            |                 or non-directed level 0         |
*         +------------+-------------------------------------------------+
```

0x0d000038	directed level 4, CPU E or non-directed level 0
0x0d00003c	directed level 4, CPU F or non-directed level 0
0x0d000040	directed level 5, CPU 0 or non-directed level 1
0x0d000044	directed level 5, CPU 1 or non-directed level 1
0x0d000048	directed level 5, CPU 2 or non-directed level 1
0x0d00004c	directed level 5, CPU 3 or non-directed level 1
0x0d000050	directed level 5, CPU 4 or non-directed level 1
0x0d000054	directed level 5, CPU 5 or non-directed level 1
0x0d000058	directed level 5, CPU 6 or non-directed level 1
0x0d00005c	directed level 5, CPU 7 or non-directed level 1
0x0d000060	directed level 5, CPU 8 or non-directed level 1
0x0d000064	directed level 5, CPU 9 or non-directed level 1
0x0d000068	directed level 5, CPU A or non-directed level 1
0x0d00006c	directed level 5, CPU B or non-directed level 1
0x0d000070	directed level 5, CPU C or non-directed level 1
0x0d000074	directed level 5, CPU D or non-directed level 1
0x0d000078	directed level 5, CPU E or non-directed level 1
0x0d00007c	directed level 5, CPU F or non-directed level 1
0x0d000080	non-directed level 2 (if SPM in slot 0)

0x0d000084	non-directed level 2 (if SPM in slot 1)
0x0d000088	non-directed level 2 (if SPM in slot 2)
0x0d00008c	non-directed level 2 (if SPM in slot 3)
0x0d000090	non-directed level 2 (if SPM in slot 4)
0x0d000094	non-directed level 2 (if SPM in slot 5)
0x0d000098	non-directed level 2 (if SPM in slot 6)
0x0d00009c	non-directed level 2 (if SPM in slot 7)
0x0d0000a0	non-directed level 2 (if SPM in slot 8)
0x0d0000a4	non-directed level 2 (if SPM in slot 9)
0x0d0000a8	non-directed level 2 (if SPM in slot A)
0x0d0000ac	non-directed level 2 (if SPM in slot B)
0x0d0000b0	non-directed level 2 (if SPM in slot C)
0x0d0000b4	non-directed level 2 (if SPM in slot D)
0x0d0000b8	non-directed level 2 (if SPM in slot E)
0x0d0000bc	non-directed level 2 (if SPM in slot F)
0x0d0000c0	non-directed level 3 (if SPM in slot 0)
0x0d0000c4	non-directed level 3 (if SPM in slot 1)
0x0d0000c8	non-directed level 3 (if SPM in slot 2)
0x0d0000cc	non-directed level 3 (if SPM in slot 3)
0x0d0000d0	non-directed level 3 (if SPM in slot 4)
0x0d0000d4	non-directed level 3 (if SPM in slot 5)
0x0d0000d8	non-directed level 3 (if SPM in slot 6)
0x0d0000dc	non-directed level 3 (if SPM in slot 7)
0x0d0000e0	non-directed level 3 (if SPM in slot 8)
0x0d0000e4	non-directed level 3 (if SPM in slot 9)
0x0d0000e8	non-directed level 3 (if SPM in slot A)
0x0d0000ec	non-directed level 3 (if SPM in slot B)
0x0d0000f0	non-directed level 3 (if SPM in slot C)

```
 *        | 0x0d0000f4 |          non-directed level 3 (if SPM in slot D) |
 *        +------------+-----------------------------------------------+
 *        | 0x0d0000f8 |          non-directed level 3 (if SPM in slot E) |
 *        +------------+-----------------------------------------------+
 *        | 0x0d0000fc |          non-directed level 3 (if SPM in slot F) |
 *        +------------+-----------------------------------------------+
 *
 *
 ***************************************************************************/

#define POINTER_RAM (unsigned *)(0x0d000000)
#define POINTER_LEN              0x100
#define POINTER_LWORDS           0x40
#define POINTER_MASK             0x7f7f0000
#define OUT_PNTR_SH              16
#define IN_PNTR_SH               24

/***************************************************************************
 *
 *      Interrupt Dispatcher Miscellaneous RAMs
 *
 *      The miscellaneous RAMs consist of the counter RAM, the service pending
 *      RAM, and the CPU configuration RAM. They are all simultaneously address-
 *      ed as a 32 bit port, with different data bits going to the different
 *      RAMs as follows:
 *
 *         31 - 28 27 - 24  23 - 16       15       14 - 10    9 - 0
 *        +-------+-------+---------+------------+---------+---------+
 *        |   x   | config|    x    | serv. pend |     x   | counter |
 *        +-------+-------+---------+------------+---------+---------+
 *                   16                 128                   8
 *                  deep                deep                 deep
 *
 *      The counter must be initialized to all 1s, the service pending must be
 *      initialized to all 0s, and the configuration RAM must be initialized
 *      with all slot ids of CPUs present in the CSS backplane which are avail-
 *      able for recieving interrupts as follows:
 *
 *                address                        data
 *
 *                CPU slot id n              CPU slot id 1
 *                CPU slot id 1              CPU slot id 2
 *                      .                          .
 *                      .                          .
 *                      .                          .
 *                CPU slot id n-1              CPU slot id n
 *                    n+1                      CPU slot id 1
 *                      .                          .
 *                      .                          .
 *                      .                          .
 *                     16                      CPU slot id 1
 *
 *        Counter
 *        -------
 *        note:       negative true logic is used to store the count
 *                  (0x3ff corresponds to 0)
```

```
*
*           Address            Bits 9 - 0
*         +--------------+----------------------+
*         | 0x0e000000   |     count level 0    |
*         +--------------+----------------------+
*         | 0x0e000004   |     count level 1    |
*         +--------------+----------------------+
*         | 0x0e000008   |     count level 2    |
*         +--------------+----------------------+
*         | 0x0e00000c   |     count level 3    |
*         +--------------+----------------------+
*         | 0x0e000010   |     count level 4    |
*         +--------------+----------------------+
*         | 0x0e000014   |     count level 5    |
*         +--------------+----------------------+
*         | 0x0e000018   |       reserved       |
*         +--------------+----------------------+
*         | 0x0e00001c   |       reserved       |
*         +--------------+----------------------+
*
*         Service Pending Ram
*         -------------------
*
*           Address                 Bit 15
*         +--------------+----------------------+
*         | 0x0e000000   |    CPU 0, level 0    |
*         +--------------+----------------------+
*         | 0x0e000004   |    CPU 1, level 0    |
*         +--------------+----------------------+
*         | 0x0e000008   |    CPU 2, level 0    |
*         +--------------+----------------------+
*         | 0x0e00000c   |    CPU 3, level 0    |
*         +--------------+----------------------+
*         | 0x0e000010   |    CPU 4, level 0    |
*         +--------------+----------------------+
*         | 0x0e000014   |    CPU 5, level 0    |
*         +--------------+----------------------+
*         | 0x0e000018   |    CPU 6, level 0    |
*         +--------------+----------------------+
*         | 0x0e00001c   |    CPU 7, level 0    |
*         +--------------+----------------------+
*         | 0x0e000020   |    CPU 8, level 0    |
*         +--------------+----------------------+
*         | 0x0e000024   |    CPU 9, level 0    |
*         +--------------+----------------------+
*         | 0x0e000028   |    CPU A, level 0    |
*         +--------------+----------------------+
*         | 0x0e00002c   |    CPU B, level 0    |
*         +--------------+----------------------+
*         | 0x0e000030   |    CPU C, level 0    |
*         +--------------+----------------------+
*         | 0x0e000034   |    CPU D, level 0    |
*         +--------------+----------------------+
*         | 0x0e000038   |    CPU E, level 0    |
*         +--------------+----------------------+
*         | 0x0e00003c   |    CPU F, level 0    |
```

```
*       +--------------+----------------------+
*       | 0x0e000040   |    CPU 0, level 1    |
*       +--------------+----------------------+
*       | 0x0e000044   |    CPU 1, level 1    |
*       +--------------+----------------------+
*       | 0x0e000048   |    CPU 2, level 1    |
*       +--------------+----------------------+
*       | 0x0e00004c   |    CPU 3, level 1    |
*       +--------------+----------------------+
*       | 0x0e000050   |    CPU 4, level 1    |
*       +--------------+----------------------+
*       | 0x0e000054   |    CPU 5, level 1    |
*       +--------------+----------------------+
*       | 0x0e000058   |    CPU 6, level 1    |
*       +--------------+----------------------+
*       | 0x0e00005c   |    CPU 7, level 1    |
*       +--------------+----------------------+
*       | 0x0e000060   |    CPU 8, level 1    |
*       +--------------+----------------------+
*       | 0x0e000064   |    CPU 9, level 1    |
*       +--------------+----------------------+
*       | 0x0e000068   |    CPU A, level 1    |
*       +--------------+----------------------+
*       | 0x0e00006c   |    CPU B, level 1    |
*       +--------------+----------------------+
*       | 0x0e000070   |    CPU C, level 1    |
*       +--------------+----------------------+
*       | 0x0e000074   |    CPU D, level 1    |
*       +--------------+----------------------+
*       | 0x0e000078   |    CPU E, level 1    |
*       +--------------+----------------------+
*       | 0x0e00007c   |    CPU F, level 1    |
*       +--------------+----------------------+
*       | 0x0e000080   |    CPU 0, level 2    |
*       +--------------+----------------------+
*       | 0x0e000084   |    CPU 1, level 2    |
*       +--------------+----------------------+
*       | 0x0e000088   |    CPU 2, level 2    |
*       +--------------+----------------------+
*       | 0x0e00008c   |    CPU 3, level 2    |
*       +--------------+----------------------+
*       | 0x0e000090   |    CPU 4, level 2    |
*       +--------------+----------------------+
*       | 0x0e000094   |    CPU 5, level 2    |
*       +--------------+----------------------+
*       | 0x0e000098   |    CPU 6, level 2    |
*       +--------------+----------------------+
*       | 0x0e00009c   |    CPU 7, level 2    |
*       +--------------+----------------------+
*       | 0x0e0000a0   |    CPU 8, level 2    |
*       +--------------+----------------------+
*       | 0x0e0000a4   |    CPU 9, level 2    |
*       +--------------+----------------------+
*       | 0x0e0000a8   |    CPU A, level 2    |
*       +--------------+----------------------+
*       | 0x0e0000ac   |    CPU B, level 2    |
```

0x0e0000b0	CPU C, level 2
0x0e0000b4	CPU D, level 2
0x0e0000b8	CPU E, level 2
0x0e0000bc	CPU F, level 2
0x0e0000c0	CPU 0, level 3
0x0e0000c4	CPU 1, level 3
0x0e0000c8	CPU 2, level 3
0x0e0000cc	CPU 3, level 3
0x0e0000d0	CPU 4, level 3
0x0e0000d4	CPU 5, level 3
0x0e0000d8	CPU 6, level 3
0x0e0000dc	CPU 7, level 3
0x0e0000e0	CPU 8, level 3
0x0e0000e4	CPU 9, level 3
0x0e0000e8	CPU A, level 3
0x0e0000ec	CPU B, level 3
0x0e0000f0	CPU C, level 3
0x0e0000f4	CPU D, level 3
0x0e0000f8	CPU E, level 3
0x0e0000fc	CPU F, level 3
0x0e000100	CPU 0, level 4
0x0e000104	CPU 1, level 4
0x0e000108	CPU 2, level 4
0x0e00010c	CPU 3, level 4
0x0e000110	CPU 4, level 4
0x0e000114	CPU 5, level 4
0x0e000118	CPU 6, level 4
0x0e00011c	CPU 7, level 4

```
 *        +--------------+----------------------+
 *        | 0x0e000120   |    CPU 8, level 4    |
 *        +--------------+----------------------+
 *        | 0x0e000124   |    CPU 9, level 4    |
 *        +--------------+----------------------+
 *        | 0x0e000128   |    CPU A, level 4    |
 *        +--------------+----------------------+
 *        | 0x0e00012c   |    CPU B, level 4    |
 *        +--------------+----------------------+
 *        | 0x0e000130   |    CPU C, level 4    |
 *        +--------------+----------------------+
 *        | 0x0e000134   |    CPU D, level 4    |
 *        +--------------+----------------------+
 *        | 0x0e000138   |    CPU E, level 4    |
 *        +--------------+----------------------+
 *        | 0x0e00013c   |    CPU F, level 4    |
 *        +--------------+----------------------+
 *        | 0x0e000140   |    CPU 0, level 5    |
 *        +--------------+----------------------+
 *        | 0x0e000144   |    CPU 1, level 5    |
 *        +--------------+----------------------+
 *        | 0x0e000148   |    CPU 2, level 5    |
 *        +--------------+----------------------+
 *        | 0x0e00014c   |    CPU 3, level 5    |
 *        +--------------+----------------------+
 *        | 0x0e000150   |    CPU 4, level 5    |
 *        +--------------+----------------------+
 *        | 0x0e000154   |    CPU 5, level 5    |
 *        +--------------+----------------------+
 *        | 0x0e000158   |    CPU 6, level 5    |
 *        +--------------+----------------------+
 *        | 0x0e00015c   |    CPU 7, level 5    |
 *        +--------------+----------------------+
 *        | 0x0e000160   |    CPU 8, level 5    |
 *        +--------------+----------------------+
 *        | 0x0e000164   |    CPU 9, level 5    |
 *        +--------------+----------------------+
 *        | 0x0e000168   |    CPU A, level 5    |
 *        +--------------+----------------------+
 *        | 0x0e00016c   |    CPU B, level 5    |
 *        +--------------+----------------------+
 *        | 0x0e000170   |    CPU C, level 5    |
 *        +--------------+----------------------+
 *        | 0x0e000174   |    CPU D, level 5    |
 *        +--------------+----------------------+
 *        | 0x0e000178   |    CPU E, level 5    |
 *        +--------------+----------------------+
 *        | 0x0e00017c   |    CPU F, level 5    |
 *        +--------------+----------------------+
 *
 *
 ******************************************************************************/

#define MISC_RAM (unsigned *)(0x0e000000)
#define MISC_LEN            0x200
#define MISC_LWORDS         0x80
```

```
#define MISC_MASK           0x0f0083ff
#define CNTR_MASK           0x000003ff
#define CNTR_LWORDS         8
#define CNTR_ZERO_BIT       0x00000400
#define SERV_MASK           0x00008000
#define SERV_LWORDS         0x60
#define CONFIG_LWORDS       16
#define CONFIG_MASK         0x0f000000
#define CONFIG_SH           24

#define PIPEDATA (unsigned *)(0x80000018)
#define PIPEADDR (unsigned *)(0x8000001c)

#define DATARESP 4


/*****************************************************************************
 *
 *      Translation table for CSS data bits to local address and data bits.
 *
 *                              CSS BUS DATA
 *      +------+------+------+------+------+------+------+------+
 *      |00  07|10  17|20  27|30  37|40  47|50  57|60  67|70  77|
 *      +------+------+------+------+------+------+------+------+
 *      |24  31|16  23|8   15|0    7|24  31|16  23|8   15|0    7|
 *      +------+------+------+------+------+------+------+------+
 *      |<-------- IN DATA -------->|<-------- IN ADDR -------->|
 *
 *                        IN DATA
 *        31 - 24   23 22-18 17 16   15 - 12    11 - 8   7     6 - 0
 *      +----------+-+------+-----+----------+----------+-+-------------+
 *      | Vector # |D|   x  |level|Dest. Slot| Src. Slot|I|I/O Src. Slot|
 *      +----------+-+------+-----+----------+----------+-+-------------+
 *
 *      Vector #
 *      --------
 *      read by the 68020 during the interrupt acknowledge cycle.
 *
 *      Directed Bit
 *      ------------
 *      directed interrupt if a 1, the interrupt must be sent to the
 *      specified destination slot. If a 0, the interrupt may be sent to
 *      any computational module accepting interrupts.
 *
 *      Int Priority
 *      ------------
 *      determines the priority of the interrupt, which increases with value.
 *
 *      Destination Slot
 *      ----------------
 *      If the directed bit is on, the interrupt is sent to the slot specified
 *      by this value. If a non-directed interrupt, this field is ignored.
 *
 *      Source Slot
 *      -----------
 *      The interrupt request came from the slot specified by this value.
```

```
 *
 *        I/O Bus Interrupt
 *        -----------------
 *        If a 1, the interrupt came from an i/o bus attached to the S-bus
 *        module. The I/O bus slot requesting the interrupt is specified by
 *        "I/O Bus Src Slot". If this bit is 0, the interrupt request came from
 *        the S-bus module.
 *
 *        I/O Bus Source Slot
 *        -------------------
 *        If the I/O Bus Interrupt bit is set, the interrupt came from the I/O
 *        bus slot specified by this value. If the I/O bus Interrupt bit is 0,
 *        this field is ignored(set to zero).
 *
 *
 *        D bit   Priority Level              Use
 *        -----   --------------              ---
 *          1      0x07 - 0x04        High priority directed interrupts;
 *                                    mapped into 68020 interrupt level 6.
 *          1      0x03 - 0x00        Low priority directed interrupts;
 *                                    mapped into 68020 interrupt level 5.
 *          0      0x0f - 0x0c        Highest priority i/o interrupts;
 *                                    mapped into 68020 interrupt level 4.
 *          0      0x0b - 0x08        High priority i/o interrupts;
 *                                    mapped into 68020 interrupt level 3.
 *          0      0x07 - 0x04        Low priority i/o interrupts;
 *                                    mapped into 68020 interrupt level 2.
 *          0      0x03 - 0x00        Lowest priority i/o interrupts;
 *                                    mapped into 68020 interrupt level 1.
 *
 *
 ************************************************************************/

/*
   Number of slots  in a css back plane.
*/

#define MAX_CSS_SLOT 16
```