DAN Processor Node: The MDH–9 Design and Ideas

Mark Hayter

16th February, 1993
15th March 1994

The DAN project is investigating building a multimedia workstation based around the Desk Area Network. One area for investigation is the use of a DAN for processor to memory interconnect. A processor node, known as the MDH–9[1], is being designed around the ARM600 RISC processor. This node will have only a small amount of local memory and it is intended that, through the use of a cache, all of its main memory will be served across the DAN. The main parts of the processor node are shown in figure 1.

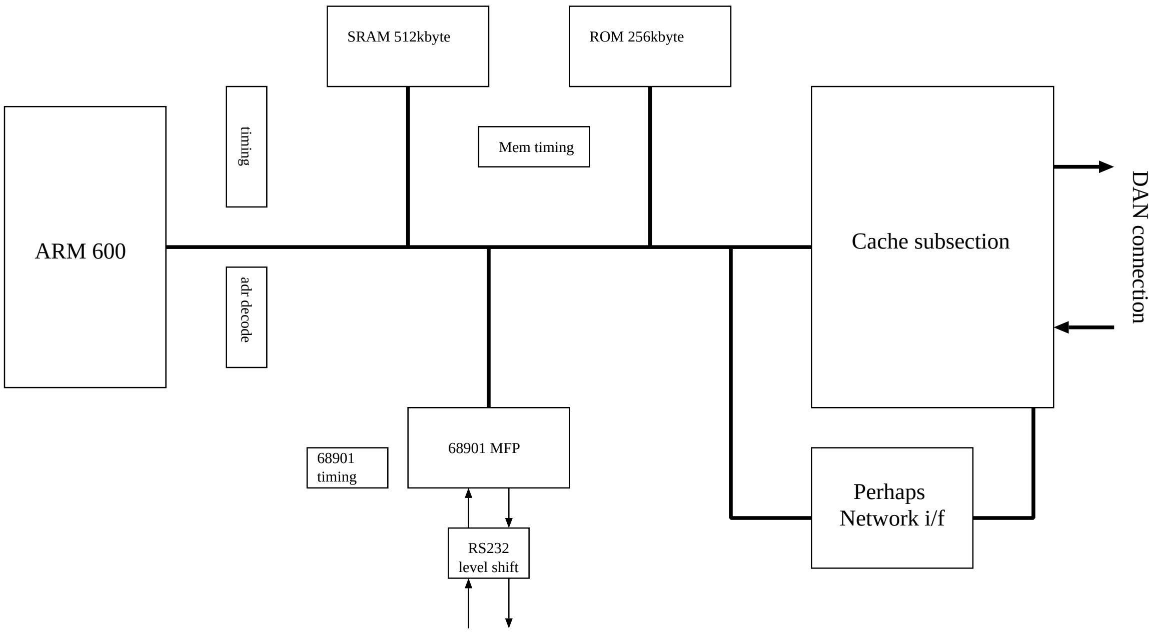

Figure 1: Overview of DAN processor node

Technical updates have been done to this document for the 1994 documentation snapshot (the Blue Book). For results using the DAN CPU node and a higher level description of the card see technical report 319.

[1]For pedants the correct name is MDH–9 and not MDH9.

1 The Processor Node

The DAN processor node is based on the ARM600 processor. This is a derivative of the ARM series of processors that are used in the Fairisle port controller. It uses the ARM6 macro cell with a 4k byte cache, a write-back buffer, and an MMU.

It is intended that the processor node run the Wanda microkernel. This requires a regular timer interrupt, mainly used to enable timeouts on semaphore operations. In addition, especially for use during debugging, it is usual for the console to be provided via a RS-232 serial line. To provide both of these a 68901 Multi-Function Peripheral chip will be put on the processor node. This chip was chosen because the Wanda device drivers have already been written for it, and we have a number in stock.

At least until the remote memory system has been debugged the processor node will require local memory to permit booting. Eventually it should be possible for all memory to be served but this is thought impractical for development of the remote system. Thus some ROM and some RAM will be provided on the node. It has been found that 128k bytes of ROM is borderline for fitting an ARM Wanda boot kernel, so it is intended to provide 256k through the use of four 27512 devices. The size of RAM provoked much discussion. Since it is only required for the development of the node, and was intended to be a "small" amount, it was thought that the additional complexity of using DRAM was not warranted. Therefore 512k bytes of SRAM are provided. There are two reasons for this size. It should be sufficient for the current boot system (which copies the ROM code and runs entirely out of RAM) (though there may be insufficient space to store the loaded image). It is also enough for a full set of MMU page tables to be kept in local memory (tr has suggested that this will take 256k).

The node will have network access, via a simple interface (the "perhaps" interface). This is based on a single cell deep transmit FIFO and a single cell deep receive FIFO. These connect directly to the DAN switch, and thus the rxFIFO not-empty condition can cause NACKs. This interface is not designed for high speed, it will be used for setting up connections to memory servers and infrequent RPC operations.

The main part of the node is the cache section. This provides access to remote memory for the processor node. The cache size has been chosen as 128k bytes, and the ATM cell size of 48 bytes suggests that a cache line of 32 bytes would be ideal. Expansion of the cache to 512k is permitted by the design, but there is only PCB space for 256k. Studies of cache systems have suggested that for such size the improvement in miss-ratio of going from direct mapped to 2-way set associative is sufficiently small that it does not justify the increase in complexity of the cache. In this case this argument is not valid - at least provided 2 way associativity does not require insertion of extra processor cycles in every memory access - since the cache line refill time is very large compared to the hit access time. Rough calculation suggests that the 0.5% change in miss-ratio is equivalent to increasing the *average* memory access time by more than one memory clock cycle. Thus the 128k of cache will be organised as two banks of 64k.

2 Memory map

The memory map was selected to require minimal decoding, thus there are many echo access points. The cache subsection takes half of the memory map, being decoded on the top address bit. To enable simple decoding by the timing PAL it was decided that for other accesses the next address bit indicate if a CPU speed or a slow access. Two additional constraints were used, the ROM must be at location zero, since this is the processor reset vector, and following the end of RAM (at its normal address) there should be a read only section. The 68901 implements the 68000 interrupt structure, which requires an IACK cycle in response to an interrupt. To allow simulation of this the address bus is decoded to give both a select and an IACK access route. This gave the memory map outlined in table 1.

A31 cache	A30 speed	A21 sel1	A20 ignore	A19 sel0	speed	size required	Use
0	0	0	0	0	250ns/150ns	256k	ROM
0	0	0	0	1	100ns	512k	RAM
0	0	0	1	0	250ns/150ns	256k	ROM (echo)
0	0	0	1	1	100ns	512k	RAM (echo)
0	0	1	0	0	slow async	32 words	68901 CS access
0	0	1	0	1	slow async	1 byte	68901 IACK access
0	0	1	1	0	slow async	32 words	68901 CS (echo)
0	0	1	1	1	slow async	1 byte	68901 IACK (echo)
0	1	0	0	0	CPU	1 byte	write TX fifo
0	1	0	0	1	CPU	1 byte	write TX, read RX fifo
0	1	0	1	0	CPU	1 byte	TX fifo (echo)
0	1	0	1	1	CPU	1 byte	TX/RX fifo (echo)
0	1	1	0	0	CPU	1k bytes	VCI dual port memory
0	1	1	0	1	CPU	1 word	read status, write ctrl
0	1	1	1	0	CPU	1k bytes	VCI memory (echo)
0	1	1	1	1	CPU	1 word	Status/ctrl (echo)
1	x	x	x	x	CPU/async	2Gb	Cache subsystem access

Table 1: Memory map

The cache has a debug and a normal mode of access, described in section 4. Combining the memory map above with that for the cache gives the base addresses shown in table 2.

3 Timing and Reset

The ARM600 is rated up to a core clock (FCLK) of 25MHz. It is usual for the memory clock (MCLK) to be approximately half this speed. The processor switches between the two as required. Since the main point of the processor node is investigating the use of the external cache it has been decided to run the memory clock at the same speed as the core. This is permitted by the original data sheet[2], although it changes some of the assumed phase relationships between signals. Using processor synchronous mode should ensure no penalty in switching between the two clocks.

The use of a fast external clock for the processor requires that wait states be inserted in accesses of all the local memory and devices. This is done by a PAL and a fast gate which will assert the nWAIT signal on every memory cycle where required. A cache access, in the event of a hit, is performed at CPU speed. The control section of the cache subsystem and the perhaps interface also work at CPU speeds. Local memory is slow, SRAM having 100ns

[2]advice from ARM Ltd also suggests it is possible

Device	base address	size
ROM	0x0000 0000	256k
SRAM	0x0008 0000	512k
68901 CS	0x0020 0000	32 (low byte of each word)
68901 IACK	0x0028 0000	1 byte
TX fifo	0x4000 0000	1 byte
RX fifo	0x4008 0000	1 byte
VCI ram	0x4020 0000	1k (low byte of each word)
Status	0x4028 0000	1 word
Command	0x4028 0000	1 word
Cache	0x8000 0000	1Gb
Debug Tag[0]	0xC000 0000	8k (lines)
Debug Tag[1]	0xD000 0000	8k (lines)
Debug Cache[0]	0xE000 0000	64k (or 128k, 256k)
Debug Cache[1]	0xF000 0000	64k (or 128k, 256k)

Table 2: Usual Base access addresses

access time, and ROM 150ns or 250ns. The 68901 uses the 68000 asynchronous bus protocol, based on a 4 MHz clock, providing an acknowledge to terminate an access after approx 250ns plus the clock synchronisation time. The timing PAL decodes the top two address bits and the cache hit signals to determine whether to signal a WAIT as shown in table 3.

A31	A30	MATCH0	MATCH1	nWAIT	Comment
1	x	0	0	Y	Cache miss
1	x	1	0	N	Cache hit bank 0
1	x	0	1	N	Cache hit bank 1
1	x	1	1	?	Cache messed up (halt ?)
0	1	x	x	N	Local CPU speed device
0	0	x	x	Y	Local slow device

Table 3: Decode for nWAIT insertion

The reset system uses a pal to count 4 MHz clock cycles to provide a valid reset pulse for all devices on the card. There are four sources of reset: the front panel switch, the fabric, the watchdog timer and the cache controller. The first two are as for port controllers. The watchdog reset is provided by the output of a timer register and enables resetting of the machine if some number of regular timer interrupts are missed. This feature has proved useful on the 68k Wanda systems, and is a link option on the MDH–9. The cache reset line is provided to enable the cache controller, or possibly reception of a control cell, to reset the card, this is programmed by the xilinx design. The reason for the last reset is available through the 68901 I/O lines as described later in table 13. *There is an outstanding problem with reset. The ARM 600 counts on the address bus while in reset, and performs two memory accesses to the next two locations on coming out of reset – if these are to an unfortunate location the nWAIT signal is asserted and the CPU will hang. This can probably be cured by* *ignoring these cycles but this has not yet been done.*

4 The cache

The cache subsystem is selected by the top address bit being set, the next bit is used to select between normal access and "debug" accesses. The other address bits are divided into five groups:

Offset. This is fixed as the low 5 bits of the address and gives the byte offset into the 32 bytes of the cache line.

Line. The line number indicates which address in the cache to look for the required data.

Tag. The cache tag is fixed at 7 bits and is used to determine if the data is in the cache at the selected line.

Region. The region is a three bit field which is part of either the line or the tag. It is used to select which VCI to use to connect to a memory server. This allows for a number of servers to be used, or for a number of objects to be mapped in. The region field *is* part of the address used to determine if an item is in the cache.

Rindex. In addition to the region an extra 5 bits are used to determine the VCI to be used to access a memory server. These bits are *not* used by the cache - all Rindex values for a given region are regarded as the same. The intention is that a region may be divided into portions with different access rights - indicated by different VCIs. The MMU is used to provide the Rindex values and perform the local check on permissions.

The cache is 2 way set associative, 128k of memory giving 2 x 64k of cache. Each cache line is 32 bytes, to fit into a cell. Thus there are 2k lines in the cache, requiring 11 address bits. The cache tag ram is 8 bits wide, but one of these must be used as a valid bit. This covers 8Mbytes of the address space. This is divided into eight regions as described above. When a cache miss occurs it may be necessary to flush a line, to ensure that only one server needs to be contacted the regions will be separately cached this is referred to as *separate regions*. This leads to wasting cache space if only few regions are used. The alternative *combined regions* case makes flushing a cache line potentially more complicated but allows a single region to use all the cache. Experiments should be done to investigate the tradeoff between the two cases, the card will have some links to select between the two. For ARM600 byte addressing table 4 gives the address assignment for 2k cache lines (128k of cache), table 5 for 4k lines (256k) and table 6 for 8k lines (512k). *The actual mapping from CPU address to line and tag is done by the "AdrPat" selection links, and the selection of the region bits by the "RegPat" links. Due to confused thinking the cache controler "AdrXil" xilinx also needs to know this mapping so the xilinx bits must match the selectors.* Figure 2 shows the pinouts of the selection links, and the correct connections for 256K cache, 2 regions as used in the "stream cache" experiments.

Each region of the cache will need an associated VCI. This consists of the fabric and port controller routeing tag and the receiver's association id. This information is obtained from a small dual ported memory. The region number and Rindex bits to index into the dual port memory forming a 10 bit address as shown below.

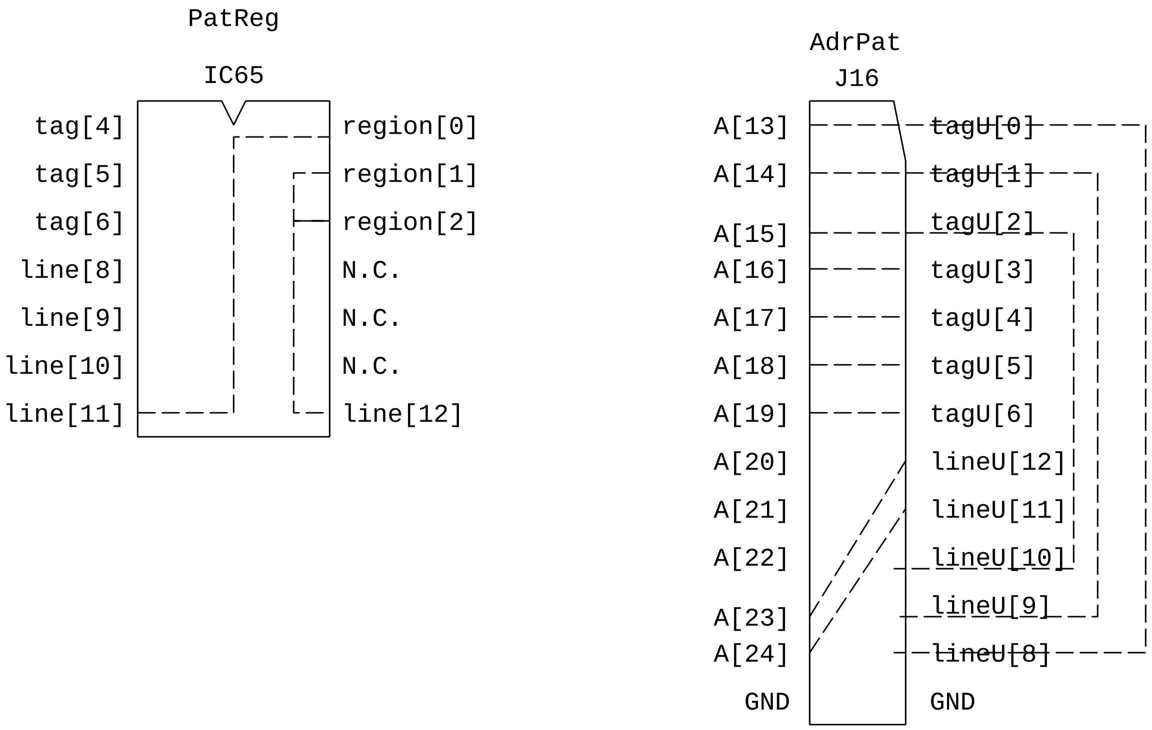

Figure 2: Address Bus to Cache Links

Name	Size (bits)	Separate regions	Combined regions	Use
Offset	5	A0-A4	A0-A4	Byte within cache line
Line	11	A5-A12,A20-A22	A5-A15	Cache line number
Tag	7	A13-A19	A16-A22	Cache tag
Region	3	A20-A22	A20-A22	Region - also in line or tag
Ignore	2	A23-A24	A23-A24	Ignore so Rindex same as 8k
Rindex	5	A25-A29	A25-A29	Index in region
Debug	1	A30	A30	Set to indicate a debug access
Select	1	A31	A31	Set to select cache access

Table 4: Address bit use (2k lines)

rindex	region	Offset (2 bits)
5..0	2..0	00-FabRou, 01-PCRou, 1x-VCI

The Rindex bits are supplied by the MMU when translating a logical address into a cache address. Since the cache resolution mechanism does not take account of the Rindex bits it is possible for several associations to be open in the same region. This would enable (for example) a region to be opened Read-Only and then a write lock be taken out on a small section of it, the region would initially be mapped R/O by the MMU with the Rindex bits setup. When the write lock is taken out (on a new association) the locked pages will be marked R/W, the association table updated and the Rindex translation in the MMU switched. Any data in the CPU cache (accessed by logical address) will still be valid, and any data in the

Name	Size (bits)	Separate regions	Combined regions	Use
Offset	5	A0-A4	A0-A4	Byte within cache line
Line	12	A5-A13,A21-A23	A5-A16	Cache line number
Tag	7	A14-A20	A17-A23	Cache tag
Region	3	A21-A23	A21-A23	Region - also in line or tag
Ignore	2	A24	A24	Ignore so Rindex same as 8k
Rindex	5	A25-A29	A25-A29	Index in region
Debug	1	A30	A30	Set to indicate a debug access
Select	1	A31	A31	Set to select cache access

Table 5: Address bit use (4k lines)

Name	Size (bits)	Separate regions	Combined regions	Use
Offset	5	A0-A4	A0-A4	Byte within cache line
Line	11	A5-A14,A22-A24	A5-A17	Cache line number
Tag	7	A15-A21	A18-A24	Cache tag
Region	3	A22-A24	A22-A24	Region - also in line or tag
Rindex	5	A25-A29	A25-A29	Index in region
Debug	1	A30	A30	Set to indicate a debug access
Select	1	A31	A31	Set to select cache access

Table 6: Address bit use (8k lines)

external cache will also be still accessible since the address the cache sees remains the same.

The memory server will have virtual circuits setup with all clients, so sharing will be explicit. Upon detection of sharing the client systems may be informed, and can take appropriate action. This suggests that the server may wish to send updates to clients - this area is still being considered but it is thought that all the extra buffers required are in place. The system may be put into "flush" in which any cache line touched that is dirty will be flushed to the memory server.

The "Debug" mode of access allows the CPU to read and write the cache memory and the tag memory. This is to allow debugging and the gathering of usage statistics. It will also allow software updates to cache lines. This mode of operation is assumed to be used infrequently and therefore the software will have to perform all the address manipulation itself – address bits are interpreted in the same way as during a standard cache access, for example when accessing the tag rams the tags will be at the 32 byte line spacing apart.. When address bits A31 and A30 are both set the next two bits indicate the device accessed as shown in table 7. An access to the cache tag memory also accesses the information memory for that line. Ten data bits are used as in table 8. The tag and valid bits may be read and written. The dirty bit will only be written if the enable bit is set. On a read the top bit indicates which of the sets least (most ?) recently had a cache hit. Accesses to the SRAM in debug mode are 32 bits wide, with the top address bit selecting which bank of the cache is used.

A29	A28	Access
0	0	Tag 0 and info 0
0	1	Tag 1 and info 1
1	0	SRAM low half (==tag 0)
1	1	SRAM top half (==tag 1)

Table 7: Access bits in debug mode

Bit	Read	Write
0-6	tag	tag
7	valid	valid
8	dirty	dirty iff bit 9
9	LRU	enable write of dirty

Table 8: Tag and Info data for a line

The cache is basically 2 way associative with LRU replacement. This is achieved by storing for each line which set was last accessed (clearly the address for this store is identical to that for the cache tag memory). This LRU data is stored in Info[2]. The output (Q) from this device is passed to the cache controller which uses it when selecting which set to use after a cache miss. A direct mapped cache may thereofe be formed by disconnecting the output from Info[2] (pull it out of its socket and bend the pin!) and pulling the line low causing all cache line fetches to be to set 1. This wastes half the cache, so an alternative is to tie the pin to one of the address bits which are part of the tag field allowing all the cache to be used direct mapped.

4.1 Cache Hardware Design

The cache is based around some fast SRAMs and cache tag memories. The tag memories are basically standard SRAMs but if a read cycle is performed with the output disabled the data supplied is compared with the data addressed and a match indication supplied. Two of these devices are used to give a two-way set associative cache. During the 'i' (idle) cycle preceding any non-sequential memory access both tag memories perform a match cycle in parallel indicating which half of the cache contains the data or if a miss has happened in addition the information memory for these cache lines is read. Assuming a hit the cache memory is accessed during the 'm' cycle, and the LRU and dirty information bits written. If further sequential accesses are performed then the cache memory is accessed again, there is a special case of a wrap over a cache line boundary when a WAIT cycle must be inserted and the tag lookup repeated. The main parts of the cache design are shown in figure 3.

To allow for the cache cycles to be performed at the speed of the processor the logic for performing the tag lookup and the access(es) in the case of a hit is put in the fast timing PAL. In case of a miss a xilinx chip takes over to perform the remote access. There are several reasons for this split. The ARM cycle time is 40ns, and the input and output times of the xilinx would use up a significant proportion of this, using "10ns" PALs however the output is valid 7ns after the clock. It seems likely that the 'hit' access logic is unlikely to

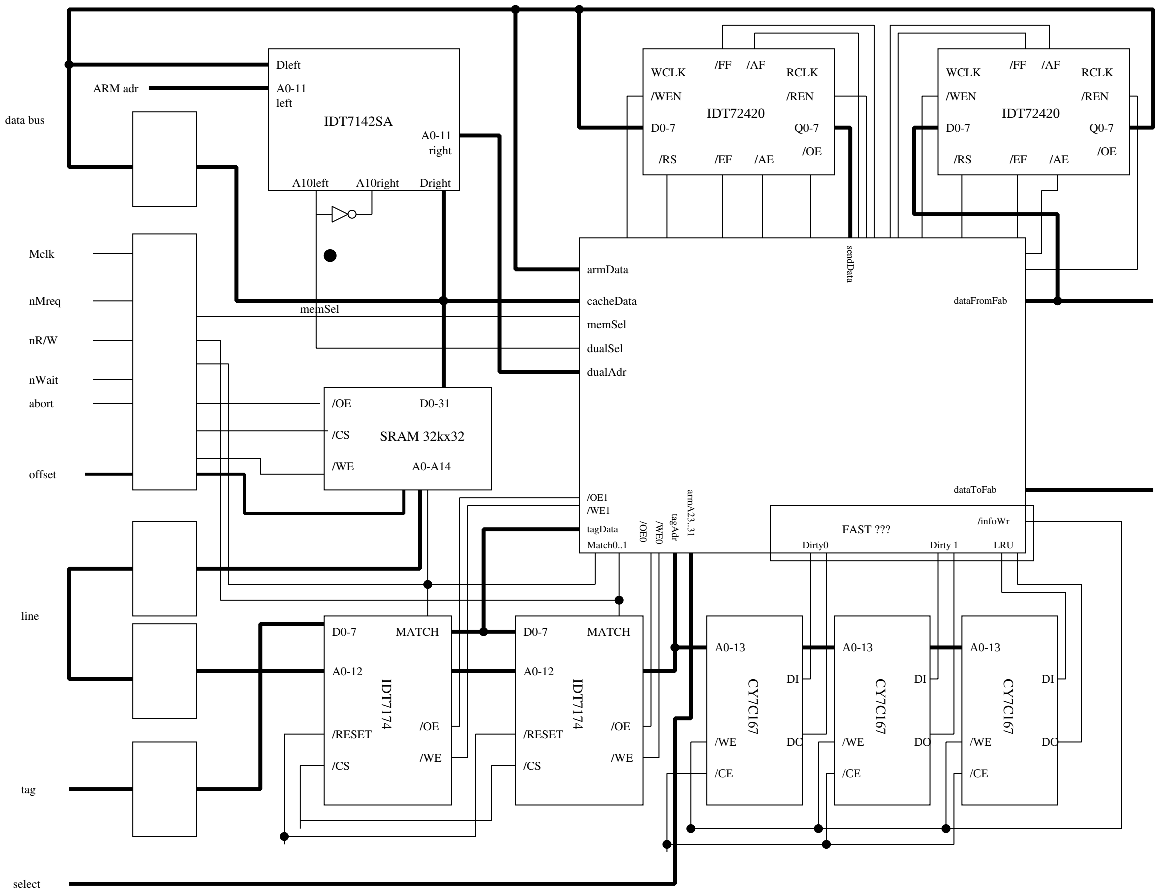

Figure 3: Main parts of the cache

change frequently and using a CMOS erasable PAL will allow easy modifications. The miss logic will involve driving the switch protocol, accessing the cache memory, maintaining the tag memory and holding the VCIs for remote accesses. This will be much more complex logic and is therefore a candidate for a Xilinx chip.

In case of a miss there are two actions that can be taken. The CPU could be stalled (in the case of the ARM600 by stretching the clock cycle) until the old cache line has been flushed and the new one fetched. This stall will last for a large number of clock cycles (especially if the memory server turns the request into an RPC to some other machine) but this is not a problem on the ARM600 which is a fully static design. The alternative is to signal a data abort to the processor, it can then spin polling the memory system but still servicing interrupts that use only local memory. This is an interesting idea, though the ARM600 requires some care to be taken with restarting aborted cycles.

To allow the data abort method, and to allow for cache lines to be updated/invalidated by the cache controller without stopping the CPU, it must be possible to fully isolate the CPU and cache. This cases a few problems as the buffers will add to the delays on data and address paths, but the use of transparent latches should solve this.

The VCI details are stored in a dual ported SRAM. To avoid address collisions the top address bit is generated by a select signal and its inverse. Thus the CPU section can write the VCI table to one half of the memory, and then switch the new table in and the old one out. The switch is done by writing to the control register of the cache subsystem.

The flush-Mode causes a reversal in the operation of the miss control system. If a cache line is hit in flush-Mode then the CPU is halted and the "miss" signal asserted allowing the cache controller to access the memory and flush the line. Thus the cpu could flush a region by doing a read for every 32 bytes of the region. A better way to flush a region (given that it is potentially 1Mb and at most 128k could be in the cache) is to use debug reads of the tag ram (a maximum of 2k reads) and flush any lines that are dirty and valid for the region. If a region is being closed then the tag can be marked invalid using a debug write. *Flush mode is probably broken by the stream cache modifications to the cdecPal. The pin used to pass the information in has been stolen for other uses. It is probably possible to put flush mode back (using Al[29] in the cdecPal) – see the PAL equations and the RCS log.*

Timing information may be extracted from the cache section. The "miss" signal indicates when the CPU is stopped due to a cache miss. Pin 22 of the time2Pal toggles on every access to the cache - i.e. using it to clock a counter gives half the number of cache accesses. Care is taken to ensure sequential accesses are counted correctly even if they cross cache line boundaries (read the equations).

5 Perhaps Interface

The Perhaps interface consists of a single cell transmit fifo and a single cell receive fifo. The fifos are a byte wide, and connect directly to the fabric. The cache controller selects between the two uses, on transmission a cache line request has priority. On reception the port controller route byte determines the destination - cells with the loopback bit set are delivered to the Perhaps interface.

Transmission is achieved by loading the entire cell, including the fabric and port controller route bytes, into the Perhaps fifo then issuing a PTX command to the cache controller. If the fifo contains less than the required 54 bytes (2 route, 4 header[3] and 48 payload) then the final byte in the fifo will be replicated into the end of the cell. If the fifo contains more than 56 bytes then transmission will not take place and an error flag is raised. Otherwise the fifo will *always* be emptied and either the transmission complete or the NACK flag will be raised. Under control of a mask these flags may cause an interrupt through the 68901. *There is currently a bug in the transmit system an additional zero byte should be written to the fifo following the last data byte. Also the clock line appears to be noisy resulting in extra bytes being strobed into the fifo giving AF errors. This should now be fixed, though problems have been seen with fabric investigation cells.* The fifo is byte wide, but a store multiple instruction may be used to write bytes from successive CPU registers.

When a cell marked with the loopback bit set is received it is offered to the Perhaps system. If the receive fifo is non-empty then the cell will be NACKed, otherwise the header and data bytes are written into the fifo. Once a cell has been received a flag is raised which may cause an interrupt, this flag is cleared when the fifo goes empty. *Single reads must be done from the fifo, use of a load multiple will probably fail*[4].

6 Cache controller

The cache controller is built from a xilinx 3064, supported by a xilinx 3020 and some fast PALs. It detects cache misses and controls sending and receiving the cells. This device also

[3] On the fabric there is no HEC byte

[4] it does in simulation anyway!

controls the perhaps interface, in the case of contention for transmission to the fabric the cache section always wins.

The cache controller contains a status register, which can be read by the processor. The bottom four bits of the status register may be used to signal a FIQ to the processor. An interrupt mask register is provided, a FIQ will be requested by any of the status bits with the mask set. The status bits are shown in table 9. Bits marked with an asterisk are latched when the condition happens. Those related to the transmit system may be cleared by a read of the status register with address bit 2 set. The cell received flag is cleared the fifo going empty.

bit	Name	Use
*0	TXAF	The perhaps tx fifo contained more than 56 bytes when transmission was attempted. The transmission was aborted.
*1	RXCE	A cell has been received into the perhaps fifo.
*2	PNACK	During transmission from the perhaps fifo the cell was NACKed. The fifo has been emptied.
*3	TXED	Transmission from the perhaps fifo has finished.
4	FMODE	Set if the cache is in flush-Mode.
5	Spare	Reads as zero.
6	TXEF	The perhaps transmit fifo is empty.
7	RXEF	The perhaps receive fifo is empty.

Table 9: Cache controller status lines

The cache controller accepts commands from the processor. The low four bits of the data indicate the operation, the top four bits are optionally some data.

Command (bits 3..0)	Name	Data bits 7	6	5	4	Description
0	SETM	x	x	x	f	Set flush-Mode f
1	TAGR	none				Reset the tag rams
2	PTX	none				Transmit from the perhaps fifo
3	SETI	m3	m2	m1	m0	Set interrupt mask
4	SWPV	none				Swap region to VCI map
5	SRED	x	x	who	d	Set red led state. If who == 0 then the controller drives the LED, if who == 1 the led is set to d (0=led on).
6	SGRN	x	x	who	d	Set green led state. (see above)
7	FIFOR	none				Reset the Perhaps fifos
8	NOP	none				No Operation

Table 10: Cache controller commands

The cache controller will send a cell to request a cache line when a miss occurs. This cell may also flush an existing dirty line. It expects a cell to be returned containing the data, and the CPU is stalled until such a cell is received. The cell format used is shown in table 11.

The cache controller gets the VCI from the dual port memory, generates the FAS, TYPE and

Byte(s)	Name	Comment	Cache Use
ATM Header			
0	FabR	Fabric route	from VCI ram
1	PCR	Portc Route	from VCI ram
2-3	VCI	Two octet	from VCI ram
4-5	FAS	Two octet FAS	1.192 (start, end bits set)
Cell Header			
6-7	TYPE	Two Octet type.	192.193 (set 0) 192.195 (set 1)
8-9	OP	Two Octet OP code.	192.x (193=rd, 194=wr, 195=both)
Arguments			
10-13	Arg1	First Argument	Read Address
14-17	Arg2	Second Argument	Write Address
18-21	Arg3	Third Argument	not used (value uncertain)
Data			
22-53	Data	Data	Cache line being moved

Table 11: Cache line cell format

OP codes itself, requests the address xilinx to output the Args, and moves the data itself. Arriving cells have their VCI compared to the region and Arg1 compared with the requested address and will be ignored until the correct values are presented. A cell arriving with a port controller route byte of 0x5 are stream cells and will always be accepted and put into the cache. Thus when a connection is set up for a stream the VCI for data arriving at the MDH–9 must have been set to have the low 3 bits equal to the region for the stream (the "CacheStream" QoS in Wanda achieves this). No checking is done on stream cell addresses.

The software memory server accepts a "reopen" cell which may be sent to restart (and reinitialise the memory) a connection on an existing VCI and should also be sent as the first cell on a new connection. It contains details of the cache organisation as shown in figure 12. The server expects an address to consist of (lsb to msb) 5 bits of offset, $lcou$ bits of line, $tcou$bits of tag and $rcou$ bits of region. If $rcou > 3$ then 3 is used. The server expects requests to conform to this address range with the given associativity and may refuse to serve any requests which suggest the cache does not match them. Code to determine the parameters is shown in the appendix.

7 Status LEDs

The card has some status LEDs. The bicolour LED lights red when the processor executes an external cycle to a local address, and green when the processor executes an external cycle to a cache address, the LED will be off if the CPU is executing out of its internal cache. The LEDs connected to the cache controller may be driven by the CPU directly, or are used as status signals by the controller. The normal state (and the reset state) is for the status indication. When under CPU control changing the LED state is a cache controller command, so completes in one cycle - this may be useful for triggering logic analysers etc.

The meanings of the other LEDs are:

Byte(s)	Name	Comment	Cache Use
ATM Header			
0	FabR	Fabric route	from VCI ram
1	PCR	Portc Route	from VCI ram
2-3	VCI	Two octet	from VCI ram
4-5	FAS	Two octet FAS	1.192 (start, end bits set)
Cell Header			
6-7	TYPE	Two Octet type.	192.193 (memory server)
8-9	OP	Two Octet OP code.	0x34.0 (reopen)
Arguments			
10	Arg1[0]	address bits in line	lcou
11	Arg1[1]	address bits in tag	tcou
12	Arg1[2]	address bits in region	rcou
13	Arg1[3]	Associativity	high 4 bits = number of sets, low 4 bits = associativity-1 (0=direct map)
14-17	Arg2	Second Argument	not used (value uncertain)
18-21	Arg3	Third Argument	not used (value uncertain)
Data			
22-53	Data	Data	Not used (value uncertain)

Table 12: Reopen memory cell format

Grn[0]	Grn[1]	Grn[2]	Grn[3]	Red[0]	Red[1]	Red[2]	Red[3]
Sending a cell	68901 grn	Xilinx grn (Receiving)	Power	Fetch Cache line	68901 red	Xilinx red (flush-Mode)	VCI map state

8 The 68901

The 68901 "MultiFunction peripheral" device is on the CPU node to provide the console serial line, and the Wanda timer. It also gives some general I/O lines, a "watch dog" reset line and a spare timer. Four of the I/O lines are used, two for LEDs and two for the reset reason, table 13 shows the gpio use. Note that the port defaults to all inputs on reset.

The 68901 timer crystal is 2.4576MHz. The inputs to timer A and B are selectable by links, and the output of timer B may be connected to cause the card to reset. Table 14 summarises the timer use.

The 68901 is the only device on the cpu IRQ line, and expects the 68k interrupt response. This is provided using the "IACK" access address. Reading from this address causes the IACK strobe rather than CS. The 68901 will then provide the reset vector formed as given in the data sheet. **Warning** when the 68901 is not interrupting it will not respond to a read of this location. Currently the machine will hang if such an access is attempted – a patch will probably be done to make the access return with bogus data.

Bit	Name	Use
0	Red68	Red LED
1	Grn68	Green LED
2-5	Spare	Not used
6-7	rr[0..1]	Reset reason.
	00	Reset switch
	01	WatchDog reset
	10	Fabric reset
	11	Cache reset

Table 13: 68901 I/O port use.

Timer	Input line	Comments
A	Cache control xilinx or "Ext" pin	General timer for experiments
B	Timer A out or Timer C out	Watch Dog. May also provide extension to timer A or C (watchdog link disabled)
C	(only internal source)	Wanda Timer - for clock interrupts
D	(only internal source)	Baud rate clock - both Tx and Rx

Table 14: 68901 timer use.

9 Links

There are ten configuration links on the PCB. Their use and "normal" setting is shown in table 15. In addition there are several ground pins for use when debugging. All of the 68901 I/O lines are brought to pads.

Link	Normal	Use
1,2	In	Connect reset reason to 68901
3	East 'S'	Rom speed.**F**ast (150ns)/**S**low (250ns)
4	East 'Ca'	TA clock.**Ca**che xilinx or **Ex**ternal
5	South 'Tco'	TB clock.**Tao** or **Tco**
6	East 'Tb'	WatchDog source.**Tb**o or **Ex**ternal
7	West 'B'	Rom size.**B**ig 27512/271024 or **S**mall 27256
8	?	Adr Xilinx program mode.**M**aster or **S**lave
9	?	Adr Xilinx program source. **D**out or **P**ROM.
10	?	Adr Xilinx Cclk link. In for slave mode

Table 15: PCB Links. (card front west)

10 BOS software

The ROMs on the MDH–9 currently contain an ARM version of the BOS monitor program found on most of the previous MDH–x machines. This is written to run with only the CPU, ROM and serial line working, so may not use a stack or scratch RAM. This version was written with the aid of an assembler and eprom eraser – which is regarded as cheating by purists. The commands available are shown in table 16. BOS will accept either upper or lower case

Command	Name	Use
		Standard Commands
.	On	Step to next memory word
,	Back	Step to previous memory word
;*hex32*	Deposit	Write word to current location
'*hex32*	Deposit	Write word to current location
:*hex32*	Dep Next	Write word to next location, step on
hex8	Dep Byte	Write byte to current location
G	Go	Run code from current location
M*hex32*	Move	Move to given location
D	Dump	Dump 64 words starting at current location to screen
N	DumpNext	Step on 64 words and execute **D**
		Booting Commands
B	Boot	Boot using Wanda Serial SBOOT protocol to 0x82000
W	Wanda Boot	(*reserved for Wanda Boot*)
X	Xboot	Extended Boot. Locate memory server, request service and start execution at 0x00300000 (Region 0 location 0))
		Machine Specific Commands
L	Loop	Turn LEDs on, run infinite loop, turn LEDs off
T	TestMem	Run tests on local SRAM, VCI SRAM and cache memory
V	Vectors	Set execption vectors to trap to BOS
E	Extended	Set ARM600 to 32 bit addresses

Table 16: Commands in MDH–9 BOS

ASCII, and always echos upper case. No CPU register access commands are implemented. When code is executed with the **G** command the return address is passed in r14, and the value returned in r4 is used as the current location.

The behaviour of the machine specific commands is:

L This command is for testing the processor. If the CPU has a poor clock then instructions may be misexecuted. To check this the 68901 LEDs are turned on and the CPU is put into an infinite loop. If the loop terminates, as happens when a bad clock is supplied, the LEDs are turned off and control is returned to BOS. The LEDs turning off may be used to trigger the logic analyser.

T Tests are run on the 512k of on board SRAM. Firstly "data" tests are performed to check that each location will store words correctly. Then the "words" test checks for addressing problems to ensure each location is unique, finally a similar test is performed to check with byte accesses. Should a test fail BOS is reentered with the fault location set as the current location. If these tests are passed then the cache investigation code is run, determining the quantity of cache memory and testing it.

V The ARM exception vectors are in ROM on the MDH–9, the ROM locations are set to jump to secondary vectors in RAM at the same offset from the base of RAM as the vectors usually are. Normally BOS does not use any vectors and exceptions will not be handled, following the **V** command the vectors are set to print a message and restart BOS, see table 17.

Name	ROM Address	RAM address	BOS message
Reset	0	N/A	Welcome to ARM BOS
Undefined	0000 0004	0008 0004	Bos-undef!
SWI	0000 0008	0008 0008	Bos-swi!
Prefetch Abort	0000 000C	0008 000C	Bos-prefetch!
Data Abort	0000 0010	0008 0010	Bos-data!
Address Exception	0000 0014	0008 0014	Bos-addr!
IRQ	0000 0018	0008 0018	Bos-irq!
FIQ	0000 001C	0008 001C	Bos-fiq!

Table 17: Exception vectors and BOS messages

E The ARM 600 starts up in 26 bit program and address space, and attempts to access addresses outside this range will address fault. This command enables 32 bit *data* accesses but leaves the core in 26 bit program mode. If neither this command nor the vector command has been given then accesses to any devices other than ROM, local SRAM and the 68901 will cause the processor to hang on an untrapped address violation.

B The boot command runs a standalone version of Wanda Serial Line Boot. This expects an xwcons[5] to be at the other end of the line. The code is loaded starting at address 0x82000 and is executed when the boot protocol succesfully terminates.

X The extended boot command causes the cache subsystem to be investigated. If a valid cache is found then the fabric will be investigated and connection requests for localhost.memoryserver sent to all ports. When a reply is received region zero of the cache is connected to it, the MMU set so 0x00300000-0x003FFFFF maps to region zero, and code is run at 0x00300000.

11 Pcb and PALs

The silk screen printed layer of the MDH–9 PCB is shown in figure 4. This has had the chip names added to ease identification. The CPU is held on a separate daughter card. This allows the PQFP device to effectively be socketed, and also terminates the unused coprocessor bus lines.

[5] Or anything with serrecv.o

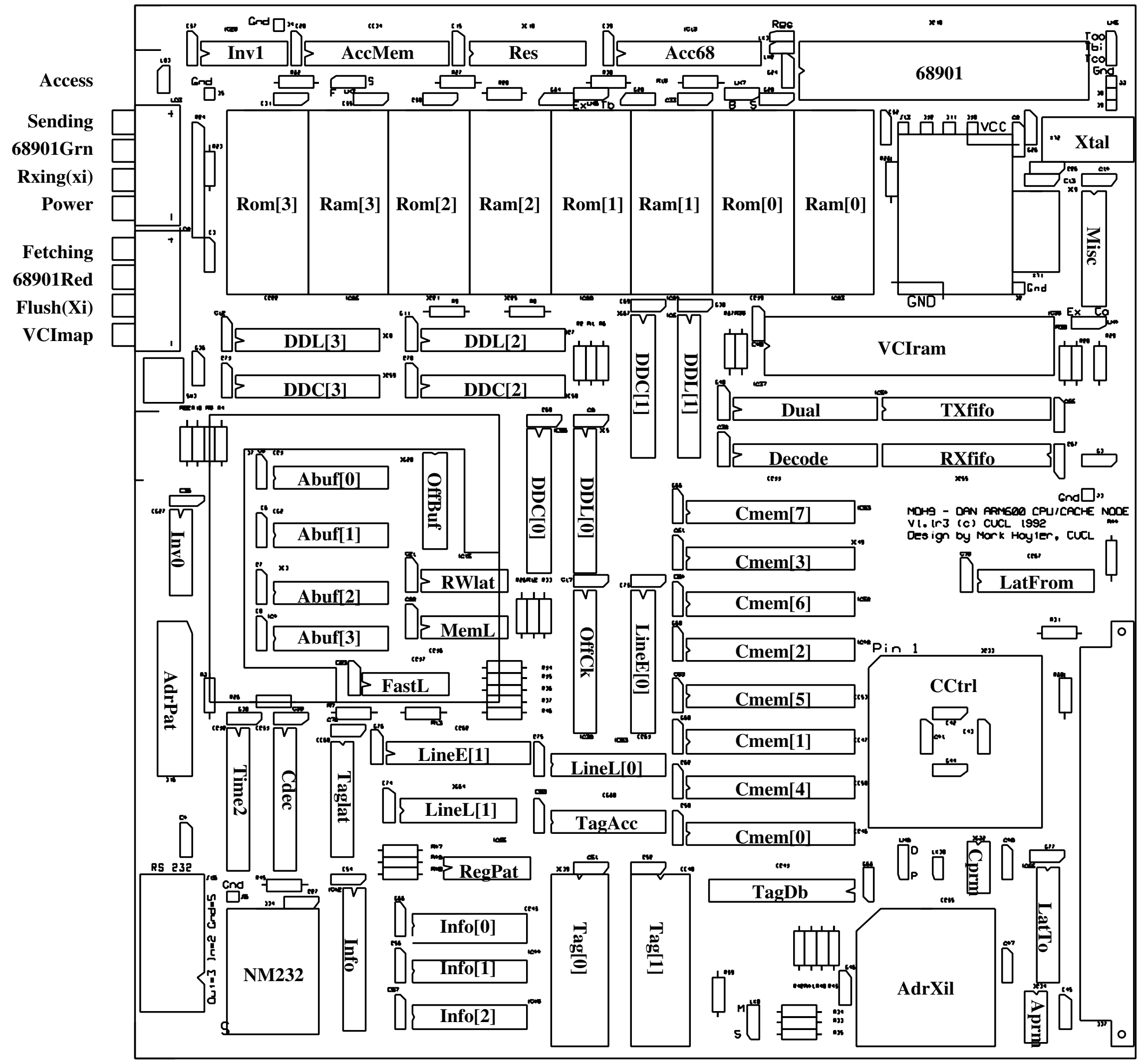

Figure 4: MDH–9 PCB chip locations

Most of the logic on the MDH–9 is provided by PALs or Xilinx FPGAs. The two xilinx devices are a 3064 which acts as the cache controller, and a 3020 which serves as the address multiplexor and comparator. Much functionality has had to be moved into devices with faster I/O timings, so the controller xilinx is supported by PALs and fast TTL logic. The PALs used on the card are:

Acc68 22V10 — Controls access to the 68901. Generates the data strobe and either CSb or IACK, and controls the asynchronous access. This PAL also combines the remove wait signals to feed the fast wait logic.

Accmem22V10-10 — Times access to ROM and local SRAM. Also generates the SRAM write pulses.

Cdec 22V10-10 — Cache Decode PAL. Enables the cache databus buffers, decodes the address and miss information to give the access mode and the cache set required. Also generates the update signal for the info system.

Decode 22V10-10 Decodes the address lines for all local devices (ie with address bit A31 clear). The ROM and RAM chip selects are also generated.

Dual 22V10 Under control of the cache control xilinx this PAL produces the address lines for fetching the VCI from the dual port ram.

Info 22V10-10 Manages reading and updating the cache information – the dirty and LRU bits.

Misc 16R8-7 Divides down the crystal to produce the CPU clocks and the 4MHz reference clock for the 68901.

OffCk 22V10-10 Generates the read and write enables for the cache RAMS and checks the offset for crossing a line boundary.

Res 16V8 The reset PAL generates the reset pulse in responce to any one of the four reset sources requesting. The reset reason is output, and may be read via the 68901.

TagAcc 16V8-10 Generates the read and write enables for the cache tag rams and the address latches.

Time2 22V10-10 Detects the need for the CPU to wait and starts the fast logic. Also generates the drive signals for the bi-colour access LED and the data bus buffers.

12 Appendix – code to size/shape the cache

This code uses "debug" accesses to the cache subsystem to work out the size and shape of the cache to pass in a cache reopen cell to the memory server.

```
/* Figure out cache configuration */

/* Check set 0 for 32 bit wide R/W memory */
if ((set0 = probedata((volatile unsigned long *) 0xE0000000,
                      (volatile unsigned long *) 0xEFFFFFFF)) != 0)
{
    outstr("Cache set 0 error. Data probe gives ");
    outhexnum(set0);
    outstr(".\n");
}
/* Check set 1 for 32 bit wide R/W memory */
if ((set1 = probedata((volatile unsigned long *) 0xF0000000,
                      (volatile unsigned long *) 0xFFFFFFFF)) != 0)
{
    outstr("Cache set 1 error. Data probe gives ");
    outhexnum(set1);
    outstr(".\n");
}

/* If neither set 0 nor 1 has 32 bit wide R/W then we can't go on */
if (set0 && set1)
{
    outstr("The cache has failed\n");
```

```
    return;
}
TRC(outstr("Check adr\n"));

/* Now see how big the cache memory is by looking at *ACTUAL* RAM*/
/* The idea is that since the lines will form a contiguous block */
/* the result from the probe will have low bits valid == line    */
/*                            next a block don't care == tag     */
/*                              finally a block valid == region  */
probeadr((volatile unsigned long *)0xE0000000,
         0x1C, 0xffffffff, &aerr, &asame);
/* lrbits is the valid bits - low valid is line, top valid is region */
lrbits = aerr | asame | 0xf0000000;
TRC(outhexnum(lrbits));

/* zeros -> bit */
TRC(outstr("find bits\n"));
/* count line bits == the number of low valid bits */
for(lcou=0, bit=32; ((lrbits & bit)==0);lcou++,bit<<=1);
/* skip over tag bits which aren't real in the debug memory */
for(tcou=0; (bit != 0) && ((lrbits & bit)!=0);tcou++,bit<<=1);
/* the top valid bits are the region ones */
for(rcou=0; (bit != 0) && ((lrbits & bit)==0);rcou++,bit<<=1);

/* Having figured out the size work out associativity */
if (set0 || set1)
    sa = 0x10; /* One set, direct mapped */
else
{
    /* Both valid, must work out if adr bit selects lru     */
    /* There are three cases:                               */
    /* a) Direct mapped, lru=0 only use set 0               */
    /* b) Direct mapped, lru=tag[4]=a17, set = a17          */
    /* c) 2way set assoc, lru=lru                           */
    volatile int *tag0 = (int *) 0xC0000000;
    volatile int *tag1 = (int *) 0xD0000000;
    volatile int *tag0a = (int *) 0xC0020000;
    volatile int *dat = (int *) 0x80000000;
    int x;

    /* First find out if it updates lru, which it only will */
    /* if it is two way assoc. The plan is to see which set */
    /* is marked as lru, set the tags to ensure a hit on it */
    /* and see if the lru bit has changed                   */
    *CCTRL_CMD = TAGR; /* NB this only resets the tag memory */
    if ((*tag0 & 0x200) == 0)
    {
        /* Now showing lru 0 */
        *tag0 = 0x80;
        x = *dat;
        if ((*tag0 & 0x200) == 0)
            /* No change -> direct map */
            sa = 0;
        else
```

```
            sa = 1;
    }
    else
    {
        /* Now showing lru 1 */
        *tag1 = 0x80;
        x = *dat;
        if ((*tag1 & 0x200) != 0)
            /* No change -> direct map */
            sa = 0;
        else
            sa = 1;
    }
    /* Now we know if it is direct or set assoc */
    if (sa)
        sa = 0x21;  /* 2 sets, assoc */
    else
    {
        /* In the direct mapped case check for two sets     */
        /* by looking to see if the lru bit depends on A17  */
        if ((*tag0 & 0x200) == (*tag0a & 0x200))
            sa = 0x10; /* 1 set, not assoc */
        else
            sa = 0x20; /* 2 sets, not assoc */
    }
}

outstr("Cache has ");
outdecnum(lcou);
outstr(" bits line, ");
outdecnum(tcou);
outstr(" bits tag, ");
outdecnum(rcou);
outstr(" bits region, ");
outhexnum(sa >> 4);
if (sa & 1)
    outstr(" set associative");
else
    outstr(" set direct map");
outch('\n');

/* Need to tell the memory server this */
reopenCell[10] = lcou;
reopenCell[11] = tcou;
reopenCell[12] = rcou;
reopenCell[13] = sa;
```