Fairisle Port Controller
Xi2 Firmware and Software

Mark Hayter and Richard Black

February 16, 1993

This document describes the Xi2 xilinx design used on the second revision Fairisle Port Controller. This was the first design in general use, and experience with its use has led to the later designs. The general outline of the port controller is given in a separate document.

1 Xi2 cell format

The cell buffer format used by the Xi2 bits is shown in table 1. The format is known as the "4-2-4" cell format. The buffer SRAM also contains a VCI translation table, which consists of the last word of every cell buffer. This word contains the data that will be sent as the first word of a cell that is not marked local and has incoming VCI equal to the buffer number.

Byte	Name	Comment
0	FabR	Fabric route byte (For local cells only)
1	PortR	Portc route byte (For local cells only)
2-3	VCIout	VCI transmitted to the fabric (For local cells only)
4-5	VCIin	VCI received from the network
6-7	FAS	Framing and Sequencing bytes
8-55	Data	User data bytes
56-59	Spare	Unused, only exist in port controller buffer
60-63	Remap	VCI table entry (format as bytes 0-3) for VCIin == buffer number

Table 1: Cell Buffer Format

The data from the network is bytes 4 to 55 of the cell buffer, the routeing tag being added by the port controller. Bytes 0 to 3 and 6 to 55 are sent into the fabric, which passes bytes 1 to 55 to the output port controller. The route byte is replaced by a start of cell control signal for transmission on the network. The network interface card may insert the header error check byte (HEC) as the fifth byte it sends, this byte is removed by the receiving network interface card. The spare bytes are never used by the network section, so are free for internal use by the processor section.

Figure 1 shows which bytes are written and read by the send and receive systems. In each case the blank locations are unused and may hold any value. The incoming data is written as a contiguous block in words 1 to 13 of the buffer. Data being injected into the

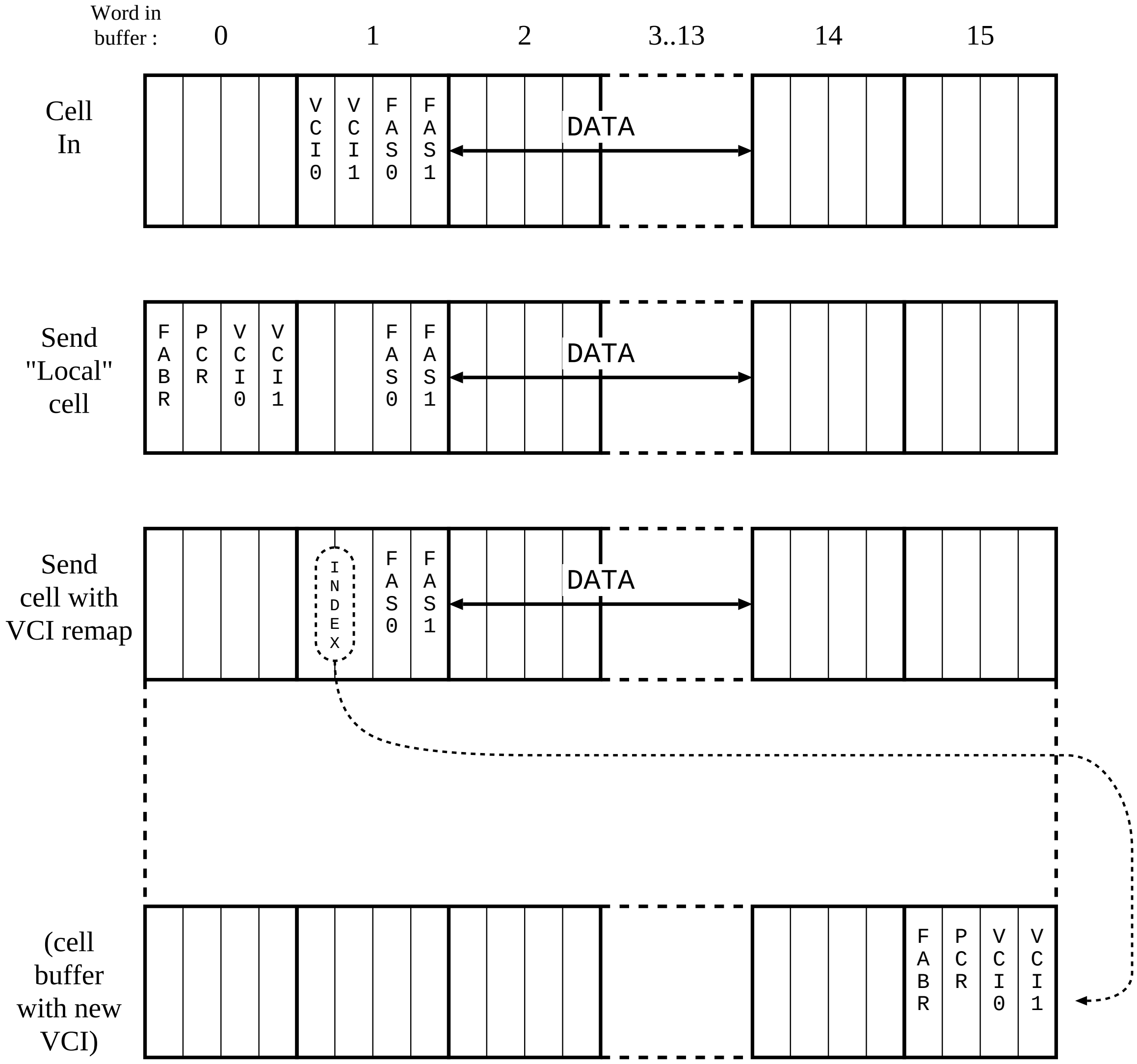

Figure 1: Cell buffer bytes accessed

fabric is in one of two forms, "local" where the cell has originated on the port controller and contains the route tags and outgoing VCI, and "forward" where the cell has arrived from the transmission system and is being forwarded into the fabric with the incoming VCI being replaced by the routeing and outgoing VCI. In the "forward" case the new information is looked up by using the old VCI as an index into the cell buffer table.

2 The xilinx DMA controller

The xilinx chip, which controls the network part of the card, maintains two queues of buffer pointers into the SRAM. Buffers in the first queue are filled by incoming cells from the network, the processor then inserts a routeing tag and outgoing VCI and adds the buffer to the second queue for injection into the fabric. The processor may be interrupted if either of the queues are not full of buffer pointers (indicating cell arrival or transmit availability), or a NACK is received from the fabric. Figure 2 shows the various parts of the DMA system and gives an outline of their interconnection. All the parts inside the grey border are on the xilinx device.

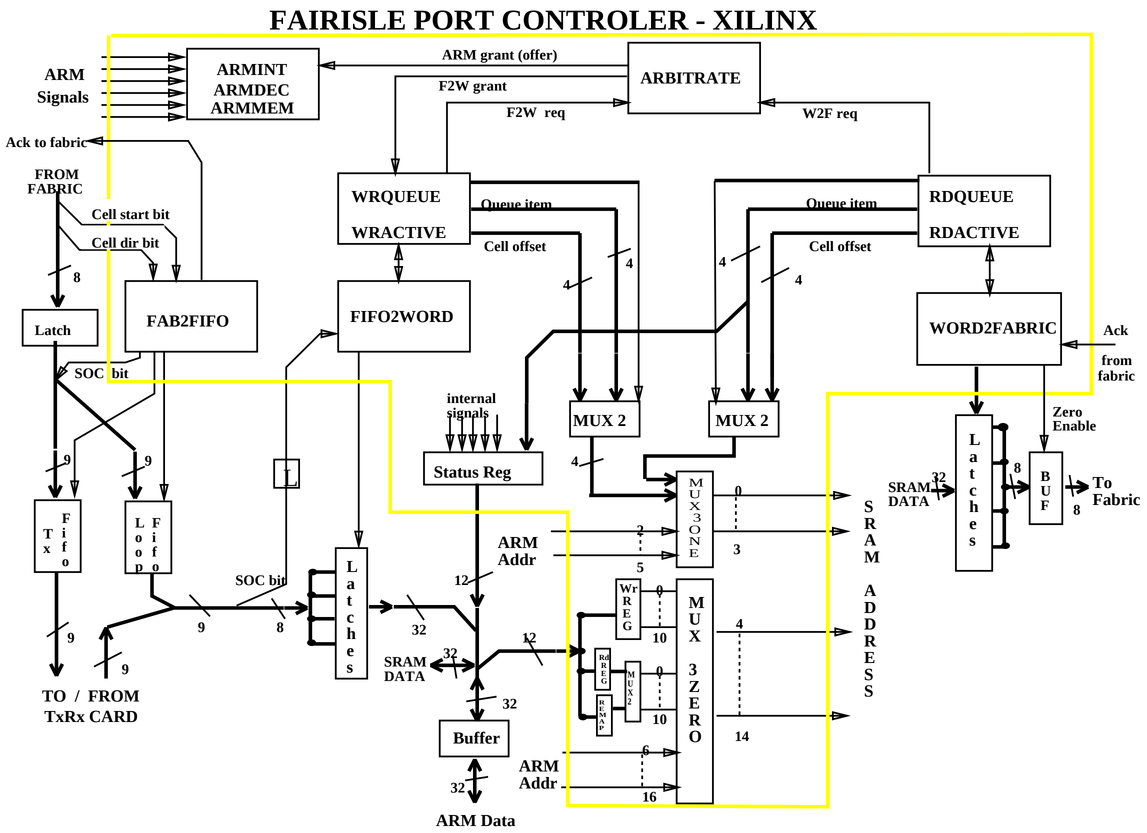

Figure 2: Port controller DMA system

2.1 SRAM interface

The xilinx is the only device which drives the SRAM, arbitrating between its DMA use and the processor. The arbiter assigns top priority to the fabric transmission requests, these are always directly granted so no grant signal is needed. Network reception requests have next priority and will be granted unless blocked by fabric requests. Any unused memory cycles are offered to the processor interface section which may use them. The grants (or offers) are generated one cycle ahead of the cycle memory may be used - thus if a fabric request happens in cycle n memory may be used by the word2fabric section in cycle n+2. The access grants control a multiplexor which drives the SRAM address bus. Enables of SRAM data bus drivers are also controlled by the arbiter. Only part of the SRAM data bus is used directly by the xilinx, to load the buffer registers and output its internal status. The width of data bus to the xilinx depends on the part used, for a 3042 data lines 0..11 and 30..31 are available, a 3064 also has lines 12..15 connected. When a cell header is read the VCI will be on the bottom 16 bits. If the SAR1A or FAS protocols are used then the start and end bits will be on bits 30 and 31.

The SRAM address is formed from registers within the xilinx as shown in figure 3. A two stage multiplexor is used, the first stage is under control of one of the subsections of

SRAM address		14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Load Read Buffer Reg		0	0	0	0	0	0	0	0	0	0	0	0	QueuePtr		
Load Output latches or Remap Reg		Read Buffer Regster 0..10											Active offset			
Load output latches during VCI remap		Ramap Register 0..10 == vci_in 0..10											1	1	1	1
Load Write Buffer Reg		0	0	0	0	0	0	0	0	0	0	0	1	QueuePtr		
Store Input latches		Write Buffer Regster 0..10											Active offset			
Arm access		Arm address 2..16 (word accesses only)														

Figure 3: Formation of the SRAM address

the xilinx and presents the address for an access by that subsection. The second stage is controlled by the arbitration unit and selects which subsection is granted the bus.

2.2 Injection to the fabric

The xilinx will start sending cells into the fabric if there are buffers in the transmit queue and the processor has strobed the RamRdGo line since reset or a fabric NACK. Transmission continues until the queue is empty, or a cell is NACKed by the fabric or destination port controller. If transmission stops because the queue becomes empty then it will restart as soon as a new item is put in the queue – strobing of the RamRdGo line is only required after NACK or xilinx reset.

To send a cell the "word2fabric" section of the xilinx first reads the next queue entry from the SRAM into the RdREG register. The buffer pointer contains a local bit. If this bit is set the cell is assumed to have originated locally and the VCI is not translated. If VCI translation is taking place then the old VCI is read from offset 1 in the cell buffer into the remap register. The SRAM is then read, either from the Remap address or from offset 0 in the cell buffer. This will latch the routeing and outgoing VCI information in the output latches. Once the first word has been sent the word from offset 1 in the cell buffer is moved to the output latches, but only the upper two bytes are transmitted. These bytes contain the FAS information for the cell. The twelve data words are then sent. Except during the remap cycle the SRAM address is formed using the register value as the top part of the address and the read active counter as the offset into the buffer. Data from the latches is clocked into the fabric a byte at a time. As the last byte of each word is being sent the next word is fetched, to ensure a continuous stream of data to the fabric. This can be done because the arbiter gives these memory reads top priority, and thus the memory grant can be predicted. The fabric uses the bottom bit to mark the start of the cell, so an external buffer is used to force this bit to a zero when no cell is being transmitted.

The Remap register is formed from the output latches of the word2fabric address multiplexor. This saves xilinx logic blocks, but means that the address is only held for one cycle. It is read in one cycle and used in the next, this is possible because the the bus grant is guaranteed.

The Ack signal from the fabric is monitored just after the header has been sent. If a NACK is seen at this point then transmission of this cell is stopped at once. If the cell buffer pointer has its retry bit clear or the cell has been retried already the transmission system is disabled. The processor can be interrupted by transmission stopping and can determine which cell was rejected (from the status register) then implement the correct retry strategy.

2.3 Fabric output to network

The details of the output of fabric data to the network interface are as described in the document on the generic port controller.

2.4 Reception from network or loopback fifo

The Xi2 generation design of the xilinx firmware implements cell reception as described in the generic port controller document except that it has a potential jam condition. This is caused because short cells are only detected by the arrival of the next Start-of-Cell indicator. If a short cell arrives in one of the fifos and there is nothing following it then the xilinx will stick waiting for the rest of the cell, and the other fifo will be allowed to fill. It may be that a cell read should timeout with a short cell indication after some number of frame pulses without new data.

2.5 The processor interface

The processor control signals are decoded and I/O accesses to the xilinx space (0x3180000 - 0x31FFFFF) are detected. The use of I/O space causes the access cycle to be asynchronous, allowing the xilinx to arbitrate the memory access. The SRAM is mapped into this area for 32 bit word accesses. The top address bits allow modifications to the access method. A basic access allows reading or writing of the memory. If the strobe bit is set then on a write operation one of the queue insertion lines will be strobed, bit 5 determining which. The final access method is a read with the status bit set, this causes a fake memory access with the data being supplied from the xilinx status register. During a status register read address bit 5 indicates whether latched status bits should be cleared. The Xi2 design uses the "slowarm" interface.

The memory map is shown in table 2. The first 8 words of the SRAM are the output queue, the next 8 the input queue. Both are circular, with the processor inserting buffer pointers, and the xilinx consuming them. It is the responsibility of the processor to ensure a new buffer pointer is stored in the correct memory location to be added as the tail of the queue. There is no protection against the processor strobing pointers into a full queue - if it does it will overwrite an existing buffer pointer but the internal tail will **not** advance. The processor can cause a queue strobe signal by writing anywhere in the memory with the appropriate high order bits set.

The SRAM is divided into cell buffers. The zeroth buffer contains the queues, the rest are available for cell DMA. The mapping from buffer pointer to buffer address is simply a left shift six bits of the bottom eleven bits of the pointer. The higher order bits of the

Start	End	Use
0x3180000	0x319FFFF	Read / Write access to SRAM
0x31A0000	0x31BFFFF	Read xilinx status word, write SRAM (bit 5 -> clear)
0x31C0000	0x31C001F	Read SRAM / Write output queue with strobe
0x31C0020	0x31C003F	Read SRAM / Write input queue with strobe
0x31C0040	0x31DFFFF	Read SRAM / Write SRAM and strobe (bit 5 -> in/out)
0x31E0000	0x31FFFFF	Read status, write with strobe

Table 2: Memory map

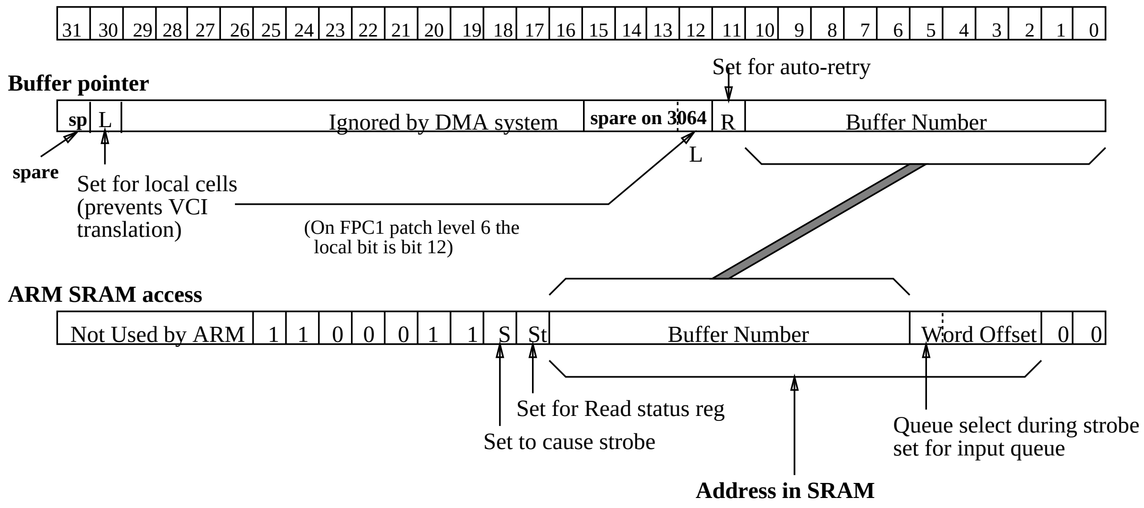

Figure 4: Buffer pointer and memory addresses

pointer are not used by the DMA system, so may be used by the control software. Note that it is the buffer pointer, and not the buffer address, that is stored in the queue. Figure 4 shows the buffer pointer value and an ARM memory address and their relationship to the SRAM address (which accesses a long word).

In the transmit queue the buffer pointer is supplemented by two control bits "retry" and "local". If the retry bit is set then the DMA system will do one automatic retry should the cell be NACKed on the first attempt. If the retry fails then transmission stops and the processor is interrupted. If the retry bit is clear then the system stops on the first NACK.[1] If the "local" bit is set then VCI translation will not be performed on the cell.

3 Status and Control

The xilinx status can be found from its internal status register, and from signals passed to the processor section I/O controller (IOC). The lines on the IOC can be set to cause a high priority, low latency, interrupt (a FIQ) to the processor. Many of the general signals are specified in the generic port controller documentation, however the Xi2 status register

[1] I have no idea if this is useful, but it is there to be tried. The number of retries could be increased easily.

and meanings for the FIQ interrupts may be found in tables 3[2], and 4 respectively.

bit	name	use
0	RamRdReady	Set if there are buffers to send. Clear if the queue is empty.
1	RamRdFull	Set if output queue is full
2-4	RdTailAdr	Buffer queue number for cell being or about to be sent. Only useful if sending has stopped, when it contains the queue number for the next cell to be sent.
5	Spare	Always zero
6	SOCseen	Set if Start of cell mark has been seen in a cell, Cleared by status register read-and-clear
7	StrayBytes	Set if bytes seen from the fifo which were not part of a cell, Cleared by status register read-and-clear
8	Spare	Not used - always zero
9	RamRdRun	Set if output system is ok, cleared by reset or fabric NACK.
10	RamRdOKbar	Set if not sending
11	CellTOG	Set if a cell was discarded due to lack of buffers. Cleared by status register read-and-clear. Always provokes StrayBytes
12-13	Spare	Not used. May read as any value. (FPC1 patch 6)
14-15	Spare	On 3064 not used, may read any value. On 3042 undefined
16-29	-	Undefined. May read as any value.
30-31	Spare	Not used. May read as any value. (FPC1 patch >6)

Table 3: xilinx status register

The xilinx status register has two main functions, to indicate unusual or error conditions and to allow the processor to determine why sending to the fabric stopped. If the system is not sending (and has been) the RamRdRun bit indicates if it stopped because the buffer queue was emptied or because of a fabric NACK. The RdTailAdr gives the RdQueue index into the output queue. This index is for the next cell to be transmitted, so if transmission was stopped by a NACK it is the previous queue entry which contains the buffer pointer for the rejected cell. Since the index value may be incremented part way through transmission of a cell it only gives useful information if the system is not sending. Some of the data bits are cleared after a status register read-and-clear, this is a read with address bit 5 set. These all latch unexpected conditions (such as short cells) seen by the xilinx, and will probably be of relevance to higher level software (for statistics gathering if nothing else). The low level software (eg the FIQ) should therefore not use the read-and-clear address.

The IOC RamRdGobar line should be strobed low to start cells being sent after the xilinx is reset or a NACK has been received, it should be set high (ie undriven) again immediately. Extra strobes of the RamRdGobar line may cause NACKs to be missed (since they could be cleared before the processor has noticed them).

The three FIQ interrupt request lines which can be generated by the xilinx device are described in table 4. The status of these lines is always software readable and they may

[2]The spare bits may be used to bring out other internal signals if required. The undefined bits are not available.

be individually masked.

bit	Name	Use
0	Xifiq[0]	Set on input queue not full
1	Xifiq[1]	Set on output queue not full
2	Xifiq[2]	Set if output has stopped

Table 4: IOC FIQ Status register

4 The current Xilinx bits - "bum3"

There are various versions of the released Xi2 design firmware for 3064-100. These are all perturbations of the "bum3" Xilinx design. The "bum3" design works with an 18MHz fabric clock, using clock distributed over the backplane. The slow version of the ARM interface is used, so all clock speeds for both processor and fabric should work. The design has been in use for a while, and the following bugs have been observed:

- The VCI remapping does not work. This is almost certainly due to the remap enable signal "w2fenR" being one cycle late, and the fault shows up under simulation. The fault probably crept in when the address bus pipeline was fixed up, so the relationship of the two should be examined.
- The spare 3064 status bits are not connected by the Xilinx, the pins being marked "prohibited locations".
- The "stray bytes" status indication has been seen to go away without a read-with-clear cycle. This was seen during testing of the Null port controller, so it is possible that there were stray bytes, but no reason can be seen for the status indication being removed.

The following alternate bits are available, being successive patches to "bum3":

bus1 — These bits have a fix for the remapping problem, but they only work at 16 MHz.

buth1 — These bits have a change to the slowarm memory interface so that they work with the Arm600 as well as the Arm3. Unfortunately they only go at 15 MHz.

bomo1 — These bits have a patch to the fab2fifo section so that they will work with a two stage fabric. The speed rating is probably 15MHz.

5 The interrupt handler

The design and implementation of the interrupt handler software changed repeatedly and significantly over the design life time of the Xi2 hardware. The details below are for the stable code now used on all Xi2 hardware. Many of the other designs and reasoning for features and changes may be found in the port controller document in the original yellow book - "Systems Research Group Technical Note 29th Feb 1992".

The FIQ is responsible for the flow of data through the port controller using the Xilinx device as a fancy DMA engine. The FIQ's tasks can be divided as follows.

- It must transfer "forward" received cells to the transmit queue for injection to the switch fabric, inserting the routeing tag and performing VCI re-mapping.
- It must keep the Xilinx receive queue full to avoid packet loss.
- It must bring management and connection setup cells to the attention of the higher level Wanda system.
- It must merge management and connection setup cells being sent by the Wanda code with cells being forwarded.
- It must maintain statistics on forwarded cells to permit the Wanda code to perform policing and quality of service operations.
- It must restart injection under circumstances of Fabric NACK.

5.1 Issues

5.1.1 Receiving

The Xilinx's receive queue is of size eight. The Xilinx will generate hard back-pressure across the loop-back fifo but will drain the receive fifo if this queue goes empty. It is thus important to keep it as full as possible.

5.1.2 Transmission

In general when transmitting into the fabric there will not be enough data to keep the Xilinx transmit queue full. Most of the time therefore it will be necessary to keep the transmit slot available interrupt masked out, but to examine its status when requiring to transmit something.

5.1.3 Nacks

Because the Xi2 design bits step the transmit queue pointer forward when a NACK condition arrises a NACK interrupt will never occur without a simultaneous transmit available interrupt. Care must be taken when dispatching interrupts that a NACK interrupt does not therefore get processed as if it was a Transmit available interrupt. Likewise when processing a NACK interrupt care must be taken because there could have been successful transmissions whose interrupts were not handled before the NACK interrupt occurred.

5.1.4 Delay Free

A cell buffer is not available for reuse until the data in it has in fact been transmitted into the fabric. This means that the buffer cannot be freed when it is placed into the transmission queue on a transmit available interrupt. In general the software has no way of knowing exactly when the transmission has successfully taken place, since it does not always have enough data to fill the transmission queue. This is known as the "Delay Free" problem.

5.2 Data Structures

There are three data structures in the Xi2 FIQ software. They are the cell queues, the free list and the forwarding map.

5.2.1 Cell Queues

Cell queues are used to put cells on when they arrive, and to take them off when they are to be transmitted. Some of these queues will be both generated and consumed by the FIQ code, and some will be generated or consumed by the Wanda networking code.

The queues must be designed to be as efficient as possible. Since the Wanda networking code runs asynchronously with the FIQ code it would be nice if the data structure could be designed such that the Wanda code would not need to disable FIQ interrupts while manipulating the queues. This is aided by the fact that a queue is never both produced and consumed by the Wanda code. Also on the transmission side when selecting from multiple queues it should be possible to test rapidly if the queue is empty. Finally access to the Xilinx chip memory is sufficiently slow that no pointers should be stored with the data in the SRAM.

Many different data structures were tried but the one that was finally settled on was to have a map[3] of index values and to implement the lists of buffers by chaining in the map. Associated with each queue were kept two index values, both the head and the tail. Additionally there is always a single buffer associated with every queue. This allows the producer and consumer to be decoupled as only the producer may update the tail and only the consumer may update the head.

The spare buffer is always considered to be at the head of the list, this is because when the data arrives it will be placed into a buffer by the Xilinx chip before the software knows what queue it will be put on. This means that when a buffer is logically removed from a queue the buffer number that is at the head of the queue must be freed and the data must then be consumed from the buffer which has become the head of the queue. This interacts subtly with the delay free problem and the free list.

Since the head and tail pointers consist of a single index value an update is a single atomic processor write instruction. It is thus possible to allow a queue user to *read* the value of the other queue pointer. This has the double advantage that an empty queue can be detected by comparing the two pointers for equality, and that no "nil" pointer value ever need by written which reduces the cost of adding a buffer to a queue. Note that the two queue pointers can also be read using an arm load multiple instruction.

5.2.2 The free list

Much time was spent in manipulating the free list. This was mostly related to the delay free problem and the associated complexity of a subsequent buffer being consumed from a queue (and hence the previous one being freed) while the data of the previous one was potentially still being transmitted.

Eventually the structure displayed in figure 5 was devised to allow the free list and the delay free to be managed with fewer writes. It relies on the observation that provided buffers freed by the Wanda networking code are not considered it is merely necessary to ensure that buffers are allocated in fifo order and that the fifo is considered empty when there are still eight (the size of the Xilinx transmit queue) buffers in it.

[3] This map is stored in the 2048 words occupying addresses 0x2000 to 0x3fff

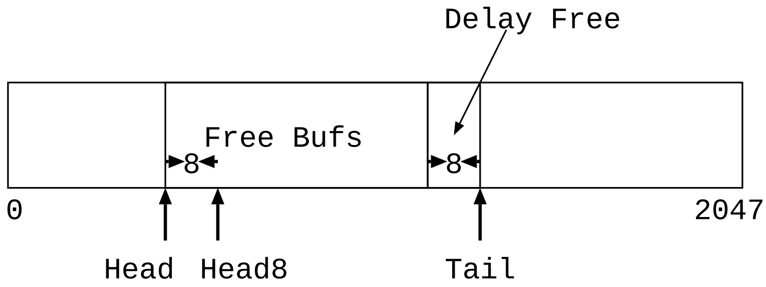

Figure 5: New arrangement of the free list.

A circular buffer is used.[4] There are three logical pointers into the buffer. They are "head", "head8" and "tail". In fact head8 is always generated from head when required since it is a simple add with wrap operation. Buffers are freed at the tail in the expected manner. However the free list is considered to be empty when head8 reaches tail rather than when head reaches tail. This prevents a buffer being allocated when it is still in the delay-free case. Additionally the head and tail values are kept in a FIQ state register so do not need to be stored to memory.

This change reduces an allocation to zero write cycles and a free to a single write cycle.

These changes mean however that the Wanda code must disable FIQs and enter FIQ mode when allocating or freeing a buffer. Also the Wanda code must *always* perform its manipulations at the head of the list since it is not involved in the delay-free procedure.

Disabling the FIQs when Wanda needs to manipulate the free queue is allowable even though it is not justified in the case of queue manipulation because the time taken to execute the code is very short and using this method allows the FIQ code to run much faster in general.

5.2.3 The forwarding map

The forwarding map has two forms depending on whether the Xi2 design in use has working remapping capability.[5] It is indexed by VCI when each cell arrives to determine what should be done with that cell. The following must be encoded:

- The queue on to which the buffer is to be placed. This is required for all entries in the map.
- Whether to reconsider servicing transmit available interrupts. As explained earlier normally the Xilinx transmit queue cannot be kept full and so the transmit available interrupt is frequently disabled. If the data is to be forwarded this flag must be set to cause dispatching of this interrupt.
- Whether to post a normal IRQ interrupt to Wanda. This is if the queue is to be consumed by Wanda and will allow Wanda to deal with the new data.
- The new VCI and routeing information that the cell should be transmitted with. This is only required if the Xilinx bits do not support hardware remapping. Again this is only required if the cell is to be forwarded.

[4]This is stored in the 2048 words in the range 0x4000 to 0x5fff

[5]See section 4

For the design which requires manual VCI remapping, each element in the forwarding map has two entries. The first is the address of the queue that cells on this VCI should be placed on. The second is the routeing information and the new VCI. The second word also doubles as the interrupt indication. If it is zero then no remapping is performed and the Wanda interrupt bit is set. If non zero then the remapping is performed and the transmit available interrupt is enabled.

For the design which does support remapping of the VCI a single bit is required to indicate which interrupt to enable. Performance is improved by encoding this bit in the same word as the queue pointer. The word stored is in fact the queue pointer shifted left by one bit with the bottom bit used to encode which interrupt to perform. A single right shift both recovers the queue address and places the controlling bit in the carry flag for the conditional branch.

5.3 Algorithms

5.3.1 Arriving Cells

When a receive available interrupt is fielded the first thing that the code does is to allocate a new cell buffer from the free list. If the free list is empty it takes a machine check (panic).

The free buffer is swapped with the buffer number in the Xilinx receive queue which gives it to the Xilinx and obtains the buffer number that actually arrived. The current active position in the Xilinx receive queue is kept in a FIQ state register.

The VCI is recovered from the received cell and indexed into the forwarding map. The cell is added to the specified queue and the appropriate interrupt is enabled. Control is finally passed to an interrupt dispatch routine which priorities transmit available interrupts over receive available, for reasons of fairness.

5.3.2 Departing cells

If enabled, on a transmit available interrupt the code searches a number of queues for a cell to transmit. This could in general be any number of queues but the current Wanda networking code uses only two queues, one for all traffic being cell forwarded through this node and one for management or other traffic being generated at this node. If a queue is non empty the head cell is removed and written to the Xilinx transmit queue. The current position within the Xilinx transmit queue is kept in a FIQ state register. The transmit retry count (see section 5.3.3) is initialised to -17. Finally control is passed to an interrupt dispatch routine which priorities receive available interrupts over transmit available, for reasons of fairness, with an important exception explained in section 5.4.

5.3.3 Blocked cells

The Fabric generates a NACK whenever there has been contention, either for an output port or due to blocking within the fabric, and this port controller's cell has lost. A NACK is also generated by the destination port controller if its fifo is becoming full. On seeing a NACK the Xilinx (possibly after one automatic retry) generates a FIQ interrupt and stops transmission.

In section 5.1.3 it was pointed out that dealing with a Nack interrupt was non-trivial, it is also very slow on a Xi2 design if further attempts are required. This is because the Xilinx chip will already have moved the head of the Xilinx transmit queue forward passed the nacked cell. To prevent VCI re-ordering it is not possible to simply restart the chip. Instead the buffers *all*[6] have to be shuffled forward first. This is performed by a sequence of seven swap operations preceeded by a load and followed by a store. The store also has the strobe bit set to increment the count of active transmissions. These nine access together take a very long time.

Before all of this takes place however the transmit retry count is incremented. If the increment caused it to reach zero then the cell is deemed to have had enough retries and the code branches straight to the final section of code which performs a "GoXilinx" command to restart transmissions and returns to the interrupt dispatch loop.

The single count mechanism used to determine the number of retries for a cell is not perfectly accurate because it is possible that a second cell could be added to the Xilinx transmit queue while a cell that had been nacked was waiting for an opportunity after having been restarted. However the probability of this is extremely low and the performance penalty of keeping separate counts is not considered appropriate for the minimal increase in accuracy. In any case the number of retries *ought* to be an attribute of the VCI.

5.4 Further Subtleties

Another area where a speedup was obtained was the manipulation of the FIQ mask register in the IOC. When a cell has arrived and needs to be forwarded the code must enable FIQ transmit available interrupts. A write cycle to the IOC is a slow way to implement this. A better method is to keep a temporary register for the duration of a FIQ with the current logical value of the mask in it. This value is loaded from the IOC when the FIQ is first entered and stored before returning.

It is necessary to reload on entering because higher level code writes the mask to enable transmission when required. The FIQ loops on something to do so instead of reading the IOC FIQ request register it reads the IOC FIQ status register and performs its own masking. State modifications are made to the FIQ mask register with a simple "bit clear" or "or". This saves writing the new state until there is no more work to do.

Additionally because the dispatcher is biased towards transmit once it it busy the transmit code when finished checks the result of the mask for receive and dispatches to there immediately rather than re-reading the status. This halves the number of status reads while busy.

6 Performance

It is difficult to determine the exact performance of the port controller because there are so many differing factors which all influence the final figure. Amongst others there are:

- Cpu type - Arm 3 or Arm 600
- Cpu FCLK speed.
- Cpu MCLK speed.
- Fabric clock speed.

[6]Due to not knowing how many are currently active

- Exact Xilinx bits version.
- Software changes as appropriate for configuration of above.
- Measurement error.

The performance is usually measured by the time the FIQ is active in dealing with a mini-burst of eight back to back cells on a circuit which is sending regular mini-bursts. A substantial number of samples are taken and averaged. The performance is based on the time and the number of *user* data bits passed - that is on the forty eight octets only.

Performances have been observed at just in excess of 40 Mbits/sec for an Arm 3 and over 48 Mbits/sec for the Arm 600. I would estimate that the performance of the Arm 600 would exceed 50 Mbits/sec if it were usable in more favourable configurations of the other parameters.

7 Towards the Xi3 design

Experience with the Xi2 design caused thought about what functions could be put into the hardware to improve performance of the port controller. This section describes the initial ideas, which evolved into the Xi3 xilinx design.

7.1 An idea for Xilinx free queue management

Since the free queue management, and interrupt status mask reads, were found to be taking a lot of the FIQ time thought is being given to putting the management into hardware. The actual management of the *free* queue is not an issue for research, but will have to be done whatever is done with the cell queues.

The proposed system is for the xilinx to have three registers for the free queue. The "Head" and "Tail" pointers perform the conventional actions, and the "Arm" pointer indicates where the processor access point is. Figure 6 shows the queue organisation. The (currently unused) 14th word of the cell buffer is used to hold the link pointers, and the list is terminated by the Tail pointer and therefore need not be NULL terminated. On initialisation the structure must be setup by the processor.

There are two interrupt conditions for the free-queue system. If the Arm pointer is unequal to the Head pointer then there are outstanding cells and an RX interrupt is raised. Should the Head pointer become equal to the Tail pointer then the list has been exhausted, this is regarded as catastrophic and the xilinx will stop and raise a fatal "BarfBarf" event[7].

Three operations may be performed on the queue:

- Cell arrives.
  Arriving cells are moved into the buffer pointed to by the Head pointer, with short cells, as now, being overwritten by the new cell. When a cell has arrived the offset pointer will be at 14, and the new Head pointer can be read directly. Moving the Head will have the side effect of raising the Rx interrupt.
- Cell is sent.
  When a cell has been sent into the fabric the buffer must be freed. This is done by writing the Tx buffer pointer into offset 14 of the Tail cell, and moving the Tx

[7] It is also proposed that the Nack retry count be raised to 16 times, and software take over if this fails.

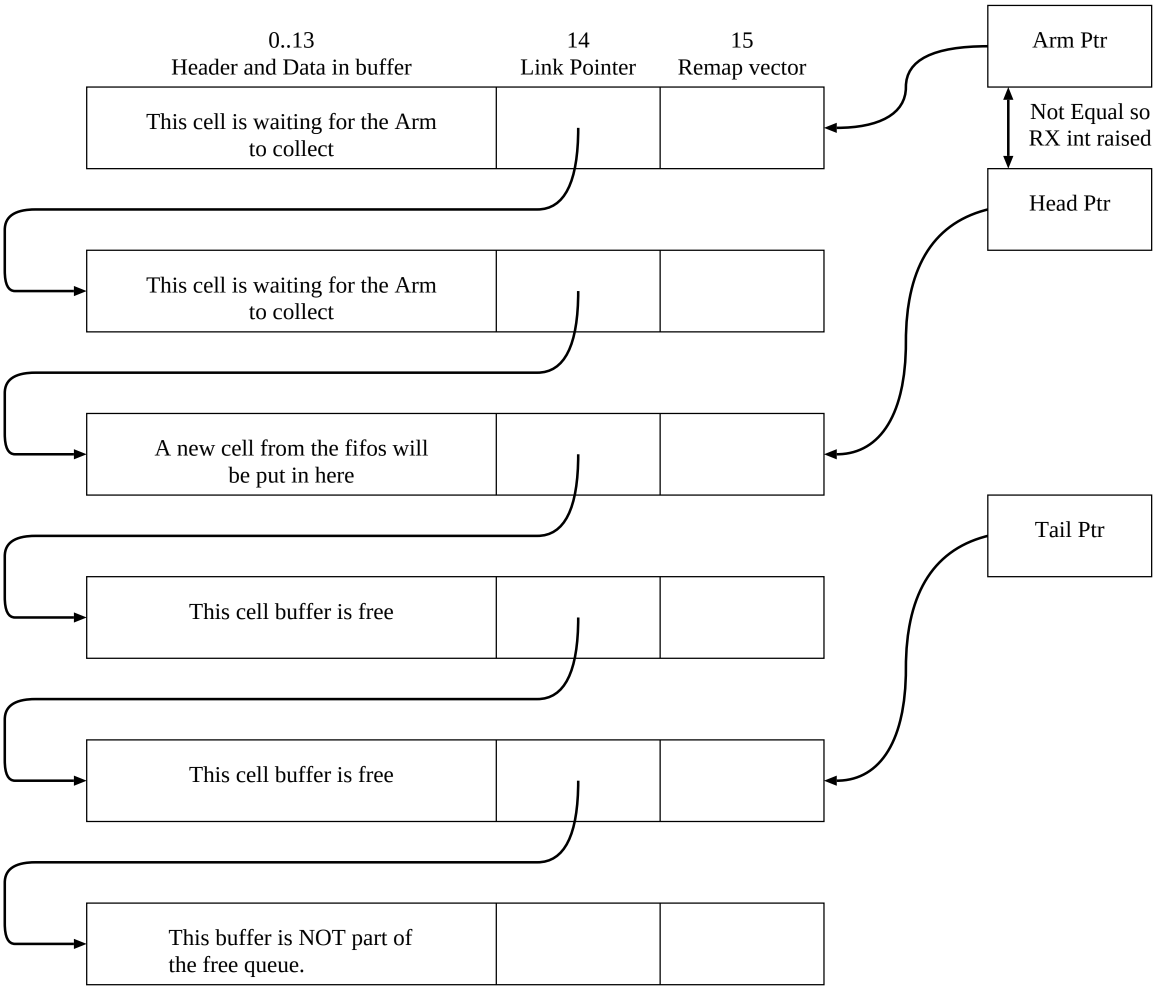

Figure 6: Suggested hardware free-queue.

pointer into the Tail register. Some cells (for example top 2 bits set) may be reserved for the management software, and will not be returned to the free list.

- Arm read.
  The processor will be able to read the status register in a way (probably indicated by the access address) that causes the Arm register **and** the interrupt status bits to be read. If the Arm register is not equal to the Head then it is stepped on by reading the link pointer. The processor thus only needs one read to identify the cause of the interrupt and (if it was a reception) the arrived cell. The xilinx only drives a number of the data lines, so it may be necessary to have a second status register for the "latched status" bits.

One problem with this system is that if many cells are received for the management entity (with few sent) it could end up with a surplus of "hardware controlled" buffers, possibly leading to free queue exhaustion. One way round this is to fill the cell with zeros and send it. Since the bottom bit is always zero the fabric will see an inactive cell, but the processor will have to write all the zeros. An alternative is to have a bit in the transmit cell buffer number to suppress transmission of the cell.

The extension of these ideas resulted in the Xi3 design, which is documented separately.