EISA ATM Interface Card

Ian M Leslie

21st July 1995

1 Overview

The EATM Interface is an EISA compliant ATM interface card. The physical transmission system used is ATM Forum 100 Mbps on TAXI 4B5B coding using both twin bore unidirectional 50Ohm coax and a 1300nm FDDI LCF-PMD compliant optical components. The interface presented to the EISA bus is a simple cell interface with some support for AAL5. It can be accessed as a memory slave. The board does *not* perform segmentation and reassembly.

The board does per cell CRC-32 (as used in AAL5) calculations in both directions. It is able to include a previous CRC residue in these calculations. For non VCI interleaved transmission no special action is required by host software to chain CRC residues across an entire AAL5 unit. When receiving non interleaved AAL5 units, host software must write a location in the EATM Interface to indicate that the current CRC residue is valid. Host software is required to save and restore CRC residues when VCI interleaving within an AAL5 unit is performed. More details can be found in section 2 below.

The board is able to interrupt on a number of conditions. These include, received cell, received cell with ATM User User (AUU) bit set, received cell FIFO over threshold, and various error conditions. The RX FIFO threshold is programmable by the host.

The major design goal of the interface is low cost via simplicity. The major components on the card are:

- ESIC: This Hitachi ESIA Slave Interface Controller device handles the EISA bus peculiarities providing control signals to the main EATM Interface controller. It handles all issues to do with address configuration.
- EATM Interface controller: This Xilinx device is the heart of the interface and:
  - performs CRC generation in both tx and rx directions
  - contains all data and non-ESIC control registers
  - interfaces the tx and RX FIFOs to the EISA bus
  - handshakes with the transmission subsystem to keep track of cells in the FIFOs
- RXFIFO: This is intended to be a 32K by 9 bit FIFO but it pin compatible with FIFO devices of various sizes.
- TXFIFO: This is intended to be a 4K by 9 bit FIFO but it pin compatible with FIFO devices of various sizes.

- Transmission subsystem controller: This Xilinx device is based on existing Fairisle and Sapphire taxi transmission controllers. It interfaces the FIFOs to the TAXI devices, performs cell framing and header CRC functions. It also handshakes with the main EATM Interface controller.
- TAXI devices: These are the two AMD TAXI devices and the associated ECL and analog gubbins.
- HP HFBR-5104: Low cost 1300nm fibre optic transceiver
- Bus drivers: These are four bidirectional latches which drive the EISA data lines. They are controlled by the main EATM Interface controller Xilinx device.
- FIFO Pulses: A delay line and a PAL can be used to generate the TX FIFO write and the RX FIFO read signals, allowing the main EATM Interface controller to behave in a completely synchronous manner.
- miscellaneous: A PROM to program the Xilinx devices. (The main Xilinx device supplies the EISA identification.)

2 Programmer's Interface

Warning: Accessing some of the registers in the EATM Interface causes changes to the internal state of the EATM Interface board, in particular on the receive side. The accessing of registers as described below should be followed closely.

2.1 EATM Interface Register Set

The EATM Interface uses the Hitachi HD64981F ESIC [Hitachi 91] to implement the EISA protocol. As well as the ESIC registers, which are located at 0xzc80, where 'z' is the EISA slot number, the EATM Interface has ten 32 bit registers which are memory mapped. Each register appears in a block of 64 contiguous word locations, ie the EISA signals LA2 to LA7 are not used in the address decoding. Each register thus uses 256 bytes. This enables store multiple instructions to be used to get cell data in or out of the interface.

The memory mappings of the start of each 256 byte bank for the registers are shown in table 1 (BASE is the address generated from the EISA slot number). The codes in the "State Change" column are Y to indicate that accessing the register will change some internal state in the EATM Interface (other than the register itself), N to indicate no change and W to indicate a change if written.

These registers and their use are described in detail below.

2.1.1 STATCON Register

The status register has the format shown in table 2. The meanings of the individual bits are as follows:

- RXCOUNT: These read only bits indicate the number of complete cells in the receive fifo.
- RXNOTEMPTY: This read only bit, when set, indicates that the rx fifo is not empty.

Location	Name	Access	State Change	Description
BASE + 0x000	STATCON	R/W	N	status and control
BASE + 0x100	RXHDR	R	Y	read cell header
BASE + 0x200	RXDATA	R	Y	read cell data
BASE + 0x300	RXCRC	R/W	W	rx CRC register
BASE + 0x400	RXCRCCONT	W	Y	rx CRC continue
BASE + 0x600	RXTHRESH	W	N	rx FIFO threshold
BASE + 0x700	TXHDR	W	Y	write cell header
BASE + 0x800	TXDATA	W	Y	write cell data
BASE + 0x900	TXCRC	R/W	N	tx CRC register

Table 1: EATM Interface IO Registers

Bit	Name	Access
16-31	RXCOUNT	R
15	RXNOTEMPTY	R
14	RXHDRERR	R
13	RXBIT	R/W
12	RXFTHR	R
11	MONITOR	R/W
10	RXFULL	R/W
9	VIOLATION	R/W
8	STRAY	R/W
7	INTRXNOTEMPTY	R/W
6	CLRXFIFO	R/W
5	INTRXBIT	R/W
4	INTRXFTHR	R/W
2	INTRXFULL	R/W
1	INTVIO	R/W
0	RESET	R/W

Table 2: STATCON Register Format

- RXHDRERR: This read only bit, when set, indicates that the last cell header read from the EATM Interface had a header error check. This is valid after a read to the RXHDR register and before the next read to the RXDATA register.
- RXBIT: This bit is set by the EATM Interface whenever a cell with the AUU bit set in a cell header is put into the RX FIFO. It is cleared only by the device driver.
- RXFTHR: This read only bit is set when the number of cells in the receive FIFO exceeds the threshold.
- MONITOR: This bit can be set by host software for testing purposes. It appears on a test point on the EATM card and can be used for example to trigger a logic analyser.
- RXFULL: This read only bit indicates that the RX FIFO has overflowed.
- VIOLATION: This read/write bit is set by the EATM Interface when the transmission system detects violation.
- STRAY: Stray bytes (ie badly formed cell) were detected. Must be cleared by the driver.
- INTRXNOTEMPTY: This read/write bit, when set enables a set RXNOTEMPTY bit to cause an interrupt.
- CLRXFIFO: This read/write bit, when set clears the RXFIFO, resets the receive cell count and resets the receive state machine. In normal operation it should be kept zero. Note that since cells may be arriving, after this bit has been asserted, host software should dally briefly (a few micro seconds) and check that the receive count is zero, otherwise reset again
- INTRXBIT: This read/write bit, when set enables a set RXBIT bit to cause an interrupt.
- INTRXFTHR: This read/write bit, when set enables a set RXFTHR bit to cause an interrupt.
- INTVIO: This read/write bit, when set enables a set VIOLATION bit to cause an interrupt.
- RESET: This read/write bit, when set clears both fifos and resets most of the state of the EATM Interface . In normal operation it should be kept zero.

2.1.2 RXHDR Register

The RXHDR register is a read only register that should be read at the start of a cell. It reads a 32 bit word from the receive FIFO but does not run the 32 bits through the CRC-32 logic. After reading the header from this location, the RXHDRERR bit in the status register is valid until the next read from the RXDATA register. Reading this register clears the RXBIT and RXNOTEMPTY status bits if appropriate.

2.1.3 RXDATA Register

The RXDATA register is a read only register that should be read (12 times in fact) after the header has been read and the CRC-32 initialised (see below). Reading from this register reads from the receive FIFO but also runs the data through the CRC-32 logic. RXHDRERR is valid after a read to RXHDR and until a read to this register.

2.1.4 RXCRC Register

This readable and writeable register contains the current state of the CRC calculation. When written to after the RXHDR register is read it starts the data running through the CRC-32 logic. *This register (or RXCRCCONT) must be written after RXHDR is read and before the first RXDATA of a cell is read.*

2.1.5 RXCRCCONT Register

This dummy register is written to (the value is ignored) to continue the CRC-32 logic without modification to the currently stored value. It allows the CRC residue in the CRC-32 register to be used without having been read and restored.

2.1.6 RXTHRESH Register

This writeable register is used to set the receive fifo threshold (which is in cells not bytes).

2.1.7 TXHDR Register

This writeable register should be written with the ATM cell header (excluding the header error check) before the data parts are written. The header is not run through the CRC-32 hardware.

2.1.8 TXDATA Register

This writeable register should be used to write the data bytes (4 bytes at a time) of a transmitted cell. These bytes are run through the CRC-32 hardware.

2.1.9 TXCRC Register

This is the CRC-32 residue register which can be read or written. Note that to send a CRC it should be read from the TXCRC register and then written to the TXDATA register — this is done in mid cell.

2.2 Initialisation

2.2.1 Finding the Sucker

As for all EISA boards, the EATM Interface should be found by investigating the EISA ID locations at 0xzc80 to 0xzc83. The manufacturer/device code is HWP18E0. *The board makes use of the Hitachi ESIC device which insists on 8 bit accesses using EISA data[0..7] only. This is known not to be supported by COMPAQ Deskpro's, although it is supported by DEC Alpha PCs and HP Vectra's.*

2.2.2 Initialising the ESIC

ESIC registers are byte-wide. A working configuration file is appended to this document. Note in particular that the ESIC device has dual function pins which must be configured correctly, in particular because the defaults are not helpful. For example there are pins which could be DMA related or just local outputs. The default on power up is to assume that they are local outputs which are at power-up driven in an unfortunate manner. On the EATM Interface they are in fact connected to EISA DMA signals.

2.3 Receive Operation Example

There are two major ways to use the device for reception, either per cell (using simply RXNOTEMPTY) or trying to use the AUU bit. The latter involves not only using RXBIT for interrupts, but also RXFTHR to avoid overruns due to multiple simultaneous receptions of blocks.

In supporting AAL5, the the driver maintains a list of partially assembled AAL5 units and their CRC-32 residues. It most efficiently processes an AAL5 unit without interleaving but is capable of interleaving – the cost is an extra EISA cycle to the EATM Interface and related software overhead to extra partial CRC.

On a receive interrupt it reads the header and determines the state of reassembly on that VCI and on the VCI just processed in order to set up the CRC-32 register appropriately:

```
IF (old_vci = new_vci) AND (old_not_finished) THEN
    <hit the RXCRCCONT>
ELSE IF (old_not_finished) THEN
         <save old CRC>
     END;
     (* The restore or init is not relevant if AAL1/2/3/4 *)
     IF (new_is_aal5) THEN
          IF (new_not_finished) THEN
              <restore old CRC>
          ELSE
              <init CRC to init val>  (* also handles old = new but finished)
          END;
     END;
END;
```

If a header error check was found or the incoming VCI is unknown, the data should be read from the RXDATA register and discarded.

If the AUU bit was not set in the header, the driver then reads the data (twelve 32 bit word transfers) into the appropriate buffer, otherwise it reads ten words to the appropriate buffer and the trailer to appropriate variables where it extracts and checks the length field and then check that the CRC-32 register now contains zero. If all is well, higher layers of software are then poked in the appropriate manner.

The driver should now look to see if it should continue; if running using the RXNOTEMPTY interrupt, if RXNOTEMPTY is still set, then we loop round; if running off RXBIT and RXFTHR, then we may decide to continue only if an interrupt is still pending, or may simply continue until the RX FIFO is empty by looking at RXNOTEMPTY (even though it is not responsible for the interrupt).

2.4 Transmit Operation Example

The EATM Interface board provides no rate control; this must be done by host software. The EISA bus is far slower than the 100 Mbps line rate so that when the driver has a cell(s) to send it can transfer it to the card immediately.

When writing the cell to the EATM Interface , the driver first writes the header and then either sets the TX CRC-32 register contents with the accumulated CRC so far, or simply starts sending data to use the current value – unlike the receive side, there is no need to explicitly issue a CRC continue. The data is then written out, either 12 writes in a row, or in the case of the last cell of an AAL5 unit, up to but not including the CRC (i.e including the pad and appropriate length fields). Then either TXCRC is then read back and written as data or, if implemented, TXCRCSEND used.

2.5 Transmission Errors

It is envisaged that these are likely to be sampled periodically without using interrupts. If error interrupts are enabled, it is suggested that they are used simply to instigate some timer based polled behaviour until they go away, as there is a danger of simply stopping a machine dead in its tracks. Note that this is less of a problem in most sensible data communications protocols as these will tend to stop sending after some time if no response is achieved – e.g. TCP ack, RPC reply, video frame credits systems – so that the machine may get overrun for a short time but then recover. The board itself will try very hard to suppress spurious cell indications due to weirdness form the line etc.

3 Asynchronous Interface

The following section is documentation of the current design of the asynchronous interface between the transmission lines and the main Xilinx – changes in this area do not effect the programming model and the final split of functions between TAXI controller and main Xilinx awaits evaluation of the real hardware.

3.1 TAXI transmission

The TAXI transmission subsystem consists of two AMD TAXI devices, one for transmission, one for reception, the TAXI control device implemented in a Xilinx FPGA, two asynchronous FIFOs, ECL buffers, some LEDs and various discrete components. Optionally it is hoped to support the low cost HP FO transceiver.

The physical interface presented to the main part of the board consists of a subset of the signals connected to the two FIFOs and the TAXI Control FPGA:

RXFIFO	rxData[0..9] rxFifoRdBar rxFifoFull rxFifoEmpty
TXFIFO	txData[0..9] txFifoWrBar txFifoFull txFifoEmpty
TAXICONTRL	rRxCell aRxCell rRxBit aRxBit rTxCell aTxCell

The FIFO signals are self explanatory, forming nine bit data paths in each direction. The signals to the TAXICONTRL device are in request-accept (r-a) pairs each controlling an asynchronous handshake.

An r-a pair is used in the following way. A *requestor* indicates that an event has occurred by raising the r-signal. It may only do this if the a-signal is low. The *acceptor* accepts this event by raising the a-signal. Upon seeing the a-signal go high, the requestor drops the r-signal. Finally the acceptor can drop the a-signal when it sees the r-signal go low, completing the handshake.

The rRxCell-aRxCell pair is used to indicate that a complete cell has been received and placed in the receive FIFO. In this case the requestor is the transmission subsystem; the acceptor is the main EATM controller. The rRxBit-aRxBit pair is similar but used to indicate that a complete cell with the AUU bit set has been placed in the receive FIFO. (A cell with the AUU bit set will cause both events to be signalled.)

The rTxCell-aTxCell pair is used to indicate that a complete cell is in the transmit FIFO and ready for transmission; here the requestor is the main EATM controller and the acceptor is the transmission subsystem.

The events generated by the transmission subsystem, the RxCell and RxBit events, are counted by the main EATM controller. Thus the EATM controller has a count of how many full cells and how many full cells with AAU bits set are in the FIFO at any given time.

On the transmission side, it is again the main EATM controller that maintains the count, feeding transmit cell events one at a time to the transmission subsystem. The transmission subsystem acknowledges an event only after it has completely transmitted the cell.

3.2 TAXI cell framing

The TAXI controller is responsible for the framing of cells. In the transmit direction this is relatively straightforward. A start of cell (SOC) code must be generated, followed by four bytes of header (read from the TX FIFO) followed by the header error check (HEC) which is generated by the TAXI controller itself. This is then followed by the 48 byte payload. SYNC characters are generated if no data is present. This conforms to the ATM Forum standard for 4B5B lines.

In the receive direction the TAXI controller must cope with possible framing errors. The receiver can be in two states, hunting or cell processing. In the hunting state the receiver places no information in the RXFIFO. The receiver stays in the hunting state until a SOC is received. It places a SOC code in the FIFO and then begins copying header bytes into the FIFO and checks the HEC. It writes a code into the FIFO (with the 9th bit set) indicating whether the CRC was correct or not. The controller also remembers if the AUU bit was set. When 48 bytes more are received the TAXI controller enters the hunting state.

During the time the receiver is cell processing, it may of course find another SOC code. If it does this, it simply writes a special code into the FIFO and carries on. Thus the RX FIFO always contains full cells. The receiver, by continually entering hunt mode at the end of each cell will re-lock onto the correct cell structure after a short cell has been received. The price paid by this simple scheme is that a short cell will also cause a full cell immediately following it to be corrupted. Note that the codes placed in the RX FIFO by the TAXI controller are interpreted by the main Xilinx to provide the appropriate interrupts and status bits.

References

[Hitachi 91]
HD64981F EISA Slave Interface Controller (ESIC), Hitachi, June 1991.

```
;---------------------------------
; EISA ATM Card Configuration File
; LAST UPDATED ON 13th March 1995
;---------------------------------

BOARD
          ID = "HWP18E0"
          NAME = "HP 100Mbps EISA ATM Card"
          MFR = "Hewlett-Packard Company 1993."
          CATEGORY = "NET"
          SLOT = EISA
          LENGTH = 220
          SKIRT = NO
          READID = YES
          IOCHECK = INVALID
          DISABLE = UNSUPPORTED


;-------------------------------------------
; MEMORY ADDRESS COMPARE REGISTER 0 - BYTE 0-2
;-------------------------------------------
IOPORT (1) = 0ZC87H
        SIZE = BYTE
IOPORT (2) = 0ZC86H
        SIZE = BYTE
IOPORT (3) = 0ZC85H
        SIZE = BYTE
;----------------------------------------
; MEMORY ADDRESS MASK REGISTER 0 - BYTE 0-2
;----------------------------------------
IOPORT (4) = 0ZC8DH
        SIZE = BYTE
IOPORT (5) = 0ZC8CH
        SIZE = BYTE
IOPORT (6) = 0ZC8BH
        SIZE = BYTE

;----------------------------------
; MODULE CONFIGURATION REGISTER 1
;----------------------------------
IOPORT (7) = 0ZCA1H
        SIZE = BYTE

;----------------------------------
; MODULE CONFIGURATION REGISTER 7
;----------------------------------
IOPORT (8) = 0ZCA7H
        SIZE = BYTE

;-------------------------------------------
; INTERRUPT CHANNEL CONFIG AND STATUS REG - 0
;-------------------------------------------
IOPORT (9) = 0ZCA9H
        SIZE = BYTE

;--------------------------
; FUNCTION CONTROL REGISTER
;--------------------------
IOPORT (10) = 0ZCAEH
        SIZE = BYTE

FUNCTION = "Register Initialization"
    SHOW = YES

    CHOICE = "INIT"
```

```
        FREE
;           INIT = IOPORT(1)  00000001
;           INIT = IOPORT(2)  00000000
;           INIT = IOPORT(3)  00000000
            INIT = IOPORT(4)  00000000
            INIT = IOPORT(5)  00000000
            INIT = IOPORT(6)  00001111
            INIT = IOPORT(7)  00100100
            INIT = IOPORT(8)  00000000
;            INIT = IOPORT(9)  00100000
            INIT = IOPORT(10) 10010000

 FUNCTION = "Hewlett-Packard EISA ATM Adapter Card"
   SHOW = YES

   TYPE = "NET"
   CHOICE = "Interupt Levels"
     FREE
        IRQ = 3 | 4 | 5 | 6
; Maybe we can change this to YES some day to support more than one card?
        SHARE = no
        TRIGGER = level
        INIT = IOPORT(9) LOC(7-0) 00001000 | 00001001 | 00001010 | 00001011

FUNCTION = "Device Address Select"
    SHOW = YES
    CHOICE = "RAM Space"
        LINK
            MEMORY = 4K
            ADDRESS = 4064M | 32M | 800K | 804K | 808K
            MEMTYPE = OTH
            WRITABLE = YES
            SIZE = DWORD
            CACHE = NO
; NB This is NOT inverted like the BUS is!
            INIT = IOPORT(1) LOC(7-0) 11111110 | 00000010 | 00000000 | 00000000 | 00000000
            INIT = IOPORT(2) LOC(7-0) 00000000 | 00000000 | 00001100 | 00001100 | 00001100
            INIT = IOPORT(3) LOC(7-0) 00000000 | 00000000 | 10000000 | 10010000 | 10100000
```