

# Notes on the Pixel Trigger processing card development

Gianluca Aglieri Rinella

July 31, 2007

## Abstract

Internal report. Notes on the design of the central processing card (BRAIN) of the Alice Pixel Trigger system.

## 1 General architecture

The Alice Pixel Trigger system is designed with a modular structure. Ten optical input boards receive data from the 120 half staves and extract the Fast-OR signals. The Fast-OR signals are processed in the FPGA of a central processing board to generate the trigger decision. The optical input boards are plugged on the central processing board as mezzanine cards. A block diagram of the system is shown in Fig.1. A virtual view of the system is shown in Fig.??.

The decision algorithm is implemented in programmable hardware. The processing FPGA on the mother board receives the FastOR signals on 600 parallel lines, coming from the ten optical input boards. The 1200 FastOR signals are time multiplexed in two consecutive 12.5 ns clock periods. The output of the processing block is sent to the Central Trigger Processor via dedicated lines. Proper buffering is foreseen.

Status monitoring and remote configuration functionality have to be provided. These are implemented via a dedicated Controller FPGA. Interfaces between the various elements of the system are implemented in this block.

The I/O signalling standard is LVTTTL.

## 2 Memory structure

The communication between the various OPTIN boards and the BRAIN board devices is done on a shared data bus. Fig. 2 shows an idealized view of the interconnection buses between the devices. The shared bus lines are actually driven by buffers controlled by the CONTROL FPGA. These are not shown for clarity.

### 2.1 DDL and SIU interface

The SIU block is the board interfacing to the DDL and constitutes the communication interface from and to the remote controlling host computer. The DDL can transfer 32 bit data in blocks. Minimum length of a data block is 1 and

maximum length is 512 Kwords, each of 32 bits. Therefore 19 bits are needed to account for the length of a data block . A DDL data block transfer is initiated by user commands STBRD or STBWR containing a 19 bit user defined parameter field. When reading a block from the Front End Electronics (FEE) to the host computer, the length of the data block transferred is reported to the host at the end of the transmission, using the 19 bit parameter field of the FESW(EOB). No information on the block length is transmitted from the host to the FEE on a request of a block read/write operation.

The DDL protocol also supports direct control words write and status words read on the FEE. The control and status words can be addressed by a 19 bit user defined parameter. In other words the host can send to the FEE a control command (FECTRL) including a 19 bit user defined parameter field. No answer from the FEE is expected by the host in this transaction. A status word can be requested to the FEE by the host with the FESTRD command also including a 19 bit parameter value. The FEE replies in this case with a FESW including a 19 bit parameter field that is the answer to the host request.

The FECTRL/FESTRD pair of commands can be used to implement fast direct access to a set of Special Purpose (SP) registers of the CONTROL FPGA dedicated to the communication control, to the status monitoring and to direct commands. No flow control is implemented in the read/write operations to these registers, ensuring minimum latency. The STBRD/STBWR pair can be used to implement access to all the registers and memory on the BRAIN card and on the ten OPTIN cards.

## 2.2 Memory on the BRAIN board

The CONTROL FPGA firmware implements various internal registers. The CONTROL FPGA also has dedicated external SRAM memory. The main use of this memory is to store the JTAG bitstream that is necessary to shift to the PROCESSING FPGA for its reprogramming. It is also necessary to read back the configuration firmware of the PROCESSING FPGA for verification of the programming. The CONTROL SRAM is constituted by 4 chips of 72 Mb. Effectively a SRAM space with a width of 32 bits and a depth of  $2^{23}$  words is available. That is  $2^{23} \times 4 = 33554432$  bytes of memory.

The PROCESSING FPGA also has dedicated SRAM memory for sampling fast data, if needed. This latter memory is half of the CONTROL memory, i.e.  $2^{22}$  words deep, 32 bit wide. The PROCESSING FPGA also has internal registers to implement masks, counters, thresholds and configuration of the main outputs.

Each OPTIN card has several registers. They are used to implement counters, thresholds, masking, optical link status monitoring and state machines monitoring.

A shared bus allows the communication between the peripherals devices and the CONTROL FPGA. This acts as bus master. A pseudo-PCI bus protocol is implemented.

All memory locations including registers in the FPGAs and SRAM memory can be accessed via the DDL as virtually belonging to a single addressing space. The address width of this virtual space is 28 bits, four out of which are used for device identification. Given that 12 devices are addressable in the system, then

spare addresses are left for broadcast (multiple target) write operations to the OPTIN cards. Table 1 details the organization of the address space.

Bus contention and management is done by the CONTROL FPGA firmware and shall be completely transparent to the DDL user. Routing to internal registers or to external SRAM chips is done in the CONTROL and PROCESSING FPGAs.

### 2.3 Accessing the 28 bit space with the 19 bit parameter field constraint on the DDL

In a generic block read/write operation one needs to transmit to the FEE the address of the starting memory position from/at which the transferred block should be read/written. This requires 28 bit to be transferred. Only 19 bit are directly available on the STBRD/STBWR commands user payloads. To solve this issue the following mechanism is implemented.

Two Special Purpose 9 bit registers  $rd\_a[27:19]$  and  $wr\_a[27:19]$  are implemented in the FEE. The 28 bit base address is obtained by padding the 19 bit transmitted with the STBRD/STBWR commands with the 9 bits stored in the register corresponding to a block read ( $rd\_a[27:19]$ ) or write ( $wr\_a[27:19]$ ) operation. The  $rd\_a[27:19]$  and  $wr\_a[27:19]$  registers can be set with a FECTRL command sent to the FEE immediately before the STBRD or STBWR commands.

In a block read (STBRD) operation also the length of the requested block must be transmitted to the FEE. The bits required to account for a maximum length of 512 kWords are 19. The Special Purpose  $len[18:0]$  register is implemented for this purpose in the FEE. It is not possible to set this register with one single FECTRL transaction. Therefore *two* consecutive FECTRL transactions must be used to set  $len[18:0]$  and must be executed before a STBRD operation. On receipt of the STBRD the FEE electronics initiates a data block read from the memory space. The base address is determined as discussed before. The block length is limited by the content of register  $len[18:0]$ . The number of words actually read from the memory space can be less than the requested value, for example because the base address and the length imply reading from a not addressable region or beyond the maximum address. The actual value of words transferred is sent to the host by the FEE using the FESTW(EOB) 19 bit payload. This value is also stored in the FEE Special Purpose register  $rd\_len[18:0]$ .

In a block write (STBWR) operation the block length is not transmitted explicitly. It is determined by the length of the data transmission itself, marked by the EOBTR word sent from host D-RORC (host) to the FEE. The length can be up to 512 kWords. The actual number of words written to the memory space is stored by the FEE in the  $wr\_len[18:0]$  Special Purpose register at the end of the transaction. This number can be less than the number of words actually transferred during the block write if, for example, the automatic address increment would write to an un-addressable region or would imply passing from a register space into a contiguous SRAM space.

The values of the  $len[18:0]$ ,  $rd\_len[18:0]$  and  $wr\_len[18:0]$  registers, containing respectively the requested length of data to be read, the actual length of data read and the actual length of data written in the last transactions, can be directly retrieved with FESTW commands.

The parameter field values of the FECTRL and FESTRD used to set and read the described Special Purpose registers are listed in Table ??.

In summary, the basic block write transaction is therefore composed of:

1. The setting of the *wr\_a[27:19]* register
2. The DDL STBWR block write transaction itself.
3. The read back of the actual value of the length of written data block from the *wr\_len[18:0]* register.

The basic block read transaction is composed instead of:

1. The setting of the *rd\_a[27:19]* register
2. The setting of the *len[27:19]* register
3. The DDL STBWR block read transaction itself, also giving back the actual value of data words read.

## 2.4 Lower communication layer function prototypes

The previous addressing mechanism is implemented by the lowest level of the PIT driver. This software routines wrap entirely the DDL library calls that are completely hidden to the upper levels. The prototypes of the lowest level PIT driver functions are described in this paragraph.

In direct correspondence with the previously defined FEE Special Purpose registers, there are the following getter functions to read via DDL each register value:

- int PITGetRdA()
- int PITGetWrA()
- int PITGetLen()
- int PITGetRdLen()
- int PITGetWrLen().

Each of the previous makes a call to the DDL library implementing a FESTRD transaction, using the parameters listed in Table ?. Notice that the getters of the address shall return directly the *full* address with all 19 bits. In case an error of the DDL is detected this should also be flagged with a proper return value.

Three of the discussed registers must also be written to, so corresponding setter functions are:

- int PITSetRdA( UINT28 address )  
{  
/\* call the DDL library function for FECTRL with proper parameter to set the *rd\_a[27:19]* register bits \*/  
/\* call the DDL library function for FESTRD with proper parameter to read back the *rd\_a[27:19]* value \*/  
/\* compare the value read back and flag possible error \*/  
}

- int PITSetWrA( UINT28 address )
 

```

      {
      /* call the DDL library function for FECTRL with proper parameter to
      set the wr_a[27:19] register bits*/
      /* call the DDL library function for FESTRD with proper parameter to
      read back the wr_a[27:19] value */
      /* compare the value read back and flag possible error */
      }
      
```
- int PITSetLen( UINT19 datalength )
 

```

      {
      /* call the DDL library function for FECTRL with proper parameters to
      set the len[18:9] register bits */
      /* call the the DDL library function for FECTRL with proper parameter
      to set the len[9:0] bits of the len register */
      /* call the DDL library function for FESTRD with proper parameter to
      read back the len[18:0] value */
      /* compare the value read back and flag possible error */
      }
      
```

Return values of the previous functions should flag eventual errors in the DDL transaction.

Notice that the two setters for the read and write address receive as parameter the entire 28 bit address value, even if they use only the upper 9 bits. In this way the extraction of the proper bits is implemented entirely inside these functions and hidden to the calling functions. If the hardware implementation changed, only the implementation of this function would need to be modified.

The two main data block read and write functions prototype and algorithms are:

- int PITReadBlock(UINT28 address, UINT19 datalength, UINT32 \*data)
 

```

      {
      /* call the PITSetLen function to set the requested data block length */
      /* call the PITSetRdA passing address to properly set the rd_a[27:19]
      register bits */
      /* call the DDL library function for STBRD with proper parameters (in-
      cluding address lower bits) to read back the data block and store it to
      *data */
      /* call PITGetRdLen to verify the number of words actually read from
      the DDL and return */
      }
      
```
- int PITWriteBlock(UINT28 address, UINT19 datalength, UINT32 \*data)
 

```

      {
      /* call the PITSetWrA function to set the wr_a[28:19] bits */
      /* call the DDL library function for STBWR with proper parameters to
      write the data block from memory location *data with length datalength
      and starting writing at the location given by address*
      /* call PITGetWrLen to check the number of words actually written to
      
```

```
memory */  
}
```

Table 1: PIT memory map. The 28 bit addresses are given in hex together with decimal value. Boundaries between the various region and unimplemented regions are also shown.

Address [hex]	Address [dec]	Rel. pos.	
000000	0	0	Beginning of CONTROL reg. space
000001	1	1	
000002	2	2	
...			
07FFFFE	8388606	8388606	
07FFFFF	8388607	8388607	End of CONTROL reg. space
080000	8388608	0	Beginning of CONTROL SRAM space
080001	8388609	1	
080002	8388610	2	
...			
0FFFFFFE	16777214	8388606	
0FFFFFFF	16777215	8388607	End of CONTROL SRAM space
100000	16777216	0	Beginning of PROCESSING reg. space
100001	16777217	1	
100002	16777218	2	
...			
17FFFFE	25165822	8388606	
17FFFFF	25165823	8388607	End of PROCESSING reg. space
180000	25165824	0	Beginning of PROC SRAM space
180001	25165825	1	
180002	25165826	2	
...			
1BFFFFE	29360126	4194302	
1BFFFFF	29360127	4194303	End of PROC SRAM space
...			Unaddressable space
200000	33554432	0	Beginning of OPTIN 0 reg. space
200001	33554433	1	
200002	33554434	2	
...			
2000FE	33554686	254	
2000FF	33554687	255	End of OPTIN 0 reg. space
...			Unaddressable space
300000	50331648	0	Beginning of OPTIN 1 reg. space
300001	50331649	1	
300002	50331650	2	
...			
3000FE	50331902	254	
3000FF	50331903	255	End of OPTIN 1 reg. space
...			Unaddressable space
400000	67108864	0	Beginning of OPTIN 2 reg. space
400001	67108865	1	
400002	67108866	2	
...			
4000FE	67109118	254	
4000FF	67109119	255	End of OPTIN 2 reg. space
...			Unaddressable space

Address [hex]	Address [dec]	Rel. pos.	
500000			Register space of OPTIN from 3 to 7
...			
90000FF			
...			Unaddressable space
A00000	167772160	0	Beginning of OPTIN 8 reg. space
A00001	167772161	1	
A00002	167772162	2	
...			
A0000FE	167772414	254	
A0000FF	167772415	255	
...			End of OPTIN 8 reg. space
...			Unaddressable space
B00000	184549376	0	Beginning of OPTIN 9 reg. space
B00001	184549377	1	
B00002	184549378	2	
...			
B0000FE	184549630	254	
B0000FF	184549631	255	
			End of OPTIN 9 reg. space

## References

- [1] ALICE Collaboration, ALICE Physics Performance Report, CERN-LHCC-2003-049, J. Phys., G 30 (2004) 1517-1763.
- [2] P. Riedler et al., Overview and status of the ALICE Silicon Pixel Detector, Proceedings of the Pixel 2005 Conference, Bonn, Germany.
- [3] A. Kluge, The ALICE silicon pixel detector front-end and read-out electronics, Nucl. Instr. and Meth. A 560 (2006) 67-70.
- [4] J. Conrad et al., Minimum Bias Triggers in Proton-Proton Collisions with the VZERO and Silicon Pixel Detectors, ALICE-INT-2005-025.



Table 2: Special Purpose Control registers access via the FECTRL and FES-TRD commands. When the number of bits set/read is less than 19, they are written/returned as *less* significant bits of the FECTRL/FESTW parameter field.

FECTRL parameter field (19 bits)			
0 0 0 0 0 0 0 0 0	0 0	control0[7:0]	control0[7:0] SP register, write value
0 0 0 0 0 0 0 0 0	0 1	control1[7:0]	control1[7:0] SP register, write value
1 1 1 1 1 1 1 1 1	1	rd_a[27:19]	rd_a[27:19] SP register, write value
1 1 1 1 1 1 1 1 1	0	wr_a[27:19]	wr_a[27:19] SP register, write value
1 1 1 1 1 1 1 0 0	len[18]	len[17:9]	len[18:9] SP register, write value
1 1 1 1 1 1 1 0 1	1	len[8:0]	len[8:0] SP register, write value

FESTRD parameter field	Replied FESTW par. field	Description
111 1111 0000 0000 0000	len[18:0]	Last requested read length
111 1110 0000 0000 0000	rd_len[18:0]	Last actual read length
111 1110 1000 0000 0000	wr_len[18:0]	Last actual write length
111 1111 1110 0000 0001	rd_a[27:19]	Last (MSB) read base address
111 1111 1110 0000 0000	rd_a[18:0]	Last (LSB) read base address
111 1111 1100 0000 0001	wr_a[27:19]	Last (MSB) write base address
111 1111 1100 0000 0000	wr_a[18:0]	Last (LSB) write base address
000 0000 0000 0000 1010	control0[7:0]	User register
000 0000 0001 0000 1011	control1[7:0]	User register
000 0000 0000 0000 0000	version[18:0]	Firmware version
000 0000 0000 0000 0100	bus_status[11:0]	Communication bus status

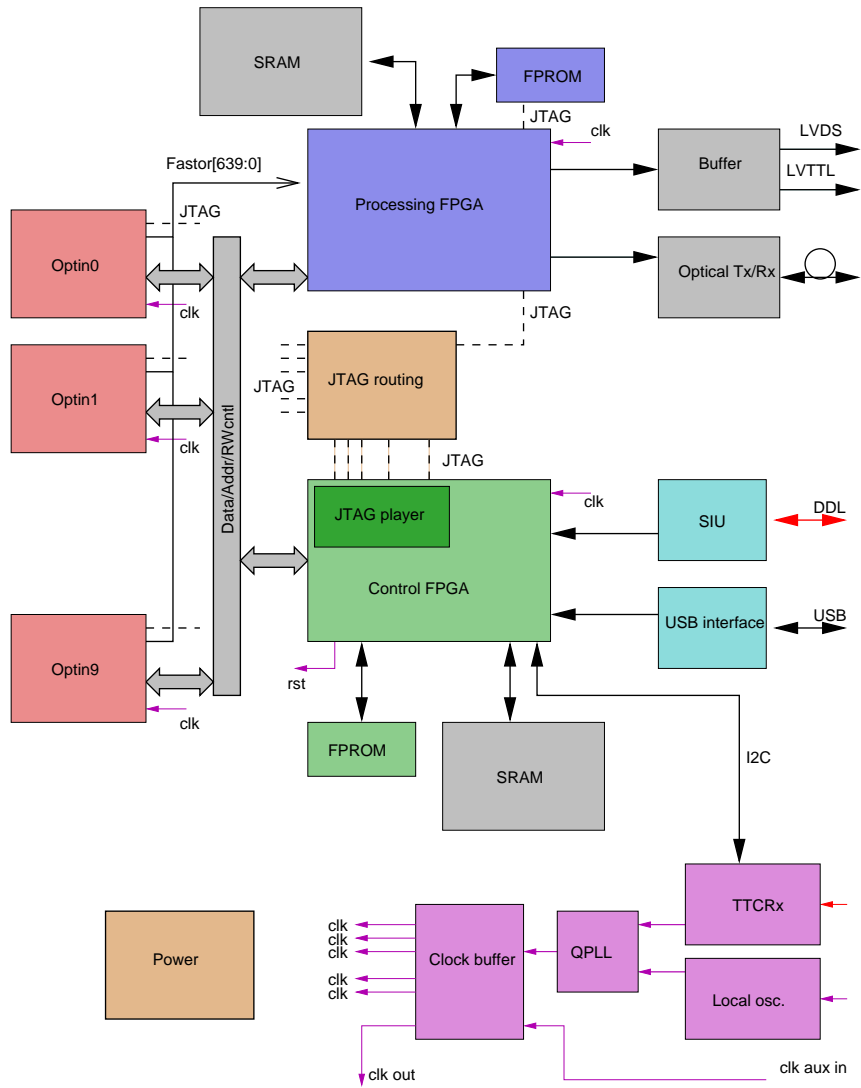


Figure 1: Block diagram of the Alice Pixel Trigger processing board

Figure 2: Communication lines between the various devices of the PIT system.

