Introduction.

The design changes in the Hardware of the new TSF prompts similar radical changes in the firmware that runs on it. Moving to the new reduced 3 FPGA Engine design, with a single Input Formatter, and combined Fast/Op/Daq (FOD) FPGA brings the chance to improve performance and lower latency.

One of the main changes incorporated is the removal of a the bus system between the FOD FPGA and the others. There no longer exists a separate address, data bus and control lines, instead there are 2 direct point to point serial links between FPGA's, one for each direction of transfer. A protocol derived from the original Clink/Dlink design is used on these links. It provides nearly all the functionality of the original design, with the exception of a few more critical control lines that are retained separately.

For the drift chamber data there is a new distribution system, again using point to point links. A 20bit link exists from the Input Formatter to each Engine FPGA, the pivot group data is sent over this to each Engine simultaneously.

DAQ data is also transferred from the Engine FPGA's over a serial link back to the FOD, being read out from each Engine simultaneously.

Design.

Input Chip.

Drift chamber data enters the chip as 20bits accompanied by the dav, cav and strobe lines from the finistar optical receiver via the TSFi board. The strobe line drives one clock of an async fifo, which receives the data, and also the cav line. A normal clock60 line drives the output of this fifo and so we are decoupled from slight clock skew issues that may happen. The data is now fed into the input of a normal fifo, the reason for this to allow us to “Reframe” in a way similar to the original TSF, which used an external fifo for this purpose.

Data that exits the second fifo is checked to see if it is accompanied by a cav strobe when we are at a clock4 transition. If now, then we flag that we are out of sync, vis the Glink Sync control line to the FOD. On receipt of a “Reframe” command the output of the fifo is halted, along with the input, and the device cleared. A delay of 1 clock4 is waited and then the output of in async fifo is watched for a cav strobe. Once this is found, the input to the second fifo is reenabled, and it begins to fill. Then, at the clock 4 edge, the output is restarted and the first word, which is now a cav word is read and we are back in sync with the Glink data flow.

The next stage is the processing of the data into an internal array, and also the sending and receiving of neighbour data off board. This code is generated via a perl script that reads in a map file describing which bit in each Glink word goes to which wire, and it then writes the code which maps these correctly, and then merges them with a vhdl template file to create the compilable vhdl.
This internal array is then processed into 3 further arrays that are shipped to the Engines. These arrays are 16bits wide, with an extra 4 control lines. A maximum of 8 words are used. The additional control lines are used to indicate that a word is a frame start, that is contains the start of a new super layer and with the last 2 bits to signify which position in the 16bit word this happens. The format of the 16bits of data are as follows: (Example data, not an actual frame)

word 1: sssssssdddddddd
word 2: ddddddddddddddd
word 3: ddddddddddddddd
word 4: dddssssssssdddd
word 5: ddddddddddddddd
word 6: ddddddddddddddd

s = Superlayer start

d = difference data.

At the start of a superlayer we ship the full 8 wires that make it up, each subsequent pivot group is made of of old data, plus the 4 new wires needed.

This data is shipped at a 30Mhz rate to the Engine FPGA’s. The reason for this rate, is that each word encodes upto 4 pivot groups, equating to 2 per clock60 which is the rate the Engines work at.

Also included in the Input Chip is a memory controller that handles playback/record memory, this is a 1Megabyte external ram, organised as 256k*32bits. This works as semi-linked Glink and external neighbour data memories. The format of the data is 20bits Glink, and in the first 4 words of a frame 7 bits of neighbour data, 28 bits total over the 4 words.

The recording of neighbour and Glink data is independent, but it is currently not possible to utilise the system in a recording Glink and playing back of neighbour, or vice versa. This is due to the 60Mhz clock frequency which the memory is run at.

Finally there is a serial-link decoder module that receives commands and data from the FOD and presents a “Fast Control” style interface to the rest of the chip internally. This is a generic module that exists inside all the FPGA's on the board. It presents the control lines, strobes and a 16bit bus to the chip. It also handles the encoding of data that is returned to the FOD, this is used when doing read operations from memory.

**Engines**

Each Engine FPGA has 2 accompanying LUT memories, and so the design of the Engine(s) is
geared to making the most effective usage of these memories. Each Engine deals with 2 superlayers, or 3 in the case of one engine on X type boards. Data arrives from the input formatter in the structure described above, and is processed by 4 parallel “Pivot Builders”, these produce completed pivot groups at the same rate as the incoming data, 30Mhz. There are occasional blank pivots produced, which occur at the start of a superlayer when only 3 per clock30 are sent from the input formatter. 2 Pivots per clock60 are output, on on each of the two pipelines.

Attached to these pipelines are the wire counters that are used to build the LUT index that is used as the memory address. Memory access happen at a rate of 1 per clock60 and the results pipelined. Owing to the data format, each superlayer needs to be reconstructed from results that are in both pipelines. The order of the required data is know, and a perl script again produces the VHDL to build the arrays that contain the pivot data for each superlayer.

At this stage we do the same as the original TSF, and use the original “Find best” code on the results of the LUT. Once this is done, there is a DAQ module, a BLT module and a ZPD module.

The BLT output is the same as the original, and as such is the same basic code, the ZPD module has to find the best segments to send to the ZPD, 3 per supercell, this is done with a simple recursive compare, that takes 5 clock60's to find the top 3 modules. The “ranking” of the segments is stored in the LUT data in the high 16bits that are not used for normal data, a feature of our wider 32bit memories. Once the segments are found the required data is extracted from the LUT result an outputted to the ZPD beginning on the Clock4 edge.

DAQ Data is processed as in the old system, but with the memory buffers changed so that there is one large duel port circular memory, one is the read, the other the write port. The offset between the write and the read points is configurable via a runtime memory location. On receipt of a L1 accept, the system starts reading out of the buffer at the required read point and send the data back to the FOD via a serial link at 60Mhz after doing the required processing. This is dependent on being in Best1, 2 or Raw modes. We ship on a L1 accept, in order to have the data ready in the DAQ buffers at the FOD for the Read Event command. -- The DAQ system is currently not written --

There is also a serial command module to handle the communication between Engine and FOD, and a memory controller to deal with reading/writing the LUT memories.

**FOD**

The original TSF had separate Fast Control, Op-Control and Daq Formatter chips. In the new reduced design these are all combined into one, the FOD FPGA. Internally it contains a slightly modified Fast Control, together with a new Op-Control and DAQ formatter. It is also linked to a 256k*32bit memory used as the output play/record memory.

Changes made to the Fast Control are related to the lower frequency clock generation, where the code responsible has been moved out of the code and into the top level of the chip. Also, some changes have been made to the timing of the read/write and address strobe lines, to make them occur earlier in the command decoding to allow our op-control some more time to process them.
Op-Control handles the serial link protocol used to transfer the Fast Control outputs to the other FPGA’s. Internally it is connected to the bus and control signals from the Fast Control, and when the state changes it propagates this over the serial link using a protocol derived from the original Clink system. A large portion of the decoder side of the link is reused, modified Fast Control code. It handles the memory mapping, deciding which FPGA is to be sent data during memory writes and reads.

DAQ Data is handled by a module inside the FPGA, but which is internally connected via a serial link, this is to ensure that it has the same delays as the external FPGA’s. Internally there is a dual port ram, used for daq data, and large enough to store four buffers worth of data. L1 and Read Event requests are pipelined, enabling the system to cope with up to 4 back to back request of each type. Data begins arriving from the Engines after a L1 accept, and is buffered in the memory, when a Read Event is received the data is read out via the other port. As the input rate is the same as the output rate, and we ship the data on L1 accepts we are able to lower the latency. The data from each Engine arrives simultaneously, and some logic deals with sorting it and storing it in the ram in the correct order. By doing things in this way we lower the time it takes to receive all the data needed. The output is directly to the Fast Control bus, via a multiplexer that disconnects the OpControl during a Read Event.

The other function of this FPGA is to handle the recording/playback of ZPD and BLT output data, this is done in a similar way to the Input Formatter, with the ZPD data taking the lower 21 bits, and 4 bits of ZPD data taking the upper 4 bits of the memory. The frame bit for the ZPD data is also produced in this part of the code, together with the ZPD data clock as well. It also contains the generic memory controller to do read/write to the playback memory.

**TSF Serial Command Format.**

Basic command structure is based on the Clink format, with a slightly smaller command/tag length.

Read from left to right..

```
ZCXXXXFFFFFF|<DD...>
  ||   |_Data for command, dependent on command.
  ||   
  ||   
  ||   
  ||   
  ||   
  ||   Command Flags. LSB First.
  ||    
  ||    
  ||    
  ||    
  ||    
  ||    
  ||    
  ||    
  ||    
  ||    
  ||    
  ||    
  ||    
  ||    
  ||    
  ||    
  ||    
  ||    
  ||    
  ||    
  ||    Command
  ||    "1"
  "0"
```

Command Codes:

Status 1:

<table>
<thead>
<tr>
<th>Opcode</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>Status Update No.1</td>
</tr>
<tr>
<td>Flags</td>
<td>D0 - D5: (6 bit number, lsb first)</td>
</tr>
<tr>
<td></td>
<td>0 = 1 Accept</td>
</tr>
<tr>
<td></td>
<td>1 = Read Event</td>
</tr>
<tr>
<td></td>
<td>2 = User Reset</td>
</tr>
</tbody>
</table>
3 = Sync – Not Used-- Has dedicated trace on PCB.
4 = Start Play/Rec
5 = Daq_reset

Data: None

Status 2:
 Opcode: 0100
 Function: Status Update No.2
 Flags:
   D0 - D5: (lsb first)
   0 = spare
   1 = spare
   2 = spare
   3 = play_record_mode (single, cont)
   4 = Daq Format
   5 = Daq Format

Data: None

Status 3:
 Opcode: 1100
 Function: Status Update No.3
 Flags:
   D0 - D5: (lsb first)
   0 = OpMode
   1 = Input Play Rec Mode
   2 = Output Play Rec Mode
   3 = Enable In
   4 = Enable Out
   5 = Enable Neighbour

Data: None

Sel_Mem:
 Opcode: 1010
 Function: Select which memory to work on.
 Flags:
   D0 - D1: (2 bit number for high mem)
   D2 - D5: (4 bit number, lsb first)
   0000 = in mem
   1000 = out mem
   0100 = mask mem
   1100 = version mem
   0010 = constant mem
   1010 = neighbour data mem
   0001 = FPGA1 LUT1
   1001 = FPGA1 LUT2
   0101 = FPGA2 LUT1
1101 = FPGA2 LUT2
0011 = FPGA3 LUT1
1011 = FPGA3 LUT2

Data: None

Write_Add:
Opcode: 0110
Function: Select an Address
Flags: D0 = 0(Lower 16bits) 1(Upper 16bits)
       D1-D5 undefined.
Data: 16bit Address, LSB First

Write:
Opcode: 1110
Function: Write data to mem
Flags:
       D0 = 1(32bit) 0(16bit)
       D1 = 0(normal) 1(Address increment)
       D2 = 0(singleword) 1(Multiword) (1st 16bit data is length)
       D3-D5 = undefined
Data: 16bits LSB first. First Word is the address, Then block size if block mode. Followed by data.
      In non block mode, data word follows.
      32bit transfers done <0-15><16-31>

Read:
Opcode: 0001
Function: Read Data from mem
Flags:
       D0 = 1(32bit) 0(16bit)
       D1 = 0(normal) 1(Address increment)
       D2 = (0) normal, (1) Block mode
       D3 = undefined
Data: 16bit Block size in 16bit words
Return Data: 16bits LSB first. N words of it. 32bit transfers done like <0-15><16-31>
            Data is returned starting with an 01 pattern as a header, to allow us to latch onto it.