## Purpose of the FPGA firmware

The purpose of this FPGA firmware is to couple an FPGA device with an AMChip in the same package. The primary target of the design will be the FTK tracker for ATLAS, while a certain amount of flexibility must be preserved so a range of other HPC applications (Imaging, medical, etc) can utilize the technology without major modifications to the design being necessary.

## Short overview of the main blocks of the FPGA firmware

The main blocks that comprise the firmware are:

* SSmap
* GTX Serial Transceiver Blocks
* Data Organizer
* Track Fitter

The SSmap is the part that calculares SSID’s from the full resolution hits. The documentation is lacking and more information about that part is needed, as its function could be as simple as discarding the lower bits of the hits but also it could be something way more complex than that.

The serial transceiver blocks utilizes the device’s GTX transceivers to send the formatted SSID data and the necessary instructions to the AMChip, and receive its output.

The Data Organizer is responsible for storing the full resolution hits coming from the Data Formatter according to the SSID they correspond to, so it can recover them when the match information from the AMChip is produced.

The Track Fitter gathers the full resolution hits that correspond to a AMChip produced road and computes a linear fir for each possible hit combination while using different sets of constants according to the sector each road belongs to.

## Data Organizer and Internal memory requirements

For each event, the Data Organizer has to store in a smart Data Base all the incoming full resolution hits so that when the AMChip produces the matched patterns, the relevant hit data is easily recovered. An DO architecture has been already designed for the AUX board firmware and targets an Altera Arria V device that has 24.140Kb of M10k memory available. The current target FPGA, a Xilinx Kintex (….. please put the code) contains 16.020Kb of BlockRAM memory.

The current DO supports a 14-bit SSID and uses about 4400Kb of on-chip memory. For a 15-bit SSID the needed memory would be 8300Kb and for 16-bit it rises to 16200Kb, more than what is available on the current target. The dependence on other parameters such as maximum hits per layer per event (1024), maximum hits per layer per SSID (7 or 31, depends on the layer being SCT or Pixel) is much more linear and an increase in those parameters doesn’t affect much the memory demands.

In case the increase of the SSID width to 16 bits is needed, a change in the architecture is being proposed. A coarse description of the original DO is presented. Full resolution hits belonging to same SSID are assumed to arrive one after the other. The content of different SSID is ordered at the DO input. As the full resolution hits are received in an event, they are stored consecutively in a memory (HLM – Hit List Memory). At the same time for each 1st hit of a specific SSID a memory (HLP – Hit List Pointer), addressed by that SSID number saves the address that hit was stored in the HLM. Also, for each hit of a specific SSID another memory (HCM – Hit Count Memory), also (in a way) addressed by that SSID number, updates the total amount of consecutive hits that belong to that SSID in that specific event. This way, to read the hits for each SSID coming from the AMMap, the initial memory address is recovered from the HLP, the number of consecutive hits to read is recovered from the HCP, and the hits are read from the HLM. The modification is in that the number of consecutive hits for each SSID is not stored, instead in parallel with the HLM another 1-bit wide memory with the same depth is implemented in Distributed RAM. Each memory cell corresponds to each full res hit data stored and indicates if the next memory location contains a hit from the same SSID or from a different one. So in order to read the hits, the starting address is looked into the HLP, and hits are read from the HLM until a zero is read from the distributed memory, which is being addressed using the same bus as the HLM.

The modified DO architecture will require 12000Kb instead of 16200Kb, leaving almost 4Mb for other functions on the FPGA (Track Fitter, various buffers) thus making it a viable option in terms of memory requirements. It will also make use of 16kb of distributed memory (for 8 layers and 1024 maximum hits/layer/event), which is a small amount (the selected device has 4000kb available). It will also save 8k registers (depending on SSID length) per layer, which the former architecture uses to reset the HCM in the beginning of each event, and would occupy a big slice percentage while possibly creating timing closure problems [What you mean here?].

It is worth noting that those memory utilization numbers regard the total resources occupied by both copies of the DO that are needed for event-pipelined AMChip operation. In non-pipelined mode, only one copy of the DO is needed and the resource demands are halved.

Finally for the Data Organizer, a Look-up table is needed to decode 17-bit RoadID a 128kpat AMChip produces. For each RoadID we need to know the 16-bit SSID that corresponds to each layer and the 2-bit DCvalue for it, for a total of 144 bits (8 layers). Given that, the LUT size should be 18Mbit. That is too big to store inside the FPGA device, so an external memory interface is necessary.

The Track Fitter memory requirements cannot be exactly predicted, as they depend on the number of sectors it should be able to handle, they can be estimated to be small though. Given that the TF that currently handles the entire AUX board requires 6.2Mb to be able to fit for 2500 sectors, it can be assumed that one AMChip will correspond to much less sectors and eventually could require about 50-500Kb of memory for the constants, which can be considered negligible.

## Track Fitter architecture

The current Track Fitter architecture aims at an extremely high 1 fit/ns speed, mainly because a track fitter instance has to deal with matches produced from all the AMChips in a LAMB board. It utilizes 9 different fitting units, 3 pixel fitters, 5 sct fitters and a nominal fitter. It is also targeted to a much more high-end device with more and more powerful DSP units. In the target device implementing just one fitter of each type will take up either 130% of the available DSPs (in performance mode) or 33% of the DSP resources and 40% of the device’s LUTs (in area optimized mode). It might be more suitable for the application in terms of area usage (and the timing closure headaches it can induce when the other parts of the design are introduced in P&R) and power consumption to develop a single track fitter unit that could fit all types, reusing a smaller set of DSP elements. The high frequency those DSP units can achieve and heavy use of pipelining could allow an implementation that shouldn’t be a bottleneck for the rate the road data is produced by a single AMChip. Also the TF could be coupled more closely to the Data Organizer – probably in the way of making the combiner a part of the DO - as to avoid extra device utilization overhead from synchronizing structures.

## External memory part selection

As pointed out, the AMMap cannot fit in the internal FPGA memory blocks, so use of an external memory chip is necessary. Some relevant properties of the three most probable solutions were researched and are presented below to choose the most suitable solution. The bandwidth numbers presented represent an estimation for totally random data transfers, extracted from the relevant datasheet information for each part. The power consumption information is extracted from testing scenarios described in the datasheets for various operations at max frequency. In that specific application the power consumption will most likely be lower than those numbers, so their value is mostly for comparison between the 3 available options. The prices listed correspond to the price found at online stores for quantities of 1 unit and 500-1000+ units.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Part no. | Size / data width [Mb / b] | Rand – burst data read BW [144b MT/s] | Power details [W, V] | Price [$] |
| DDR3-1066 | MT41J64M16JT-125 | 1024 / 16 | 18 - 120 | 0.3-0.6, 1.5 | 4-6 |
| QDRII+ SRAM | CY7C2165KV18-550BZC | 18 / 36 | 275 - 275 | 1.3-2.0, 1.8 | 51-66 |
| RLDRAMII | MT49H8M36BM-25 | 288 / 36 | 50 - 200 | 0.9-1.4, 1.8 | 18-32 |

1. The DDR3 option, even though it is the cheapest and has the largest capacity by far. In order to fully exploit the AMChip’s potential though, we need a sustainable bandwidth of 50M totally random reads/s which it can’t provide. Maybe for a somewhat slower application with less strict latency demands and larger data blocks to recover for each match it would be a very cost effective solution, with the added benefit of low power consumption.
2. The QDRII+ option is an overkill for the application since it can only hold data for the AMMap and for that only 50M reads/s are needed. If more than 500b of data were needed for each pattern match so we could use the extra bandwidth it would make sense to move to a bigger QDRII+ SRAM memory component but that would increase the cost to about 250$ a piece and also mean significantly bigger power consumption.
3. The RLDRAMII option seems the best in that it delivers just the necessary random read data rate while at the same time providing plenty of extra storage. While it can sustain the minimum data rate to keep up with the AMChip output, at the end of each event more data can be accessed at a much higher burst transfer rate of That extra space could be accessed using much faster burst transfers at the end of each event. It can be faster than needed but the extra bandwidth can be used to access the additional data that can be stored and might be useful on some applications.

## Verification plans – choice of HDL

This as a whole is a big FPGA project with a lot of data moving in both serial and parallel form, a lot of synchronization stages and many complex FSMs. Extensive simulation and verification in the form of functional coverage and validation of each part is a must to minimize bug-hunting in the complete design. Thus it is very important to use good verification practices from the design stage. SystemVerilog provides random stimulus generation, object-oriented methodology and abstraction and many features to enable more efficient functional verification. At the same time it is as least as good as VHDL for design entry, with the added benefit of having mechanisms to prevent accidental latch extraction and some common simulation-synthesis mismatches. In terms of vendor support, Altera fully supports SV for all devices and Xilinx introduced support in the Vivado toolchain which is the recommended environment to target 7-series and future devices. It is also supported on the Synopsys Synplify Pro toolchain. For those reasons, both design entry and testbench creation would benefit from using SystemVerilog.