FTK TrackFitter Firmware Design Summary

Jordan Webster

March 26, 2013

Abstract

Track fitting is one of the primary functions of the FTK AUX. This document describes the purpose and responsibilities of the TrackFitter, and then summarizes the current firmware design [First draft: March 23, 2013].

Contents

List of Tables

1 Fitting tracks on the AUX

The ATLAS detector is a many-layered system, designed to reconstruct particle trajectories. Particles produced in LHC collisions pass through the detector and deposit energy in each layer. The locations of these energy deposits, or *hits*, can then be used to reconstruct particle tracks. Due to the combinatorics in helical trajectories, reconstructing tracks using a full helical-fit is time consuming. FTK is a hardware tracker that approximates a helical fit so that track candidates can be identified quickly (prior to the LVL2 trigger).

The TrackFitter on is responsible for this calculation. It uses hits from 8 detector layers, three of which (*Pixels*) have 2 hit coordinates and 5 of which (*SCT*) have one hit coordinate. Tracks which have similar trajectories deposit hits in the same regions of these detector layers and we define these regions as sectors. Sectors are narrow enough that a helical fit can be replaced by a linear approximation. Sets of constants are precomputed for each sector and are multiplied by hits in the sectors to approximate the χ^2, a goodness of fit parameter, of the track fit. If a particular combination of hits comes partially from one track, and partially from another, then there will be no reasonable trajectory passing through all of the hits, and the resulting χ^2 from the linear approximation will be large. On the other hand, if the combination of hits happens to come from a single track, then it is likely that the χ^2 will be small. The χ^2 calculation is as follows:

$$\chi^2 = \sum_{i=1}^{N_\chi} \left(\sum_{j=1}^{N_{hits}} S_{ij} x_j + h_i \right)^2 \tag{1}$$

Here, x_j are hit coordinates from all 8 layers (11 coordinates in all), and S_{ij} and h_i are precalculated constants. If the resulting χ^2 is below a certain threshold, then the combination of hits is deemed a track.

Due to detector inefficiencies, it is possible for a track to not have hits in every layer. For cases like this, a fit can be done by first guessing the missing hit positions, and then calculating χ^2 associated with the combined set of real and guessed hits. The calculation is more complicated than the one showen in Eq. 1, and requires additional stored constants. An example χ^2 calculation for a track candidate with a single missing hit is carried out in Appendix A.2, but the details are not terribly important for understanding the structure of the TF firmware.

When a particle misses a layer, it is possible for hits to show up in the missed layer if they happen to come from a different particle passing nearby. For cases like this, the nominal fit from Eq. 1 can result in a bad χ^2 because one of the coordinates in the calculation comes from the wrong particule. Thus, if all hit coordinates are availible but the hit combination still fails the χ^2 cut, a *majority recovery* procedure is performed where hits are dropped from one layer at a time, and the χ^2 is recalculated.

2 Firmware responsibilities

The TF sits downstream of the Data Organizer (DO) on the AUX. Collections of hits called *roads* are fed in by the DO, and the TF is responsible for calculating χ^2 values and determining which hit combinations are tracks. The design goal is to do at least one fit per ns on average, where a single fit corresponds to a single χ^2 calculation.

Furthermore, The firmware needs to be capable of doing the different types of fits mentioned above. First and foremost, it must be capable of handling fits with no missing hits. However, combinations that are missing a single hit must also be allowed. The structure of these missing-hit fits depends on the layer that is missing a hit; when there is a missing Pixel hit, 2 coordinates need to be guessed, but when there is a missing SCT hit only 1 coordinate needs to be guessed and the calculation is slightly easier. Finally, the TF needs to be able to do majority recovery calculations for cases where all hits are present, but the calculated χ^2 still fails the track cut. Each of the fit types are defined again below, in the order of increasing complexity:

Nominal fit: A fit calculation for a combination with no missing hits (using Eq. 1)

SCT fit: A fit calculation for a combination with 1 missing SCT hit. In this case, 1 coordinate needs to be guessed before χ^2 can be calculated.

Pixel fit: A fit calculation for a combination with 1 missing Pixel hit. In this case, 2 coordinates need to be guessed before χ^2 can be calculated.

Recovery fit: When there are no missing hits but the χ^2 is above threshold, the fit is redone 8 times after dropping the coordinates from each layer, one at a time. This is by far the most expensive calculation.

2.1 Inputs

The TF recieves coordinates for one road at a time from the DO. One coordinate per layer is written on each clock edge. Since a road can have multiple hits in each layer, it often takes multiple clock edges to write a single road. Each input is described briefly below:

- **clk:** Global clock, with a frequency of 200 MHz. The maximum frequency allowed by the firmware is currently closer to 120 MHz.
- **reset:** Global reset (currently grounded).
- **read_enable:** Flow control from the FIFO downstream (currently hooked to VCC).
- **write_enable:** Flow control from the FIFO upstream (currently hooked to VCC).
- **sector_in[16]:** The sector ID corresponding to the road currently being written (from DO).
- **road_in[32]:** The road ID corresponding to the road currently being written (from DO).
- **layermap_in[8]:** A bitmask corresponding to the road currently being written (from DO). Each bit corresponds to a different layer. A value of '0' is used when the corersponding layer has no hits.
- **inpixlayers[3][32]:** A set of Pixel layer coordinates for all 3 layers for the road currently being written (from DO).
- **inpixeor[3]:** A bitmask with one bit for each Pixel layer (from DO). A value of '1' is used when the hit being written for the corresponding layer is the last hit in the road for that layer.
- **insctlayers[5][16]:** A set of SCT layer coordinates for all 5 SCT layers for the road currently being written (from DO).
- **inscteor[5]:** A bitmask with one bit for each SCT layer (from DO). A value of '1' is used when the hit being written for the corresponding layer is the last hit in the road for that layer.
- **vme_write_enable:** Enable signal for writing to constants memory (from VME).
- **vme_clk:** Clock with arbitrary frequency for writing to constants memory (from VME).
- **vme_address[20]:** RAM address for writing to constants memory (from VME).
- **vme_data[30]:** Input data to write to constants memory (from VME).

2.2 Outputs

The TF is responsible for outputing the hit information for passing tracks. One coordinate per layer can be dumped on each clock edge. Since each track has no more than one hit per layer, it takes a single clock edge to output a track. All of the outputs are currently hooked up to "dummy" MUXs that will eventually be replaced with a FIFO and more logic.

- **read_ready:** Flow control to the FIFO downstream.
- **write_ready:** Flow control to the FIFO upstream.
- **sector_out:** The sector ID for the track.

- **layermap_out[8]:** A layer bitmask for the track. A value of '0' is used when the corresponding layer is missing a hit.
- **outpixlayers[3][32]:** The set of Pixel layer coordinates for the track.
- **outsctlayers[5][16]:** The set of SCT layer coordinates for the track.

3 A top-down summary of the TF

A schematic of the entire TF design is shown in Fig. 1. The overall structure of this design is summarized and motivated in this section. The components are described in more detail in Section 4.

Given that the target clock period is 5 ns, and that the goal of the TF is to do at least one fit per ns, it is clear that fits need to be done in parallel. More specifically, to achieve this goal there need to be at least 5 fitting calculators working simultaneously. In the current design, there are 9 parallel fitters (which is actually necessary because the timing goal of 200 MHz not been reached). Each fitter can only handle 1 type of fit. There are 3 Pixel Fitters for handling fits with one missing Pixel layer and 5 SCT Fitters for handling fits with one missing SCT layer. Roughly 90% of the incoming roads have a missing layer and are handled by one of these fitters. A single Nominal Fitter is also included to handle the remaining 10% of the roads. Immediately downstream of the Nominal Fitter is a Recovery Fitter, which is used to perform majority recovery on nominal combinations. This is done for all nominal combinations, and not just those with a bad χ^2 (this is explained more in Section 4).

The road information that enters the TF starts at the Road Organizer, which is responsible for determining what type of fit needs to be done based on the input layermap (defined in Section 2.1). The road information is then passed in the direction of an appropriate fitter. For example, if the incoming layermap indicates that a single Pixel layer is missing, then the Road Organizer will pass the corresponding road information to a write-ready Road FIFO that is upstream from a Pixel Fitter.

A Combiner sits immediately downstream of each Road FIFO. The Combiner has two main responsibilities. First, it is responsible for loading the constants necessary for each fit from a Constants Memory on the chip. Each χ^2 calculation requires a set of 77-89 `signed` constants[1] that is chosen based on the sector ID. All of the Combiners on the chip talk to the Constants Memory through a Constants MUX, which allows one Combiner at a time to read from the memory. The entire set of constants is read in one clock period, and written to the Combiner where it can be passed downstream to fitters. The second responsibility of the Combiner is to produce combinations. Each road can have multiple hits in a given layer. When fitting tracks, all possible combinations of layer coordinates need to be tested. For example, if there are 2 hits in the first 2 layers, and 1 hit in every other layer, then we first need to come up with all the possible combinations–

combination 1: hit 1 from layer 1, hit 1 from layer 2, ...

combination 2: hit 2 from layer 1, hit 1 from layer 2, ...

combination 3: hit 1 from layer 1, hit 2 from layer 2, ...

combination 4: hit 2 from layer 1, hit 2 from layer 2, ...

–and then perform fits on all combinations. The expected number of combinations per road is around 5-10, but it is possible for a road to have hundreds of combinations. The Combiner produces all combinations for the road, and passes them one-by-one, along with the constants, to the Fitter downstream.

In between the Combiner and Fitter, however, is a Hit Extractor. This is a simple component that converts the ATLAS style hit coordinates into hit coordinates represented by `signed` 27-bit vectors. This needs to be done before the χ^2 calculation can begin because hits are multiplied by constants using using 27×27 bit `signed` DSP multipliers (27 bit precision is necessary due to the range of the constants). The incoming ATLAS style coordinates are 10 bit `unsigned` vectors, buried inside 16 (for SCT) or 32 (for Pixel) bit words. The auxiliary bits in each ATLAS hit word are not necessary to the TF. After the Hit Extractor converts the coordinates into 27 bit `signed` vectors, they are passed on to the fitter where the χ^2 calculation begins.

[1] Constants are stored as 27-bit vectors, so a single set of constants for a given road contains 2079-2403 bits.

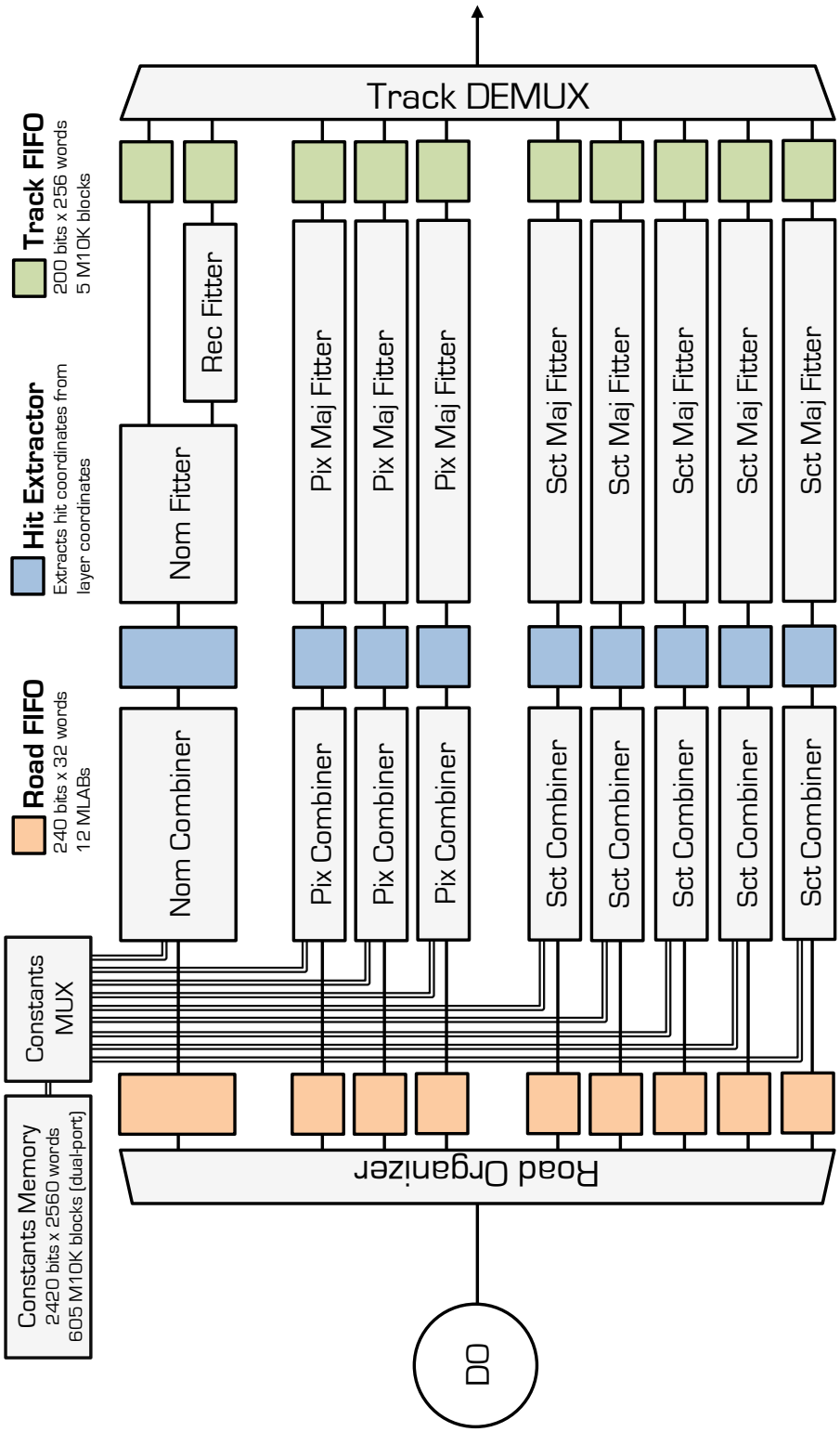

Figure 1: Schematic of the full TrackFitter design. Inputs are fed in from the left by the DO. The logic flow moves from bottom-to-top. Outputs come from the Track DEMUX. The colored components are defined above the schematic.

The fitter is a series of pipelined DSP multipliers, parallel adders, and arithmetic shift registers (ASRs). The length of each fitter pipeline is 64 clock edges. At the end of each fitter is a comparator for checking if the resulting χ^2 is less than the track threshold. If the χ^2 passes the cut, then a `write_enable` signal is sent to the adjacent Track FIFO downstream. Though it is not shown in the shematic, each fitter works in parallel with a Fitter Shift Register that has an equally long pipeline of 64 clock edges. The Fitter Shift Register reads and writes combination information–the sector ID, layermap, and hits. The outputs are connected to the Track FIFO in addition to the `write_ready` signal from the fitter, so when the fitter finds a passing track the corresponding combination information is passed on to the FIFO.

Finally, after the Track FIFOs is a Track DEMUX, which is responsible reading from one Track FIFO at a time, and passing the information to the TF outputs.

4 Descriptions of individual components

The individual components of the TF are described in detail in this section, starting from the Road Organizer, and moving downstream.

4.1 Road Organizer

The Road Organizer is the first component in the TF, so it has inputs identical to those defined above in Section 2.1. This component must have logic for (1) determining what type of fit needs to be done based on the incoming layermap, and (2) only writing to Road FIFOs that are write-ready. The former is done using some straightforward combinational logic:

```
-- Use the incoming layermap to determine what type of fit needs to be done
-- pix := missing Pixel layer
-- sct := missing SCT layer
-- nom := no missing layers
fittype <= pix when( layermap_in(0) = '0' or layermap_in(1) = '0' or layermap_in(2) = '0' ) else
          sct when( layermap_in(3) = '0' or layermap_in(4) = '0' or layermap_in(5) = '0' or
                    layermap_in(6) = '0' or layermap_in(7) = '0' ) else nom;
```

This logic is the used for determining where to write. The incoming road ID is used to determine when a new road is beginning. When this happens, the Road Organizer checks `fittype` and determines if the current Road FIFO corresponding to a fitter of that type is free (i.e. less than half full). If it is free, then the Road Organizer begins writing the road to the FIFO. If not, then it looks to the next FIFO on the next clock edge, and so on. A resource usage summary for the Road Organizer is shown in Tab. 1.

4.2 Road FIFO

The Road FIFO stores hits, layermaps, and sector IDs that are passed in from the Road Organizer. There is no reason for this FIFO to be very deep, so it is constructed from MLABs.[2] In particular, 12 MLAB FIFOs are used, each 20 bits wide and 32 words deep (640 bits each). A resource usage summary for the Road FIFO is shown in Tab. 2

4.3 Constants Memory

Each sector has a corresponding set of constants that occupy 2403 bits, and the TF needs to be able to handle around 2500 sectors. The Constants Memory is designed to hold 2560 sets of constants, or 6.2 Mb. The memory is written using a VME interface, so it is built using dual-port RAMs constructed from M10K blocks.

More specifically, a "stitched" dual-port RAM is created by combining a one containing 2048 addresses with another containing 512 addresses, each 20 bits wide.[3] The stitched result has a width of 20 bits and a

[2]FIFOs constructed from MLABs (640 bits) have a minimum depth of 32 words, while those constructed from M10K blocks (10240 bits) have a minimum depth of 256 words

[3]This is done because the Quartus MegaWizard cannot efficiently create a dual-port RAM with > 2048 addresses. For example, the MegaWizard uses 10 M10K blocks to produce a dual-port RAM with a width of 20 bits and depth of 2560 words, even though it should be possible to only use 5.

Analysis & Synthesis Resource Usage Summary	
Resource	Usage
Estimate of Logic utilization (ALMs needed)	1240
Combinational ALUT usage for logic	93
-- 7 input functions	0
-- 6 input functions	26
-- 5 input functions	6
-- 4 input functions	8
-- <=3 input functions	53
Dedicated logic registers	455
Virtual pins	2094
I/O pins	1
Total DSP Blocks	0
Maximum fan-out	455
Total fan-out	3097
Average fan-out	1.17

Table 1: Synthesis-level resource usage summary for the Road Organizer.

depth of 2560 words. 121 copies are put side-by-side, so that 2420 bits can be read from the memory every clock cycle. A resource usage summary for the Constants Memory is shown in Tab. 3.

4.4 Constants Memory MUX

The Constants Memory MUX recieves `consts_write_ready` signals and sector IDs from all of the Combiners in the design. On each clock edge, it checks to see if the any of the write ready signals are '1', looking first at the Nominal Combiner, then the Pixel Combiners, and finally the SCT Combiners. If a '1' is found, then the corresponding sector ID is passed to the Constants Memory to load the set of constants. A `consts_read_ready` signal is also sent to the corresponding Combiner so the Combiner knows when to write the constants. A short pipeline is added to the Constants Memory MUX outputs (1-3 clock cycles) to give the Quartus Fitter more flexibility. A resource usage summary for the Constants Memory MUX is shown in Tab. 4.

4.5 Combiner

A rough schematic of the Combiner is shown in Fig. 2. The Combiner is a 3-state machine with READ, WRITE, and RESET states. It is designed to behave like a clocked nested for-loop using a series of RAMs and counters. For each layer, it contains a RAM for storing hits, a counter for keeping track of the number of hits in the layer, and a second counter for iterating over RAM addresses. All three components are declared abstractly (without using megafunctions), as follows:

Analysis & Synthesis Resource Usage Summary	
Resource	Usage
Estimate of Logic utilization (ALMs needed)	491
Combinational ALUT usage for logic	334
-- 7 input functions	0
-- 6 input functions	0
-- 5 input functions	44
-- 4 input functions	90
-- <=3 input functions	200
Memory ALUT usage	208
-- 64-address deep	0
-- 32-address deep	208
Dedicated logic registers	528
Virtual pins	420
I/O pins	2
Total MLAB memory bits	6656
Total block memory bits	0
Total DSP Blocks	0
Maximum fan-out	736
Total fan-out	5603
Average fan-out	3.75

Table 2: Synthesis-level resource usage summary for the Road FIFO.

Analysis & Synthesis Resource Usage Summary	
Resource	Usage
Estimate of Logic utilization (ALMs needed)	2554
Combinational ALUT usage for logic	2672
-- 7 input functions	0
-- 6 input functions	2
-- 5 input functions	101
-- 4 input functions	127
-- <=3 input functions	2442
Dedicated logic registers	2404
Virtual pins	2433
I/O pins	35
Total MLAB memory bits	0
Total block memory bits	6151680
Total DSP Blocks	0
Maximum fan-out	7210
Total fan-out	130937
Average fan-out	10.57

Table 3: Synthesis-level resource usage summary for the Constants Memory.

Analysis & Synthesis Resource Usage Summary	
Resource	Usage
Estimate of Logic utilization (ALMs needed)	9155
Combinational ALUT usage for logic	2864
-- 7 input functions	0
-- 6 input functions	60
-- 5 input functions	109
-- 4 input functions	244
-- <=3 input functions	2451
Dedicated logic registers	4981
Virtual pins	14594
I/O pins	2
Total MLAB memory bits	0
Total block memory bits	6151680
Total DSP Blocks	0
Maximum fan-out	9787
Total fan-out	149212
Average fan-out	5.48

Table 4: Synthesis-level resource usage summary for the Constants Memory MUX (assuming an output pipeline of 1 clock edge)

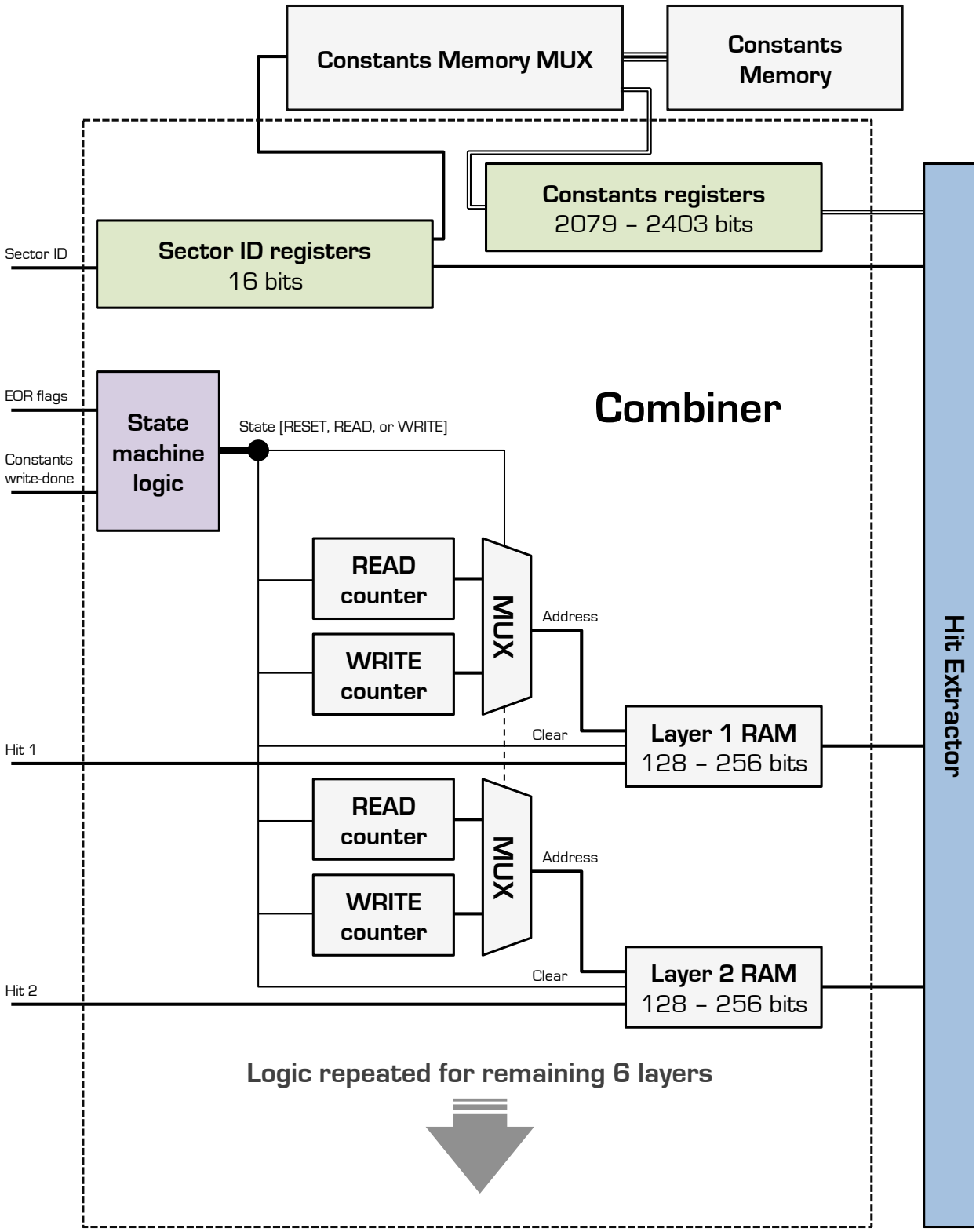

Figure 2: Schematic of a Combiner

```
-- define some new types to behave like an 8-port RAM for storing layer coordinates
type pixlayers_ram_arr is array( depth-1 downto 0 ) of tf_pixlayer;
type pixlayers_ram_matrix is array( NPIXLAYERS-1 downto 0 ) of pixlayers_ram_arr;
type sctlayers_ram_arr is array( depth-1 downto 0 ) of tf_sctlayer;
type sctlayers_ram_matrix is array( NSCTLAYERS-1 downto 0 ) of sctlayers_ram_arr;
signal pix_ram : pixlayers_ram_matrix := ( others => ( others => ( others => '0' ) ) );
signal sct_ram : sctlayers_ram_matrix := ( others => ( others => ( others => '0' ) ) );

-- use counters for read/write addresses
constant address_width : integer := required_bits(depth);
type pix_ram_count is array( NPIXLAYERS-1 downto 0 ) of integer range 0 to depth-1;
type sct_ram_count is array( NSCTLAYERS-1 downto 0 ) of integer range 0 to depth-1;
signal pix_write_addr : pix_ram_count := ( others => 0 );
signal pix_read_addr : pix_ram_count := ( others => 0 );
signal sct_write_addr : sct_ram_count := ( others => 0 );
signal sct_read_addr : sct_ram_count := ( others => 0 );
```

The Combiner is initialized in RESET mode. The RESET state clears all of the RAMs and counters, and then changes the state to WRITE for the next clock edge. In WRITE mode, sets of Pixel and SCT hits are written to the RAMs on each clock edge, and each RAM counter is incremented by 1. Each layer continues to write new hits until it sees an EOR (end-of-road) flag from the Road FIFO. This indicates that the hit that is currently being written is the last hit in the layer for the given road. When this happens, the RAM stops writing, and the corresopnding counter freezes so the total number of hits in the layer is known. Once an EOR flag is seen in every layer, and the constants and sector ID have also been written to the Combiner, the state is changed to READ. Here, `read_ready` is set to '1' and a different combination of hits is output on every rising clock edge. The process starts by outputing the first element in each RAM. On the next clock edge, the address of the first layer RAM is incremented by 1. This continues until the address of the first layer RAM is equal to the number of hits in the layer, at which point the address is reset to 0, and the address of the second layer RAM is incremented by 1. The process is finished when the address of the last layer is equal to the number of hits in the last layer, at which point the state switches back to RESET. A resource usage summary for the Combiner is shown in Tab. 5.

4.6 Hit Extractor

The Hit Extractor is designed to convert ATLAS layer coordinates into actual `signed` hit positions. The details behind this conversion are not yet finalized in the firmware, but the idea is simple. Each Pixel hit has 2 coordinates embedded in a 32 bit word. The Hit Extractor grabs the important bits from this word and converts them into two 27-bit `signed` vectors. Similarly, each SCT hit has 1 coordinate embedded in a 16 bit word, so the Hit Extractor grabs the important bits from this word and convert them into a single 27-bit `signed` vector. the resulting output consists of 12 `signed` vectors–1 for each coordinate plus a dummy word equal to 1 to be multiplied by the additive constant h_i from Eq. 1. If any hits are missing, then the corresponding coordinates are set to 0. The constants also pass through the Hit Extractor on their way to a fitter. A resource usage summary for the Hit Extractor is shown in Tab. 6.

4.7 Nominal and Recovery Fitters

A rough schematic of the Nominal Fitter is shown in Fig. 3. This fitter is designed to perform the calculation from Eq. 1. The calculation is done by first multiplying hits by constants, and then summing together the results. For example, χ_1 is calculated first by dotting a hit vector $(x_1, x_2, ..., x_{11}, 1)$ with a constants vector $(S_{1,1}, S_{1,2}, ..., S_{1,11}, h_1)$. This requires 12 separate $27 \times 27 \rightarrow 54$ bit DSP multipliers, each with a pipeline of 3 clock edges. The products are then summed by a 54 bit $\times$ 12 word $\rightarrow$ 58 bit parallel adder with a pipeline of 3 clock edges. Finally, an ASR is used to shift and then shorten the result back to 27 bits, yielding χ_1 as a 27 bit `signed` vector that can be fed into another multiplier.[4] This logic is copied 6 times to produce all 6 χ-components.

To get the final χ^2 the 6 components need to be squared and summed. For this 6 more DSP multipliers are used, and one more parallel adder for adding 6 54 bit words. The 57 bit result is then passed to a

[4]Note that the ASRs, which are used to shift the adder outputs (essentially multiplying them by a power of 2) and then shorten them to 27 bits, all keep track of overflows. If an overflow occurs at any calculation stage, the combination is dropped as a track candidate. This is true for the ASRs used in all fitters.

Analysis & Synthesis Resource Usage Summary	
Resource	Usage
Estimate of Logic utilization (ALMs needed)	5285
Combinational ALUT usage for logic	3294
-- 7 input functions	1
-- 6 input functions	478
-- 5 input functions	37
-- 4 input functions	2542
-- <=3 input functions	236
Dedicated logic registers	6770
Virtual pins	5515
I/O pins	1
Total DSP Blocks	0
Maximum fan-out	6770
Total fan-out	31856
Average fan-out	2.04

Table 5: Synthesis-level resource usage summary for the Combiner.
This report is for a generalized Combiner that can work with all fitter types. When connected to a fitter the Quartus compiler synthesizes away all ports except for those that are used by the fitter, so the implemented resource usage varies slightly from what is shown here and depends on the type of the connected fitter. This represents an upper limit.

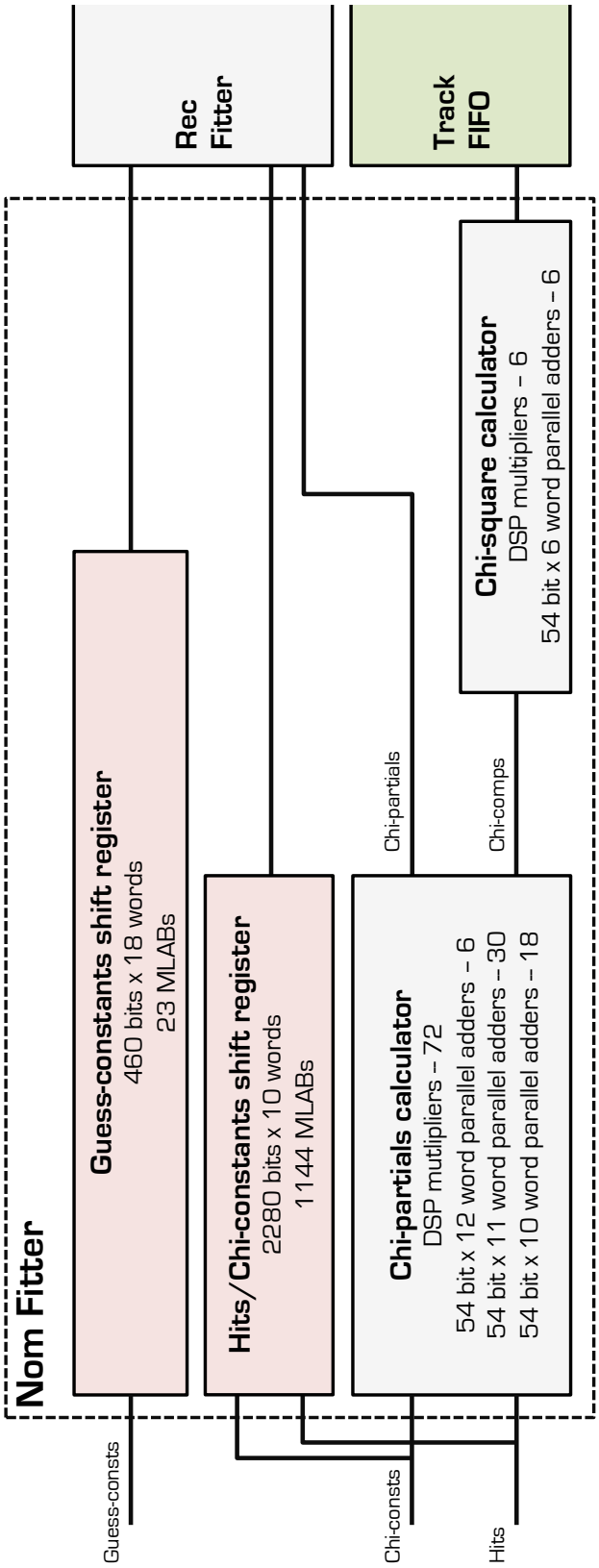

Figure 3: Schematic of a Nominal Fitter

Analysis & Synthesis Resource Usage Summary	
Resource	Usage
Estimate of Logic utilization (ALMs needed)	4141
Combinational ALUT usage for logic	0
-- 7 input functions	0
-- 6 input functions	0
-- 5 input functions	0
-- 4 input functions	0
-- <=3 input functions	0
Dedicated logic registers	2843
Virtual pins	5592
I/O pins	1
Total DSP Blocks	0
Maximum fan-out	2843
Total fan-out	8530
Average fan-out	1.01

Table 6: Synthesis-level resource usage summary for the Hit Extractor.

comparator for determining if the χ^2 value satisfies the track cut.

In addition to all this, the Nominal Fitter needs to prepare inputs for the Recovery Fitter. This means χ-partials (described in Appendix A.2) need to be calculated in addition to the χ-components. There are 8 χ-partials corresponding to each χ-component (one for each layer), and they can be calculated by summing all possible 7-layer combinations from the initial dot product. This requires 8 more parallel adders (one 54 bit $\times$ 11 word $\rightarrow$ 58 bit adder for each SCT layer, and one 54 bit $\times$ 10 word $\rightarrow$ 58 bit adder for each Pixel layer), which receive inputs from the first layer of DSP multipliers. The outputs of these adders are shifted by an ASR and reduced to 27 bits, and then passed on to the Recovery Fitter. The recovery calculation also requires the entire constants set (including all of the hit-guessing constants) and all of the hit coordinates. Thus, in addition to being used in the nominal fit calculation, the constants and hits are placed in shift registers inside the Nominal Fitter. As the χ-partials are passed into the Recovery Fitter, so are the corresponding constants and hits.[5]

Since recovery fits begin part way through each nominal fit–as soon as the χ-partials are calculated–recovery fits are performed for every nominal combination and not just the ones that fail the nominal χ^2 cut. The disadvantage of this setup is that it increases data flow downstream. For cases where the nominal fit passes the cut, the recovery fit results are not important (and are eventually discarded by logic downstream from the TF). If the recovery fits did not begin until the nominal fits finished, then recovery fits could be done only for cases where we know that the nominal fit failed. The advantage of this layout, however, is that the shift registers for storing hits and constants in the Nominal Fitter do not need to be as deep, so the resource usage is reduced.

The Recovery Fitter is essentially just a bunch of Pixel and SCT Fitters (described in the following two sections) working in parallel, so I will not describe the details here. A resource usage summary for the combined Nominal and Recovery Fitters is shown in Tab. 7.

[5]There are some subtleties here associated with synchronization. Hit-guessing constants are only used by the Recovery Fitter 8 clock edges into the pipeline, so the shift register used to store hit-guessing constants in the Nominal Fitter is 8 clock edges longer than that used to store hits and nominal constants.

Analysis & Synthesis Resource Usage Summary	
Resource	Usage
Estimate of Logic utilization (ALMs needed)	33684
Combinational ALUT usage for logic	21467
-- 7 input functions	0
-- 6 input functions	141
-- 5 input functions	1423
-- 4 input functions	336
-- <=3 input functions	19567
Memory ALUT usage	5319
-- 64-address deep	0
-- 32-address deep	5319
Dedicated logic registers	59015
Virtual pins	2728
I/O pins	2
Total MLAB memory bits	75330
Total block memory bits	24549
Total DSP Blocks	275
Maximum fan-out	70108
Total fan-out	283244
Average fan-out	2.99

Table 7: Synthesis-level resource usage summary for the Nominal and Recovery Fitters.

Analysis & Synthesis Resource Usage Summary	
Resource	Usage
Estimate of Logic utilization (ALMs needed)	11017
Combinational ALUT usage for logic	3395
-- 7 input functions	0
-- 6 input functions	17
-- 5 input functions	518
-- 4 input functions	55
-- <=3 input functions	2805
Memory ALUT usage	918
-- 64-address deep	0
-- 32-address deep	918
Dedicated logic registers	18437
Virtual pins	2380
I/O pins	2
Total MLAB memory bits	12096
Total block memory bits	3629
Total DSP Blocks	106
Maximum fan-out	20229
Total fan-out	66058
Average fan-out	2.53

Table 8: Synthesis-level resource usage summary for the Pixel Fitter.

4.8 Pixel Fitter

A rough schematic of the Pixel Fitter is shown in Fig. 4. This fitter is designed to carry out the calculation described in Appendix A.2. In short, before any χ-components can be calculated the missing coordinates need to be guessed. The hit-guessing process begins with the calculation of χ-partials, so the standard χ-comnponent calculation is carried out with the missing hits set to 0. As described in the previous section, this requires 72 DSP multipliers, 6 parallel adders, and 6 ASRs. The resulting 6 χ-partials are then multiplied by more constants to calculate a t-vector with 2 elements. Each t-vector component calculation requires 6 more DSP multipliers for multiplying χ-partials by constants, followed by a 54 bit $\times$ 6 word $\rightarrow$ 57 bit parallel adder. Furthermore, since the t-vector components need to be used in a subsequent multiplication, they must be shifted by an ASR and reduced back down to 27 bits. The t-vector components are then multiplied by hit-guessing constants to get the missing hit coordinates, requiring 2 more multipliers, followed by 2 54 bit $\times$ 2 word $\rightarrow$ 55 bit adders, followed by 2 ASRs. These missing hits can then be multiplied by constants and added back to the χ-partials to get the full guessed χ-components. This step requires 12 multipliers, followed by 12 ASRs, followed by 6 27 bit $\times$ 3 word $\rightarrow$ 29 bit parallel adders, followed by one final ASR. The χ-components can then be squared and summed just as they are in the Nominal Fitter.

This calculation has many stages, and each stage requires inputs from the previous stage, along with a particular set of constants. Shift registers are used within the Pixel Fitter to store incoming constants. Their depths are chosen to synchronize each collection of constants with the output from each calculation stage, as shown in Fig. 4. For example, as the t-vector components leave the t-vector calculator, the corresponding hit-guessing constants also leave the Guess-constants Shift Register so they can be multiplied to calculate missing hits. A resource usage summary for the Pixel Fitter is shown in Tab. 8.

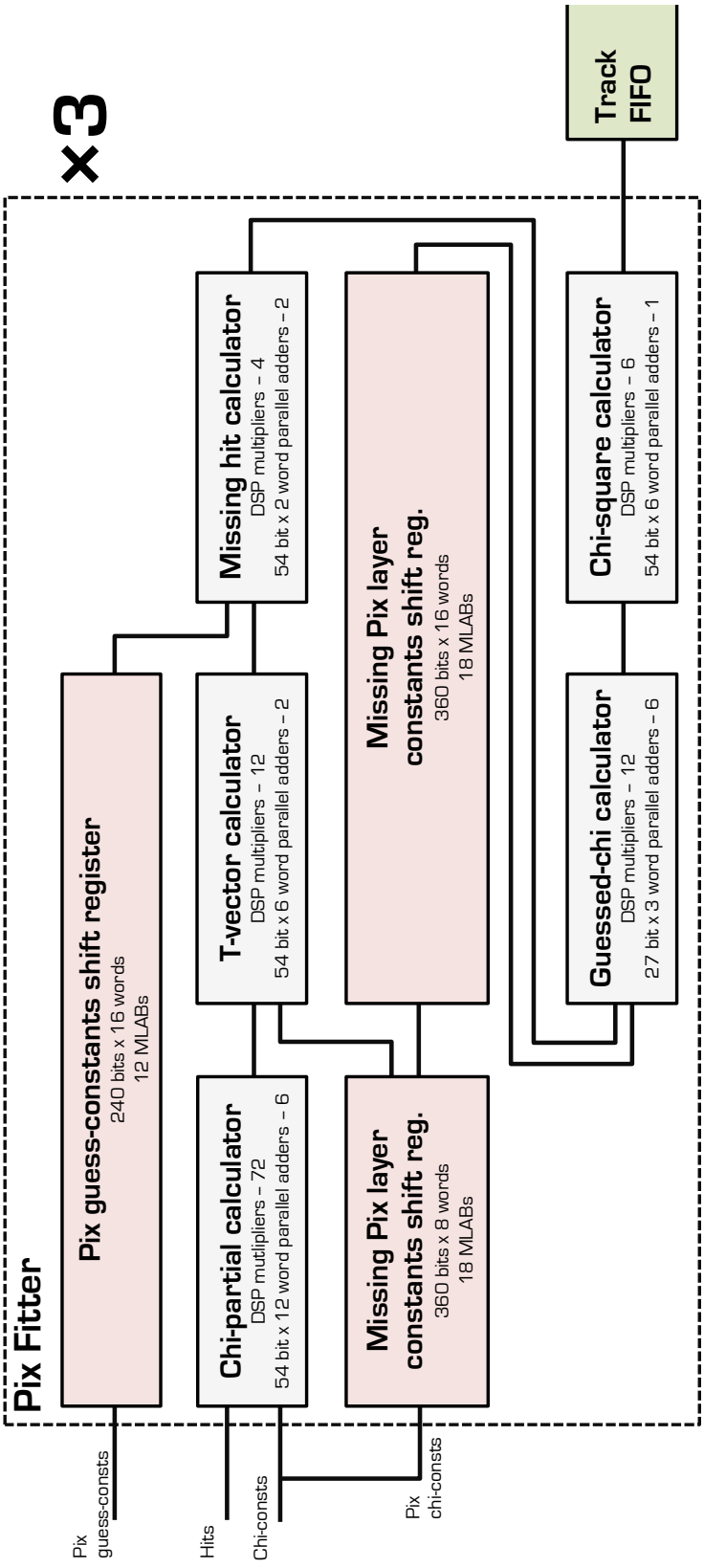

Figure 4: Schematic of a Missing Pixel Layer Fitter

Analysis & Synthesis Resource Usage Summary	
Resource	Usage
Estimate of Logic utilization (ALMs needed)	9808
Combinational ALUT usage for logic	2918
-- 7 input functions	324
-- 6 input functions	16
-- 5 input functions	145
-- 4 input functions	37
-- <=3 input functions	2396
Memory ALUT usage	513
-- 64-address deep	0
-- 32-address deep	513
Dedicated logic registers	16094
Virtual pins	2301
I/O pins	2
Total MLAB memory bits	6912
Total block memory bits	2327
Total DSP Blocks	91
Maximum fan-out	17210
Total fan-out	55415
Average fan-out	2.46

Table 9: Synthesis-level resource usage summary for the SCT Fitter.

4.9 SCT Fitter

A rough schematic of the SCT Fitter is shown in Fig. 5. This fitter is almost identical to the Pixel Fitter, except the calculation is slightly simpler because SCT hits have 1 rather than 2 coordinates. The subtle differences can be seen in the schematic by comparing the resource requirements from each component to those for the Pixel Fitter. A resource usage summary for the SCT Fitter is shown in Tab. 9.

4.10 Fitter Shift Register

Each fitter in the design has a Fitter Shift Register parallel to it that keeps track of the sector ID, layermap, and hits for each combination in the fitter pipeline. The Fitter Shift Register pipeline needs to perfectly match that of the fitter, so it is set to 64 clock edges. Since this is relatively shallow, the Shift Register is contructed from MLABs, each 20 bits wide. 11 MLABs are used to keep track of a total of 220 bits per combination. A resource usage summary for the Fitter Shift Register is shown in Tab. 10.

4.11 Fit FIFO

The Fit FIFO contains track information (layermap, coordinates, and sector ID) for output tracks. It needs to be at least as deep as the fitter pipeline. As it turns out, the shallowest FIFO that can be built from an M10K block is 256 words, so this is the depth used for the Fit FIFO. Since it needs a width of 200 bits to hang on to all the necessary combination information, it is contructed using 5 M10K FIFOs, each 40 bits wide and 256 words deep. An `almost_full` signal is used for flow control upstream. A resource usage summary for the Fit FIFO is shown in Tab. 11.

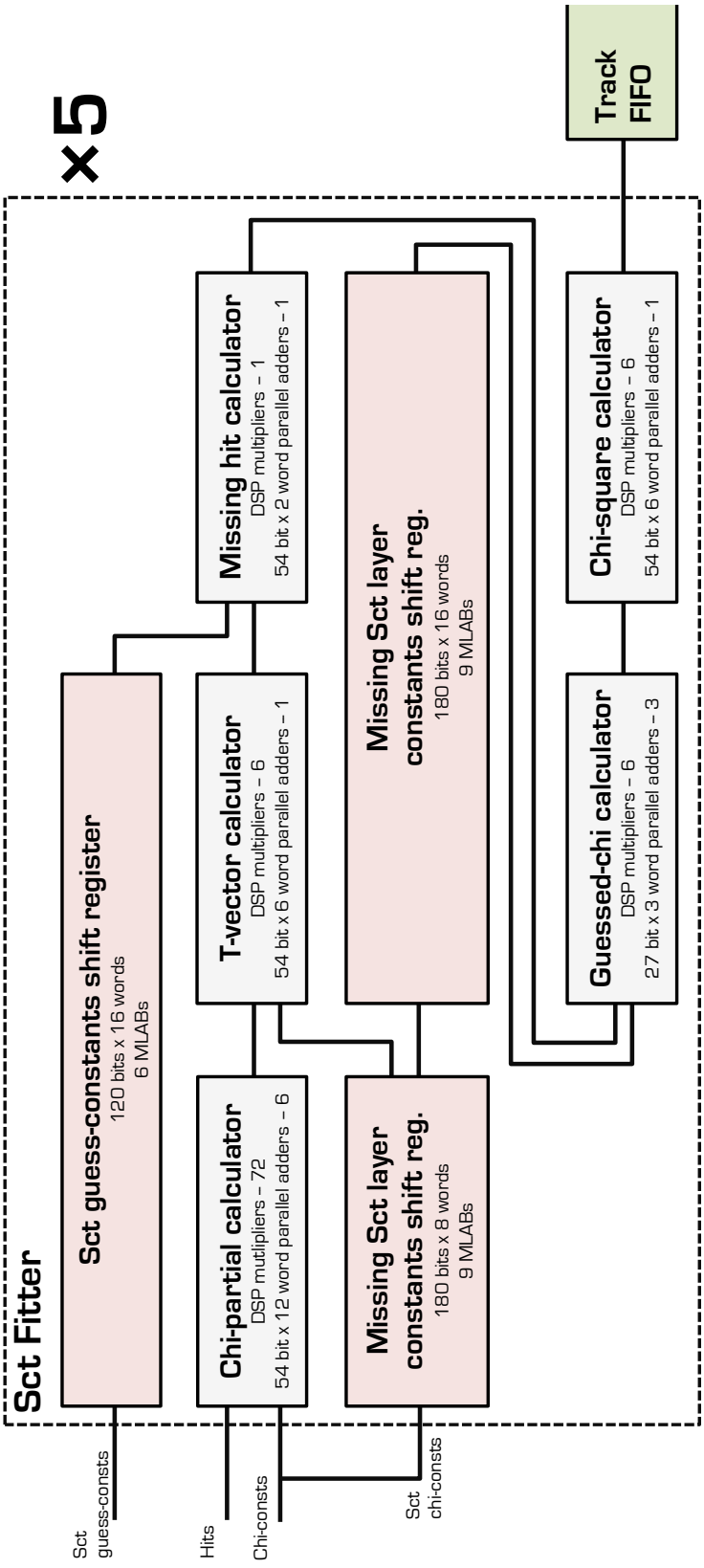

Figure 5: Schematic of a Missing SCT Layer Fitter

Analysis & Synthesis Resource Usage Summary	
Resource	Usage
Estimate of Logic utilization (ALMs needed)	475
Combinational ALUT usage for logic	267
-- 7 input functions	0
-- 6 input functions	0
-- 5 input functions	0
-- 4 input functions	0
-- <=3 input functions	267
Memory ALUT usage	402
-- 64-address deep	0
-- 32-address deep	402
Dedicated logic registers	66
Virtual pins	402
I/O pins	1
Total MLAB memory bits	12864
Total block memory bits	0
Total DSP Blocks	0
Maximum fan-out	468
Total fan-out	4274
Average fan-out	3.75

Table 10: Synthesis-level resource usage summary for the Fitter Shift Register.

Analysis & Synthesis Resource Usage Summary	
Resource	Usage
Estimate of Logic utilization (ALMs needed)	322
Combinational ALUT usage for logic	218
-- 7 input functions	0
-- 6 input functions	21
-- 5 input functions	0
-- 4 input functions	56
-- <=3 input functions	141
Dedicated logic registers	181
Virtual pins	404
I/O pins	2
Total MLAB memory bits	0
Total block memory bits	51200
Total DSP Blocks	0
Maximum fan-out	381
Total fan-out	5520
Average fan-out	5.48

Table 11: Synthesis-level resource usage summary for the Fit FIFO.

Analysis & Synthesis Resource Usage Summary	
Resource	Usage
Estimate of Logic utilization (ALMs needed)	1205
Combinational ALUT usage for logic	417
-- 7 input functions	0
-- 6 input functions	405
-- 5 input functions	2
-- 4 input functions	1
-- <=3 input functions	9
Dedicated logic registers	209
Virtual pins	1586
I/O pins	1
Total DSP Blocks	0
Maximum fan-out	209
Total fan-out	3290
Average fan-out	1.49

Table 12: Synthesis-level resource usage summary for the Fit DEMUX.

4.12 Fit DEMUX

The Fit DEMUX is a two-state machine with READ and WRITE states. In WRITE mode, it looks for a Fit FIFO that is not empty, writes the track information to registers, and then flips to READ mode. To find a non-empty Fit FIFO, it looks first at the Nominal Fitter FIFO, then to the Pixel Fitter FIFOs, and finally to the Sct Fitter FIFOs. The search for a non-empty FIFO is redone on every clock edge when in WRITE mode. In READ mode, the Fit DEMUX waits for a `read_enable` signal, and then switches back to WRITE mode. A resource usage summary for the Fit DEMUX is shown in Tab. 12.

5 Full resource usage summary

See Tab. 13

6 Design challenges

Listed below are some fo the biggest design bottlenecks and the areas where there seems to be room for improvement:

- The full AUX does not make timing goals, and this is probably mostly due to the TF since it uses the majority of the resources on the chip. The timing can probably be improved by adding pipelines in the right places, and reducing fanout. That largest fanout occurs in the Constants Memory MUX, where constants sets need to be sent out to 9 different fitters.
- The constants sets needed for each fit are large (2403 bits) and have to be passed all over the chip for DSP multiplications. If this can somehow be done more efficiently, it would probably help with timing and resource usage.

Analysis & Synthesis Resource Usage Summary	
Resource	Usage
Estimate of Logic utilization (ALMs needed)	147643
Combinational ALUT usage for logic	81300
-- 7 input functions	18
-- 6 input functions	7003
-- 5 input functions	4034
-- 4 input functions	21246
-- <=3 input functions	48999
Memory ALUT usage	14180
-- 64-address deep	0
-- 32-address deep	14180
Dedicated logic registers	272716
Virtual pins	496
I/O pins	2
Total MLAB memory bits	259522
Total block memory bits	7119327
Total DSP Blocks	1048
Maximum fan-out	306709
Total fan-out	1275079
Average fan-out	3.27

Table 13: Synthesis-level resource usage summary for the full TF.

- It might make more sense to load the constants for each road inside the Road Organizer. That way, the Constants Memory MUX could be removed, and the different Combiners would no longer have to fight over read-access to the Constants Memory. This idea was originally squandered because it would require the Road FIFOs to be much wider so they could store the entire constants set.
- It might be possible to remove the Road FIFO altogether if some more intelligent logic is used in the Road Organizer.
- Depending on the run conditions, the largest bottleneck could be at the TF inputs. Suppose the maximum number of hits per layer in a road is 2 on average. Assuming the timing goal of 200 MHz is reached, it would take an average of 10 ns to write each road (because the TF receives one hit per layer on each clock edge). The average number of combinations or fits per road may be around 5, in which case the maximum number of fits per ns would be around 0.5. This potential problem becomes much more of an issue if the timing goal is not reached.
- There are some "unlucky" scenarios that could cause the TF to get backed up. For example, suppose 10 roads with no missing layers are fed into the TF back-to-back. Since there is only one Nominal Fitter, all of these roads need to get processed by the same Combiner. If the first of these roads happens to have a lot of combinations, then the Combiner will get stuck processing the first road, and the rest of the combiners and fitters will sit idly waiting for the nominal Combiner to finish.
- FIXME – definitely more stuff to add...

7 Some general notes about the firmware

- Nearly all of the types and constants used in the firmware are defined in `./packages/TFGen.vhdl`.
- Nearly all of the ports in the design that are either `std_logic_vector`s or arrays of `std_logic_vector`s in the design are labeled with "in"/"out" plus a descriptive title. In almost all cases, if the port is just a `std_logic_vector`, then "in"/"out" is written after the descriptive title (e.g. layermap_in, sector_out). On the other hand, if "in"/"out" is written before the descriptive title, then the port is almost always an array of `std_logic_vector`s (e.g. inpixlayers, outsctlayers).
- In an attempt to help with timing, shift registers were added to the ends of several components to pipeline the outputs. This was accomplished by initializing an array, and shifting the elements in the array on each clock edge, as in the following example:

```
architecture example of TFComponent is

  type output_shiftregister_array is array( OUTPUT_PIPELINE_LENGTH-1 downto 0 ) of tf_output_type;
  signal output_shiftregister : output_shiftregister_array;

begin

  process( clk )
  begin
    if( clk'event and clk = '1' ) then

      -- shift all elements in the pipeline to make room for the next output
      for i in 0 to OUTPUT_PIPELINE_LENGTH-2 loop
        output_shiftregister(i) <= output_shiftregister(i+1);
      end loop;

      -- some logic for setting the first element in the pipeline
      output_shiftregister(OUTPUT_PIPELINE_LENGTH-1) <= determine_output(input);

    end if;
  end process;

  -- define the actual output at the end of the pipeline
  output <= output_shiftregister(0);

end example;
```

In some cases, this makes the firmware look more complicated than it really is.

Appendix

A χ^2 Calculation

A.1 Calculation for nominal fits

To calculate the χ^2 for a given road, the TrackFitter needs to compute N_χ χ-components, where N_χ is equal to the number of degrees of freedom in the linear approximation (i.e. the number of hit coordinates minus the number of helix parameters). For the 8 layer configuration that is currently used, there are 6 χ-components. Each χ-component can be calulated using the following:

$$\chi_i = \sum_{j=1}^{N_{hits}} S_{ij} x_j + h_i \quad : \quad i = 1, ..., N_\chi \tag{2}$$

where N_{hits} is the total number of hit coordinates x_j, and the matrix S_{ij} and vector h_i consist of constants to be stored in a memory on chip. The χ^2 is then calculated by squaring the components and summing them:

$$\chi^2 = \sum_{i=1}^{N_\chi} \chi_i^2 \tag{3}$$

A.2 Calculation for fits with missing hits

Occasionally, due to detector inefficiencies, a track will not have hits in all 8 layers used by FTK. When this happens, the TF does not have enough information to perform the linear approximation in Eq. 2 because one of the x_j coordinates is missing. Fortunately, it is possible to guess missing hit positions by calculating the hit coordinates that minimize the resulting χ^2. This proves to be a useful strategy when only one layer is missing; rather than discard the combination, the TF benefits by guessing the missing hit and then continuing with the full fit χ^2 calculation.

At high luminosity (when there are lots of tracks), when a track is missing a hit, it is probable that the superstrip will contain other hits that are not associated with the track. For these cases the TF uses a majority recover procedure: when a combination with hits in all layers has a χ^2 value above the track threshold but below a certain upper limit, the fit is redone one layer at a time after systematically dropping the hit in the layer and guessing the dropped hit coordinates.

Missing hit coordinates can be calculated on-the-fly by the TrackFitter using a series of linear calculations, provided some additional constants are stored in memory. The details behind these calculations are summarized in the following document: https://twiki.cern.ch/twiki/pub/Atlas/FastTracker/ftk_majority_v3.pdf. The steps that are most relavent for the firmware design are described here.

The first step is to calculate χ-partials. These are the χ-components calculated with the missing coordinates set to 0. Suppose layer ℓ is missing a hit, and $\hat{\ell}$ is the set of missing hit indices. Then the χ-partials, $\chi_i^{\hat{\ell}}$, are defined by the following equation:

$$\chi_i^{\hat{\ell}} = \chi_i - \sum_{\hat{j} \epsilon \hat{\ell}} S_{i\hat{j}} x_{\hat{j}} \quad : \quad i = 1, ..., N_\chi \tag{4}$$

If the χ-components are not known in advance, as is true for Pixel and SCT fits, then it makes more sense to write the equation as follows, where $\check{\ell}$ is the set of hit indices that are not missing:

$$\chi_i^{\hat{\ell}} = \sum_{j \epsilon \check{\ell}} S_{ij} x_j + h_i \quad : \quad i = 1, ..., N_\chi \tag{5}$$

Once the χ-partials are known, they are multiplied by constants S_{ij} to calculate a t-vector with length equal to the number of missing coordinates (2 for a missing Pixel hit, 1 for a missing SCT hit):

$$t_j = -\sum_{i=1}^{N_\chi} S_{ij} \chi_i^{\hat{\ell}} \tag{6}$$

Elements of the t-vector are then multiplied by elements of an auxiliary constants matrix that is used only for hit-guessing, C^{-1}, to arrive at the missing hit values. For cases where an SCT layer is missing, only one coordinate needs to be guessed, and the following equation is used:

$$\hat{x}_a = C_{aa}^{-1} t_a \tag{7}$$

For missing Pixel layers, since the t-vector has 2 elements, the missing hits are calculated using the following equations:

$$x_a = C_{aa}^{-1} t_a + C_{ab}^{-1} t_b \tag{8}$$

$$x_b = C_{ba}^{-1} t_a + C_{bb}^{-1} t_b \tag{9}$$

This means that guessing a missing Pixel hit requires 4 additional constants, and guessing a missing SCT hit requires 1 additional constant. For each sector, in addition to the nominal constants stored on chip, 12 constants need to be stored for guessing missing Pixel hits (4 per layer), and 5 need to be stored for guessing missing SCT hits (1 per layer).

Once the missing hit coordinates are guessed, the χ-components can be calculated from the χ-partials in just a few clock cycles by adding on the contributions due to the guessed hits. Then the recovered χ-components can be squared as in Eq. 3 to yield a final result.