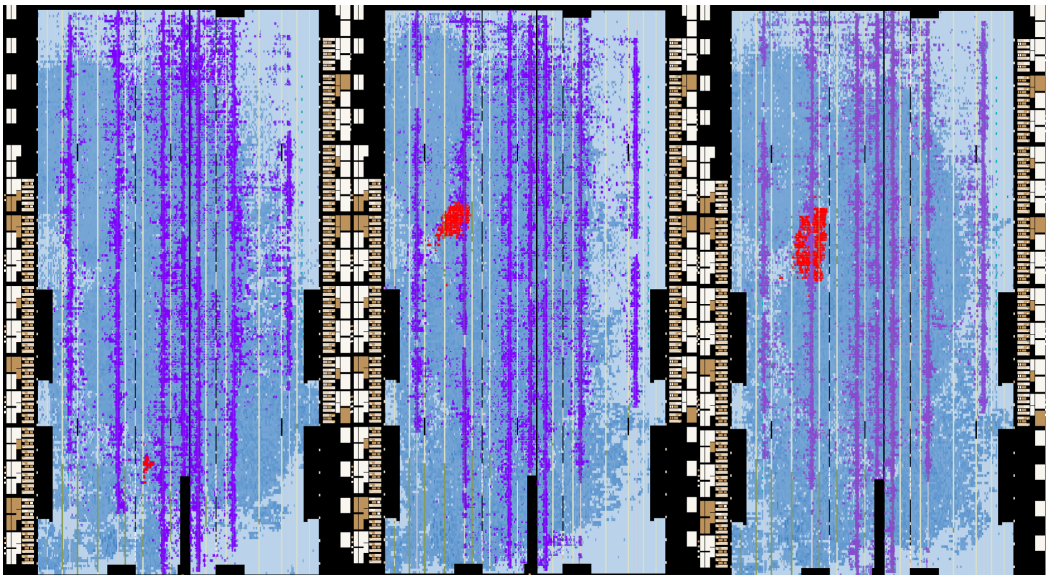
CERN Summer Student Programme 2025

ATLAS Experiment

Testing and Implementing Address Decoders of the LATOME Firmware



Bertil Cold

MSc. Student, Niels Bohr Institute, University of Copenhagen.

1 The ATLAS Experiment - LAr Calorimeter

The ATLAS (**A** **T**oroidal **L**HC **A**pparatu**S**) Experiment is one of two multipurpose collider-detectors at CERN's Large Hadron Collider (LHC). As many other collider-detectors before it, ATLAS has an 'onion-like' structure with a vertex detector, tracking chamber, electromagnetic and hadronic calorimeters and muon chambers, along with a forward calorimeter and endcap. This project revolves around the firmware of ATLAS' electromagnetic-, endcap- and forward calorimeter, the so-called Liquid Argon (LAr) Calorimeter (LArCal).

One of the primary goals of ATLAS were to discover the Higgs boson via the channels $H \rightarrow \gamma\gamma$ or $H \rightarrow 4l$, which it did in 2012[3]. This requires an effective and precise EM calorimeter and a triggering system looking for high energy deposits in the EM calorimeter. The LArCal is structured as an accordian with interleaving layers of lead absorbers, liquid argon and electrodes[3]. A relativistic charged particle going through the calorimeter will initiate particle showers in the absorber, the particles of which will then ionize the argon, and the free electrons will drift to the electrodes and generate an electrical signal. The electrical signal is sent to an Analog-to-Digital-Converter (ADC) which digitizes it and sends it out of the experimental cavern and to the backend electronics in an adjacent cavern. The shape of the ADC signal is seen in Fig. 1.

From the proton-proton collisions delivered by the LHC every 25ns ATLAS generates on the order of $\sim$ 60 terabytes of data pr. second. Most of the data comes from low-energy strong interactions, which are uninteresting to most physics analyses. Therefore, the amount of data needs to be severely reduced in multiple steps. The first of these data-reduction steps is the Level 1 triggering system, which is designed to save or discard an event with a latency of only 2.5μs[3]. The level 1 triggering system related to the LArCal triggers on high transverse energy, E_T[3][1]. A specially designed piece of hardware electronics, called a LATOME (LAr Trigger prOcessing MEzzanines), calculates the transverse energy at a very high rate using FPGAs (Field Programmable Gate Array). Originally, the LArCal was segmented into $0.1\Delta\eta \times 0.1\Delta\varphi$ 'trigger-towers', but with the increase in the LHC's luminosity in Run-3 a finer granularity of the triggering system was needed[1]. For this purpose the LArCal is segmented into $\sim 0.025\Delta\eta \times 0.1\Delta\varphi$ so-called Super-Cells (SC). As such, one LATOME board, ie. one FPGA, is designed to calculate and transmit energies of up to 320 SCs with an estimated latency of 350ns[2].

[1] Transverse energy is defined similar to transverse momentum and for light relativistic particles, the two are approximately equal.

2 LATOME Firmware

The LATOME firmware consists of 4 major blocks along with minor control and monitoring functions[2]:

1. **Input Stage**: ADC data is collected and buffered at a clock rate of 320MHz.
2. **Configurable Remapping**: Data from nearby SCs is grouped together and sent on to the User Code.
3. **User Code**: Energy is calculated from SC data sent by the Configurable Remapping using a FIR filter.
4. **Output Summing**: Group the SC energies into specified $\Delta\eta, \Delta\phi$ regions.

At the heart of the User Code is the FIR filter. A FIR filter is basically a discrete convolution, where you convolve an input signal with a pre-defined set of coefficients, a_i. The result of this convolution is then the transverse energy detected in a given SC at a given bunch crossing. Mathematically this is expressed as[2]:

$$E_T = \sum_{i=0}^{3} a_i \cdot (ADC_i - ped - base_i), \tag{1}$$

where *ped* is a pedestal value of the SC, ie. the null-signal, and $base_i$ is a baseline correction dependent on the current bunch crossing's number in the orbit. The baseline correction has to do with the fact that an ADC signal lasts way longer than 25ns, and therefore over the course of one orbit signals will overlap. This overlap causes a statistically predictable distortion to the signals later in the orbit. See Fig. 1 for an idea of the shape of a single ADC signal. The coefficients, a_i, in Eq. 1 are determined via calibration and are unique for each SC. Along with calculating E_T, the UserCode must also calculates $E_T\tau$, with τ being the phase of the pulse. This requires an equation mathcing Eq. 1, but with new constants. One FPGA must thus be able to write, read and store at least 3200 $(= 320 \cdot (4+1) \cdot 2)$ constants in 32-bit registers.

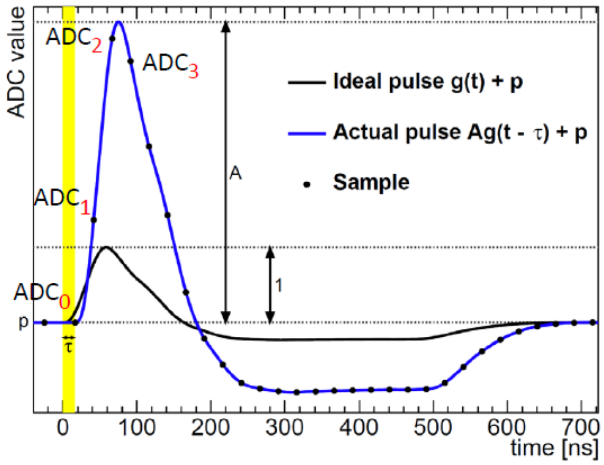


Figure 1: *Sketch of an ADC signal with markers indicating different samples and pedestal value, adapted from [2]. Not included is the baseline.*

2.1 Address Decoding

When writing values to a large set of registers one can of course not connect 3200 wires from the input device to the registers on the FPGA. Instead, one must employ an address decoder. In Fig. 2 the basic idea of an address decoder is illustrated. It takes one input signal - the constant to be written - and one control signal - the address of the register to write it to. For a 2-bit decoder like the one in Fig. 2 up to $2^2 = 4$ different registers can be addressed. A 32-bit address decoder will be able to address $2^{32} = 4\,294\,967\,296$ addresses, and so on. If we naively apply this address decoding to every single register, this will result in a lot of routing across the FPGA. To overcome and minimize this routing problem, we use what is called 'shift registers'.

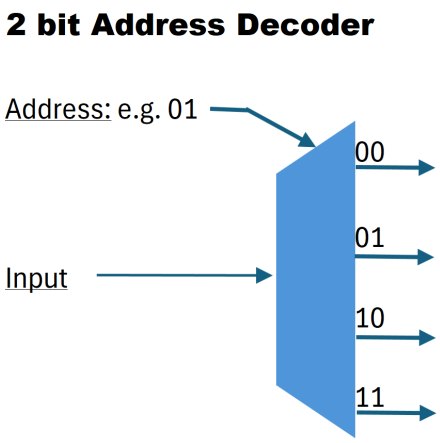


Figure 2: *Illustration of a 2-bit address decoder*

2.2 Shift Registers

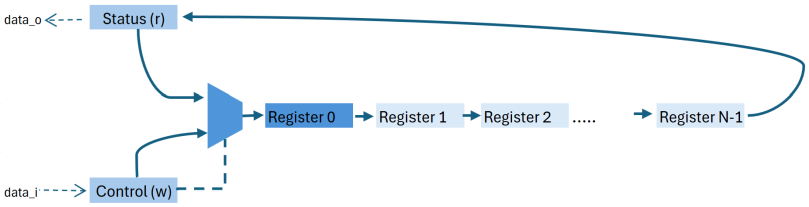


Figure 3: *Illustration of a shift register with 'Control' as input and 'Status' as output to the IP-Control. The trapezoid symbolizes a multiplexer controlled by 'Control', allowing to choose which values are inserted to the front of the shift register.*

A shift register consists of a 'daisy chain' of registers. This means that the input of one register is connected to the output of a previous register. Of course the output of the registers is also connected to the FIR filter itself. The front and back of the chain is connected to the IP-Control (IPctrl), ie. the functionality that writes ('Control') and reads ('Status') coefficients. This is all illustrated in Fig. 3 above. To ensure that nothing is lost from the daisy chain, whenever something is read from the registers it must also be written back into them. This is taken care of by the multiplexer symbolized by the trapezoid of Fig. 3. This controls what to write to the front of the shift register - either what comes from Status or what comes from Control.

2.3 The AxN Architecture

In this project different architectures of shift registers and address decoders were investigated, tested, and implemented. Stacking A shift registers of depth N on top of each other yields AxN registers - a number which should equal around 3200. Fig. 5 in Appendix A illustrates the idea. For this project four different address decoders were tested:

1. **8x320:** In this architecture each 'coloumn' of registers represent 1 SC. Taking advantage of the fact that the coefficients, a_i, take up 18 bits and pedestals 14 bits, the pedestal of each SC is written in the same register (32 bits) of one of the coefficients. Thus, only 8 registers are needed pr. SC.

2. **108x30 (legacy):** In this architecture, 108 'streams' ($\sim$ addresses) contain constants and pedestals of E_T or $E_T\tau$ of 6 SCs ((4 coefficients + 1 pedestal)$\cdot 6 =$ 30 registers). This is what is currently implemented in v6 of the LATOME firmware.

3. **54x60:** This archirectures employs the same scheme as above, with 6 SCs pr stream, but here E_T and $E_T\tau$ coefficients are written into the same stream, thus halfing the number of streams, but doubling their depth.

4. **3200x1:** Every coefficient/pedestal gains its own address and register. No daisy chaining of registers is employed.

3 Results

The following section shows the results after compiling the different address decoder architectures with the full LATOME Firmware using Intel's Quartus Prime software. The firmware was placed and routed onto an FPGA of the Arria^{TM} Family[2]. The main figures of merrit will be the needed number of Adaptive Logic Modules (ALMs) and the impact on the routing.

3.1 Resources

In Table 1 key numbers related to the needed amount of ALMs are presented for the compilations of the different address decoders. For the first three architectures, the needed amount of ALMs differs only on the order of $\sim$1000, ie. not very much compared to the total. However it is clear that 8x320 needs the fewest ALMs. The naive 3200x1 architecture needs far more ALMs than the rest, and in fact the compilation didn't get further than this - the routing stage failed due to congestion.

ALMs Needed	**8x320**	**108x30**	**54x60**	**3200x1**
Total	157 208	159 317	158 161	183 070
For registers	75 809	75 701	75 487	79 034
Full User Code	6 612	8 841	7 746	32 384
Top Level	321	1 246	774	...
Address Decoder	124	1 446	732	25 668

Table 1: *Table showing key numbers from the Quartus compilations. The dots in the 5th coloumn of the 5th row indicate that this data was not obtained from a compilation and deleted due to human error.*

3.2 Placement

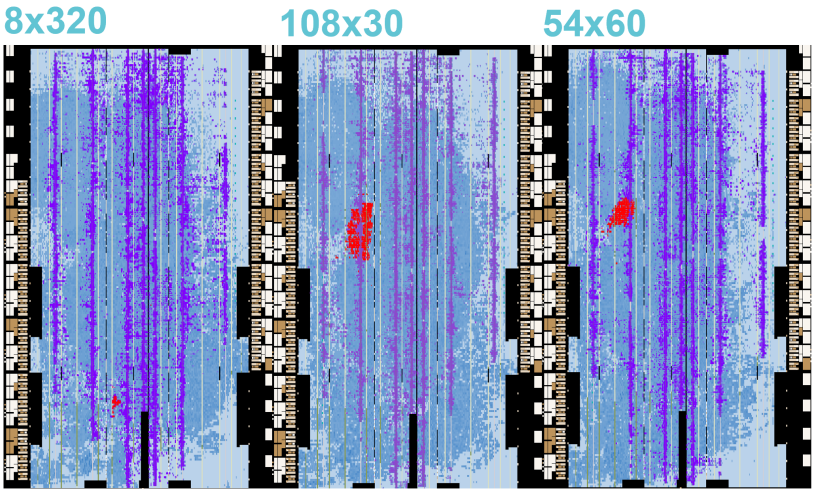


Figure 4: *Maps of the FPGA showing in blue the placement of logic devices for the entire firmware, in red for just the user code address decoder (this project), and in purple the entire user code*

Above is shown the placement of logic devices on the FPGA for the aforementioned three architectures (excluding 3200x1 since it didn't compile properly and is 'out of the picture'). The placement clearly resembles the picture drawn by Tab. 1 - the more ALMs needed, the more space the address decoder (shown in red) takes up.

3.3 Routing and Timing

For briefness, the heatmaps of wire usage across the FPGA for the compilations of the three different architectures are shown in appendix B for long wires and C for short wires. They are found to be very similar. When compiling the same project in Quartus multiple times, it was observed that the wire usage would change slightly every time along with the timing. More wire usage would mean fewer timing violations and vice versa. Because of this relationship and the fluctuations in the compilation results, the slight difference in routing and timing found among the three architectures is taken to be statistically insignificant.

4 Conclusion and Discussion

An initial goal of this project was to investigate the effectiveness of using what is known as a 'Partial' address decoder. By default, addresses from the IPctrl are set to be 32 bits wide. However, when decoding eg. only 8 addresses in hexidecimal code, only 4 bits are needed to differentiate the addresses. Thus, not all 32 bits of the incoming address signal must be compared to find the correct address - enter the 'Partial Address Decoder'. This idea of partial address decoding was tested in Quartus, for the 108x30 architecture and no difference was found. It is therefore, believed that Quartus itself optimizes this and implements a partial address decoding whenever possible.

Another curious feature of Quartus is, what is chosen to be called 'Dense Packing'. Consider the following example: if 18 bits are written thrice into a shift register of depth 3 as defined in the VHDL file, Quartus is believed to only employ 2 physical registers on the FPGA. This is because $3 \cdot 18\,\texttt{bits} = 54\,\texttt{bits} < 2 \cdot 32\,\texttt{bits}$, ie. if Quartus can pack the 54 bits of information into only 2 registers instead of 3 it will do so. This becomes an issue for the 108x30 and 54x60 architectures, because there we must always make place for 18 bits of information when writing, since both 18 bit coefficients and 14 bit pedestals enter into the same daisy chain of registers. Therefore, effectively, the pedestals take up 18 bits on the FPGA, 4 of which are unused. This is also why the trick of writing the pedestal in the same register as one of the coefficients is not used for 108x30 and 54x60 architectures. If we did so, then we would always have to make space for 32 bits when writing and a lot bits would be unused. In the 8x320 architecture, all registers of a given daisy chain are set to store the same 'kind' of information (all words are of same width) and there are thus no unused bits.

To conclude: the 8x320 architecture is optimal, but only slightly. This is evident from the numbers shown in Tab. 1 and maps in Fig. 4. The routing does not seem to be affected by the change in address decoder architecture, but more compilations should be made to get a definitive answer and obtain better precision on routing/timing results. As more functionality is packed into the firmware, more constants must be store onto the FPGA, increasing the need of a reliant and resourceful address decoder. The point made above, concerning 'dense packing' will also grow more important as the variety of information stored increases. To maximize Quartus' dense packing feature, constants of different bit width should not be stored in the same shift register.

Acknowledgments

Thank you to the LATOME HLS team - Marcos, Dabson, Nick, Melissa, Huacheng, Yi-Ling and Adriana - for taking their time to explain simple concepts of firmware design and digital logic to a complete newcomer, and for creating a nice and pleasant working environment.

References

[1] G. Aad et al. "The Phase-I trigger readout electronics upgrade of the ATLAS Liquid Argon calorimeters". In: Journal of Instrumentation 17.05 (May 2022), P05024. DOI: 10.1088/1748-0221/17/05/P05024. URL: https://dx.doi.org/10.1088/1748-0221/17/05/P05024.

[2] G. Aad et. al. "ATLAS LAr Calorimeter trigger electronics phase I 2 upgrade: LATOME Firmware Specification". In: DRAFT VERSION (2023).

[3] The ATLAS Collaboration. ATLAS: A 25-Year Insider Story of the LHC EXperiment. World Scientific, 2019.

Appendix

A - 2D Shift Register

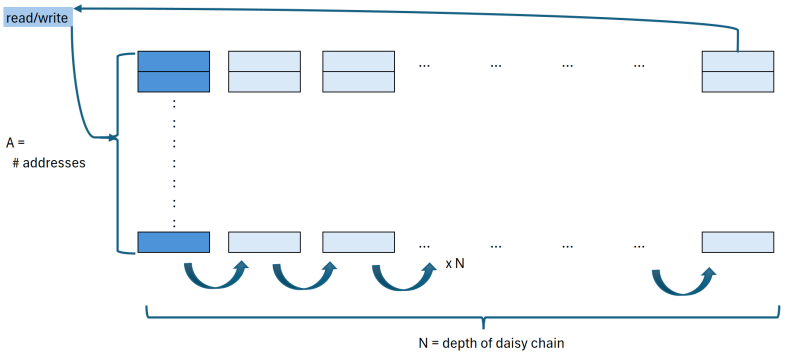


Figure 5: *Illustration of the AxN architecture of shift registers*

B - Routing, Long Wires

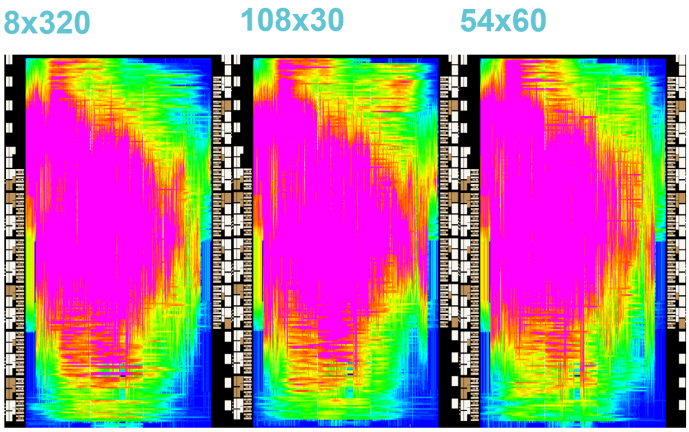


Figure 6: *Heatmaps of how the long wires are used across the FPGA for compilations of the different architectures*

C - Routing, Short Wires

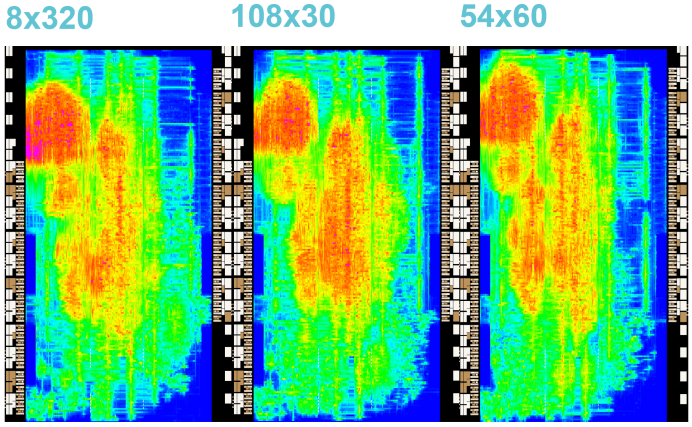


Figure 7: *Heatmaps of how the short wires are used across the FPGA for compilations of the different architectures*