Conception of digital architectures - Architecture Report.

Matias Barla and Tomas Gonzalez

November 2019

1 Introduction

1.1 Objective of the report.

The objective of this report is to present and explain the selected architecture. Also present its estimated performance, speed, cost and size and state possible improvements to the architecture.

1.2 The Fast Fourier Transform

The algorithm to implement is a Fast Fourier Transform using the Butterfly implementation of Cooley-Tuckey. Despite that in the example C code given the numbers are represented in floating point, for this implementation fixed point representation will be used. This number representation is better suited for hardware implementation and limits the complexity of the operations. For this architecture (18,5) was selected, 18 bits before the point and 5 after, but the architecture can be easily adapted to more.

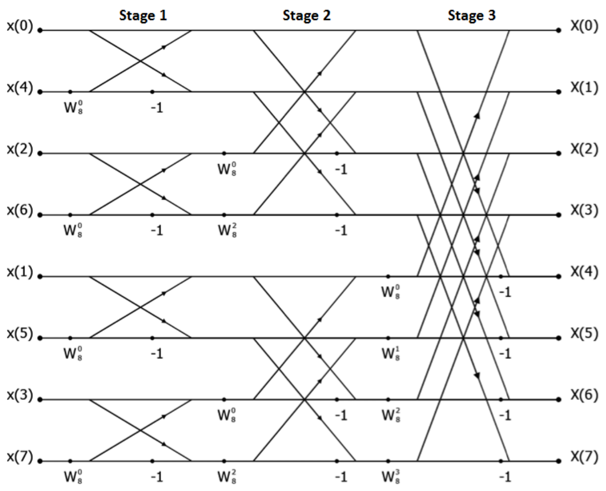

Figure 1: Representation of the FFT

As we can see, this fast Fourier transform algorithm uses 4 butterflies per stage and 3 stages to complete the transformation. After analyzing this algorithm we decided that it could be possible to implement it with just 1 butterfly that changes its entries and weights depending on the stage and moment.

2 FFT Architecture

2.1 Resume

The decided architecture is a recursive one. Our intention was to make it as recursive as possible , considering the amount of clock cycles that we can use in audio decoding without a problem. For that purpose, we created a butterfly implementation that only uses 2 multipliers and is controlled by an FSM. We also implemented a Stage architecture that uses only one Butterfly and memory reading and writing , and an Top FSM that controls each stage of the FFT and the inputs of the system. The outputs are sent in parallel from a register. With this architecture recursive is possible to minimize the surface and cost of the implementation.

2.2 Butterfly

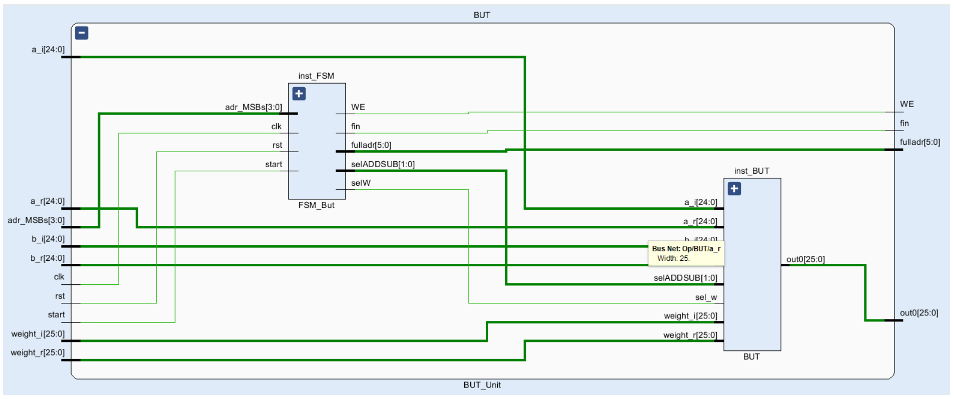

Figure 2: Butterfly Unit. FSM and Operation Unit

The Butterfly implementation receives the 4 inputs (a_r, a_i, b_r, b_i) and returns in memory the 4 outputs 4 clock cycles after. In order to know where in the memory to put the outputs, an address input is required , that marks the start of the memory address. The four outputs are then put in order in the memory starting from that address, whereas in the last stage the outputs are put in different out registers in order to send them to the block ”frequency processing” in parallel. The FSM of the implementation is in charge of controlling the memory and selecting which operation is done.

2.2.1 Operation Unit for the Butterfly

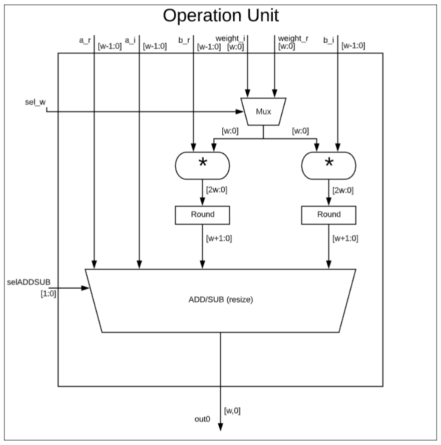

Figure 3: Butterfly implementation. Operative unit.

One key feature of our architecture is this operating unit that is capable of using only 2 multipliers to perform all its operations (with the help of the FSM). By reducing the multipliers used we greatly reduce the cost and surface of our architecture, but increment the latency of the process. Since our implementation is completely recursive, this block will be implemented only once, thus, the biggest possible size must be used in the implementation. Given how the size of the information increments by one bit in each stage of the FFT process, the size of the Operation Unit will be 25 input bits and 26 output bits that will be managed differently depending of the stage. When viewing the net-list of the implementation we observed that the ADD/SUB block was implemented with 6 add/sub blocks and 3 multiplexers, which can be optimized at a second developing stage.

2.3 Stage

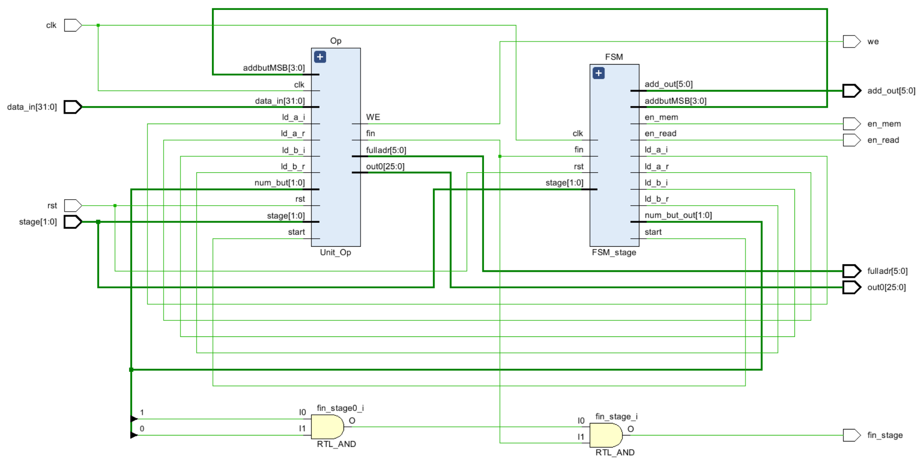

Figure 4: Stage Unit. FSM and Operating unit

The Stage Unit is in charge of re-using the Butterfly Unit 4 times in order to complete a stage of the FFT. The FSM gives the address to the memory according to the stage it is in and the number of Butterfly (of the 4 possible per stage) its implementing. Using those memory addresses the FSM controls the loading of the registers of the Operative Unit. The registers are needed because the 4 inputs must be available to the Butterfly unit during the hole operation. Also knowing the stage and number of Butterfly the FSM controls which weights to put as inputs for the Butterfly Unit, weight that are constants saved in hardware. The Butterfly number that is being implemented is counted internally but the stage is given to the unit by a top unit. Simply put, the stage unit connects to the memory and loads the correct values that will be used in each butterfly of each state, thus, keeping the architecture fully recursive.

2.3.1 Operation Unit for a stage

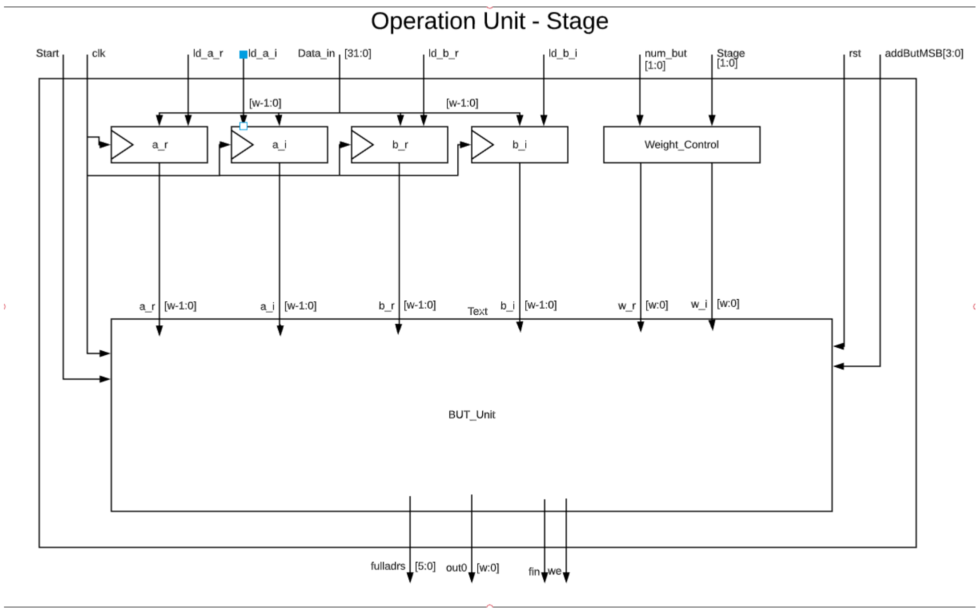

Figure 5: Stage Unit. Operation Unit

As described before, the Butterfly implementation will have the biggest entry size needed for the hole FFT. So from the 32 bits of memory, 23 and 24 bits will be taken for the first and second stages and resized to 25. When the operation unit receives the order from the FSM that all the registers have the correct data, it starts the Butterfly and 4 clock cycles after, has delivered the output into the signaled address in memory.

2.4 FFT

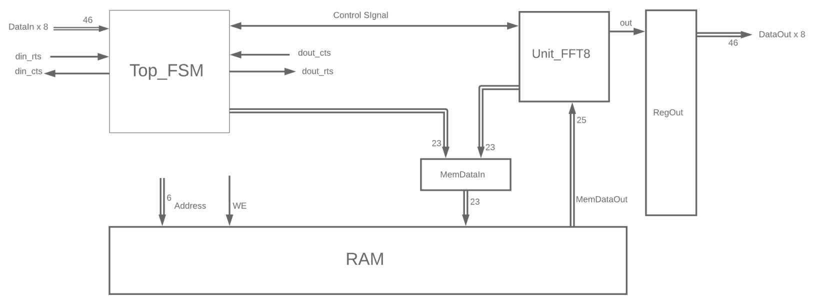

Figure 6: Highest circuit diagram

The whole calculation is controlled by a FSM called "Top-FSM". Firstly, this FSM waits for the $din - rts$ signal to be 1 and then it copies the 8 values that are sent in parallel into the first 16 places of the memory, taking into consideration that all the values are complex numbers. Once this is done, it puts the $din - cts$ signal to high and it starts the iteration of the stages. After the last stage is completed, the FSM keeps high the signal $dout - rts$ while the output signals remain unchanged and it waits for the answer of the block "frequency processing" on dout-cts.

Finally, the TopUnit is in charge of connecting this TopFSM, the memory, the out registers and the other FSMs together.

2.5 Memory

To make the iteration work, a memory is vital. Therefore, we are using a memory RAM of 64 32-bits addresses. This RAM has one bus of address, one write enable, one bus for reading and one bus for writing. What is more, we use 8 out registers of 46 bits each to send the information in parallel to the next block.

3 Conclusion

Our model has dramatically decreased the cost of production and the surface, since the multiplier is a big and very "expensive" component. In order to do this, we had to lose a lot of clock cycles. We believe this is not a big problem because of the low speed of an audio analyzer in comparison with the 100MHz frequency of the clock. Assuming that the audio is sampled at 48kHz, that means that for each 8 values that are sent to the fft, the fft has nearly 2000 cycles to analyze them and our implementation actually needs between 120 and 140 clock cycles to return the result.