Bruno E. Gracia Villalobos

EE 4513

64-bit Multiply-Add-Fused (MAF) Integrated Circuit Design

October 30, 2019

Path to folder in server: /home/UTSARR/ohf614/lab9hw/maf_unit

Introduction

This report describes the design of a 64-bit Multiply-Add-Fused (MAF) cell as an integrated circuit using Verilog and Cadence Encounter. The design is pipelined in two stages, effectively separating the multiply and add operations. For the abstraction level used, the design is developed at RTL level except for the behavioral definition of a 2:1 Multiplexer (MUX) using an inline conditional statement.

First, the MAF architecture is discussed. The RTL design of the MAF and its verification in Verilog and simulation respectively follow. Lastly, the integrated circuit is presented as engineered with Cadence Encounter.

Architecture

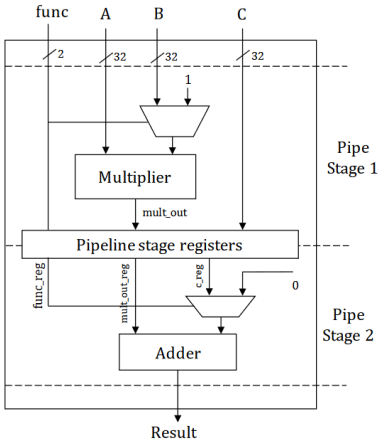


Figure 1: MAF Architecture

The MAF architecture is based on a simple design shown in Figure 1. The following logic elements are used:

- Two 32-bit 2:1 MUXes,
- One 64x64 Array Multiplier
- One 64-bit Carry-Lookahead-Adder (CLA)
- One 64-bit register for mult_out_reg
- One 2-bit register for func_reg
- One 32-bit register for c_reg
- One NOR-2 gate for the select line of MUX

From previous assignments, all the above components are repurposed except for the MUX and NOR-2 gate—which is a Verilog primitive and does not need to be modularized. The pipeline stage registers block consists of the func, mult_out, and c registers used to "pass on" the information from stage 1 onto stage 2. Given there is only one level of registers, the entire design takes 1 clock cycle to output the result.

Function	Pipe Stage 1	Pipe Stage 2	Desired Output
00	A × B	(A × B) + 0	A × B
01	A × 1	(A × 1) + C	A + C
10	A × B	(A × B) + C	(A × B) + C
11	Don't Care	Don't Care	Don't Care

Figure 2: MAF functions

The MAF has three functions, and they are presented in Figure 2. This presented a design issue, because the func control signal does not select the appropriate bus for both stage 1 and stage 2 MUXes, and the following

Karnaugh-Maps (KMAPs) are used to employ the synchronization between both stages.

F0\F1	0	1
0	1	0
1	0	X

MUX_PS1 = F0

F0\F1	0	1
0	1	0
1	0	X

MUX_PS2 = ~F0 & ~F1

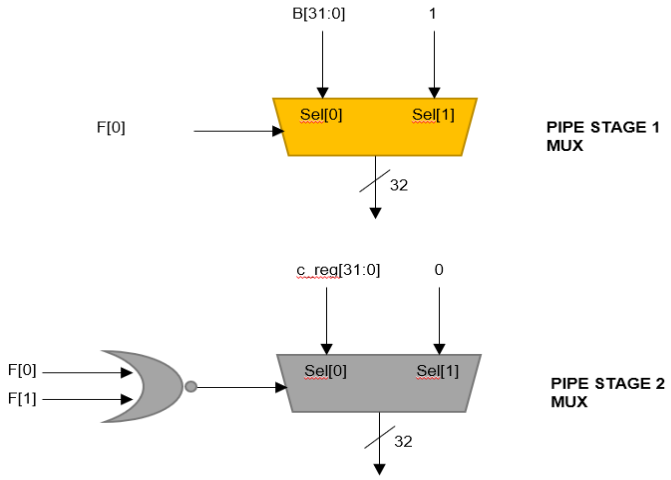


Figure 1: KMAPs to synchronize pipeline stage MUXes

The 2-bit bus F in the figure above is the func control signal, and a table was used to understand what signals need to pass through the MUXes for each function described in Figure 2. Given there are 3 functions, two different signals are needed to select the input lines for both pipeline stages, and these are mapped in each KMAP as a 1 or 0 depending on the signal needed to pass through. The pipeline stage 1 MUX only needs the Least-significant-bit (LSB) of the func signal, and the pipeline stage 2 MUX needs a NOR gate to select the appropriate signal.

It is also important to mention the registers are the only clocked logic elements in this design, and therefore supports the 1 clock cycle delay. Another caveat in the architecture above is the width of the MUX in stage 2: c_reg is 32-bits wide, but mult_out_reg is 64-bits wide. The next section describes how this problem is annihilated.

RTL Design

Given the architecture resembles a high-level view of a datapath for a controller, its implementation in Verilog is designed as such. However, the controller design is omitted in this assignment to focus on the actual MAF functionality. The MAF block is modeled as the top module in this project called “DATAP” and Figure 3 shows the elaborated design using Xilinx Vivado.

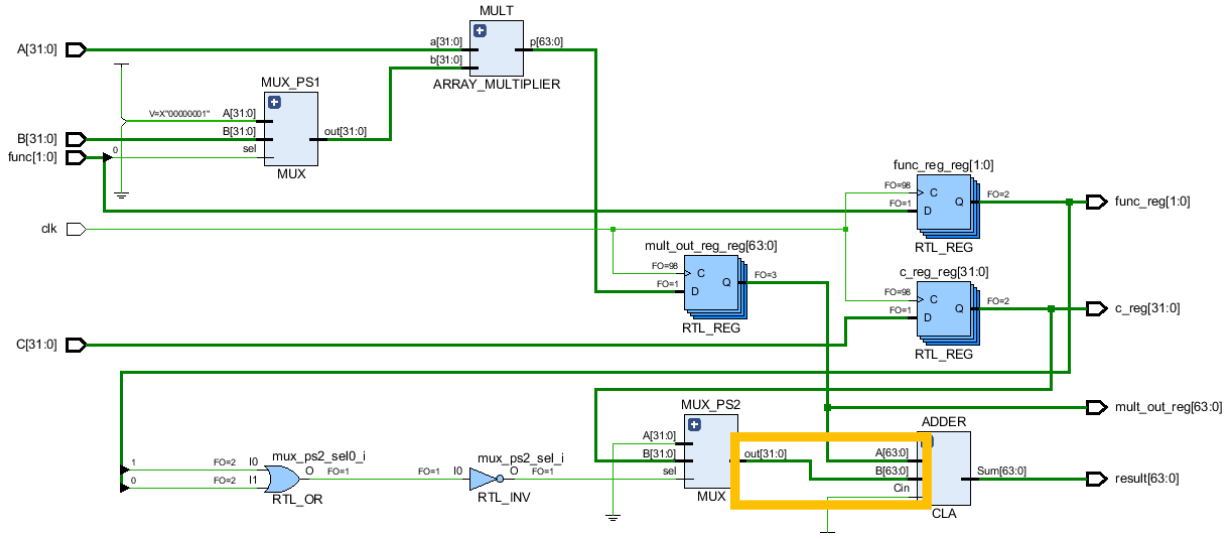


Figure 2: Elaborated design of the MAF in Xilinx Vivado

```
//have to sign extend the output of mux for adder
//since result of multiplier is bits*2
assign mux_ps2_ext = { {bits{mux_ps2[bits-1]}}, mux_ps2 };

MUX #(bits) MUX_PS2(
.A(1'b0),
.B(c_reg),
.sel(mux_ps2_sel),

.out(mux_ps2)
);


CLA #(bits*2) ADDER(
.A(mult_out_reg),
.B(mux_ps2_ext),
.Cin(1'b0),
```

Figure 3: Replication operator sign-extends 32 to 64

As mentioned earlier, the output of the MUX in stage 2 is 32-bits wide, but the CLA takes in a 64-bit input—shown in Figure 2 as the orange outline. The Verilog design implements a sign extension in-line to accommodate for this problem and is shown in Figure 4. The replication operator is used to sign extend the output of the 32-bit mux_ps2 wire bus to 64-bits by replicating the Most-significant-bit (MSB).

```
always@(posedge clk) begin
    func_reg <= func;
    mult_out_reg <= mult_out;
    c_reg <= C;
    //result <= result_w;
end
```

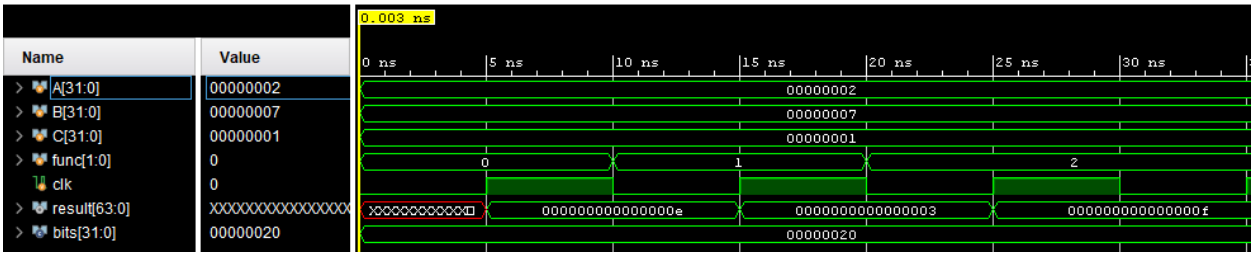
Figure 4: Pipeline stage registers

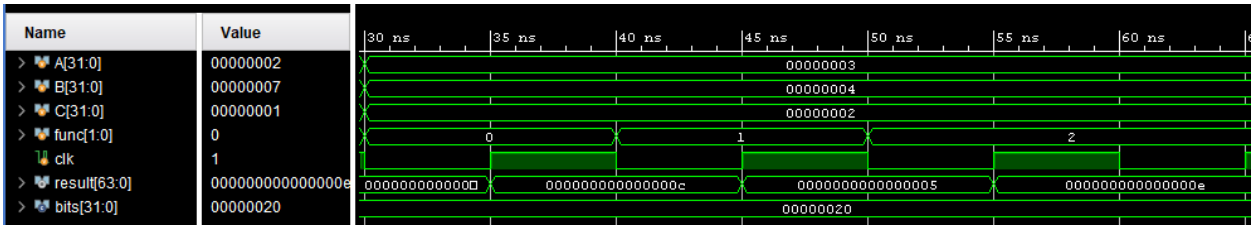
Apart from the ordinary instantiations of the necessary logic to build the MAF, the other object of interest are the pipeline registers in between stages. A procedural always block triggered by the rising edge of the clock is used to register in the values of c, the func control signal, and the multiplication result. The Figure to the left displays the procedural block.

Results and Simulation

Once the elaborated design developed as expected, the next step simulates the MAF with two distinct test cases. Consider the cases presented in the table and waveforms below.

INPUTS	FUNC	RESULT
A = 0x0000_0002 B = 0x0000_0007 C = 0x0000_0001	00	0x0000_0000_0000_000E
	01	0x0000_0000_0000_0003
	10	0x0000_0000_0000_000F
A = 0x0000_0003 B = 0x0000_0004 C = 0x0000_0002	00	0x0000_0000_0000_000C
	01	0x0000_0000_0000_0005
	10	0x0000_0000_0000_000E

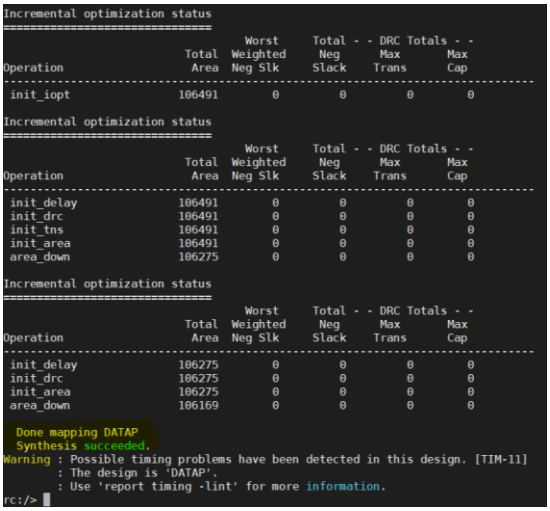


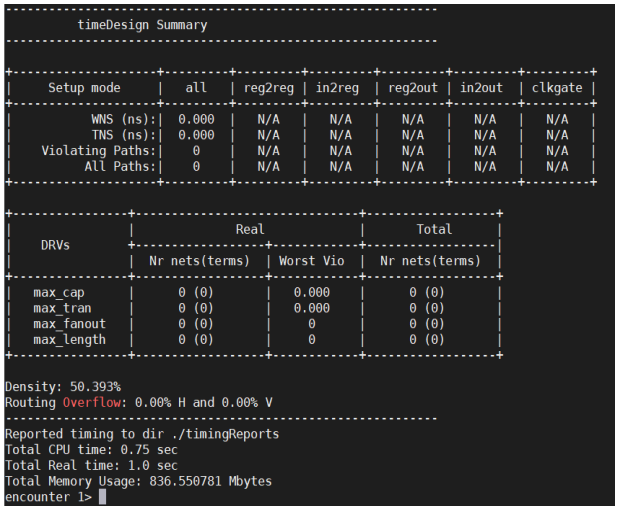


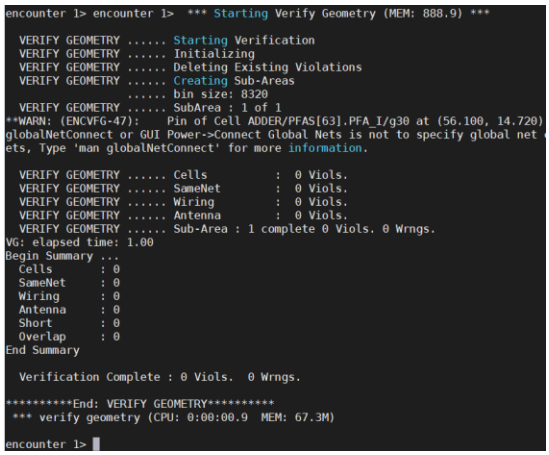
Although an FPGA is not the target for this design, the synthesized result from the XST is shown to the right for informational purposes. In total, 3159 LUTs are used.

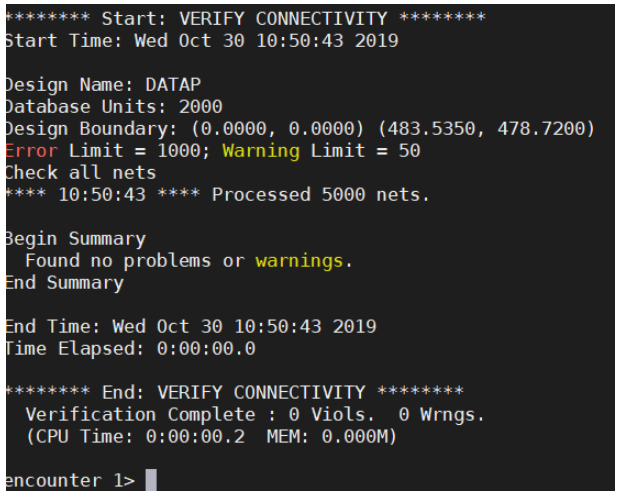
Integrated Circuit Design

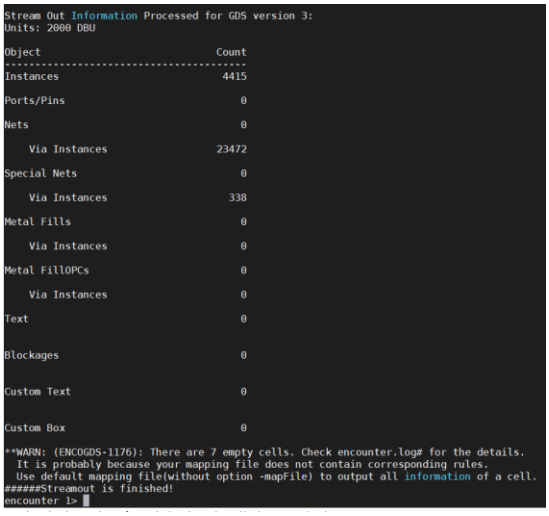
Now that the design is verified, the next step is realizing the design using Cadence RTL Compiler and Cadence Encounter. The synthesis, timing, verification of geometry and connectivity, and GDS generation reports are presented next, in order and from left to right.



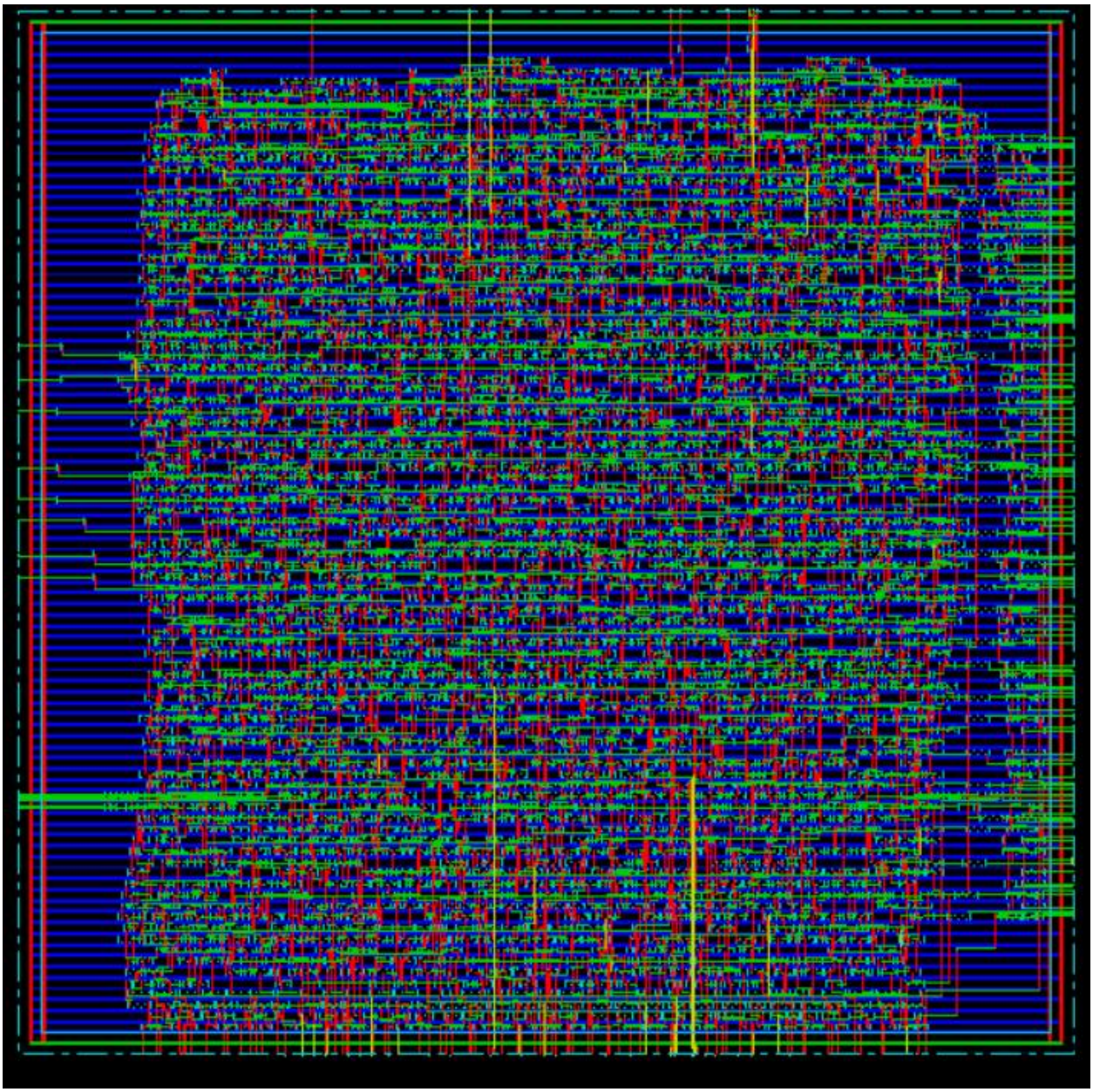




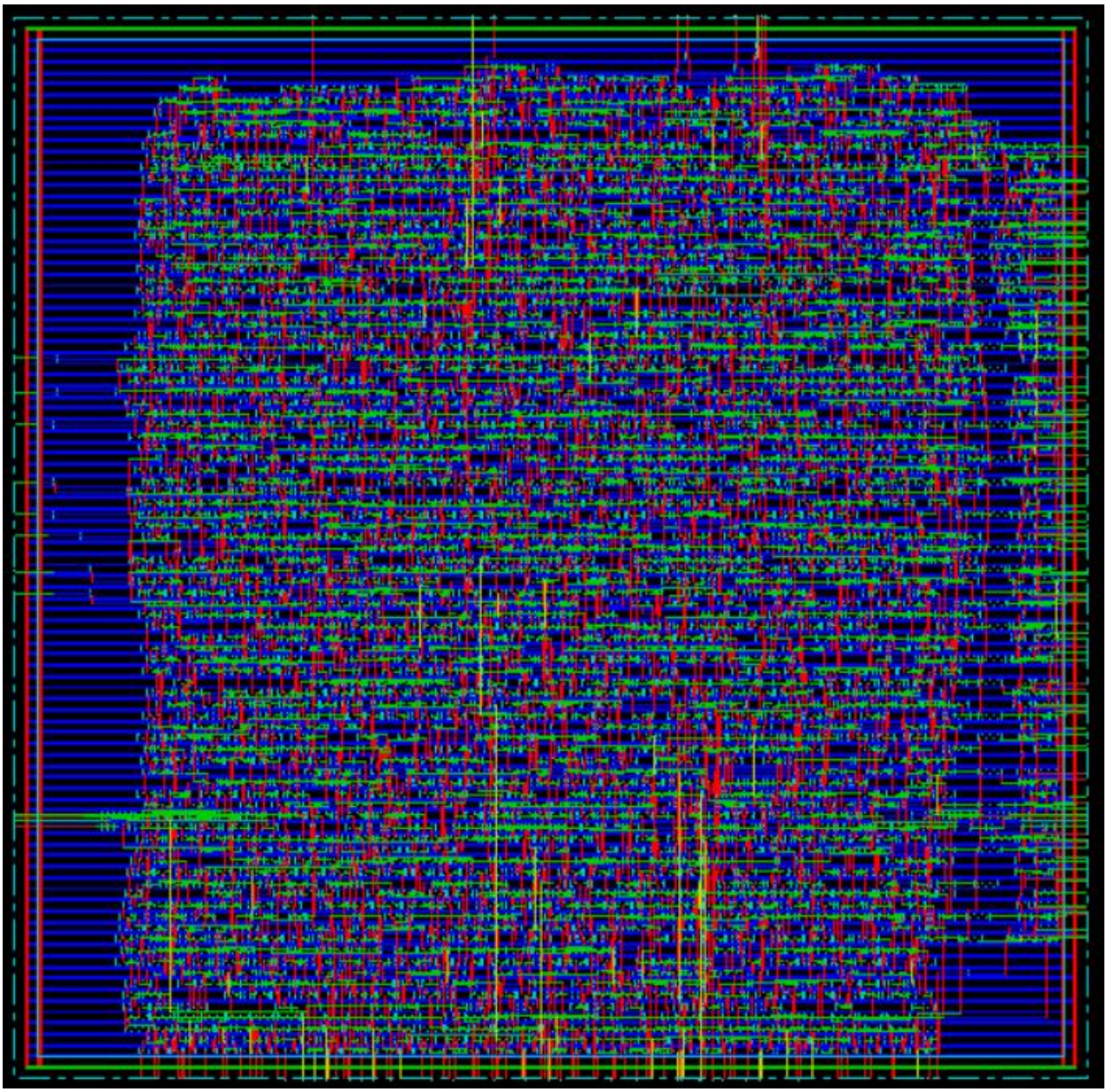




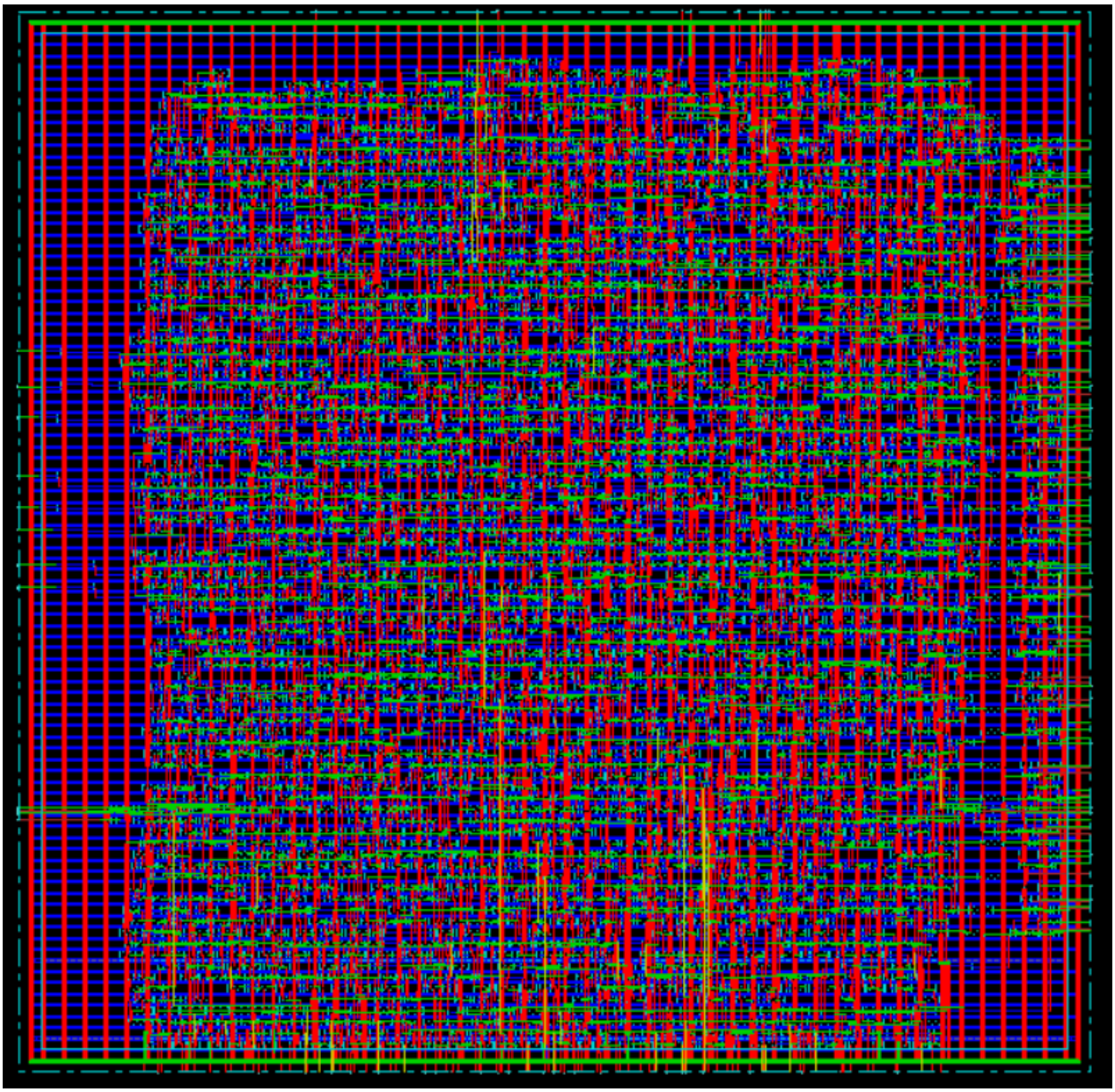
MAF Integrated Circuit Layout



After Nanoroute



50 Power Strips for Reduced V=IR Drops



Schematic Views

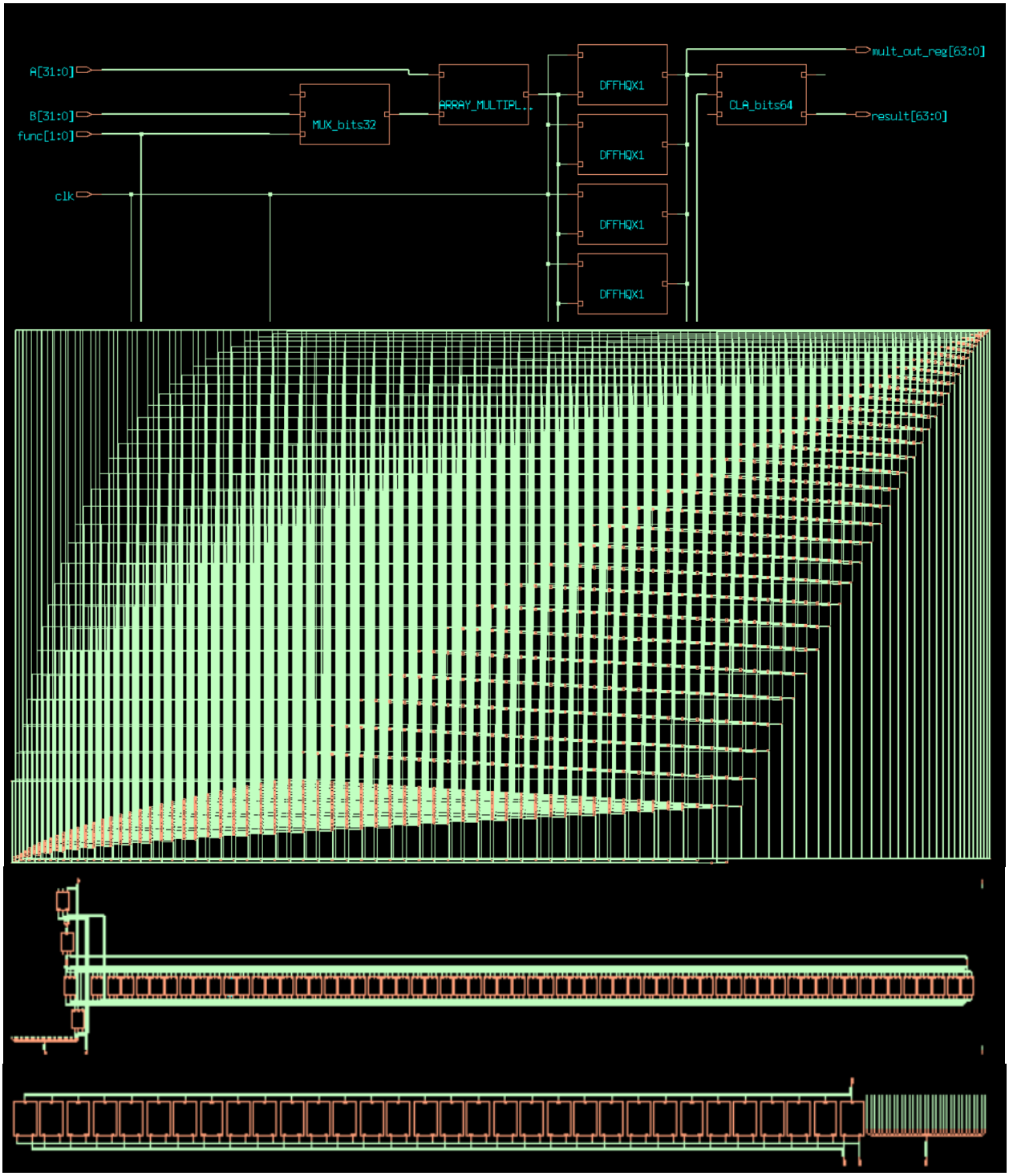


Figure 5: (FROM TOP TO BOTTOM) MAF schematic showing main components. 32x32 Array Multiplier. 64-bit CLA. 32-bit 2-1 MUX.

Conclusion

Compared to the previous assignments, the realization of the MAF architecture was not as difficult because most of the structures used were previously developed and simulated. The extra effort put in the previous modules to program the HDL to generate variable data sized logic elements was well worth it. It was also refreshing to go back to the drawing board when I had to figure out how to engineer the inputs to both MUXes so that they are synchronized with the given function control signal across pipeline stages. For future implementations, I would like to implement a control unit with a given set of operations to carry out to better understand the benefits of having a pipelined datapath.