Design, Simulation and Hardware implementation of a Multiply and Accumulate (MAC) Core

EE519P - CMOS Digital IC Design Practicum

Course Instructor: Dr. Hitesh Shrimali

Shrinivas Khatavkar(B18144)
Electrical Engineering
IIT Mandi
Mandi(H.P.), India
Group 8
b18144@students.iitmandi.ac.in

Abstract

Multiply and Accumulate is an essential operation in Digital Signal Processing and in video cards. The MAC unit is the main factor in the speed of the overall system. Hence a fast, power efficient and area efficient MAC unit is absolutely essential for these operations. In this report we design a MAC unit using a 16-bit Ripple Carry Adder and an 8-bit Booth Multiplier. We then describe the MAC unit on Verilog HDL and perform Synthesis. The Synthesized design is then implemented onto a ZYBO FPGA Board after generating the bitstream.

I. Introduction

The Multiply and Accumulate operation works on two inputs. The two inputs are multiplied together and their product added to the previous value of the output. This operation can be represented as

$$F \leftarrow (A * B) + F$$

The MAC unit was first conceptualized by the Irish scientist Percy Ludgate. Since then it has become a crucial part in DSP applications, to such a degree that it always lies in the critical path of the circuit. MAC unit is also essential for video cards, as it is useful in matrix multiplication used in video processing. MAC is also finding its way into general-purpose processors.

When it comes to floating point arithmetic, FMA (Fused Multiply Add) calculates the MAC expression and rounds it to the specified number of significant bits. It can speed up the calculation of dot product, matrix multiplication, solving polynomials, and convolutions.

II. Working of MAC core

The MAC unit receives two input. These inputs are first sent to a radix-2 multiplier. In the multiplication operation the result contains double the bits of the inputs. This product is then sent as one of the inputs to the adder unit. The other input to the adder is the previously accumulated value stored in the register.

This entire operation is represented in the following diagram.

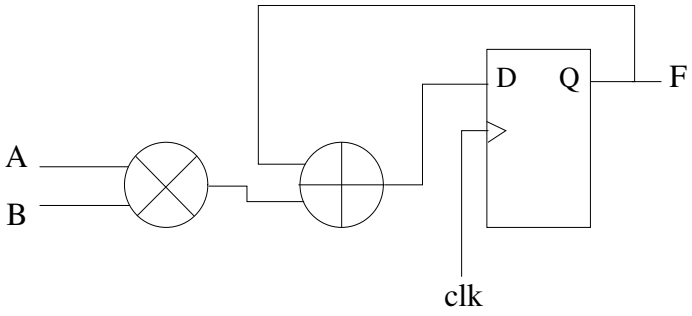

Figure 1. Block diagram representing MAC operation

III. Design Specifications and Components

The MAC unit described in this report is designed for 8-bit signed inputs. In 2's complement notation, an n-bit number can represent values ranging from -2^{n-1} to $(2^{n-1} - 1)$. Hence our inputs can range from the values -128 to 127. The output will be a 16-bit signed integer which can range from -32,768 to +32,767.

As evident from the flowchart, we will require a multiplier and an adder circuit to perform the operation. For multiplication, we have used an 8-bit booth multiplier. The booth multiplier will consume less area than a combinational multiplier at the cost of speed. For addition, a 16-bit Ripple Carry Adder is used.

Although this circuit is designed to work with integers, it can easily be scaled to work with fixed point numbers by replacing the adder and multiplier unit.

A. Ripple Carry Adder

The full adder is the building block for a ripple carry adder. It is a simple combinational circuit with three inputs which are two input bits and one input carry bit and two outputs, which are sum and output carry. It can be made using basic logic gates. It can be described by the equations:

$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = AB + C_{in}(A \oplus B)$$

Ripple Carry Adder is a combinational adder which is basically an array of full adders. The output carry of one full adder is fed to the next. For the sum to be calculated, each full adder must wait for the carry to be calculated by the previous adders.

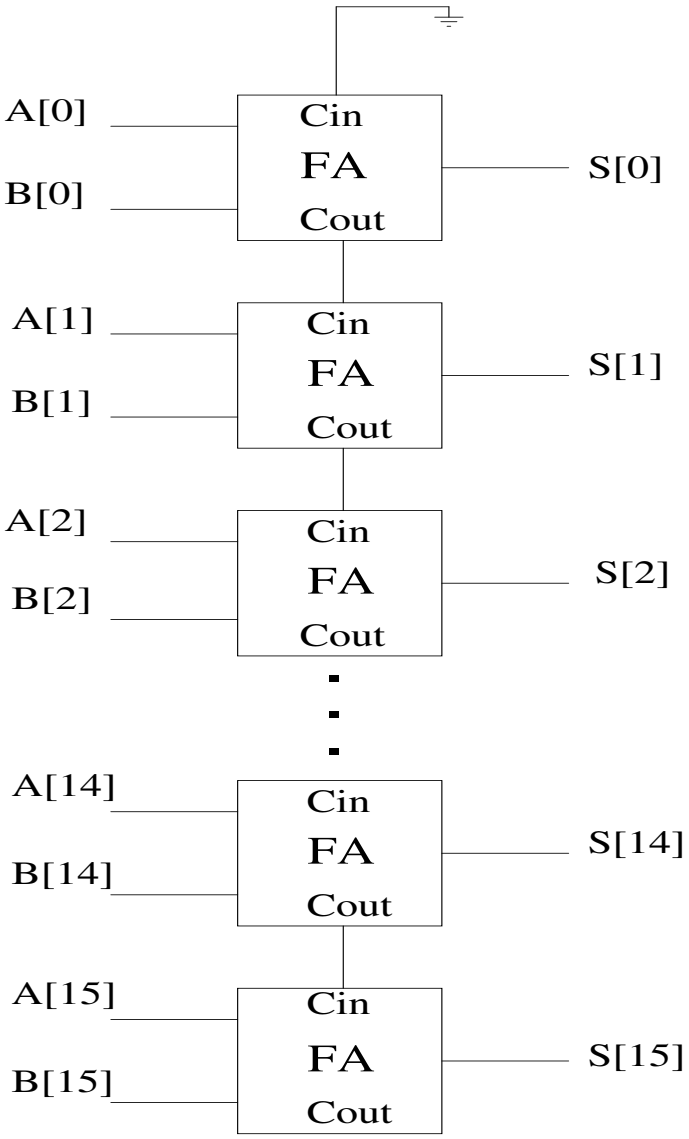

Figure 2. 16-bit ripple carry adder circuit

As seen in the figure, the last output carry is discarded. This is because the last carry is always discarded in 2's complement addition. In our code the full adder and ripple carry adders are described on a gate level.

B. Booth Multiplier

The booth multiplier can be used for the multiplication of signed numbers represented in 2's complement notation. The least significant bit of the multiplier and register q' are used to decide the operation to be performed. Depending upon their values, the multiplicand can be subtracted or added to register A. At the end of each cycle A, Q and q' are right shifted together.

An n-bit booth multiplier takes n clock cycles to return the final product. So for our 8 bit inputs, 8 clock cycles will be required to finish the operation.

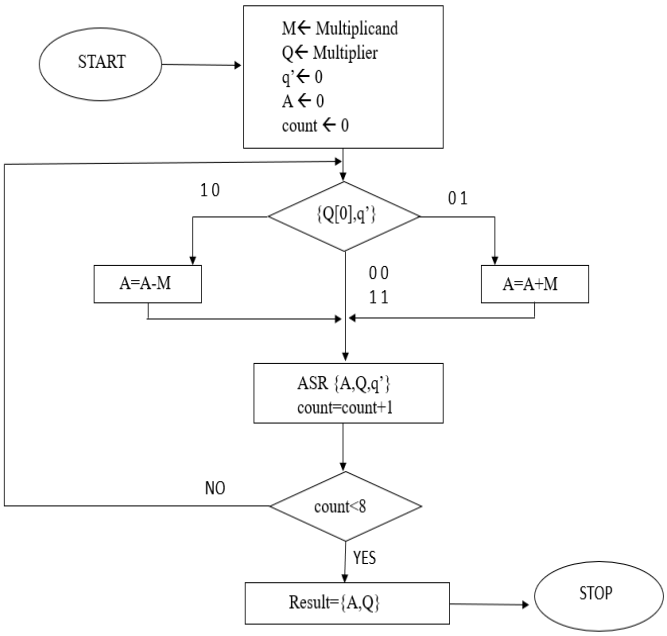

Figure 3. Flowchart of booth multiplication

The hardware implementation of booth multiplier is flexible and can be achieved in many different ways. For the purposes of our design, we have described the algorithm on RTL level. The code was synthesizable and produced hardware implementation upon running synthesis.

There were five ports in our booth multiplier. Three ports were for the two inputs and output. One port was for 'start' signal, which cleared the registers and loaded the inputs. The final output port was for the 'busy' signal remained high while the multiplier was working, and low when output was reached.

IV. Multiply and Accumulate Unit

A. Pins of MAC unit

The MAC unit designed in this report has 8 pins. A representational diagram is shown below.

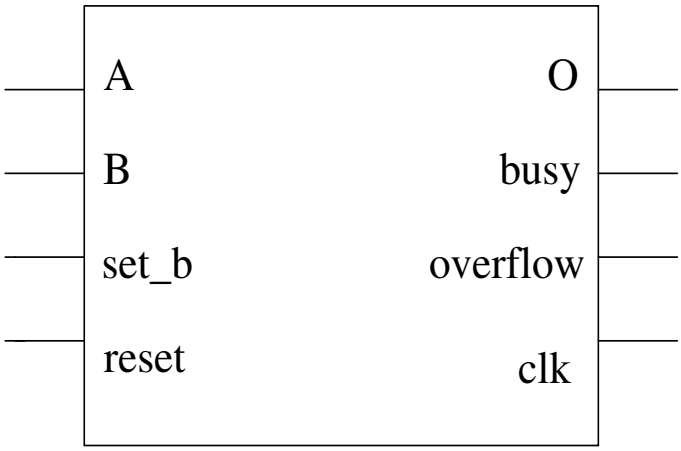

Figure 4. Pins of our MAC circuit (only representative view)

A brief description of all the pins is given below:

A: This is the first 8-bit signed input to the MAC unit

B: This is the second 8-bit signed input to the MAC unit

set_b: This input should be set high when giving new inputs to the MAC unit

reset: This input resets the accumulated value of the register

O: 16-bit signed output

busy: This output signal is high when booth multiplier is working and no new inputs can be provided

overflow: This output signal is high if an overflow occurs in the accumulation register of MAC unit

clk: Clock provided to this pin

B. Hardware description

The MAC unit was described on RTL level using the ripple carry adder and booth multiplier. Code for the MAC unit can be found in appendix B.

Just as the flowchart suggests, inputs A and B are fed to the 8-bit booth multiplier. The output pin 'busy' and input pin 'set_b' are also connected to the respective ports on the booth multiplier. The product from the booth multiplier is fed to the ripple carry adder along with the accumulation register. The output wire 'temp' is also connected to the accumulation register.

The 'always' block determines what kind of flip-flop a registered is synthesized from. For asynchronous reset operation, the always block was set to any changes in combinational input.

At every positive edge of the clock, if 'set_b' is high, inputs A and B are fed to the booth multiplier. This will set the 'busy' signal to high for the next 8 clock cycles. At the negative edge of the 9^{th} clock cycle, the wire 'temp' from the ripple carry adder is used to update the accumulation register. The output O is assigned to the accumulation register. Also at this point, the 'overflow' output will become high if an overflow has occurred in the accumulation register. The logic for overflow detection is discussed in the next section

C. Detection of overflow

In 2's complement notation, the Most Significant Bit (MSB) essentially stores the sign of the number. If the MSB is '1' it denotes negative number while '0' denotes positive number. An overflow occurs when a register is forced to a value higher than it can represent in the 2's complement notation with its limited size. For example, if a 4-bit register is storing it's highest possible positive number (0111), and '1' is added to it, the carried '1' from the addition will overflow into the MSB, turning the value stored in the register negative (1000). Similarly if '1' is subtracted from the lowest possible negative number (highest negative in magnitude) which is 1000, the value will turn positive (0111).

Whenever new inputs are loaded into the MAC unit, their MSB are sampled. The sign of the two inputs is used to determine the sign of their product. The MSB of the current accumulated value is also sampled. After the accumulation register is updated in the 9^{th} clock cycle, if the new sign of the accumulated value becomes negative from positive when the product was positive, then overflow signal is set to '1'. Similarly if the accumulated value becomes positive from negative when the product was negative, overflow has occurred.

D. Expectations from the control unit

When the MAC unit designed by us is used in a processor, there are some constraints the control unit should follow.

Firstly, after a set of inputs has been given, the next set of inputs can only be given on the positive edge of the 10^{th} clock cycle after receiving the output on the negative edge of the 9^{th} clock cycle. Also the 'set_b' signal should be set to '1' while giving the inputs.

Secondly in the case of an overflow, the control unit should send the reset signal and discard the last received output. The

MAC unit does not correct itself in the event of an overflow and continues to work with the wrong accumulated value.

E. Problem with asynchronous reset

As mentioned previously, the MAC unit was originally meant to have an asynchronous reset. However, in the synthesis stage it was realized that asynchronous reset creates an unnecessary latch to drive the output (along with the intended accumulation register. This causes an error as the output can not be driven with two registers holding different values. Therefore the asynchronous reset was replaced with a synchronous reset for the design to become implementable in hardware.

V. Simulation

A. Normal functioning

These simulations were performed with a clock of frequency 50MHz.

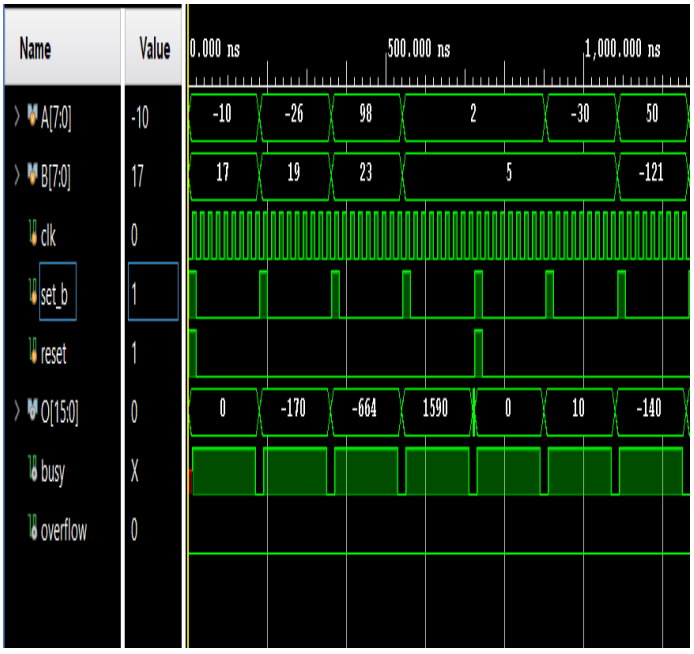

Figure 5. Simulation of MAC unit with reset signal given in between

As expected, the value of the output is only updated at the negative edge of the 9^{th} clock cycle. The inputs are only sampled when the set_b signal is high. Since the accumulated value is inside bounds, the overflow signal remains low. It should also be noted that a reset signal was given in the middle of the operation, which cleared the accumulated value of the MAC unit.

All of the MAC operations performed have been cross-checked to be accurate. Hence we can see that our MAC unit works well under normal conditions. The next figure shows the output for another set of inputs, without the reset signal this time.

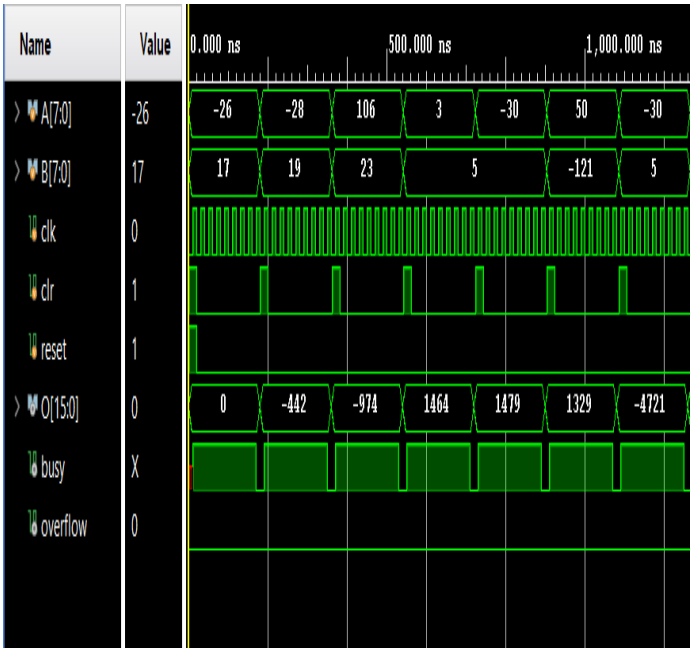

Figure 6. Simulation of MAC unit without reset signal given in between

B. Positive overflow

The following figure shows a scenario where the MAC unit has been given a series of large positive inputs causing an overflow. Notice the overflow signal has become high to alert the control unit.

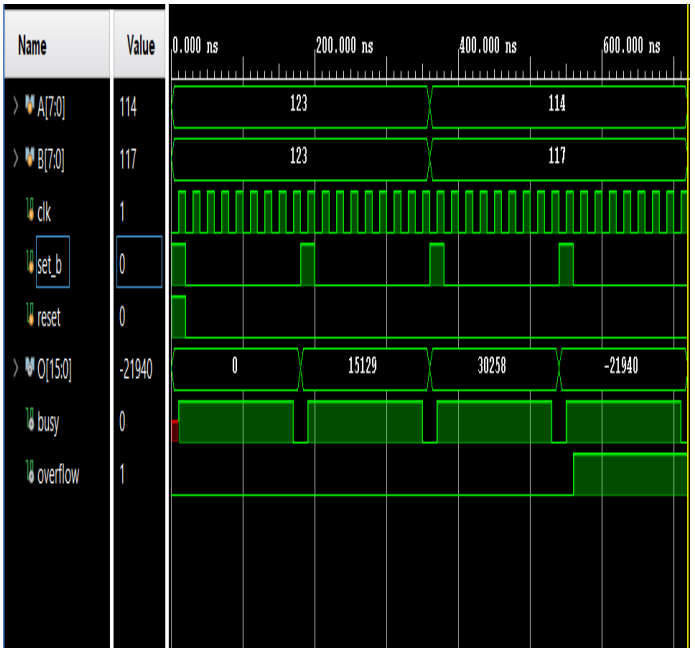

Figure 7. Simulation of MAC unit with positive overflow

Hence we conclude that our circuit is capable of detecting positive overflows.

C. Negative overflow

The following figure shows a scenario where the MAC unit has been given a series of large inputs with opposite signs causing a negative overflow. Notice the overflow signal has become high to alert the control unit.

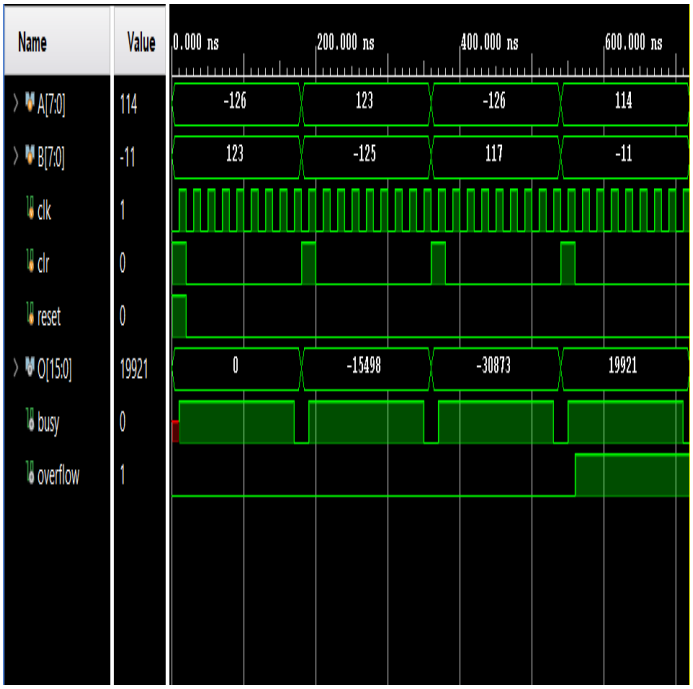

Figure 8. Simulation of MAC unit with negative overflow

Hence we conclude that our circuit is capable of detecting negative overflows. Thus any error caused by overflow will not go undetected and compromise the integrity of the final result. However it is still expected from the control unit to provide the reset signal in such a situation since the MAC does not correct itself.

VI. Analysis of Elaborated Designs

The synthesized design gives us a more accurate view of the implemented design than the elaborated design. However, elaborated design can give us a better and more intuitive insight into the implementation of the code. In this section I've only discussed the elaborated designs of booth multiplier and the MAC unit. The elaborated design for ripple carry adder looks exactly as we expect since it was described on a lower level.

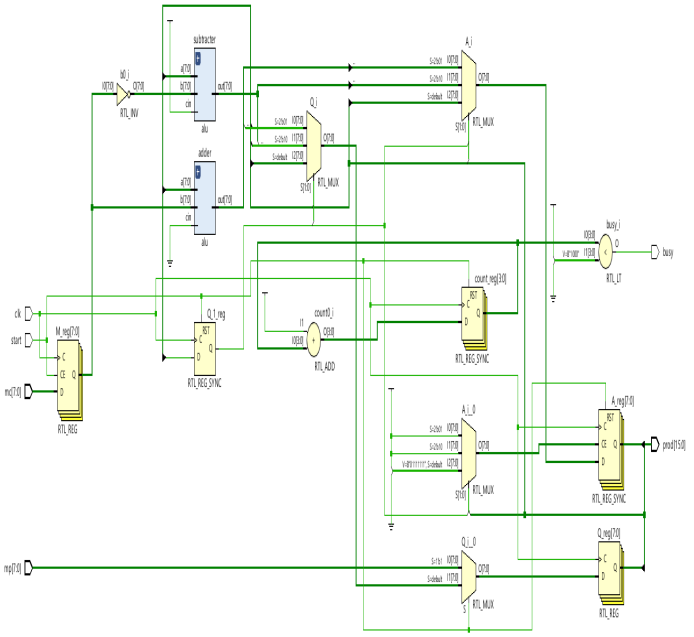

Figure 9. Elaborated design of booth multiplier

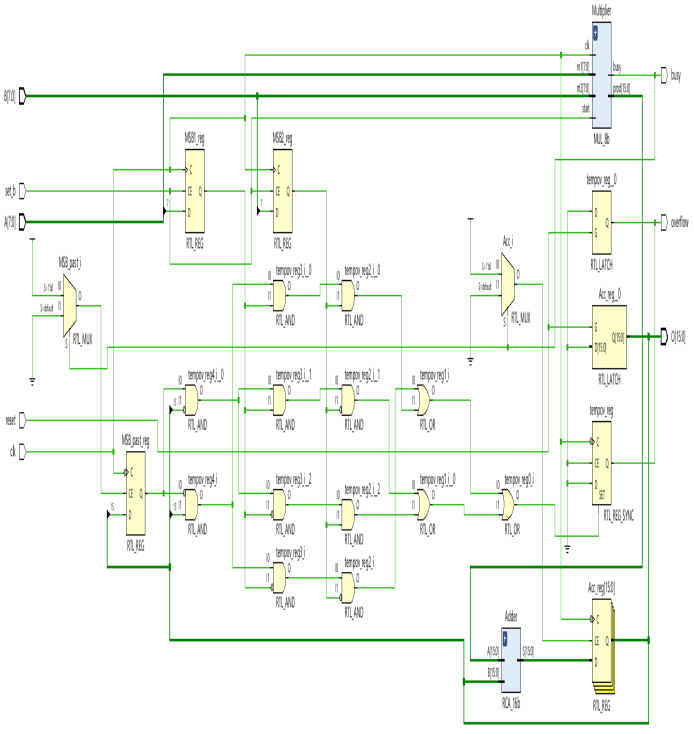

Figure 10. Elaborated design of MAC unit

In figure 9 we can see the elaborated design of the booth multiplier. We can clearly see the registers A, Q and M as described in the flowchart. We can also see A and M being fed into the adder and subtracter unit. We can see various multiplexers being used for case operations and barrel shifting. We can also see a counter circuit, which counts the positive edge of the clocks.

In figure 10 we can see the circuit for MAC unit. As the flowchart suggested, the inputs A and B are fed into the multiplier and then the product is accumulated using the adder. We can also see a number of logic gates used to determine the overflow using the MSB registers. It should be noticed that this elaborated design is for the circuit with asynchronous reset, as is apparent from the additional latch formed at the output. The circuit used for synthesis will have synchronous reset and will be same as this circuit except for that particular latch.

VII. Synthesis

The synthesis of the module was successful with the following schematic as output.

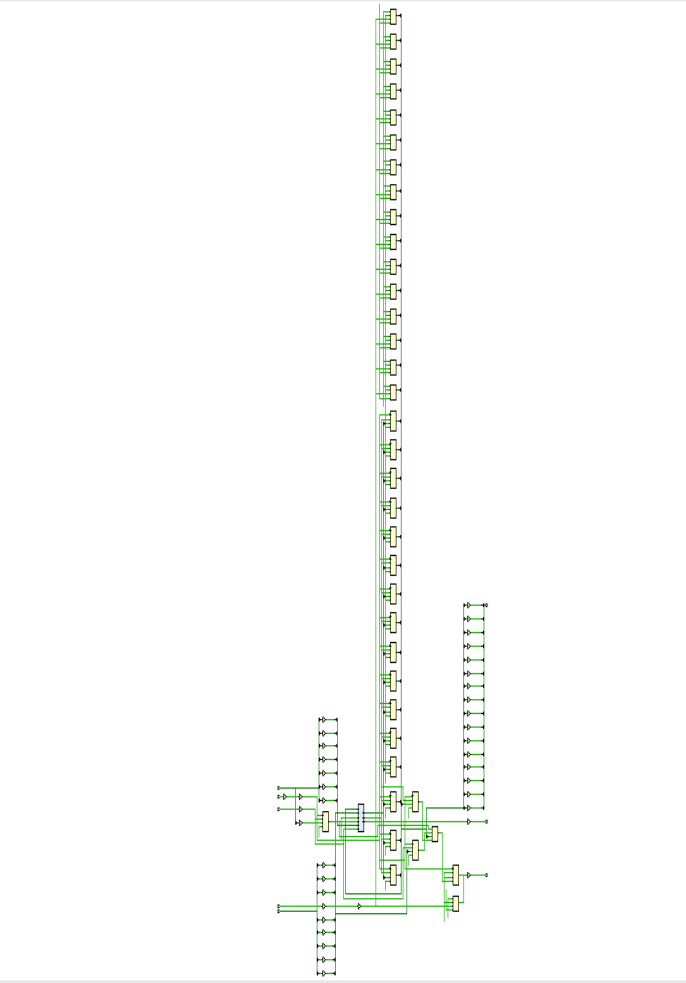

Figure 11. Synthesized Schematic

Due to the large size of the schematic it isn't properly discernible on this report. Despite this, we can clearly make out several features like the input and output pins and the modules used.

VIII. Implementation and Bitstream generation

After the synthesis, the implementation was also successful. Surprisingly, some of the modules were missing from the final implemented schematic, but all of the pins were present.

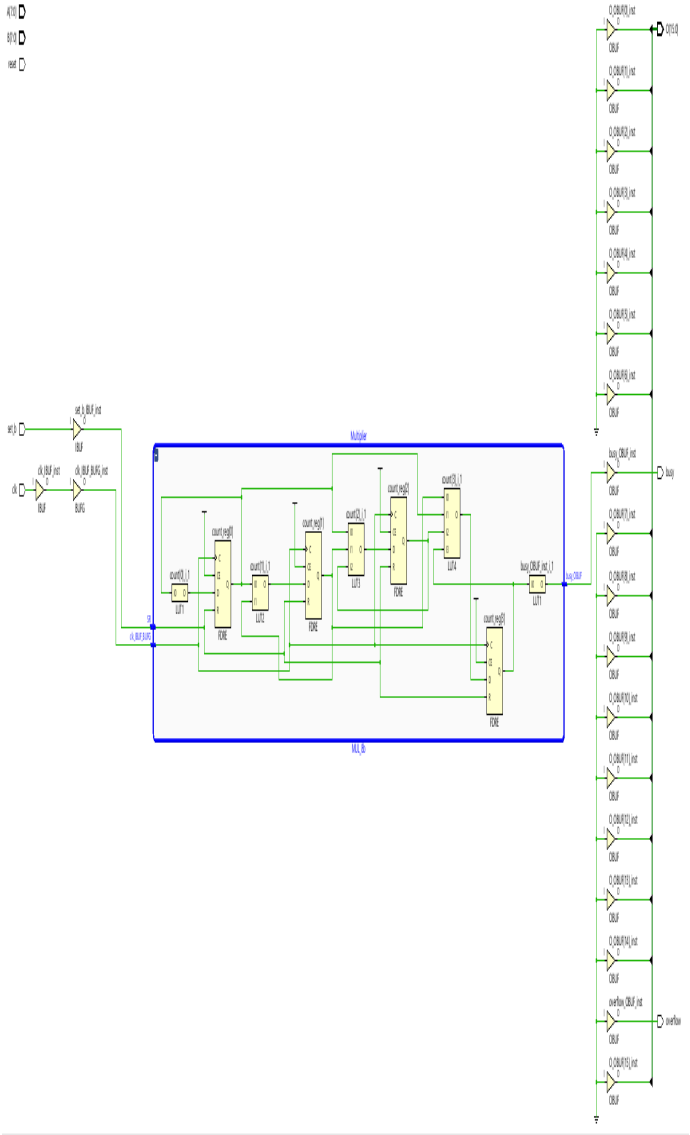

Figure 12. Implemented Schematic

After this, the bitstream generation was also successful. Therefore our code is ready to be implemented on a ZYBO FPGA board.

A. *Power Consumption*

The power consumption was calculated after the implementation of the design. The ambient temperature was assumed to be 30° C and the load capacitance 30pF.

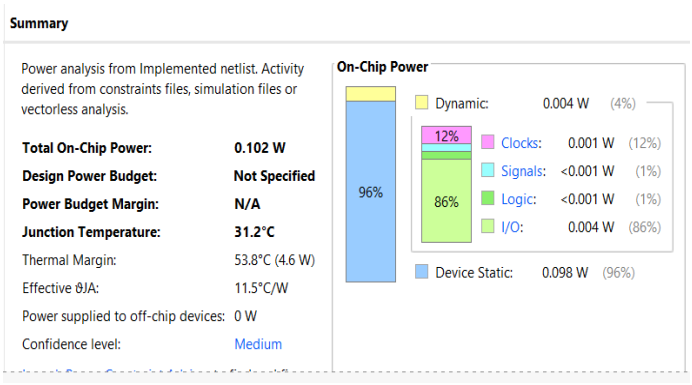

Figure 13. Power consumption

As we can see, most of the power consumption is static in nature.

B. *Utilization Report*

The final design used 3 Slice LUTs and 4 Slice Registers. All of the registers were synthesized as Flip-Flops.

IX. Conclusion

We designed and described a MAC unit in Verilog HDL. We wrote a testbench for it and performed simulations for all possible scenarios. The code was then synthesized and converted into a gate-level netlist. The design was then implemented and we analyzed the power consumption and FPGA Utilization. Finally, we generated bitstream to implement the code on a ZYBO FPGA board.

References

[1] Samir Palnitkar, *Verilog HDL - A guide to Digital Design and Synthesis*. Prentice Hall(Pearson) Publications, Hoboken, New Jersey(US), Feb 2003.

[2] Booth, Andrew Donald *"A signed binary multiplication technique"*. Oxford University Press, Aug 1950

[3] Maroju Sai Kumar, D. Ashok Kumar *"Design and performance analysis of Multiply-Accumulate (MAC) unit"*. 2014 International Conference on Circuits

Appendix A: MAC Unit Code

```
module MAC(
input [7:0]A,  //input 1
input [7:0]B,  //input 2
input clk,     //clock
input set_b,
input reset,
output [15:0]O,  //output
output busy,
output overflow
);

reg MSB1, MSB2, MSB_past;   // for overflow
reg [15:0] Acc; //register for accumulation
wire [15:0] temp; //temporary RCA output
wire [15:0] tempprod;
reg tempov; //for calc of overflow

MUL_8b Multiplier(tempprod,busy,A,B,clk,set_b);
RCA_16b Adder(tempprod,Acc,temp);

//always @(*)   //uncomment for asynch reset
 //   begin
 //   if(reset==1)
 //       begin
 //       Acc=16'b0000000000000000;
 //       tempov=1'b0;
 //       end
 //   end

always @(posedge clk)
    begin
    if(set_b==1)
        begin
        MSB1<=A[7];
        MSB2<=B[7];
        end
    if(reset==1)    //comment this out for synch r
        begin
        Acc=16'b0000000000000000;
        tempov=1'b0;
        end
    end
always @(negedge clk)
    begin
    if(busy==0)
        begin
        MSB_past<=Acc[15];
        Acc<=temp;
        end
    if ((~MSB_past&&Acc[15]&&~MSB1&&~MSB2
    ||~MSB_past&&Acc[15]&&MSB1&&MSB2)
    || (MSB_past&&~Acc[15]&&MSB1&&~MSB2
    ||MSB_past&&~Acc[15]&&~MSB1&&MSB2)
        begin
```

```
        tempov<=1'b1;  //overflow occurs
        end
end

assign O=Acc;  //MAC output

assign overflow=tempov; //overflow output

endmodule
```

APPENDIX B: BOOTH MULTIPLIER CODE

```
module MUL_8b(
output [15:0] prod,
output busy,
input [7:0] m1,
input [7:0] m2,
input clk,
input start);

reg [7:0] A, Q, M;
reg Q_1;
reg [3:0] count;
wire [7:0] sum, difference;
always @(posedge clk)
    begin
    if (start)
        begin
        A <= 8'b0;
        M <= m1;
        Q <= m2;
        Q_1 <= 1'b0;
        count <= 4'b0;
        end
    else
        begin
        case ({Q[0], Q_1})
            2'b0_1 : {A, Q, Q_1}
            <= {sum[7], sum, Q};
            2'b1_0 : {A, Q, Q_1}
            <=
            {difference[7], difference, Q};
            default: {A, Q, Q_1}
            <= {A[7], A, Q};
        endcase
        count <= count + 1'b1;
        end
    end

alu adder (sum, A, M, 1'b0);
alu subtracter (difference, A, ~M, 1'b1);
assign prod = {A, Q};
assign busy = (count < 8);

endmodule
```

```
module alu(out, a, b, cin);
output [7:0] out;
input [7:0] a;
input [7:0] b;
input cin;
assign out = a + b + cin;
endmodule
```

APPENDIX C: RIPPLE CARRY ADDER CODE

```
 module RCA_16b(
input [15:0] A,
input [15:0] B,
output [15:0] S
    );
wire [15:0] C;

FA F0(A[0], B[0],1'b0,C[0],S[0]);
FA F1(A[1], B[1],C[0],C[1],S[1]);
FA F2(A[2], B[2],C[1],C[2],S[2]);
FA F3(A[3], B[3],C[2],C[3],S[3]);
FA F4(A[4], B[4],C[3],C[4],S[4]);
FA F5(A[5], B[5],C[4],C[5],S[5]);
FA F6(A[6], B[6],C[5],C[6],S[6]);
FA F7(A[7], B[7],C[6],C[7],S[7]);
FA F8(A[8], B[8],C[7],C[8],S[8]);
FA F9(A[9], B[9],C[8],C[9],S[9]);
FA F10(A[10], B[10],C[9],C[10],S[10]);
FA F11(A[11], B[11],C[10],C[11],S[11]);
FA F12(A[12], B[12],C[11],C[12],S[12]);
FA F13(A[13], B[13],C[12],C[13],S[13]);
FA F14(A[14], B[14],C[13],C[14],S[14]);
FA F15(A[15], B[15],C[14],C[15],S[15]);



endmodule

module FA(input A,
input B,
input Cin,
output Cout,
output S
);

assign S  = (A ^ B) ^ Cin;

assign Cout = (A & B) | (B & Cin) | (Cin & A);

endmodule
```