

Chapter 3

4-by-4 Bit Multiplier Accumulator

The standard design method for multiply accumulator (MAC) is defined. A multiply accumulator (MAC) performs complex repetitive tasks of convolution neural network . It has various applications in the areas of DSP such as, digital filters, images and graphical applications. Up speed, low power consumption and customization of design barriers facing design engineer.

This chapter describes a design methodology of an 8-by-8 unsigned multiplier accumulator. At the highest level of output, the project starts with the concept of functional blocks to accommodate multiplication, addition and accumulation of data as a general requirement. Based on the design details, a Clear Structural Design is constructed; at this level, the block diagram consists of major blocks, connections between blocks and signal flow direction. Principles of design strategies such as hierarchy, regularity, modularity and locality coupled with constraints of area, speed and power dissipation are considered. This consideration also determines the use of data for each active block. Each performance block will be built, tested and performed. The design will be displayed and encoded in Verilog and is designed for optimal performance and time. Simulation effects, build drawings and Verilog codes will be included.

3.1 GENERAL OBJECTIVE

The objective is to design a Multiplier-Accumulator(MAC) circuit. 8 pairs of operands have to be multiplied and added.

$$C = \sum_{i=0}^7 A_i B_i \quad (3.1)$$

MAC acquires A_i and B_i operators, each with 8 unsigned binary numbers. The output results C , is the summation of these results. Each pair of operand AB is loaded parallel in the RA RB registers at start of each recurrence cycle. Each cycle of recurrence begins with a BEGIN signal . result of each multiplication is added to the previously accumulated data in the output register.

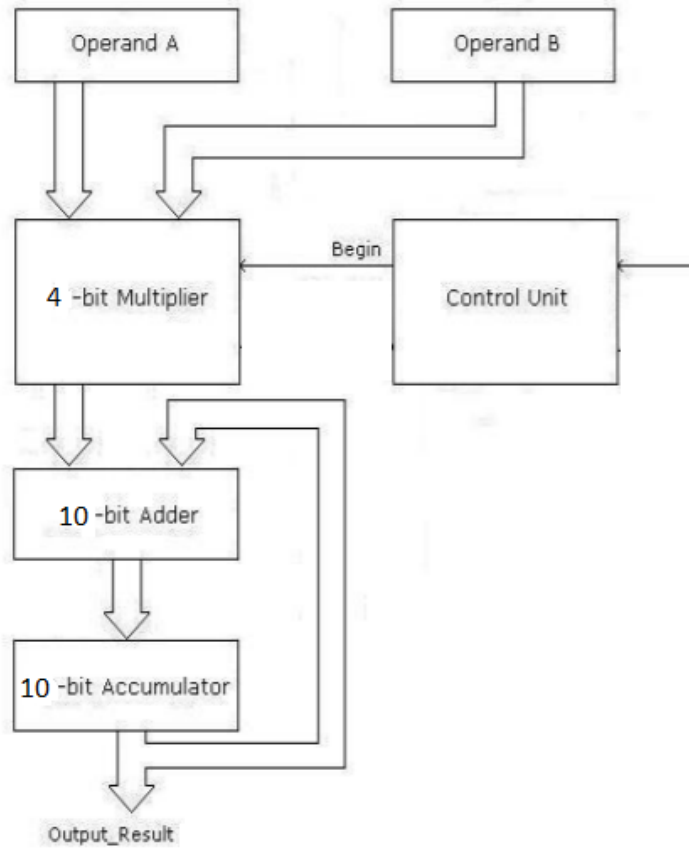


Figure 3.1: MAC Block Diagram

3.1.1 Design Description

Design a Multiply and Accumulation(MAC) with two 4- bits inputs.

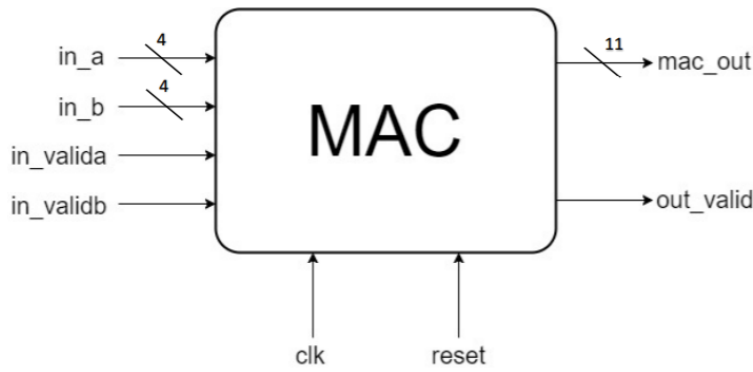


Figure 3.2: Block Diagram

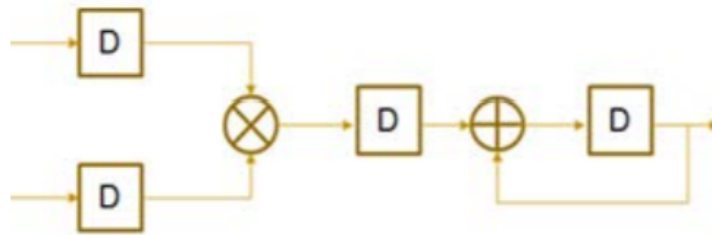


Figure 3.3: MAC Implementation

3.1.2 Specifications

Average filters are used for smoothing image data. Filtering creates a new pixel with coordinates as the center of neighbourhood. If filtering is done linearly then it is linear spatial filters otherwise it is nonlinear. Smooth linear filters are also known as averaging filters because they calculate the average of the pixels contained in the neighbourhood of the filter mask. By averaging we get the resultant image which is the reduced sharp transitions in intensities. The averaging filter thus reduces noise as well as the irrelevant details in an image.

- Top module name : mac (File name: mac.v)
- Input pins: in-a[3:0], in-b[3:0], in-valida, in-validb, reset, clk
- Output pins: mac-out[10:0], out-valid,

- All inputs and outputs are synchronized at clock positive edge.
- It is synchronous-reset architecture, both outs become 0 when the reset equal to 1.
- Note that in-a, in-b are 4-bits signed integer number. Please design the circuit without overflow.
- The two operands are valid when the control signals “in-valida” and “in-validb” are 1, respectively.
- After 8 MACs, output the result and assert the “out-valid” signal
- Note that two control signals are in a “one-to-one” manner, so you do not have to buffer the operands.
- In order to decrease the critical path, insert a retiming DFF(introduce one more pipeline stage) at the output of the multiplier.

3.1.3 Simulation

this the simulation and timing result of the MAC controller. The active signal puts the controller in IDLE state and temporary counter to 8. Temporary counter keeps track the number of operand pairs loaded into MAC.; the transition happens on the rising clock edge. During MAC state, the controller activates a signal to the input registers (register A and register B) to accept a pair of operands. When the multiplier unit completes its tasks, it generates a multiplication complete signal to the controller, the controller commands

the add accumulator unit to add and accumulate the multiplication result(out-valid) . The simulation also shows that the state changes back to IDLE state when a reset signal is applied to the controller.

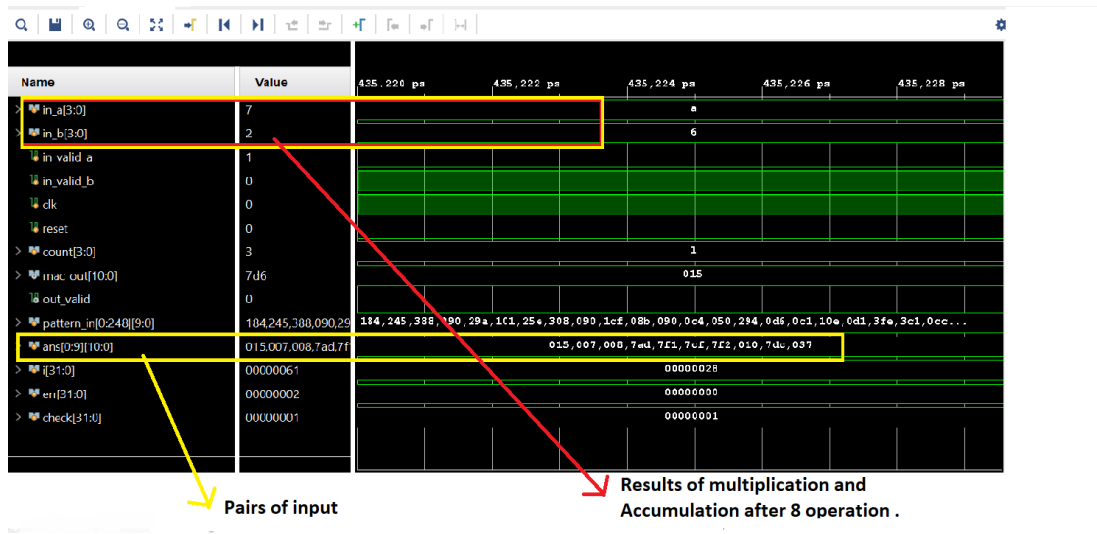


Figure 3.4: MAC Controller Simulation Timing Diagram

3.2 Testbench Verilog

Testbench provides inputs to the design unit and monitors any outputs. Figure below shows a structure of a testbench and its design under test (D.U.T). All Testbench source code is included in Appendix A

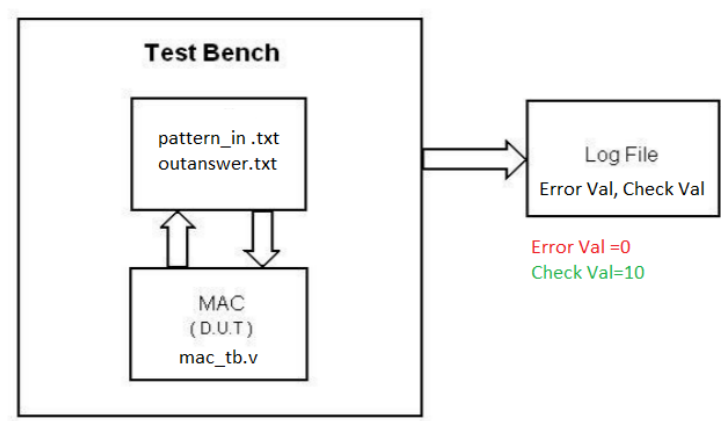


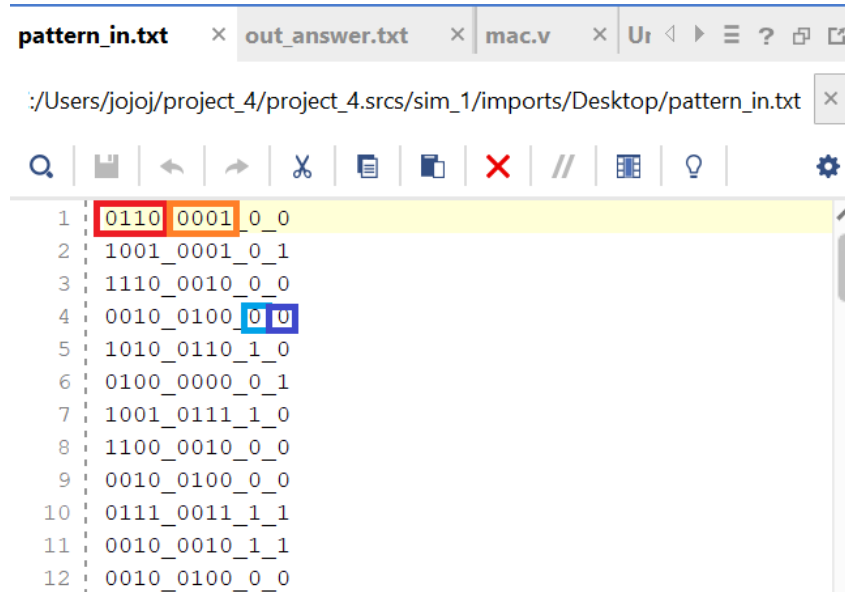
Figure 3.5: Testbench Simulation Block Diagram

3.3 Timing and Simulation

To verify functionality of the design of the MAC, a test bench is written. It is to verify that the

MAC is multiplying and accumulating, and the other one is to verify the add-accumulator holds the maximum possible accumulated value.

Input is given to test bench as pattern-in.txt and Calculated value of output is given to testbench as out-answer.txt .



```
pattern_in.txt  x  out_answer.txt  x  mac.v  x  UI  ◀ ▶ ≡ ? ☐ ☐
: /Users/fojoj/project_4/project_4.srscs/sim_1/imports/Desktop/pattern_in.txt  x
🔍 📁 ⏪ ⏩ ✂ 📄 📋 ✖ // 📊 💡 ⚙
1  0110_0001_0_0
2  1001_0001_0_1
3  1110_0010_0_0
4  0010_0100_0_0
5  1010_0110_1_0
6  0100_0000_0_1
7  1001_0111_1_0
8  1100_0010_0_0
9  0010_0100_0_0
10 0111_0011_1_1
11 0010_0010_1_1
12 0010_0100_0_0
```

Figure 3.6: Testbench Input pattern-in.txt

3.4 CONCLUSION

A complete design of an 4-by-4 bit signed Multiplier Accumulator was presented. The project aimed to design a high speed MAC with good regularity to increase CNN performace; this was analysed on Vivado .An Add/Shift Multiplier is selected because it generates a signal when it completes its multiplication process; this feature is one of the design requirements. All components were modeled with Verilog at behavior level and RTL level. The Verilog source codes were written and simulated on Vivado 2020 and windows-based CAD tools such as ModelSim. More projects could be done to explore and extend the power of MAC and improving CNN algorithm by accepting floating-point

```

1 000_0001_0101
2 000_0000_0111
3 000_0000_1000
4 111_1010_1101
5 111_1111_0001
6 111_1100_1111
7 111_1111_0010
8 000_0001_0000
9 111_1101_1100
10 000_0011_0111

```

Figure 3.7: Testbench Input out-answer.txt

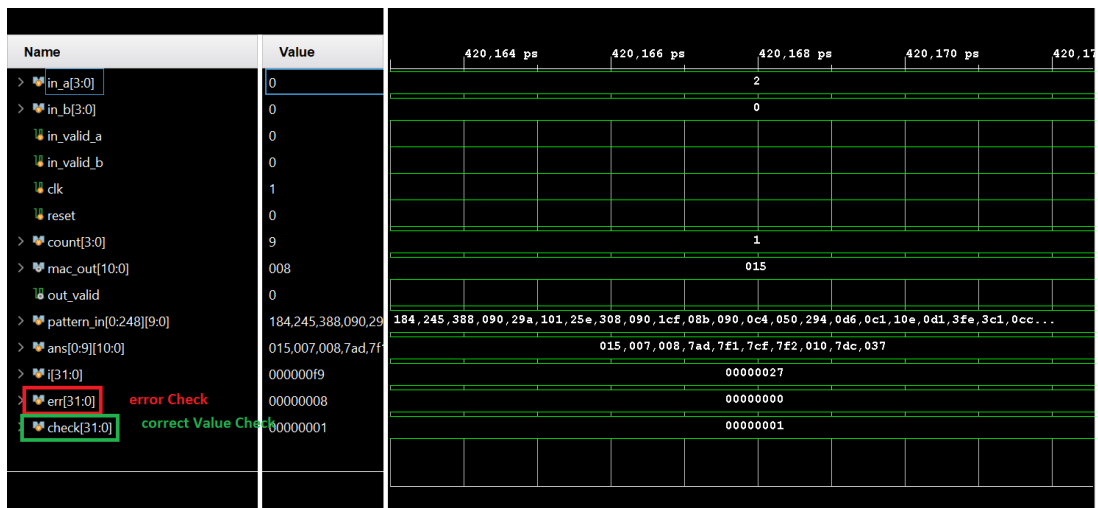


Figure 3.8: Error/Correct Val Check in Timing Diagram

numbers, optimizing the control unit and increasing speed by using Booth-Wallace Tree multiplier.