COL215 – Digital Logic and System Design

Department of Computer Science & Engineering, IIT Delhi Semester I, 2025–26

Lab Assignment 5 (Friday)

Vector Dot Product Controller using MAC unit

Developed By:
Bharmal Bhakar (2024CS10403)
Ritik Raj (2024CS10086)

1 Introduction

In this assignment, we designed and implemented a **Vector Dot Product Controller** on the Basys3 FPGA board. The objective was to compute the dot product of two vectors, each containing four 8-bit elements, by reusing the Multiply-Accumulate (MAC) unit designed in the previous assignment. The dot product is defined as:

$$\text{Dot Product} = \sum_{i=0}^{3} A[i] \times B[i]$$

where $A[i]$ and $B[i]$ are the i^{th} elements of vectors A and B, respectively.

Through this exercise, we became familiar with:

- Capturing and storing vector elements using FPGA switches.
- Resetting a design to a known initial state using BTNC.
- Indicating special states (reset, overflow) on the 7-segment display.
- Displaying dot product results on LEDs and the 7-segment display.

2 Design Decisions

We approached the problem with the following design considerations:

1. Input Handling

- Vectors *A* and *B* are stored as arrays of four 8-bit registers each.
- Switches SW[7:0] provide the data input, while SW8--SW9 select the element index (0–3).
- Write enables (SW12 for *A*, SW13 for *B*) control when an element is stored.
- sw14 and sw 15 are used to read the value of vector at the position specified by the index sw8 and sw 9. when both are they will show theproduct
- Rest anodes(an0-an1), cathodes(a-g), led, BTNC and clk are same as in assignment 4

2. Internal wires, parameters and reg

```
reg [7:0] vecA [3:0];   // stores the vectors
reg [7:0] vecB [3:0];

localparam FSM_IDLE=3'd0, FSM_LOAD_A=3'd1, FSM_LOAD_B=3'd2, FSM_ACCUM=3'd3;
// the keep track of which values to given to the mac for the calculation
reg [2:0] state = FSM_IDLE;  //idle state for FSM
reg [2:0] index = 3'd0;      // keeps track of index which is to be multiplied int he mac

reg mac_load_b_en, mac_load_c_en, mac_accum_en;
  //these are the variables which will used to communcate with the mac and the dot_product
reg [7:0] mac_data_in;
  //this is the input data which is given to the MAC
wire [15:0] dot_product_result;
wire mac_overflow;   //flag for overflow
reg done = 0;        //flag when the vector is calculated once
```

(meaning is same as shown in the picture)

3. Dot Product Computation(using MAC unit

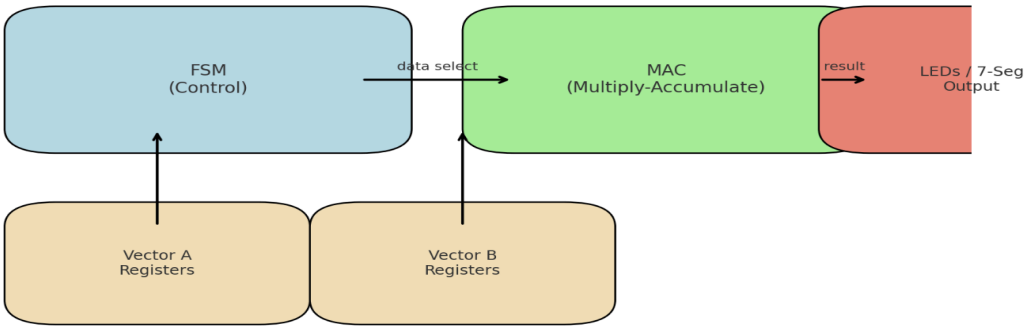
- Instanciating the MAC unit
- Dot product is calculated by the MAC unit A counter index iterates through indices 0 to 3, multiplying corresponding elements and accumulating the sum.

- The accumulator in the MAC is a 16-bit register, and we have another 17-bit variable that checks the overflow and assigns the first 16 bits to the accumulator.
- If overflow occurs, “OFLO” is displayed on the 7-segment display, and the least significant 16 bits will be shown on the LEDs.

```
// Instantiate the MAC from the assignment 4
MAC mac_unit (
    .clk(clk),
    .rst(BTNC),
    .data_in(mac_data_in),
    .load_b_en(mac_load_b_en),
    .load_c_en(mac_load_c_en),
    .accum_en(mac_accum_en),
    .result_out(dot_product_result),
    .overflow(mac_overflow)
);
```

working of code

1. The vectors **vecA** and **vecB** are loaded from the switches (sw12 for A, sw13 for B), storing values at the index given by sw8–sw9.
2. A small FSM (FSM_LOAD_A → FSM_LOAD_B → FSM_ACCUM) sequences the inputs going into the MAC.
3. In the **FSM_LOAD_A** state, the selected element of **vecA** is sent to mac_data_in with mac_load_b_en=1.
4. In the **FSM_LOAD_B** state, the corresponding element of **vecB** is sent to mac_data_in with mac_load_c_en=1.
5. In the **FSM_ACCUM** state, the MAC multiplies the latched A and B values and accumulates the partial sum when mac_accum_en=1.
6. The final **dot_product_result** from the MAC is continuously driven to the led output and optionally shown on the 7-segment display.

7. 

This way, your FSM controls the data flow into the MAC, step by step, and then reads out the accumulated dot product result.

4. Reset Handling

- The reset button (BTNC) clears vectors, the accumulator, and internal states.
- Upon reset, the 7-segment display displays "-rSt" for 5 seconds, (the code for that 5 secs is in the MAC unit).
- All the values in the vector becomes 0 and accumulator also beocomes 0.

```
// FSM State Transition Logic for the MAC to calculate
always @(posedge clk) begin
    if (BTNC) {state, index} <= {FSM_IDLE, 2'd0};
```

```
always @(posedge clk) begin
    if (BTNC) {vecA[0],vecA[1],vecA[2],vecA[3],vecB[0],vecB[1],vecB[2],vecB[3]} <= 0;
```

5. Output Display

- The accumulator result (dot product) is continuously displayed on the 16 LEDs in binary.
- The 7-segment display is used to display vector elements when read enables for any one vector. If both the read switches are on simultaneously, then the dot product will be displayed. It will display "-rst" and "OFLO" for reset and overflow, respectively.

- SW14 and SW15 act as read enables for vector *A* and *B,* respectively. Both ON simultaneously display the dot product on the 7-segment display and LEDs.

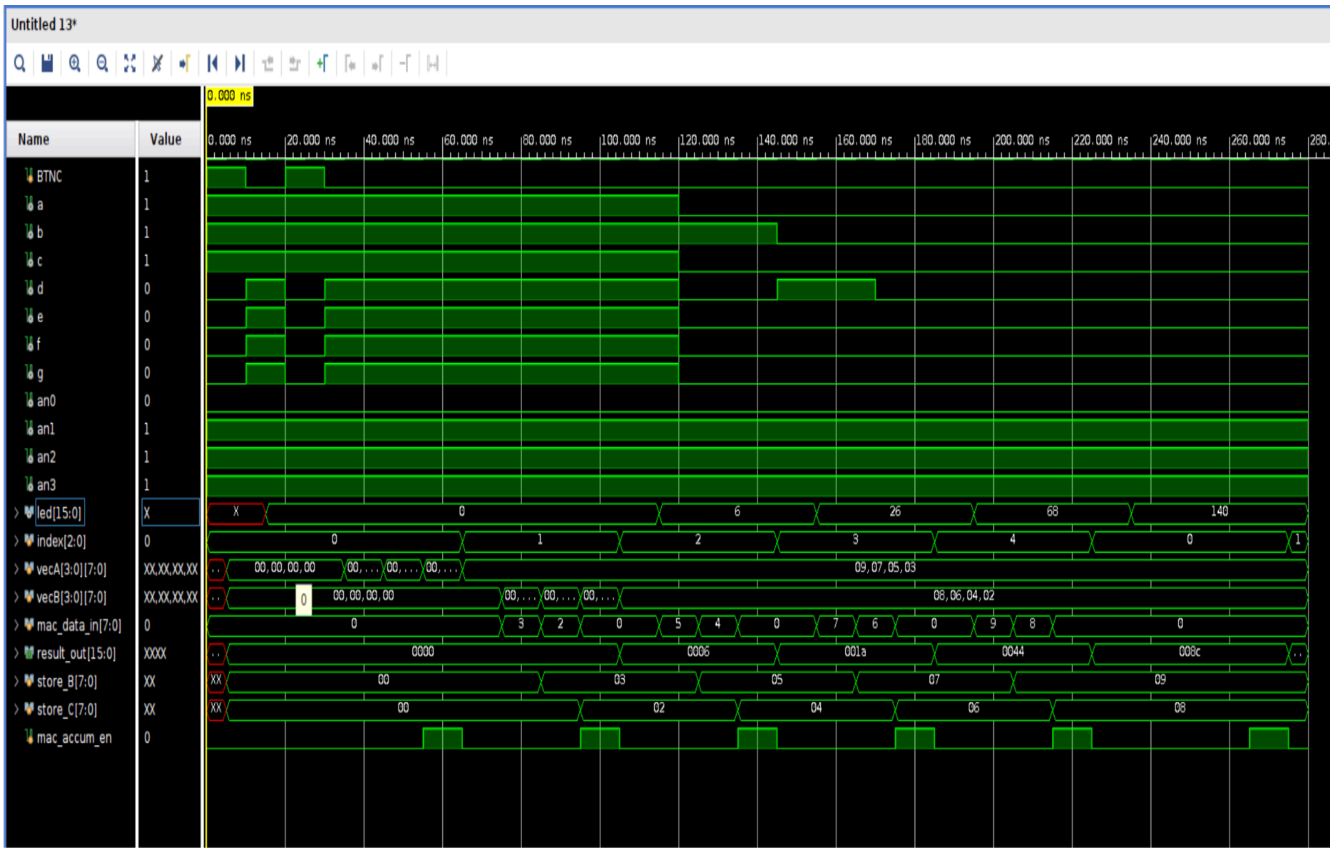
```
always @(*) begin
    {an3,an2,an1,an0} = 4'b1111; digit = 17;
    case(refresh_counter[19:18])
       2'b00: begin an0=0; if(BTNC) digit=20; else if(mac_overflow) digit=0; else if(sw14&sw15) digit=dot_product_result[3:0];
       else if(sw14) digit=vecA[{sw9,sw8}][3:0]; else if(sw15) digit=vecB[{sw9,sw8}][3:0]; end
       2'b01: begin an1=0; if(BTNC) digit=5; else if(mac_overflow) digit=18; else if(sw14&sw15) digit=dot_product_result[7:4];
       else if(sw14) digit=vecA[{sw9,sw8}][7:4]; else if(sw15) digit=vecB[{sw9,sw8}][7:4]; end
       2'b10: begin an2=0; if(BTNC) digit=19; else if(mac_overflow) digit=16; else if(sw14&sw15) digit=dot_product_result[11:8]; end
       2'b11: begin an3=0; if(BTNC) digit=15; else if(mac_overflow) digit=0; else if(sw14&sw15) digit=dot_product_result[15:12]; end
    endcase

    case(digit)
       4'h0: seg=7'b0000001; 4'h1: seg=7'b1001111; 4'h2: seg=7'b0010010; 4'h3: seg=7'b0000110;
       4'h4: seg=7'b1001100; 4'h5: seg=7'b0100100; 4'h6: seg=7'b0100000; 4'h7: seg=7'b0001111;
       4'h8: seg=7'b0000000; 4'h9: seg=7'b0000100; 4'hA: seg=7'b0001000; 4'hB: seg=7'b1100000;
       4'hC: seg=7'b0110001; 4'hD: seg=7'b1000010; 4'hE: seg=7'b0110000;
       5'd15: seg=7'b1111110; 5'd16: seg=7'b0111000; 5'd17: seg=7'b1111111; 5'd18: seg=7'b0111001;
       5'd19: seg=7'b0111110; 5'd20: seg=7'b1110000;
       default: seg=7'b1111111;
    endcase
end
```

3 Simulation Snapshots

We simulated the design to validate correct operation. Snapshots include:

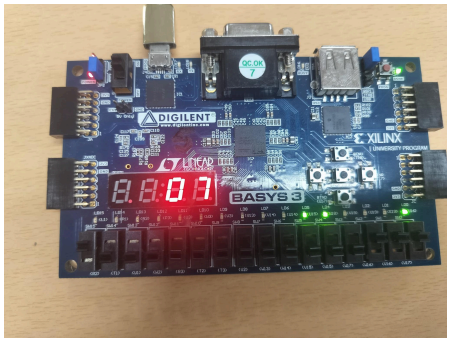
- Taking correct input and giving correct output
- Leds related to the output are correctly shown
- Normal dot product computation is done correctly
- The result on the output and all the other signals like index, mac-XX , leds, vecA/B are also corrretly and dot_product can be seen on the snapshot



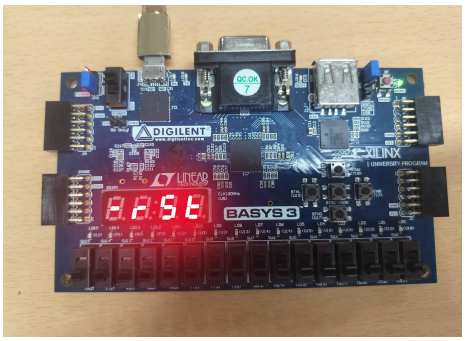
4 On-Board Implementation

The design was tested on the Basys3 FPGA board. Key observations:

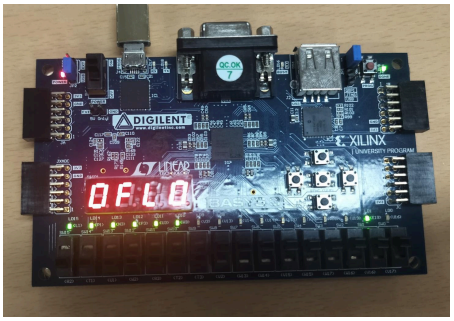
- The LEDs successfully display the dot product value in binary as well as on the display.
- The 7-segment display shows correct vector values on read enable.
- Overflow and reset states were correctly displayed.



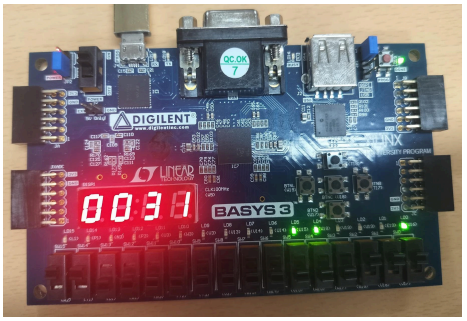
seg7 display showing value of B[0]



reset for 5 secs



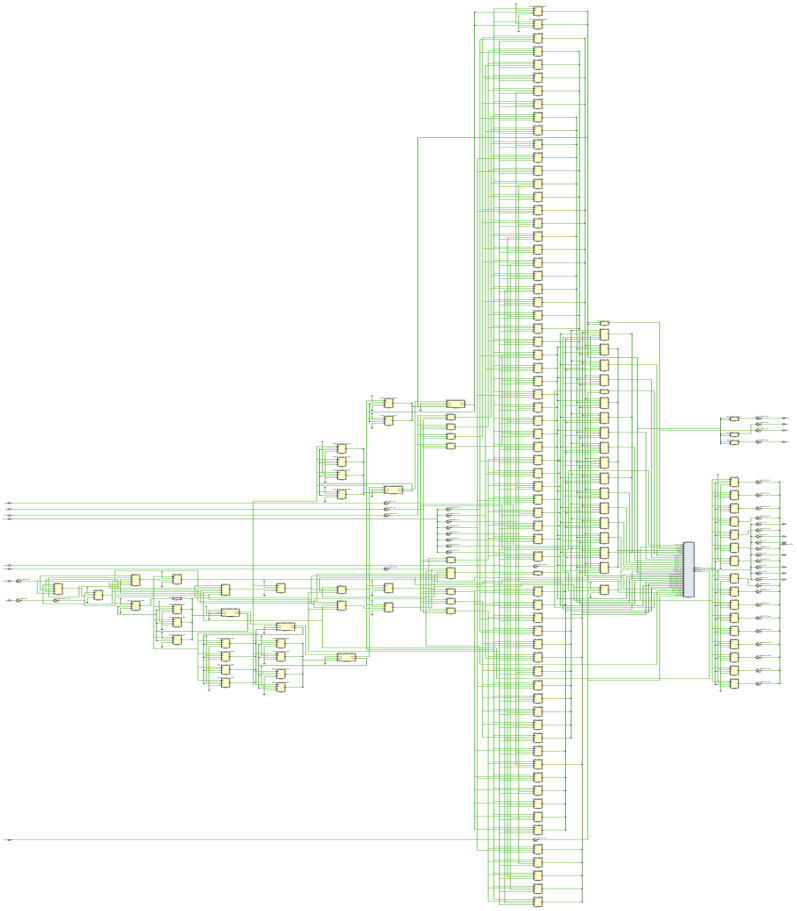
overflow



dot product

5 Generated Schematics

The synthesised schematic confirms the instantiation of registers, counters, and arithmetic blocks used in the controller. And as seen the MAC unit is successfully used (in gray color)



6 Synthesis Report

Info relates to LUTs, BRAM, Flip-Flops in the synthesis report

Resource Type	Used
Flip-Flops (FDRE)	139
LUTs	201 (sum of LUT1–LUT6)
BRAMs	0
DSPs	0

Report Cell Usage:

	Cell	Count
1	BUFG	1
2	CARRY4	20
3	LUT1	1
4	LUT2	49
5	LUT3	12
6	LUT4	22
7	LUT5	29
8	LUT6	88
9	FDRE	139
10	IBUF	16
11	OBUF	27

Name	Constraints	Status	WNS	TNS	WHS	THS	WBSS	TPWS	Total Power	Failed Routes	Methodology	RQA Score	QoR Suggestions	LUT	FF	BRAM	URAM	DSP	Start
✓ synth_1	constrs_1	synth_design Complete!												175	139	0	0	0	8/30/25,
✓ impl_1	constrs_1	write_bitstream Complete!	1.274	0.000	0.238	0.000		0.000	0.096	0	40 Warn			169	139	0	0	0	8/30/25,

7 Conclusion

In this assignment, we successfully designed and implemented a vector dot product controller on the Basys3 FPGA board. The design reused concepts from the MAC unit while introducing new mechanisms for handling multiple inputs, counters, overflow detection, and user interaction(i.e. .have used the previous assignment's module for the calculation part)

This project enhanced our understanding of integrating arithmetic logic with system-level control and user feedback mechanisms. We have learnt how to use different modules for different purposes.