MANISH PUNDIR
2022PVL0063
2022pvl0063@iitjammu.ac.in

12th December 2022

Major Project 1
Modified Booth Multiplier

---

1 Problem Statement

You have to design and implement a modified Booth algorithm based multiplier. The input word size must be equal to the last two digits of your roll number. The multiplier shall be able handle both signed and unsigned numbers. Also, it shall be able to handle all the exceptions and produce required flag signals as well.

2 Solution

2.1 What is Modified Booth multiplier ?

This modified booth multiplier is used to perform high-speed multiplications using modified booth algorithm. It is possible to reduce number of partial products by half, by using this modified algorithm.

2.2 Working of Modified Booth Multiplier

The basic idea is that, instead of shifting and adding of every column of the multiplier term and multiplying by 1 and 0, we only take every second column, and multiply by 0 or +1 or +2 or -1 or -2 to obtain the same result. This booth encoder performs the process of encoding the multiplicand based on multiplier bits. It will compare 3 bits at time with overlapping technique. Grouping start from the LSB, in which only two bits of the booth multiplier are used by the first block and a zero is assumed as third bit as shown in the below table :

Multiplier Bits Block			Recoded 1-bit pair		2-bit booth
i+1	i	i-1	i+1	i	Multiplier Value
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	1	-1	1
0	1	1	1	0	2
1	0	0	-1	0	-2
1	0	1	-1	1	-1
1	1	0	0	-1	-1
1	1	1	0	0	0

Figure 1: Booth Recoding Table

Modified booth multiplication algorithm consists of three major steps as:

- Generation of partial products from multiplier bits called as Recoding.
- Reducing the number of partial products in two rows.
- Addition of partial products that gives the final Result.

2.3 Inputs Signals

- 63 bit Multiplicand
- 63 bit Multiplier

2.4 Output Signals

- 127 bit Product Register

2.5 Architecture of Modified Booth Multiplier

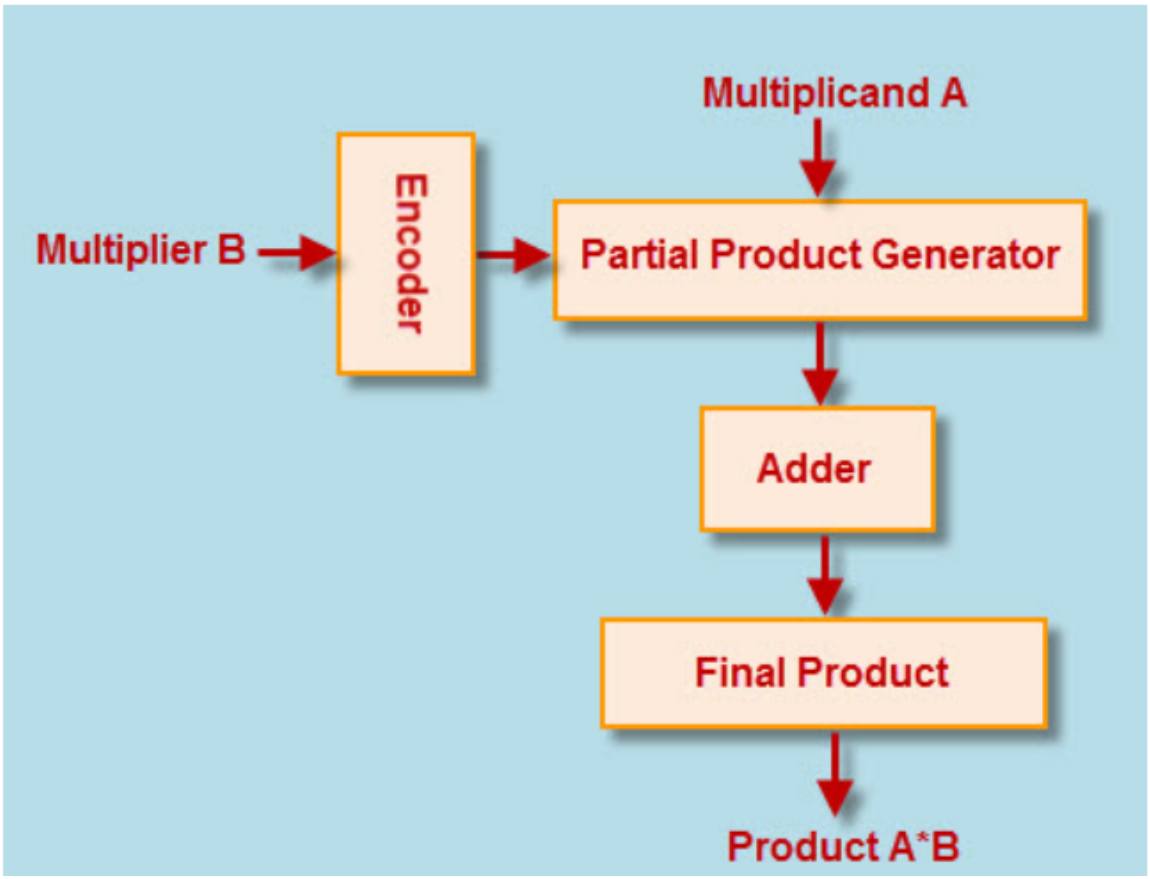

Figure 2: Architecture of Modified Booth Multiplier

3 Verilog HDL Code for Modified Booth Multiplier

```
module multiplier_64x64 (
input [63:0] mltpcd,
input [63:0] mltplr,
output reg [128:0] product // 1 bit extra for sign adjustment.
);
integer i;
reg [2:0] enc;
reg [128:0] pp; // for partial product generation
always @(*)
begin
product = 129'b0;
for (i = 0; i < 30; i = i+1)
begin
enc = (i == 0)? {mltpcd[1:0],1'b0}: // first encode
(i == 1)? mltpcd[3:1] :(i == 2)? mltpcd[5:3] :(i == 3)? mltpcd[7:5] :
(i == 4)? mltpcd[9:7]:(i == 5)? mltpcd[11:9]:(i == 6)? mltpcd[13:11]:
(i == 7)? mltpcd[15:13]:(i == 8)? mltpcd[17:15]: (i == 9)? mltpcd[19:17]:
(i == 10)? mltpcd[21:19]: (i == 11)? mltpcd[23:21]: (i == 12)? mltpcd[25:23]:
(i == 13)? mltpcd[27:25]:(i == 14)?mltpcd[29:27]: (i == 15)? mltpcd[31:29]:
(i == 16)? mltpcd[33:31]: (i == 17)? mltpcd[35:33]:(i == 18)? mltpcd[37:35]:
(i == 19)? mltpcd[39:37]: (i == 20)? mltpcd[41:39]: (i == 21)? mltpcd[43:41]:
(i == 22)? mltpcd[45:43]:(i == 23)? mltpcd[47:45]: (i == 24)? mltpcd[49:47]:
(i == 25)? mltpcd[51:49]: (i == 26)? mltpcd[53:51]: (i == 27)? mltpcd[55:53]:
(i == 28)? mltpcd[57:55]:(i == 29)? mltpcd[59:57]:
(i == 30)? mltpcd[61:59]: mltpcd[63:61];

//partial product generation
pp = (i == 0)? {{65{mltplr[63]}},mltplr[63:0]}:
(i == 1)? {{63{mltplr[63]}},mltplr[63:0],2'b00}:
(i == 2)? {{61{mltplr[63]}},mltplr[63:0],4'b0000}:
(i == 3)? {{59{mltplr[63]}},mltplr[63:0],6'b0000}:
(i == 4)? {{57{mltplr[63]}},mltplr[63:0],8'b0000}:
(i == 5)? {{55{mltplr[63]}},mltplr[63:0],10'b0000}:
(i == 6)? {{53{mltplr[63]}},mltplr[63:0],12'b0000}:
(i == 7)? {{51{mltplr[63]}},mltplr[63:0],14'b0000}:
(i == 8)? {{49{mltplr[63]}},mltplr[63:0],16'b0000}:
(i == 9)? {{47{mltplr[63]}},mltplr[63:0],18'b0000}:
(i == 10)? {{45{mltplr[63]}},mltplr[63:0],20'b0000}:
(i == 11)? {{43{mltplr[63]}},mltplr[63:0],22'b0000}:
(i == 12)? {{41{mltplr[63]}},mltplr[63:0],24'b0000}:
(i == 13)? {{39{mltplr[63]}},mltplr[63:0],26'b0000}:
(i == 14)? {{37{mltplr[63]}},mltplr[63:0],28'b0000}:
(i == 15)? {{35{mltplr[63]}},mltplr[63:0],30'b0000}:
(i == 16)? {{33{mltplr[63]}},mltplr[63:0],32'b0000}:
(i == 17)? {{31{mltplr[63]}},mltplr[63:0],34'b0000}:
```

```
(i == 18)? {{29{mltplr[63]}},mltplr[63:0],36'b0000}:
(i == 19)? {{27{mltplr[63]}},mltplr[63:0],38'b0000}:
(i == 20)? {{25{mltplr[63]}},mltplr[63:0],40'b0000}:
(i == 21)? {{23{mltplr[63]}},mltplr[63:0],42'b0000}:
(i == 22)? {{21{mltplr[63]}},mltplr[63:0],44'b0000}:
(i == 23)? {{19{mltplr[63]}},mltplr[63:0],46'b0000}:
(i == 24)? {{17{mltplr[63]}},mltplr[63:0],48'b0000}:
(i == 25)? {{15{mltplr[63]}},mltplr[63:0],50'b0000}:
(i == 26)? {{13{mltplr[63]}},mltplr[63:0],52'b0000}:
(i == 27)? {{11{mltplr[63]}},mltplr[63:0],54'b0000}:
(i == 28)? {{9{mltplr[63]}},mltplr[63:0],56'b0000}:
(i == 29)? {{7{mltplr[63]}},mltplr[63:0],58'b0000}:
(i == 30)? {{5{mltplr[63]}},mltplr[63:0],60'b0000}:
 {{3{mltplr[63]}},mltplr[63:0],62'b0000};

case (enc) // partial product gen
3'b000: pp = 129'b0; // 0M
3'b001: pp = pp; // 1M
3'b010: pp = pp; // 1M
3'b011: pp = pp << 1; // 2M
3'b100: pp = ~(pp << 1) + 129'b1; // -2M
3'b101: pp = ~pp + 129'b1; // -M
3'b110: pp = ~pp + 129'b1; // -M
3'b111: pp = 129'b0; // 0M
endcase
product = product + pp; // accumulate pp
end
end
endmodule
```

4 Test Cases

4.1 Both Multiplier and Multiplicand are Postive

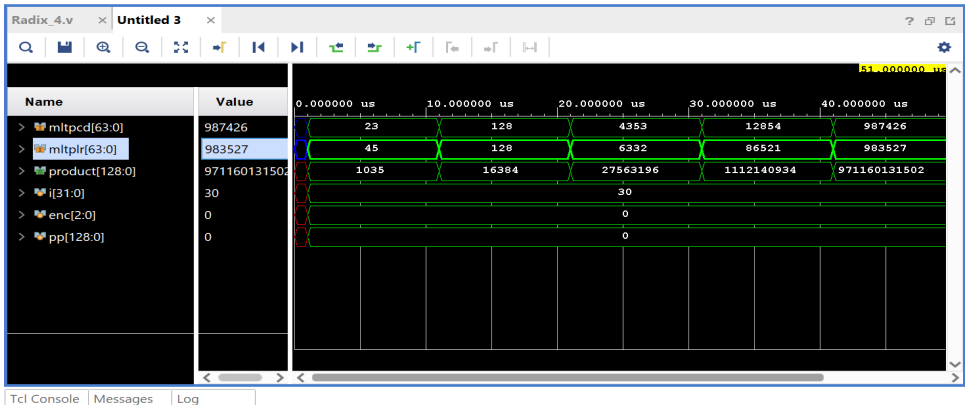

Figure 3: Vivado Simulation when both Multiplier and Multiplicand are Positive

4.2 When Multiplier is Negative and Multiplicand is Positive

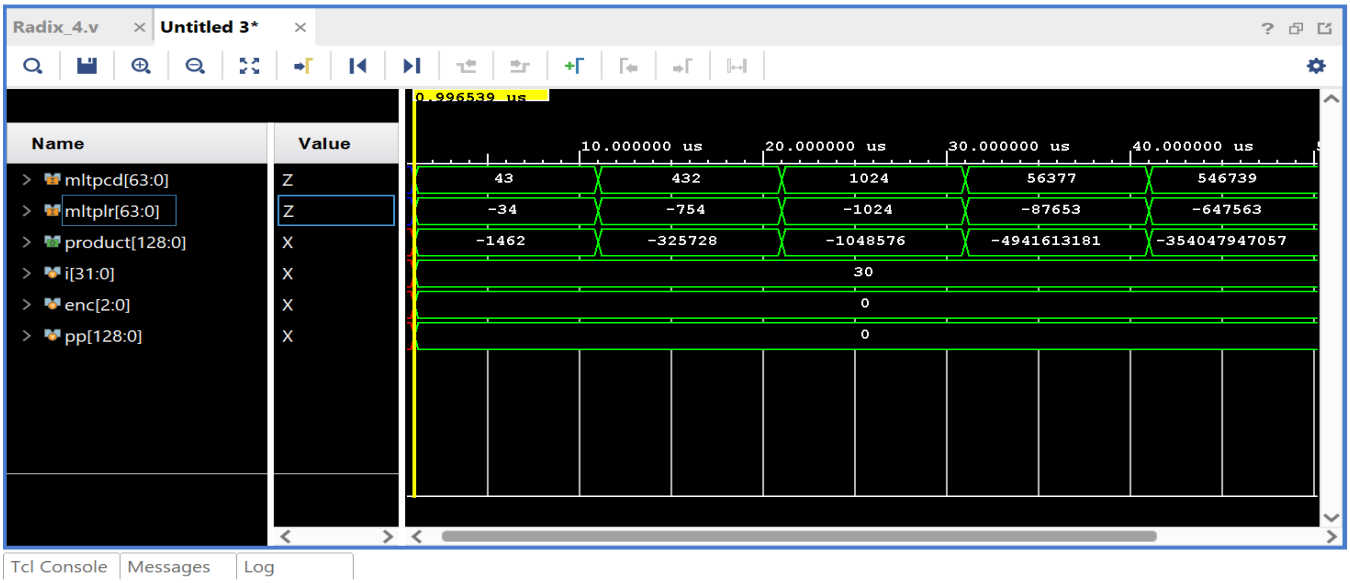

Figure 4: Vivado Simulation when Multiplier is negative and Multiplicand is Positive

4.3 When both Multiplier and Multiplicand is negative

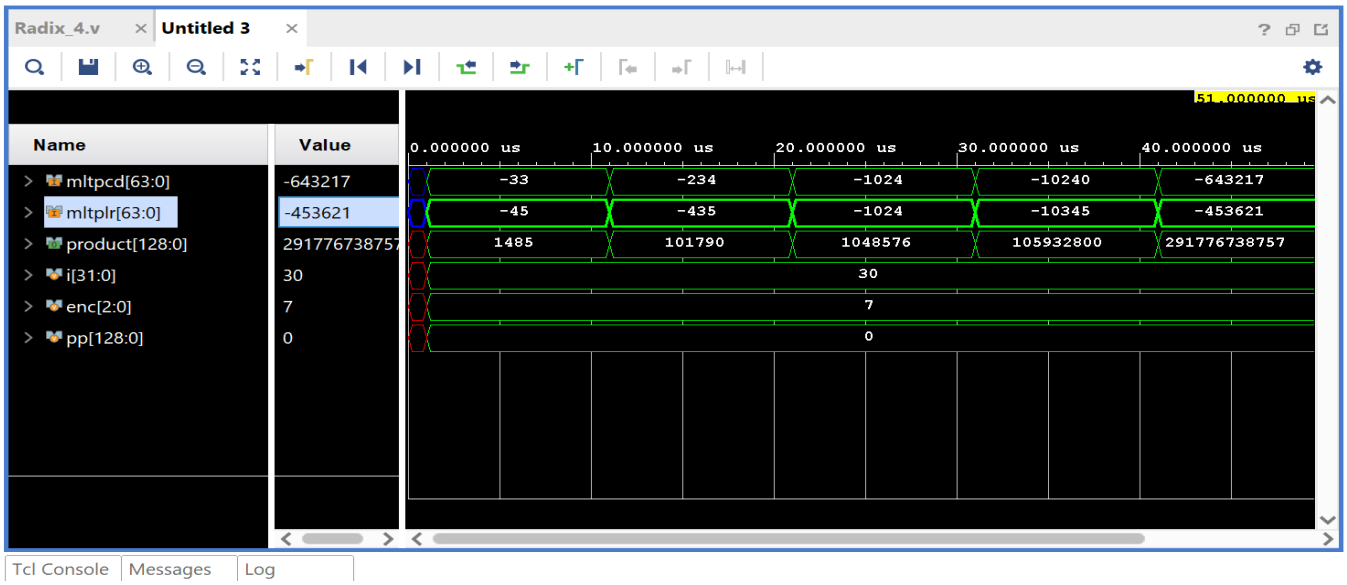

Figure 5: Vivado Simulation when When both Multiplier and Multiplicand is negative

5 Elaborated Design

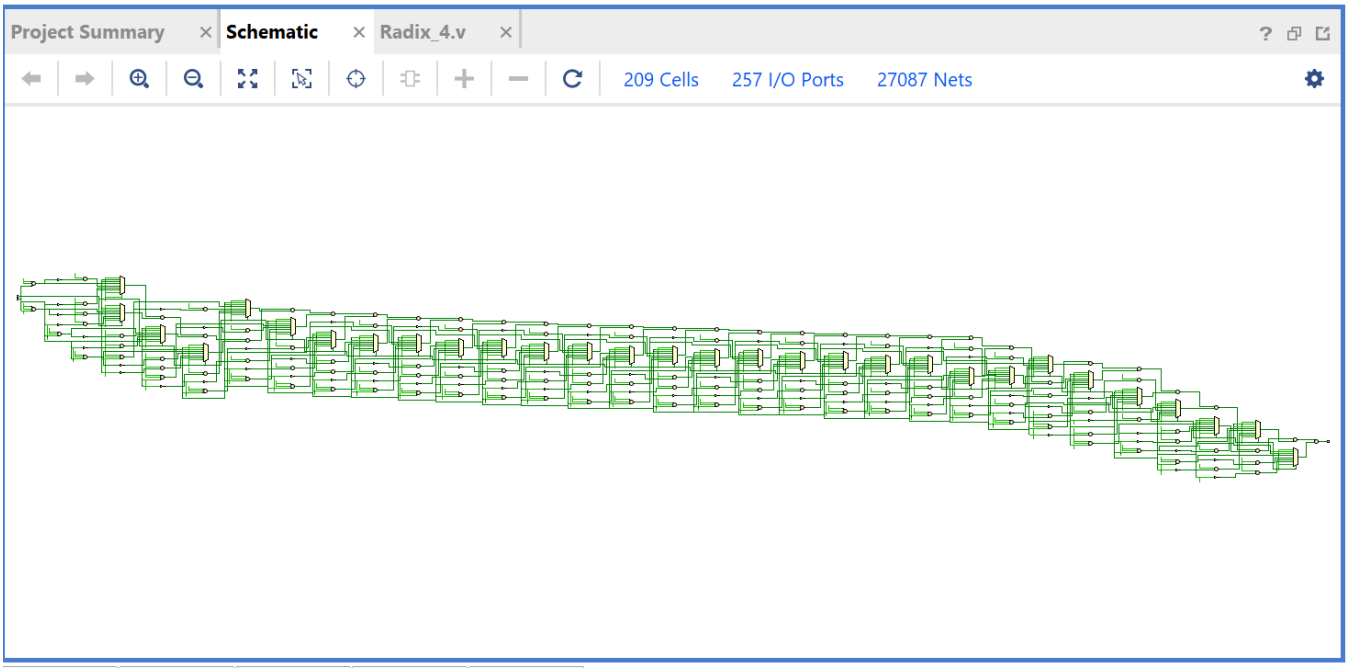

Figure 6: Elaborated Design of Modified Booth Algorithm using Vivado