COL216 : Assignment 0

Abhinav Rajesh Shripad (2022CS11596)
Jahnabi Roy (2022CS11094)

January 7, 2024

1 Introduction

The report explains the design principles used in implementing fixedLatencyMultiplier and variableLatecyMultiplier modules in Verilog, with their area's and time taken for synthesis. a, b are 16 bit input numbers in signals and we output 32 bit number c.

2 Implementation Details

2.1 Fixed Latency Multiplier

We initialize $c = 0$ and then go through each (i^{th}) bit (starting from 0) of the a, and if the corresponding bit is 1, we add $2^i b$ ($b << i$) to c.
For each bit we take a single clock cycle thus for making n bit multiplier we need n clock cycles, which makes it a fixed latency multiplier.

2.2 Variable Latency Multiplier (Booth's Algorithm)

We again initialize $c = 0$, and make 2 pointers each pointing to a bit in a, we use them to find consecutive $1's$ or $0's$. We find each consecutive stream in a single clock cycle and then update c accordingly.

3 Synthesis and Simulation

Download the submitted zip file and extract the files from it.
To run fixedLatencyMultiplier write the following command in terminal

- `$ iverilog -o fixedLatencyMultiplier fixedLatencyMultiplier.v`
- `$ vvp fixedLatencyMultiplier`

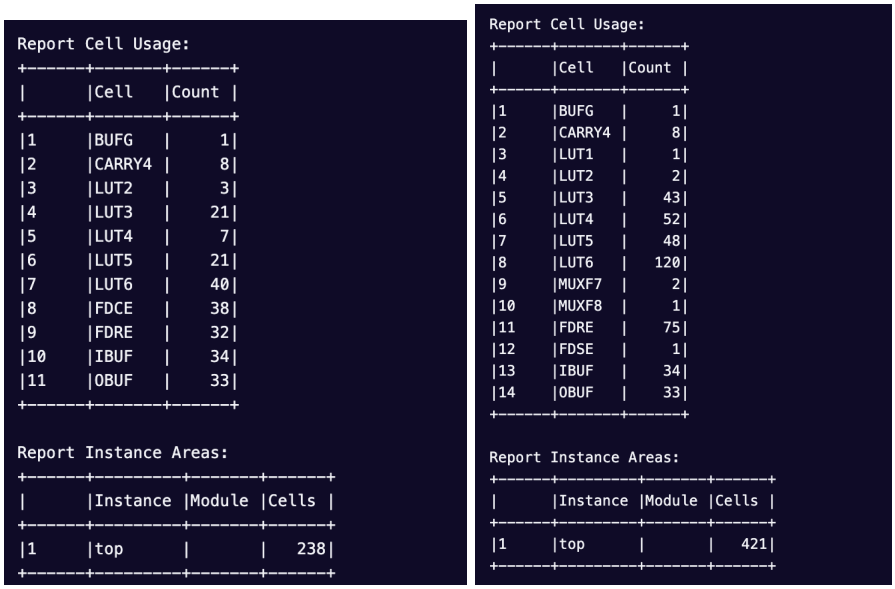
To run variableLatencyMultiplier write the following command in terminal

- `$ iverilog -o variableLatencyMultiplier variableLatencyMultiplier.v`
- `$ vvp variableLatencyMultiplier`

Each thing will generate 2 `.txt` files which contain the data for the number of Clock cycles taken by the code for computing the results with corresponding values of a,b,c and $a * b$ computed by inbuilt multiplication along with the testcase number.
We have also submitted the synthesis report generated by Vivado and also attached Screenshot's from same for comparing area.

4 Clock Cycles and Area

We can see from the files generated above and by synthesis report that for fixedLatencyMultiplier, the number of clock cycles is constant 16. For variableLatencyMultiplier it is worst case 17 and on average 8.052 . Also observe that total simulation time is also 50 percent.



Report Cell Usage:

	Cell	Count
1	BUFG	1
2	CARRY4	8
3	LUT2	3
4	LUT3	21
5	LUT4	7
6	LUT5	21
7	LUT6	40
8	FDCE	38
9	FDRE	32
10	IBUF	34
11	OBUF	33

Report Instance Areas:

	Instance	Module	Cells
1	top		238

Report Cell Usage:

	Cell	Count
1	BUFG	1
2	CARRY4	8
3	LUT1	1
4	LUT2	2
5	LUT3	43
6	LUT4	52
7	LUT5	48
8	LUT6	120
9	MUXF7	2
10	MUXF8	1
11	FDRE	75
12	FDSE	1
13	IBUF	34
14	OBUF	33

Report Instance Areas:

	Instance	Module	Cells
1	top		421

Figure 1: Area taken by (1)fixedLatency (2)VariableLatency

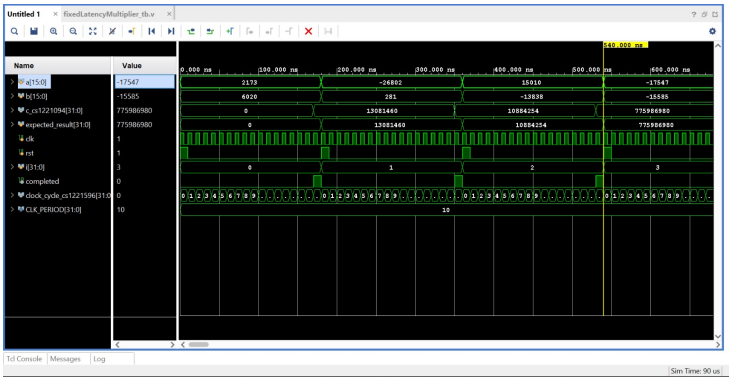


Figure 2: fixedLatency Simulation

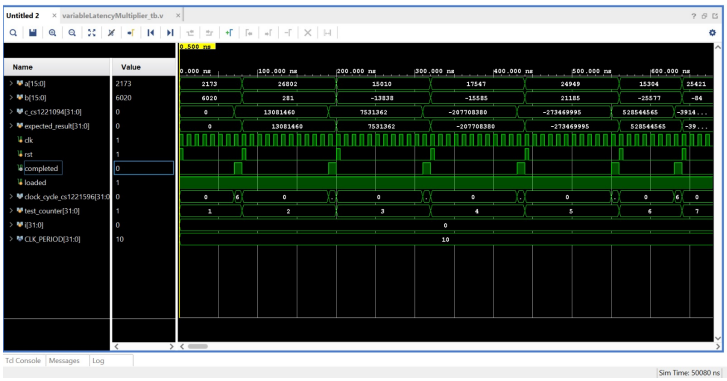


Figure 3: VariableLatency Simulation

5 Performance Comparison

We can see that Booth Algorithm increase the area taken by the circuit by approx 76 percent and decreases the number of clock cycles by approx 50 percent. Few of the worst performing testcases in Booth's algorithm are:-

a	b	ClockCycles
1010101010101010	1010101010101010	17
1010101010101010	1111111111111111	16
1010101010101001	110111011101111	15
1010101010100101	110111011101111	15
1010101011000101	110111011101111	14

6 Submission Details

The submission consists of a `.zip` file which includes the testcase folders and `main.py` file. Each testcase folder contains the following items:

1. `Report.pdf`
2. `fixedLatencyMultiplier.v`
3. `variableLatencyMultiplier.v`
4. `tb_fixedLatencyMultiplier.v`
5. `tb_variableLatencyMultiplier.v`
6. `fixedLatencyMultiplier.vds`
7. `variableLatencyMultiplier.vds`
8. `fixedLatencyMultiplier.txt`(Contains data for 500 testcases)
9. `variableLatencyMultiplier.txt`(Contains data for 500 testcases)
10. `Screenshot of Running Simulation`