ECE 385

Fall 2023

Experiment #4

An 8-Bit Multiplier in SystemVerilog

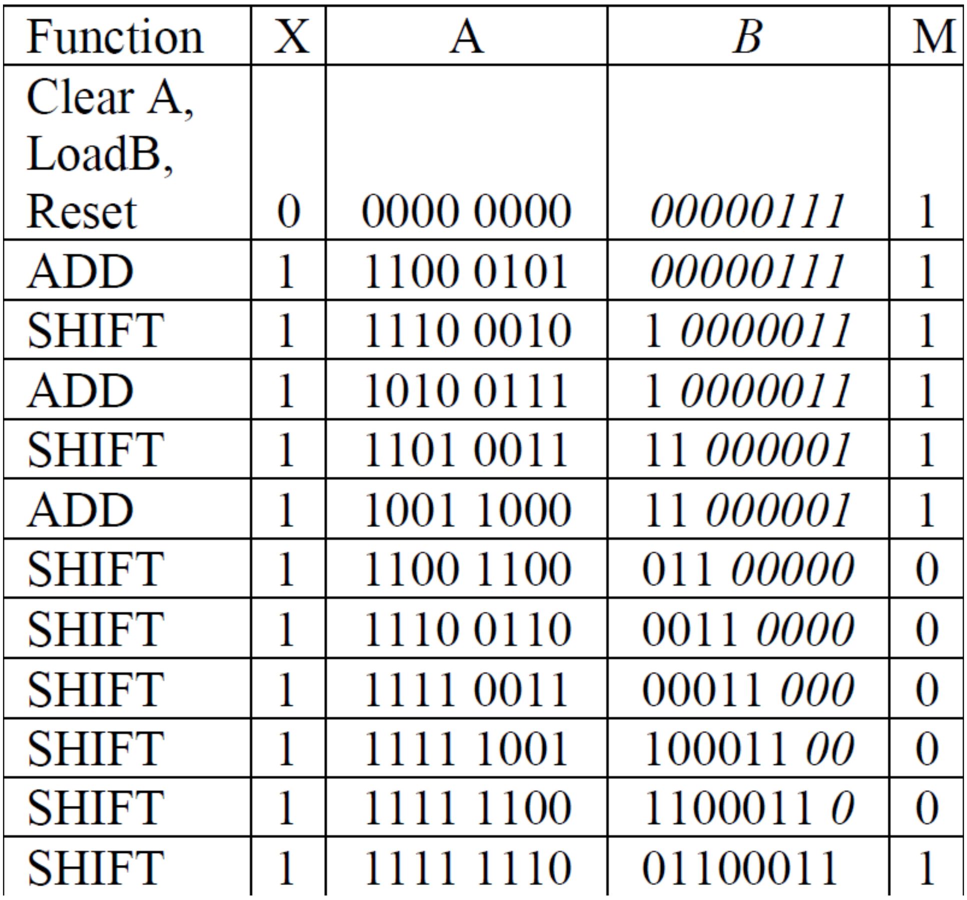
Ziyi Lin
Lai Zhouyang
TA: Jerry Wang

1.Introduction

This multiplier circuit uses shift and add method to complete multiply functionality. It takes an input with reset button being pushed and store the data in SW into B, an eight-bit shift register. Then, once the run button is pushed, SW multiplies with register B, creating a 16-bit data, store in A and B registers. Also, there is an X register with 1 bit wide, it is used to storge the positive or negative prosperity of the number stored in A and B register. Pushing run again without reset will reset A and X value, while again repeat the procedure of multiply with the bits stored in register B and the signal in SW.

2.Pre-lab questions

a. See graph 2.1



Function	X	A	*B*	M
Clear A, LoadB, Reset	0	0000 0000	*00000111*	1
ADD	1	1100 0101	*00000111*	1
SHIFT	1	1110 0010	1 *0000011*	1
ADD	1	1010 0111	1 *0000011*	1
SHIFT	1	1101 0011	11 *000001*	1
ADD	1	1001 1000	11 *000001*	1
SHIFT	1	1100 1100	011 *00000*	0
SHIFT	1	1110 0110	0011 *0000*	0
SHIFT	1	1111 0011	00011 *000*	0
SHIFT	1	1111 1001	100011 *00*	0
SHIFT	1	1111 1100	1100011 *0*	0
SHIFT	1	1111 1110	01100011	1

Graph 2.1(pre-lab question a)

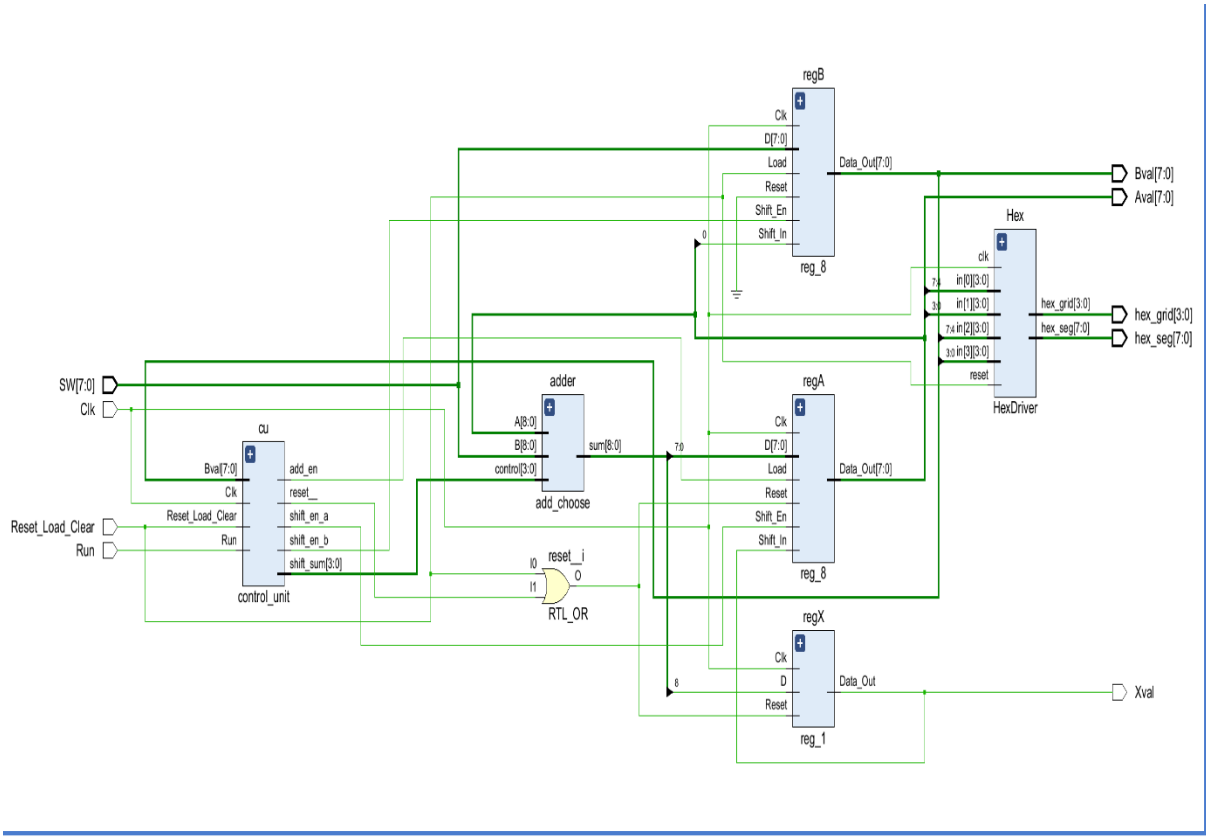
3.Written description and diagrams of mutiplier circuit

a.

This multiplier circuit uses the shift-and-add technique to achieve its multiplication

function. When the reset button is pressed, the data from SW is stored in B, an eight-bit shift register. Upon pressing the run button, SW multiplies with the content of register B, resulting in a 16-bit output stored across registers A and B. Additionally, there's a 1-bit wide X register used to indicate the sign (positive or negative) of the number held in registers A and B. If the run button is pressed again without a prior reset, registers A and X are cleared, but the multiplication process repeats using the data in register B and the current input in SW.

b.

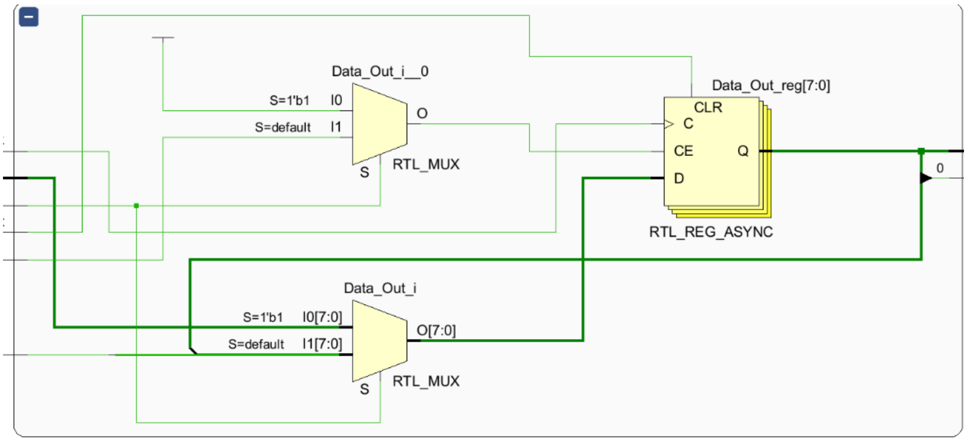


Graph 3.1（toplevel block diagram）

c.
Module 1:
Module: mutiply_toplevel
Input:[7:0] SW, Clk, Reset_Load_Clear
Output:[3:0] hex_grid, [7:0] hex_seg, [7:0] Aval, [7:0] Bval, Xval
Description: toplevel of the mutiplier circuit.
Purpose: combine other module together.
(diagram already provided above)

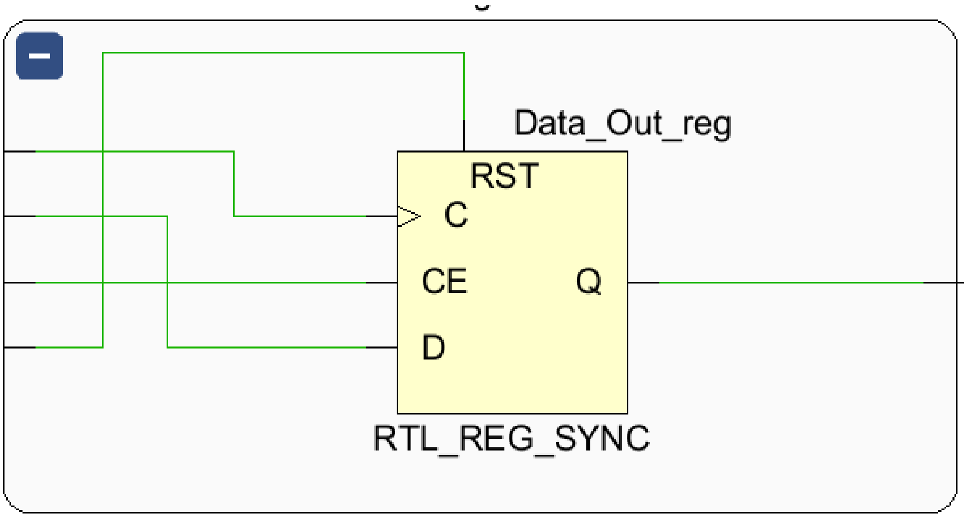
Module 2:
Module: reg_8

Input: Clk, Reset, Shift_en, shift_in, Load_en,[7:0] D
Output:[7:0] Data_Out
Description: a module for 8-bit shift-register.
Purpose: store register A and B



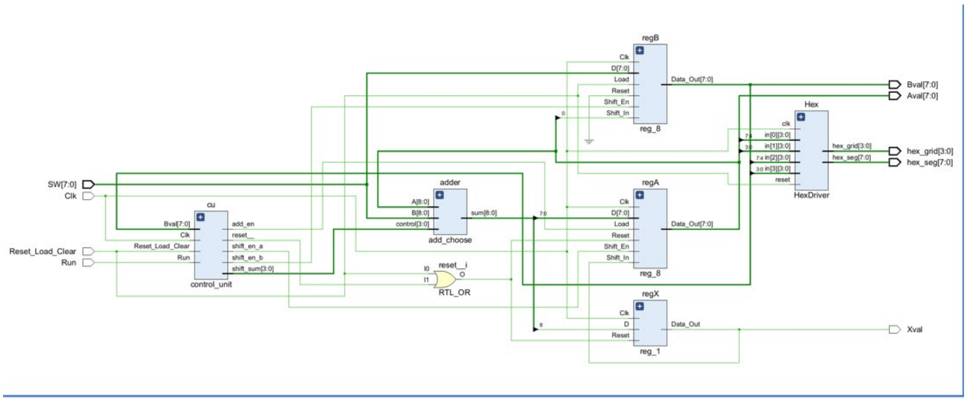
Graph 3.31(reg_8)

Module 3:
Module: reg_1
Input: Clk, Reset, Load_en, D
Output: Data_Out
Description:1-bit register
Purpose: store register X



Graph 3.32(reg_1)

Module 4:
Module: control unit
Input: Clk,Reset_Load_Clear, Clk,Run, [7:0] Bval
Output: reset__, adden, shift_en_a, shift_en_b, [3:0] shift_sum
Description: state machine to output add and shift enable signals

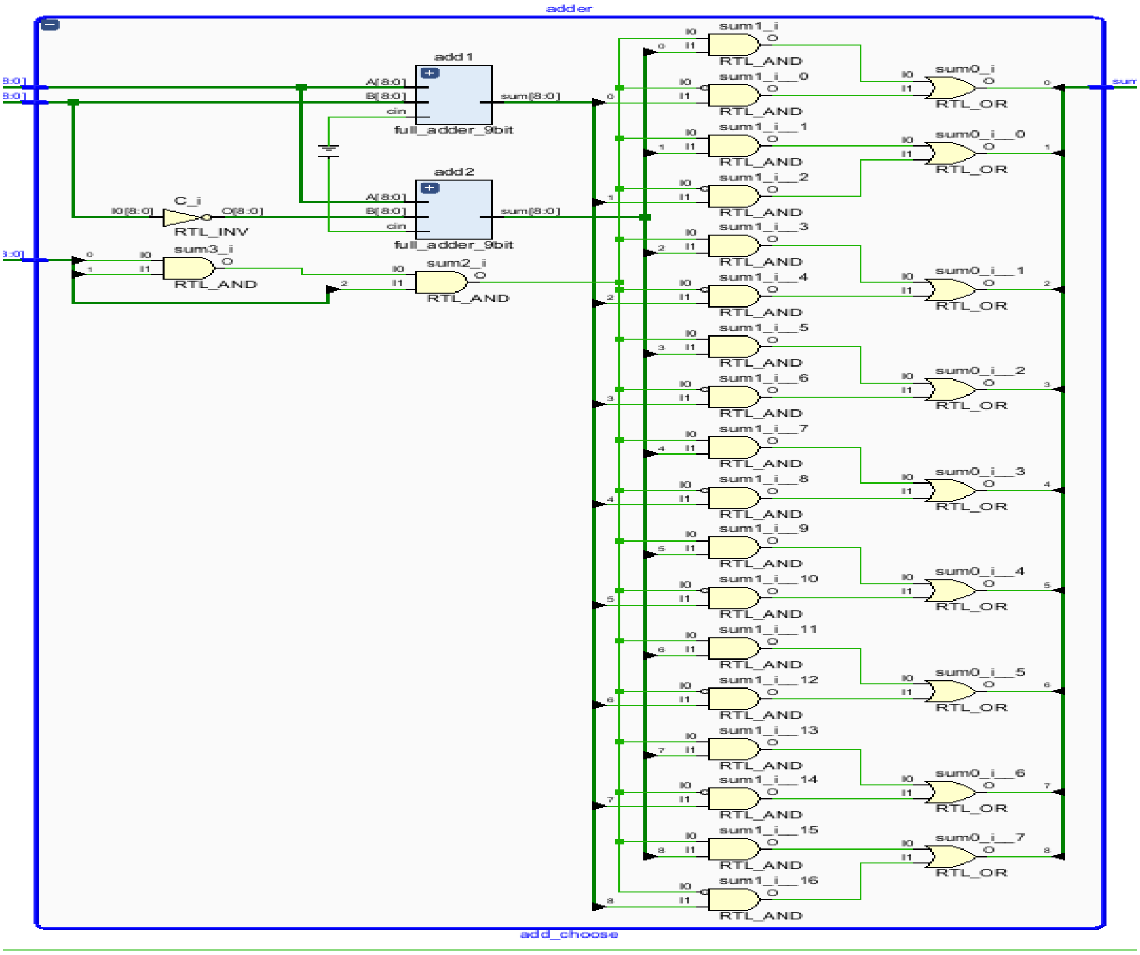


Graph 3.33 (control_unit)

Purpose: to determine when to add or shift or halt when the operation has finished.

Module 5:
Module: hex_driver
Input: Clk, reset, [3:0]in[4]
Output: [7:0]hex_seg,[3:0]hex_grid
Desciption: hex display driver
Purpose: to display the output in hex form.

Module 6:
Module: add_choose
Input:[8:0]A,[8:0]B,[3:0] control
Output: [8:0] sum
Desciption: choose from subtration or addtion
Purpose: when it is the last shift possible and your M = 1 you should perform subtration otherwise perform addtion

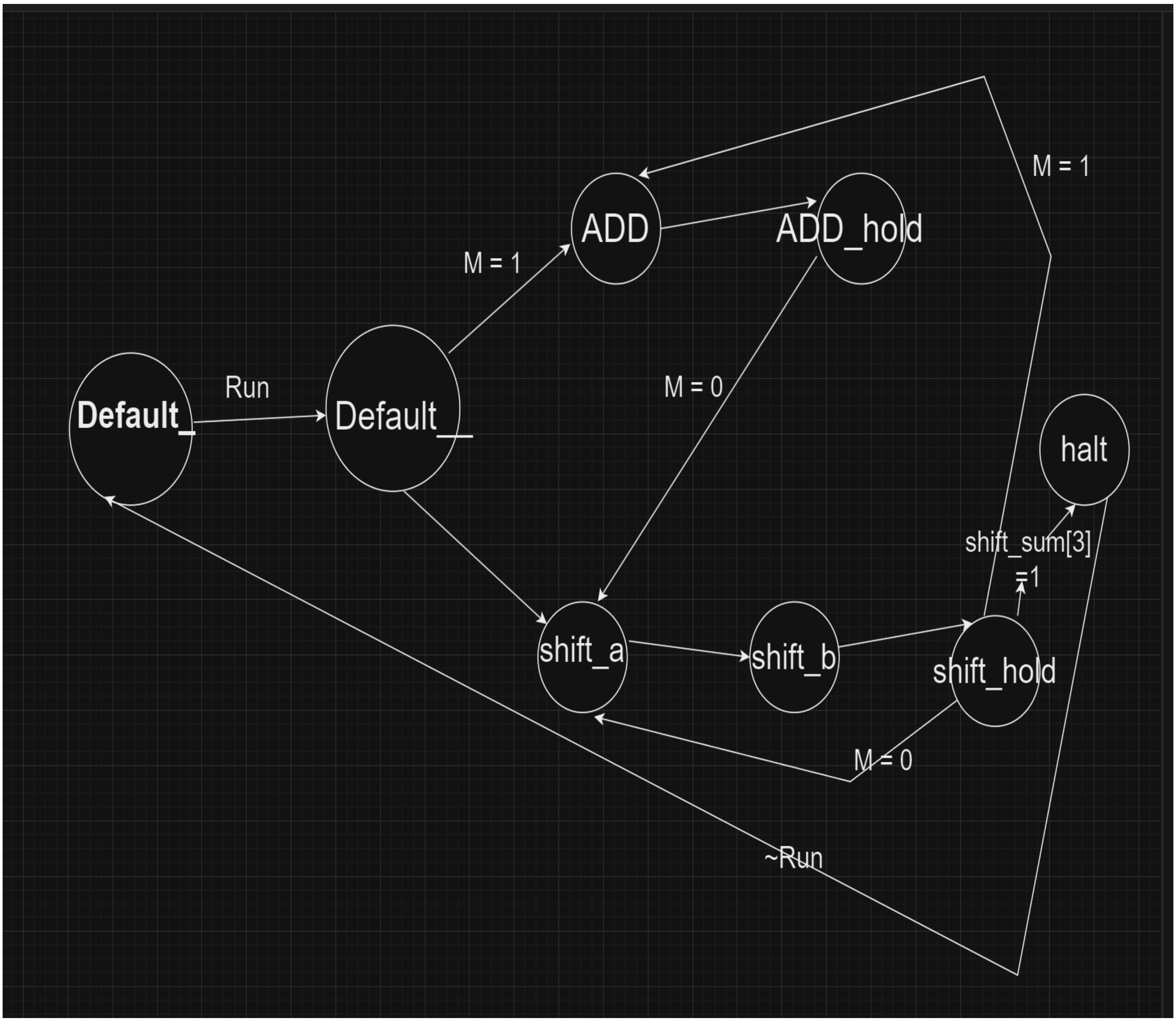


Graph 3.34（add_choose）

Module 7:
Module: full_adder_9bit
Input: [8:0]A,[8:0]B,cin
Output: [8:0] sum, cout
Description: 9-bit adder
Purpose: used for A = A + SW

Module 8:
Module: ripple_adder_1bit
Input: A,B,cin
Output :sum, cout
Description:1-bit full adder
Purpose: construct full_adder_1bit

d.



Graph 3.4 (state diagram)

4.Annotated pre-lab simulation waveforms.

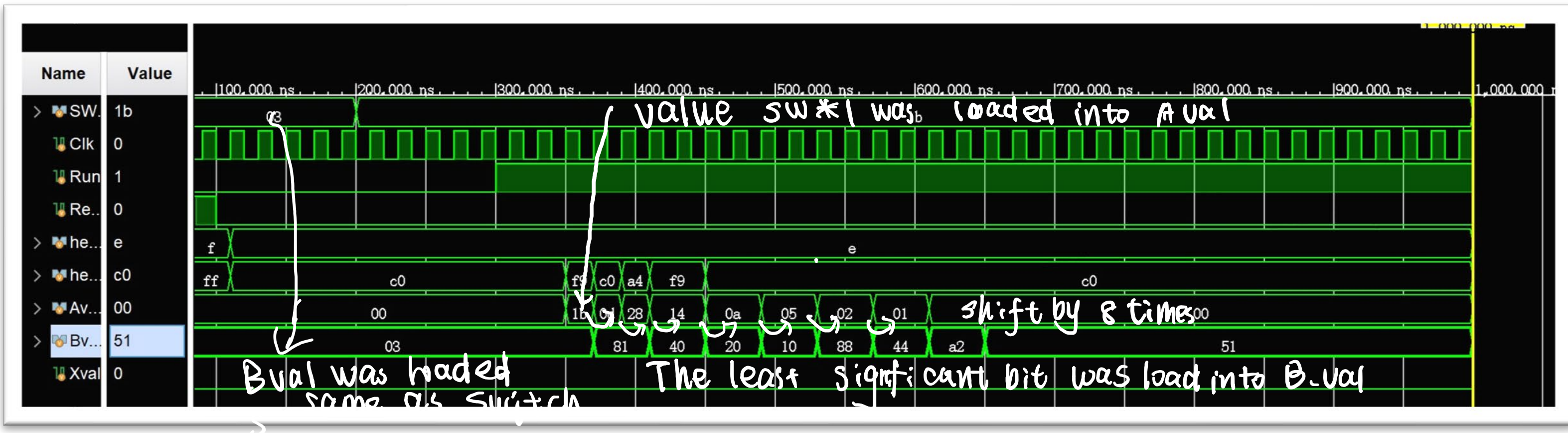


Figure.1 multiplication of (27*3=81)

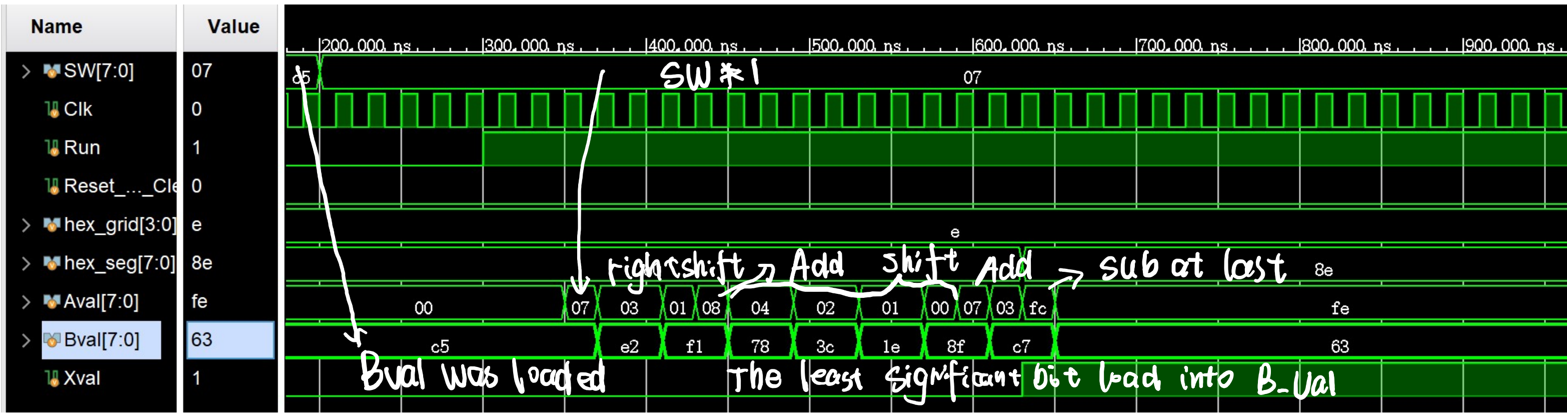


Figure 4.2 multiplication of （-59*7=-413）

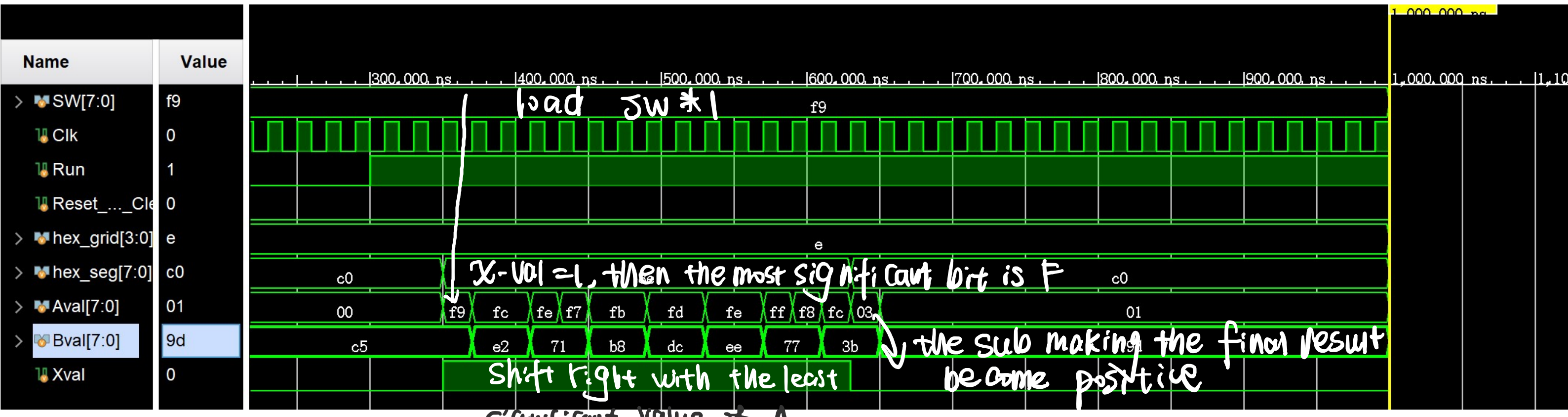


Figure 4.3 (multiplication of -59*-7=413)

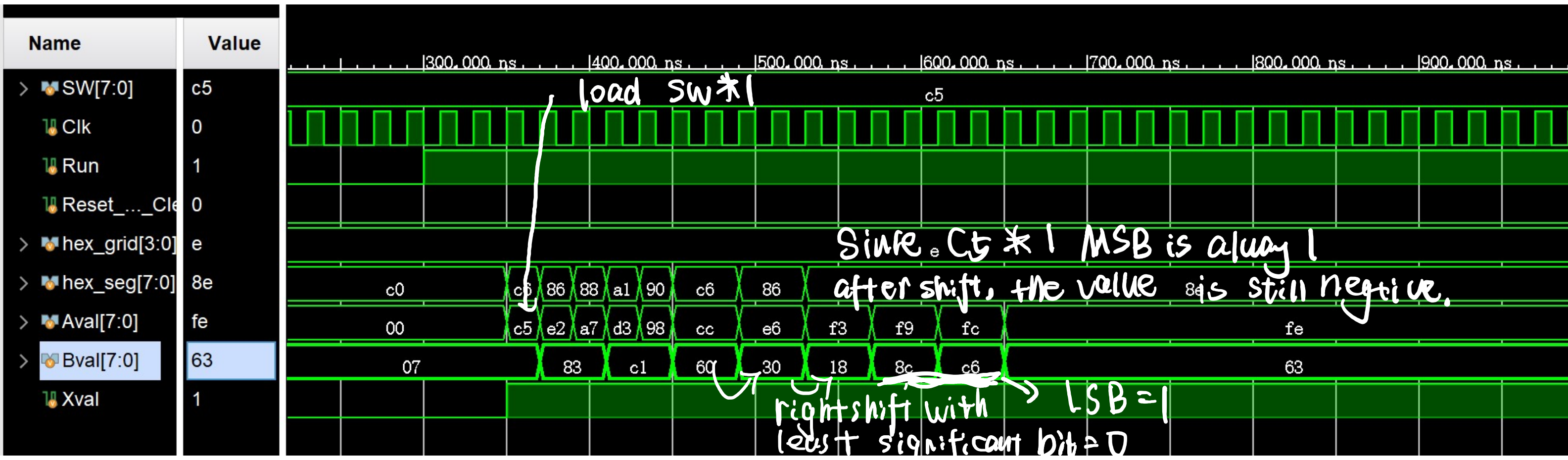
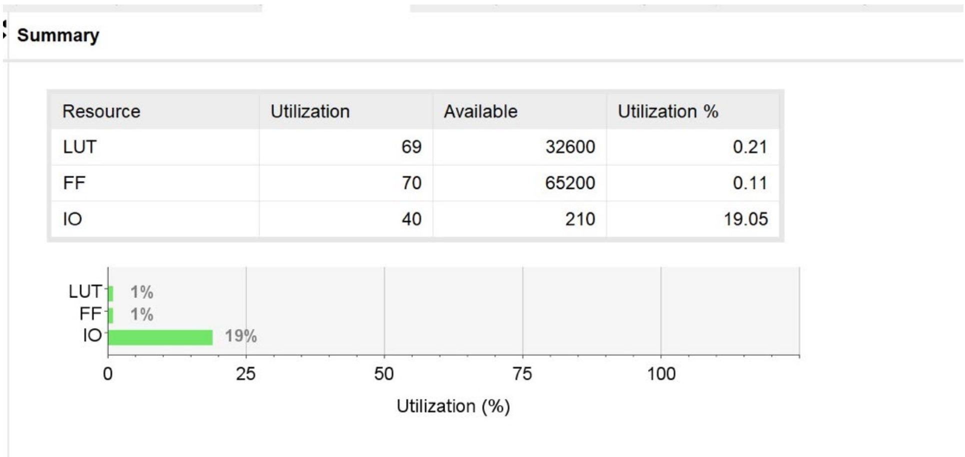


Figure 4.3 (multiplication of 59*-7=-413)

5.Answers to post-lab questions

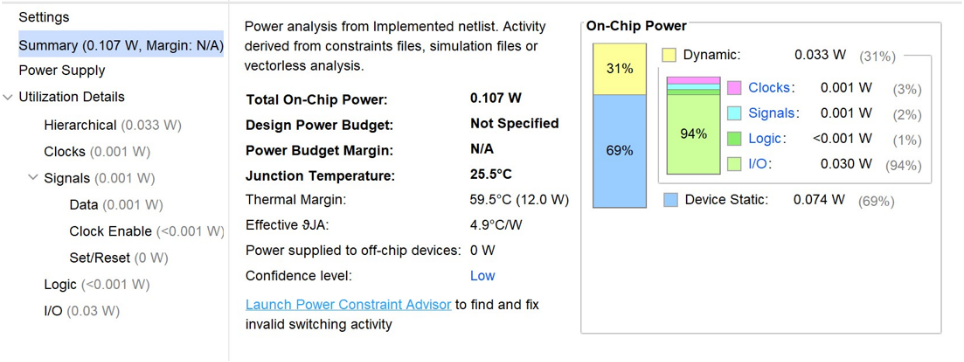
5.1


Summary

Resource	Utilization	Available	Utilization %
LUT	69	32600	0.21
FF	70	65200	0.11
IO	40	210	19.05

Figure 5.1 - Implementation Summary Table

LUT	**69**
DSP	**0**
Memory(BRAM)	**0**
Flip_Flop	**70**
Latches*	**0**



Settings
Summary (0.107 W, Margin: N/A)
Power Supply
Utilization Details
- Hierarchical (0.033 W)
- Clocks (0.001 W)
- Signals (0.001 W)
  - Data (0.001 W)
  - Clock Enable (<0.001 W)
  - Set/Reset (0 W)
- Logic (<0.001 W)
- I/O (0.03 W)

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.107 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.5°C**
Thermal Margin:	59.5°C (12.0 W)
Effective ϑJA:	4.9°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

🟨	Dynamic:	0.033 W	(31%)
🟪	Clocks:	0.001 W	(3%)
🟦	Signals:	0.001 W	(2%)
🟩	Logic:	<0.001 W	(1%)
🟩	I/O:	0.030 W	(94%)
🟦	Device Static:	0.074 W	(69%)

Figure 5.2 - Power Summary

Frequency	**198.846MHZ**
Static Power	**0.074w**
Dynamic Power	**0.033w**
Total Power	**0.107w**

how you might optimize your design to decrease the total gate count and/or to increase maximum frequency by changing your code for the design?

Ans:

decrease the total gate count: in the FSM, we use binary encoding which means we need more logic gates to compare each bit which is costly. So we use one-hot encoding so that each time, we only need to compare one signal bit.

increase maximun frequency: we can use select ripple adder instead of carry ripple adder to increase the speed because we eliminate the delay.

5.2 Make sure your lab report answers at least the following questions:

5.2.1:

x-reg's value is the next state of A_reg. When the computation is done, the value should be stored and wait until the next positive clock cycle and load into A_reg. If we don’t use X_reg, the computation may lost before the shift operation.

X_register get set after the computation and clear when we click clearA_loadB.

5.2.2:

when we use 8-bit adder, the computation result will go wrong if we operate

Subtraction. Ex.(8’b11111110+8’b000000001=8’b11111111) if we use 9 bit adder, the result is –1 but if we use 8 bit adder, the value is 255.

5.2.3:

the success of consecutive multiplications without resetting depends on whether the product of each multiplication operation falls within the valid 8-bit range [-128,127] after truncation. If it does, the consecutive multiplications are expected to work as described.[1]

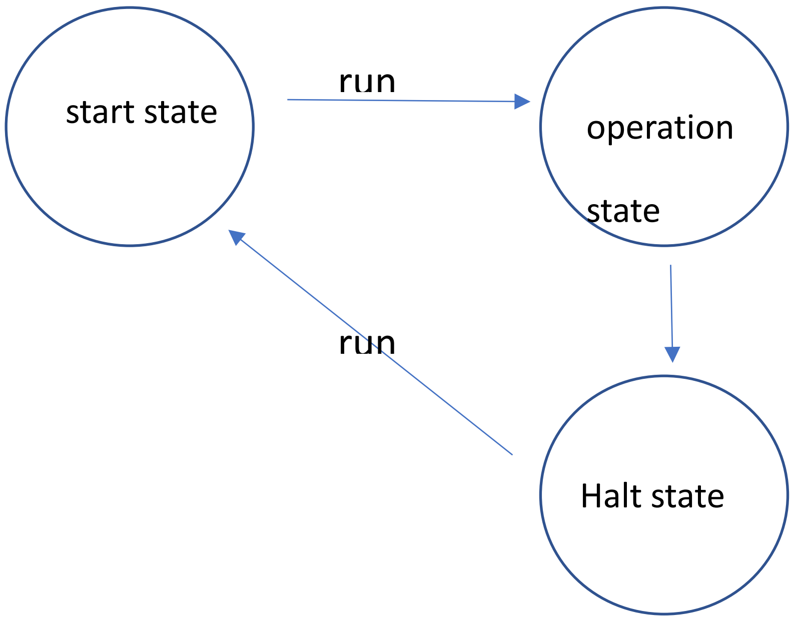
5.2.4:

Disadvantage: Implemented multiplication algorithms can be complex because we need more logic to and different condition detecting whether the M value is 1 and whether we need to add or sub.
Advantage: the operation is faster because it does not need to add all 8 bit’s result together when M is 0.

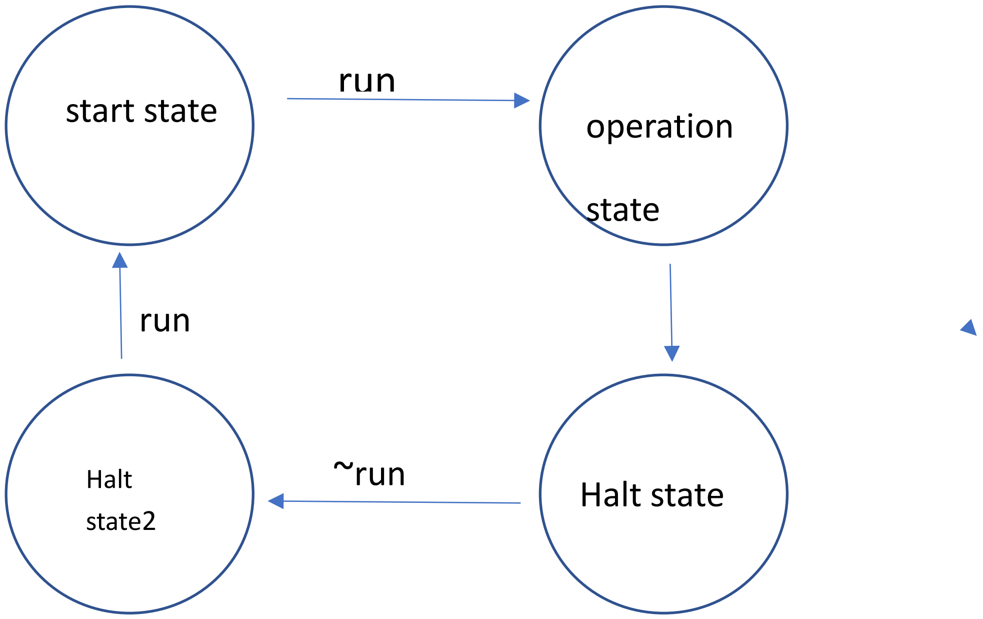
6. Conclusion

6.1 Discuss functionality of your design. If parts of your design did not work, discuss what could be done to fix it.

The simulation is right but when we implement it on board, for example (3*3) when we click run, the correct value (9) will shown on board quickly then changed into wrong value all of a sudden. This indicate that our FSM can’t get into halt state after we click run. We find that the total running time of multiplcation is so small that when we click run, FSM can’t stop.



Therefore we add a new state



Then the state can stop after one mulitiplcation.

6.2

The resources are quite scattered, sometimes in the wiki, and sometimes in the PDFs. We hope that the information can be more integrated.