ECE 385

Spring 2023

Experiment #4

8-Bit Multiplier

Max Ma & Dhruv Kulgod

NL / Friday 9:15 AM - 9:30 AM

Nick Lu

Purpose of Circuit

In this lab, we had to design an 8-bit multiplier in SystemVerilog. The procedure for multiplication is not a repeated addition style, because this becomes very slow for large numbers. Instead, we follow an "add-shift" method similar to the pen-and-paper method of multiplying two numbers. Because we are dealing with binary numbers, if we want to multiply $(001)_2$ and $(010)_2$, we add $0 * (001)_2$, $1 * (0010)_2$ and $0 * (00100)_2$, which equates to $(010)_2$. An important consideration to keep in mind is that these are signed numbers. Thus, to deal with negative multiplication, we subtract the last value instead of adding.

Written Description of Circuit

Because multiplication often yields numbers much larger than the two operands, we store the result of multiplying two 8-bit numbers in a 16-bit register. We have two 8-bit registers, A and B, as well as a 1-bit register X. First, we store the multiplier in register B using the switches. X and A are then initialized to zero.

The multiplicand is loaded on the switches. Depending on the lowest bit value in register B (value M), we decide whether or not to add the value of the switches to register A. The addition of the switches and Register A is sign extended to 9-bits. Register X holds the 9th bit of this sum. This is important, because it also holds the sign of the resultant partial sum. After adding, we shift the whole 17 bit value in XAB to the right in an arithmetic shift (which just means we hold the value in X). We perform this check-add-shift operation 7 times. On the last bit instead of adding, we would perform a subtraction operation instead when M = 1. This is achieved by adding the 2's complement value of the switches.

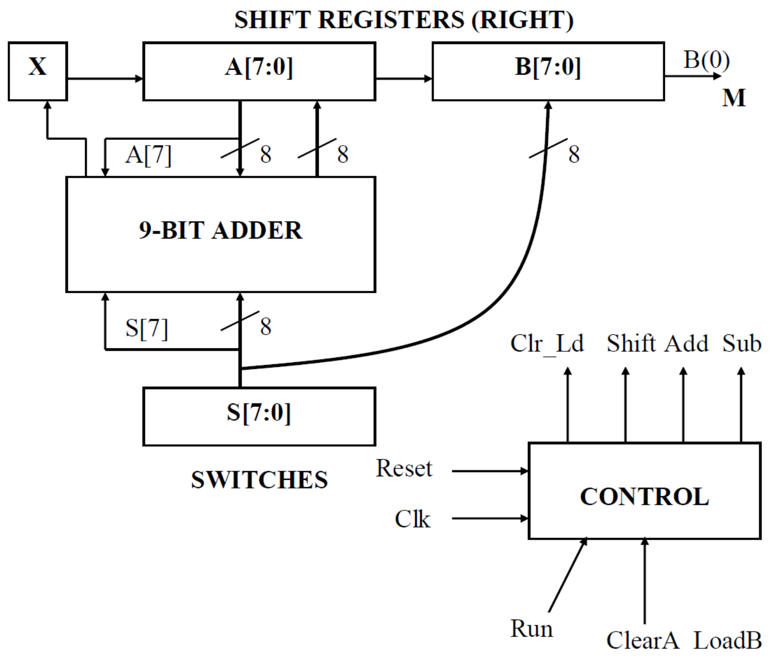

Figure 1: Logical structure of the multiplier

The 9-bit adder we used in this lab was a carry-lookahead adder from the previous lab. This was chosen because of its minimal delay in calculation.

We control the multiplier with the switches as mentioned above, as well as two buttons - Reset_Load_Clear and Run. R_L_C essentially restarts the machine, clearing all registers and loading the switch value into register B. Run can be used to perform recurring operations without clearing.

State Diagrams and Tables

The Moore state machine design used in this experiment is described below in Figures 2 and 3. The design controls the shift, addition, subtraction, register resetting, and register loading operations. This was handled using the appropriate outputs for each state. The state transitions and outputs are defined in Figure 2. A synchronous counter was used to track the number of shifts as shown in Figure 3, which reduced the number of total states in the design.

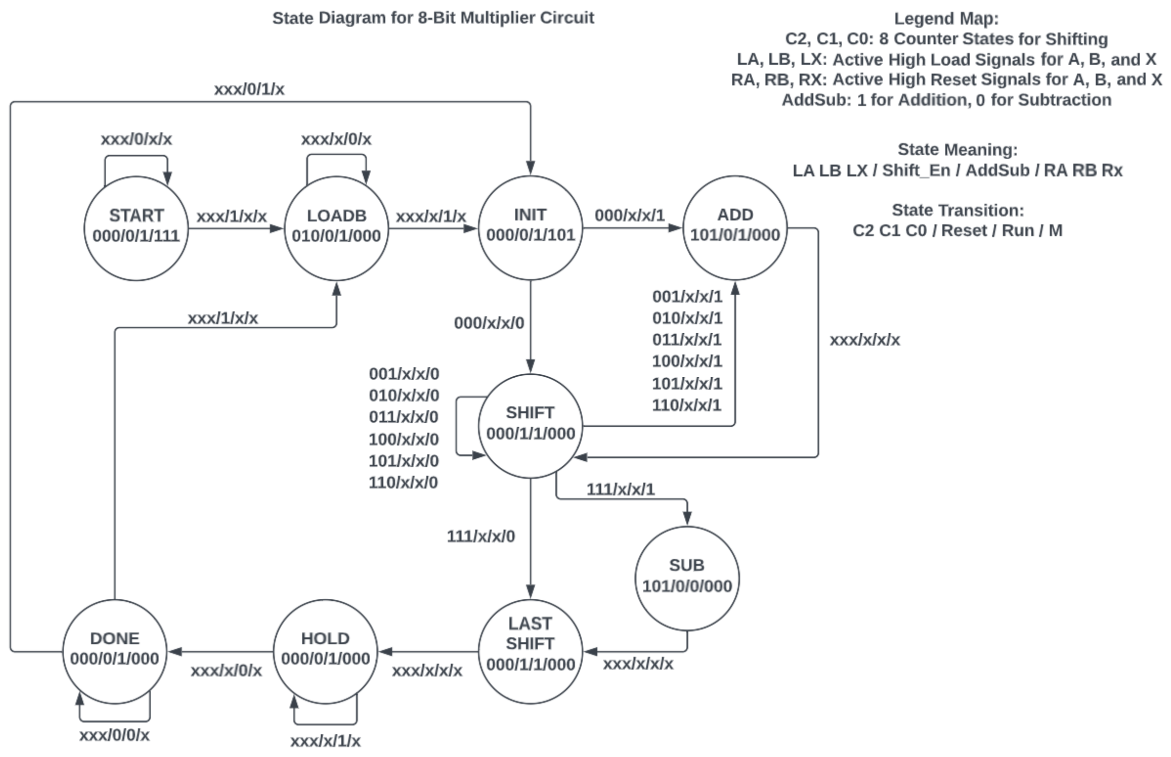

Figure 2: State Diagram for 8-bit Multiplier Circuit

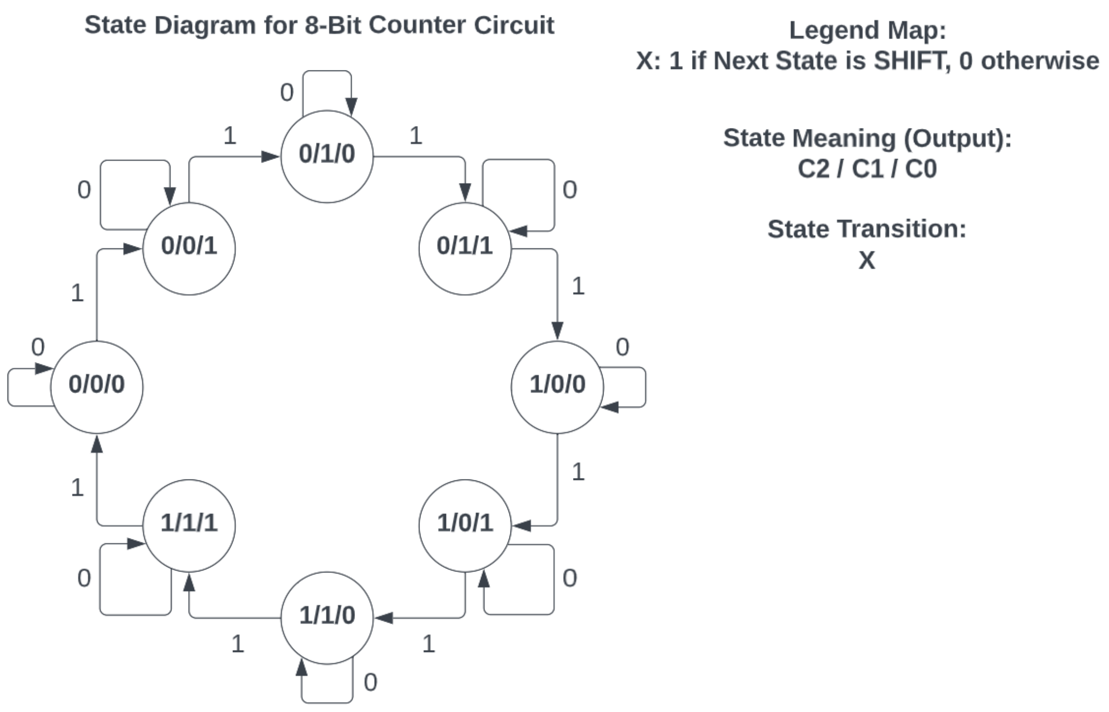

Figure 3: State Diagram for 8-bit Counter Circuit

Answers to Pre-Lab

a. Rework the multiplication example on page 5.2 of the lab manual, as in compute 11000101 * 00000111 in a table like the example. Note that the order of the multiplicand and multiplier are reversed from the example.

Function	X	A	B	M	Comments
Clr A, Load B, Reset	0	0000 0000	0000 0111	1	M=1, so add multiplicand to A
ADD	1	1100 0101	0000 0111	1	Shift XAB by one
SHIFT	1	1110 0010	1000 0011	1	ADD next
ADD	1	1010 0111	1000 0011	1	Shift XAB by one
SHIFT	1	1101 0011	1100 0001	1	ADD next
ADD	1	1001 1000	1100 0001	1	Shift XAB by one
SHIFT	1	1100 1100	0110 0000	0	M=0. Do not add. Just shift.

SHIFT	1	1110 0110	0011 0000	0	M=0. Do not add. Just shift.
SHIFT	1	1111 0011	0001 1000	0	M=0. Do not add. Just shift.
SHIFT	1	1111 1001	1000 1100	0	M=0. Do not add. Just shift.
SHIFT	1	1111 1100	1100 0110	0	M=0. Do not add. Just shift.
SHIFT	1	1111 1110	0110 0011	0	8th shift done. Do not shift anymore. Answer is AB.

Answers to Post-Lab

1. Come up with a few ideas on 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.
    a. Instead of using a synchronous counter to track the eight shifts, we could use a 3-bit register with an incrementer. This could reduce the gates needed to calculate the next state logic.
    b. We could try to reduce the number of states in our overall design. For example, removing the START state in Figure 2 would bring the state machine to 8 states. This would reduce the design by 1 flip-flop since we can represent 8 states using 3 flip-flops.
    c. Although our design already has a high maximum frequency, we could try to find critical paths and try to bypass or reduce these slownesses. For instance, if the path between the serial output of Register A and serial input of Register B is a critical path, we could try to change the design to incorporate Registers A and B as a single 16-bit register.

Design Resources and Statistics Table for 8-Bit Multiplier	
LUT	123
DSP	0
Memory (BRAM)	0
Flip-Flop	61
Frequency	82.24 MHz

Static Power	89.97 mW
Dynamic Power	1.89 mW
Total Power	104.86 mW

Figure 4: Design Resources and Statistics Table for 8-Bit Multiplier

2. What is the purpose of the X register? When does the X register get set/cleared?
    a. The purpose of the X register is to store the signed bit of the addition between A and the Switches. This is then shifted into Register A during a shift operation. The X register gets set after the 9-bit addition of the Switches and Register A.
3. What would happen if you used the carry out of an 8-bit adder instead of output of a 9-bit adder for X?
    a. The carry out would provide a problem when the 8-bit adder overflows. Adding two large, non-negative numbers could result in an overflow; however, this would store the carry out bit in X, indicating that the result is a negative number. This would, of course, provide incorrect results for any possible overflow conditions.
4. What are the limitations of continuous multiplications? Under what circumstances will the implemented algorithm fail?
    a. Continuous multiplications quickly explode to numbers of large magnitude. For this experiment, we are able to output 16-bit 2's complement results. Thus, valid results can only take values between −32,768 to 32,767. The algorithm could fail in cases of overflow that surpass this range, which is inevitable when doing continuous multiplications.
5. What are the advantages (and disadvantages?) of the implemented multiplication algorithm over the pencil-and-paper method discussed in the introduction?
    a. The implementation of the multiplication algorithm is advantageous because the operation can be performed using only two 8-bit registers and a simple control unit. In the pencil-and-paper method, multiple computations of various bit lengths must be stored as well as added/subtracted at the end. This would increase the required number of flip-flops and the complexity of the design substantially.
    b. On the other hand, the multiplication algorithm will be slower due to the additional shifts on top of the additions.

FPGA Labs: Block Diagram

The block diagram for this experiment, as shown in Figure 5, was composed of synchronizer, control unit, adder, register, and hex driver modules.

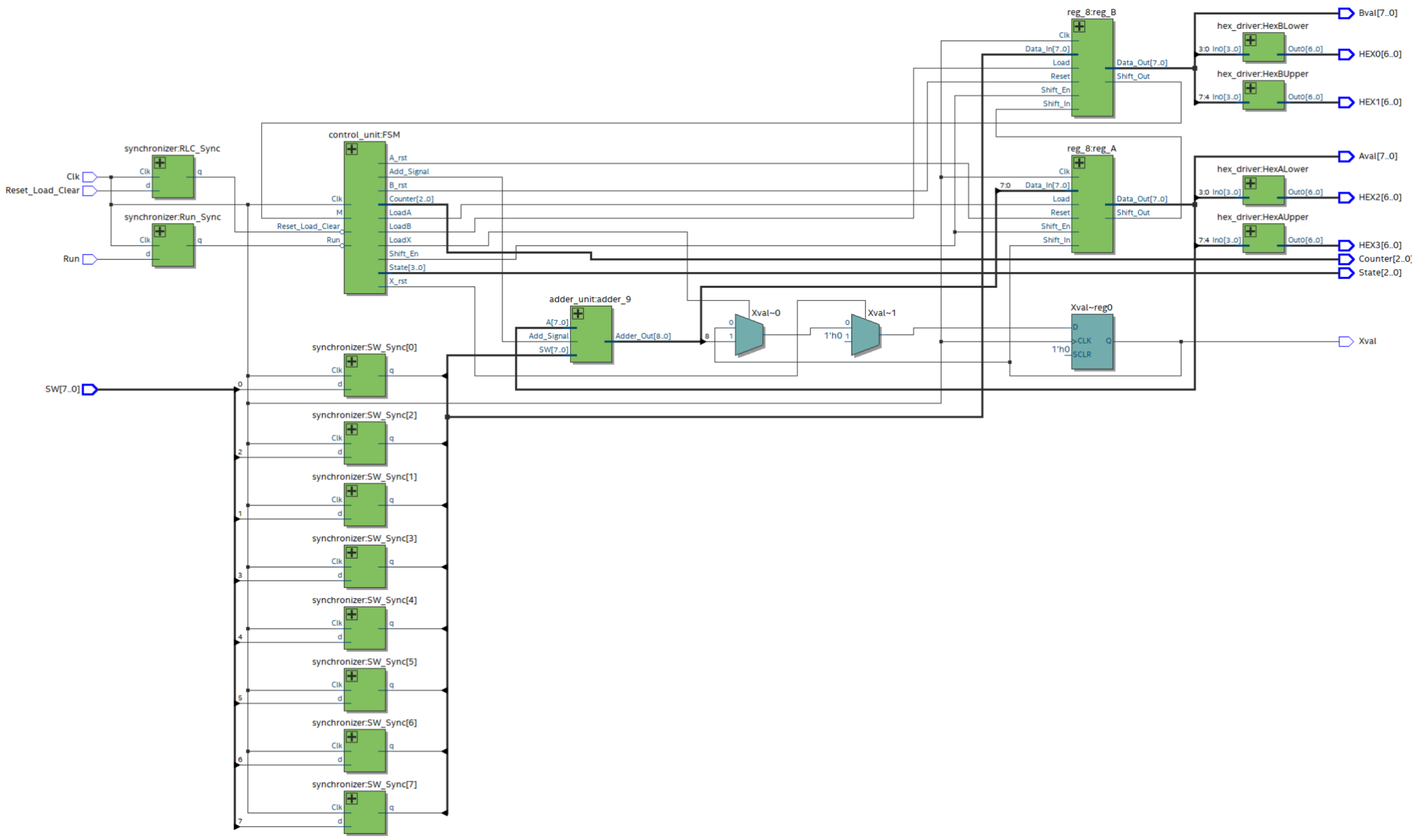

Figure 5: FPGA Block Diagram

FPGA Labs: Design Module Information

You will need to write the purpose and operation for each SystemVerilog module. For each module, you will need to list the inputs, outputs, and what the module does (i.e., is it a register? Control unit? Computation unit? What does it do to the inputs? What are the role of the outputs in the whole circuit?)
synchronizer, control unit, adder, register, and hex driver

Control Unit:

Inputs: Clk, Reset_Load_Clear, Run, M

Outputs: LoadA, LoadB, LoadX, Shift_En, Add_Signal, A_rst, B_rst, X_rst, [3:0] State, [2:0] Counter

Description: Our state machine to decide what operation must be done, and control the number of shift-adds. Rather than create a long FSM, we implemented a counter to count the number of shifts we had done. We initialize in the Reset_Load_Clear state (LOAD_B), after which we add and shift depending on the counter values and the lowest bit in register B. Finally, once the operation is done, we implement a WAIT state that waits till the Run button is no longer pressed. This prevents unwanted repeated multiplications. Once the operation is done, we can either run again or reset.

Synchronizer:

Inputs: Clk, d

Outputs: q

Description: Synchronizes our physical buttons and switches i.e. ensures their values are only changed at each rising edge of the clock. This helps mitigate timing issues in the design.

Reg_8:

Inputs: Clk, Reset, Shift_In, Load, Shift_En, [7:0] Data_In

Outputs: Shift_Out, [7:0] Data_Out

Description: Shift register module. Allows us to easily load, reset and shift bits in an 8-bit register. The Load_En and Shift_En signals tell us what operation to perform on the register. We are always able to read the value of the register as well.

HexDriver:

Inputs: [3:0] In0

Outputs: [6:0] Out0

Description: This module acts as a translator between binary to hexadecimal. A set of four binary bits are taken as input and linked to their corresponding pattern on the 7-Segment Hex Display using a case statement.

Adder_Unit:

Inputs: Add_Signal, [7:0] A, [7:0] SW

Outputs: [8:0] Adder_Out

Description: 9 bit adder unit implemented as a carry-lookahead for performance. 2x4 hierarchical design plus an additional bit. Add_Signal is used to decide whether to simply add the switch value to A or to add the 2’s complement of the switches (subtraction). Output stored in registers XA.

FPGA Labs: Annotated Simulation Waveform (Pre-Lab)

The annotated simulation waveform shown in Figure 6 demonstrates successful computation of four test cases where operands have signs (+*+), (+*-), (-*+) and (-*-).

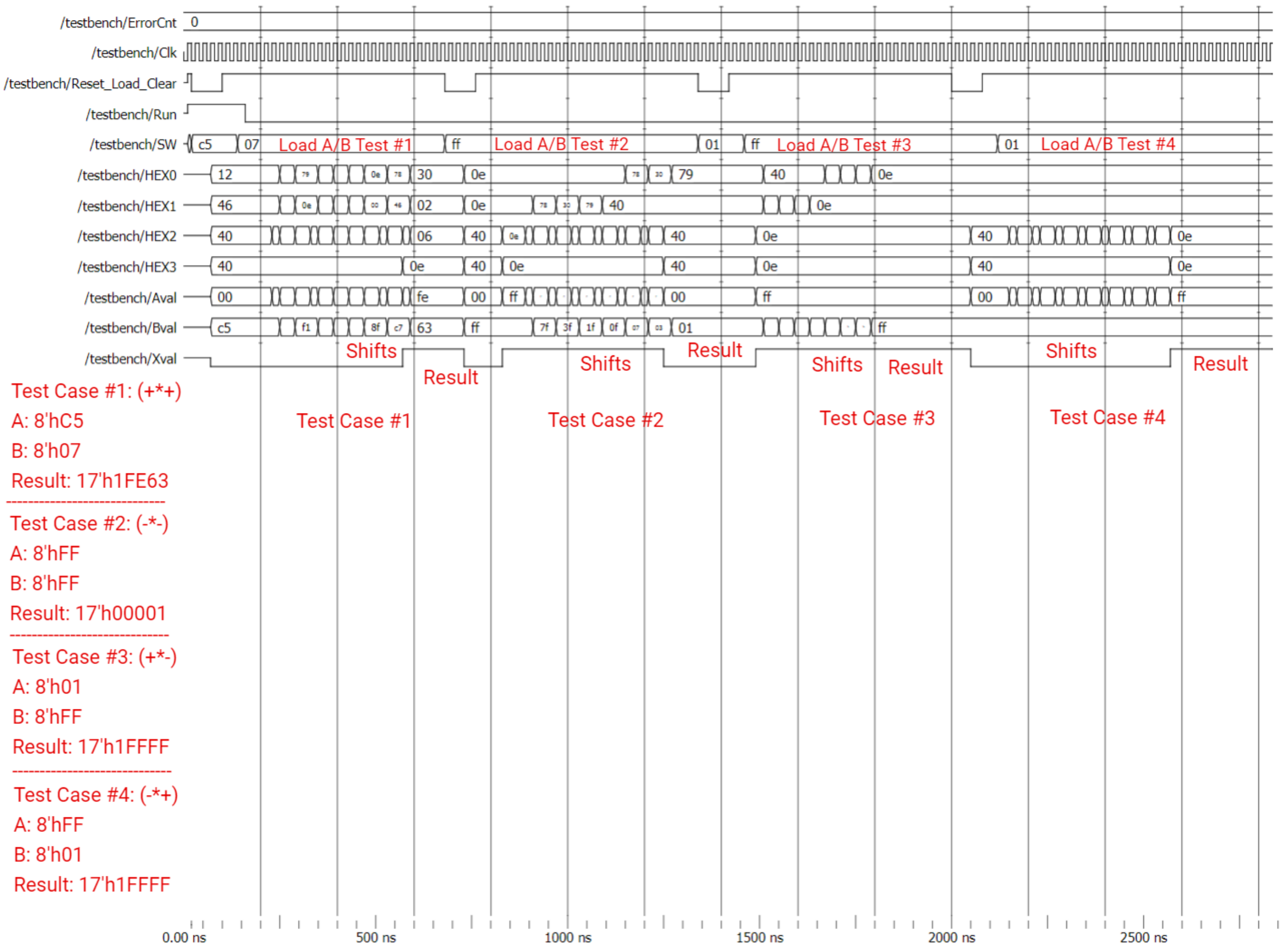

Figure 6: Annotated Simulation Waveform

Conclusions

We learnt about the difficulty in designing robust FSMs without testing. Our initial design gave us a good overview of what we had to achieve, but there were many bugs we had to fix. We found ourselves adding ill-defined states at the end of our design as a band-aid measure. While this technique worked, we couldn't help but think there is a better approach to FSM design out there. We hope to explore a more structured approach the next time we are asked to design an FSM.