An 8-bit Multiplier in SystemVerilog

Hongbo Zheng, Yuhao Yuan

Summer 2021

ECE 385

Section AB1

TA: Hanfei Wang

Introduction

The purpose of this lab is to design an 8-bit Multiplier which is able to perform multiplication of two 8-bit 2's complement numbers using SystemVerilog, and then test the SystemVerilog design in both ModelSim with testbench and on the DE10-Lite FPGA board with switches.

Written Description of the Circuit

Summary of operation

The 8-bit Multiplier uses a add-shift algorithm to compute the product of two 8-bit 2's complement numbers. The design utilizes two 8-bit right shift registers, register A and register B, which are connected together in sequence (8-bit register A -> 8-bit register B) to store the final result in a total of 16-bit registers in order to prevent overflow. Moreover, there is a 1-bit register, register X, which is used to keep track of the sign bit.

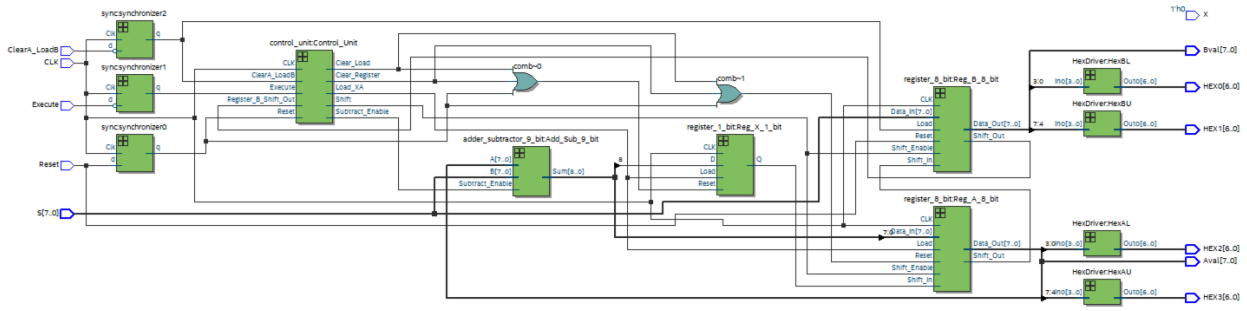
At the beginning of the operation, the Multiplier is set through the switches and loaded into register B by pressing the ClearA_LoadB pushbutton. Then, the switches are changed to the value of Multiplicand and kept during the entire operation. After that, the circuit is ready to compute the product of two 8-bit 2's complement numbers. Once the Run pushbutton is clicked, the circuit is going to through 8 consecutive add-shift operations which are controlled by the least significant bit of the multiplier stored in register B.

When B[0] = 1, the value in register A and the value from switches are both sign-extended to 9 bits, and then added together. The result of the 9-bit adder is then stored into register XA. Then, the entire 17-bit value is arithmetically shifted right by 1 bit. However, when B[0] = 0, the ADD state is skipped, and the entire 17-bit value is arithmetically shifted right by 1 bit.

In addition, there is a special case that needs to be taken care of. If the Multiplier stored in register B is a negative number, its most significant bit (8^{th} bit) has a binary value of '1'. Then both Add state and Shift state have to perform because B[0] = 1 at the 8^{th} add-shift operation. However, the add-shift is replaced by subtract-shift at the last operation in order to make sure register A still has the correct sign bit after performing the last right-shift. Therefore, at the 8^{th} add-shift operation, if B[0] = 1, subtract-shift operation is performed instead of add-shift operation to ensure the final result has the correct sign bit.

Block Diagrams and Module Descriptions

High-Level Block Diagram



Module Description

Module: Multiplier_8_bit.sv

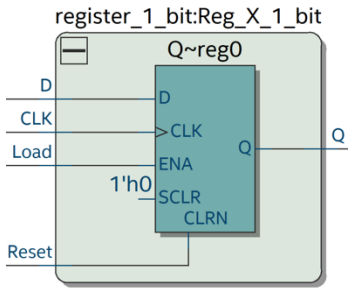
Input: CLK, Reset, ClearA_LoadB, Execute, [7:0] S,

Output: [7:0] Aval, [7:0] Bval, [6:0] HEX0, [6:0] HEX1, [6:0] HEX2, [6:0] HEX3

Description: This component contains 9-bit Adder/Subtractor, two 8-bit registers, 1-bit register, Control Unit, HexDrivers, and Synchronizers. ClearA_LoadB signal and Execute signal are changed from active-high to active-low because they are all connected to synchronizers in order to change from asynchronous signals to synchronous signals. The 8-bit input [7:0] S is the input to the 8-bit Register B and the 9-bit Adder/Subtractor controlled by ClearA_LoadB. When ClearA_LoadB is low, the register A is cleared (store the binary value 0), and the 8-bit input [7:0] S is loaded into register B. When Reset is high, the [7:0] Aval and [7:0] Bval are reset to 8-bit binary '0', and [6:0] AhexL, [6:0] AhexU, [6:0] BhexL, and [6:0] BhexU are set to 7-bit binary value which displays 0 on the HEXs. When Execute is low, the 8-bit Multiplier start to compute the product of two 8-bit 2's complement numbers. The final result of the multiplication will store in register A and register B, and it will display either through [7:0] Aval and [7:0] Bval in the testbench or the decoded 7-bit values [6:0] HEX0, [6:0] HEX1, [6:0] HEX2, and [6:0] HEX3 on the 4 HEX units.

Purpose: This module is the high-level block of the 8-bit Multiplier which contains 9-bit Adder/Subtractor, two 8-bit registers, 1-bit register, Control Unit, HexDrivers, and Synchronizers.

1-bit Register X Block Diagram



Module Description

Module: 1-bit_Register.sv

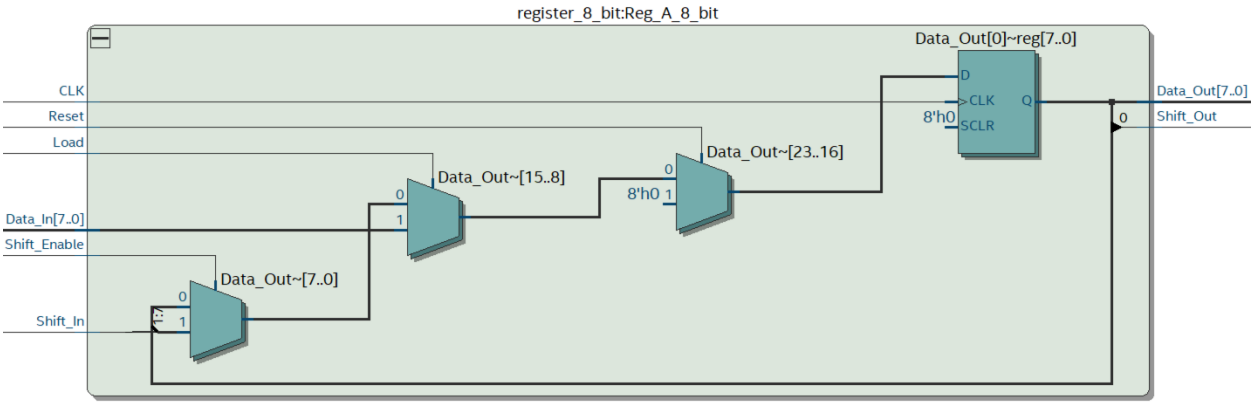
Input: CLK, Reset, Load, D

Output: Q

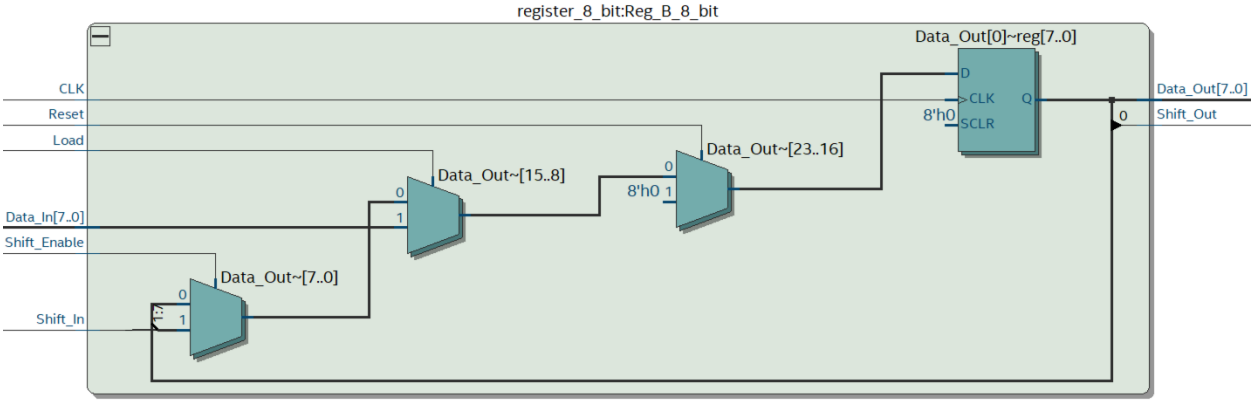
Description: This component is a positive-edge triggered DFF with synchronous Reset and synchronous Load. When Reset is high, the DFF is cleared (store the binary value of 0) on the positive edge of CLK. When Load is high, data is loaded from D into the DFF on the positive edge of CLK.

Purpose: This module is used to store the sign bit (most significant bit) of the result of the 9-bit Adder/Subtractor in the 8-bit Multiplier circuit.

8-bit Register A Block Diagram



8-bit Register B Block Diagram



Module Description

Module: 8-bit_Register.sv

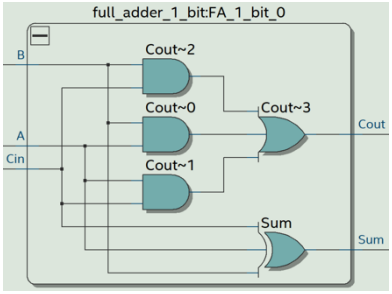
Input: CLK, Reset, Load, Shift_Enable, Shift_In, [7:0] Data_In

Output: Shift_Out, [7:0] Data_Out

Description: This is a positive-edge triggered 8-bit right shift register with synchronous reset and synchronous load. When Reset is high, all the registers are cleared (store the binary value 0) on the positive edge of CLK. When Load is high, data is loaded from [7:0] Data_In into the registers on the positive edge of CLK. When Shift_Enable is high, the values in the registers start to shift right by 1 bit on the positive edge of CLK, Shift_In updates the value of the left most register (register 7), and Shift_Out is the output of the right most register (register 0). Data_Out is the 8-bit output of all the register units (register 0 – register 7).

Purpose: This module is not only used to store the 8-bit result of the 9-bit Adder/Subtractor, but also used to store the value of Multiplier B in the 8-bit Multiplier Circuit.

1-bit Full Adder Block Diagram



Module Description

Module: 1-bit_Full_Adder.sv

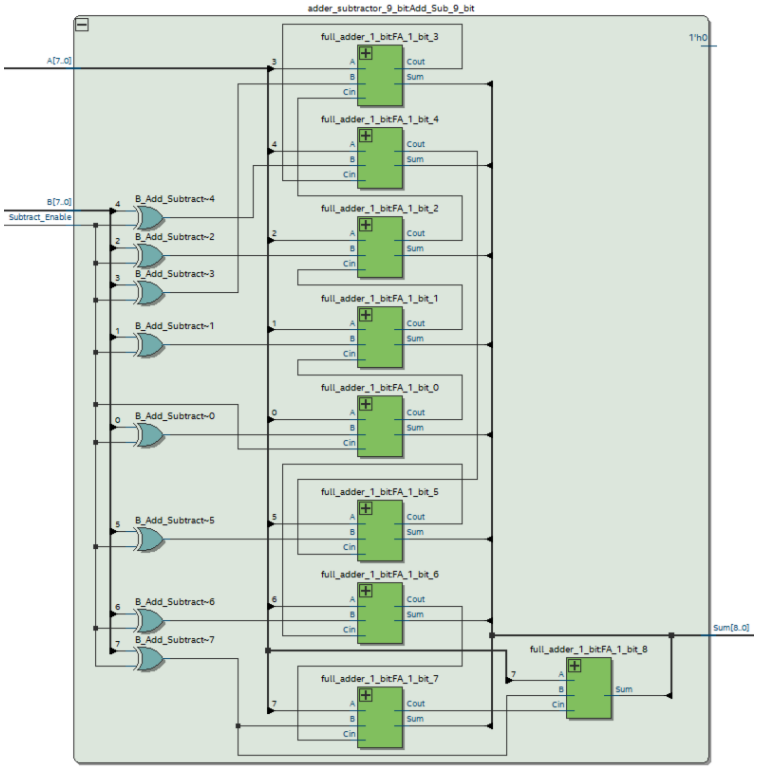
Input: A, B, Cin

Output: Sum, Cout

Description: This component is a 1-bit full-adder which add the three 1-bit inputs: A, B, Cin, and produce two 1-bit output: Sum and Cout. The boolean expression of Sum is $Sum = A \oplus B \oplus C_{in}$, and the boolean expression of Cout is $Sum = (A \cdot B) + (A \cdot C_{in}) + (B \cdot C_{in})$.

Purpose: This module is an 1-bit full-adder which is used as the fundamental building block of the 9-bit Adder/Subtractor in the 8-bit Multiplier circuit.

9-bit Adder/Subtractor Block Diagram



Module Description

Module: 9-bit_Adder_Subtractor.sv

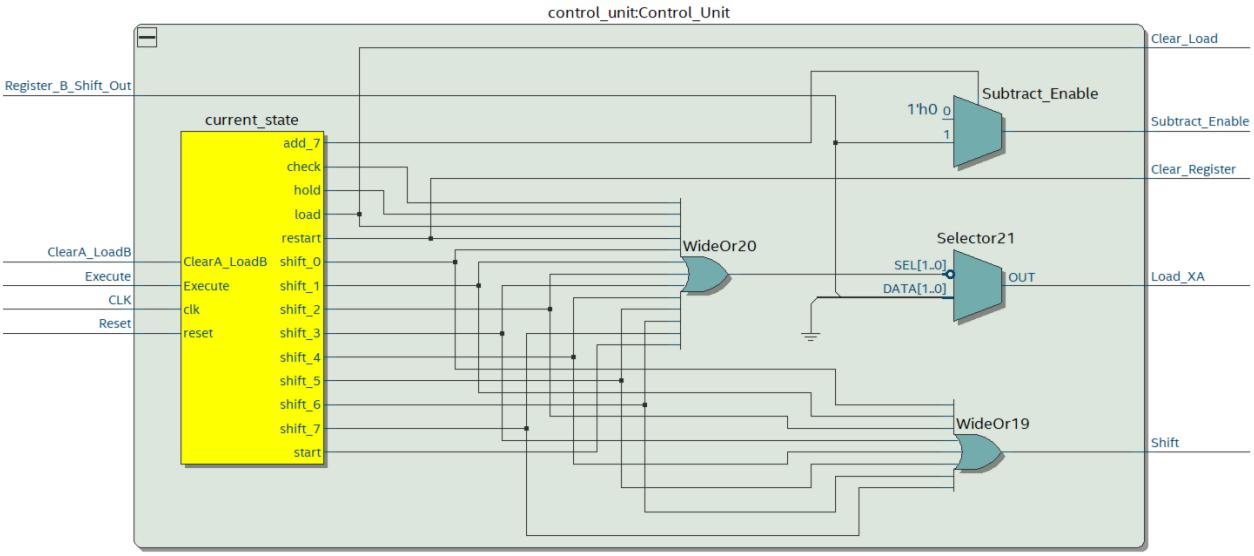
Input: [7:0] A, [7:0] B, Subtract_Enable,

Output: [8:0] Sum, Cout

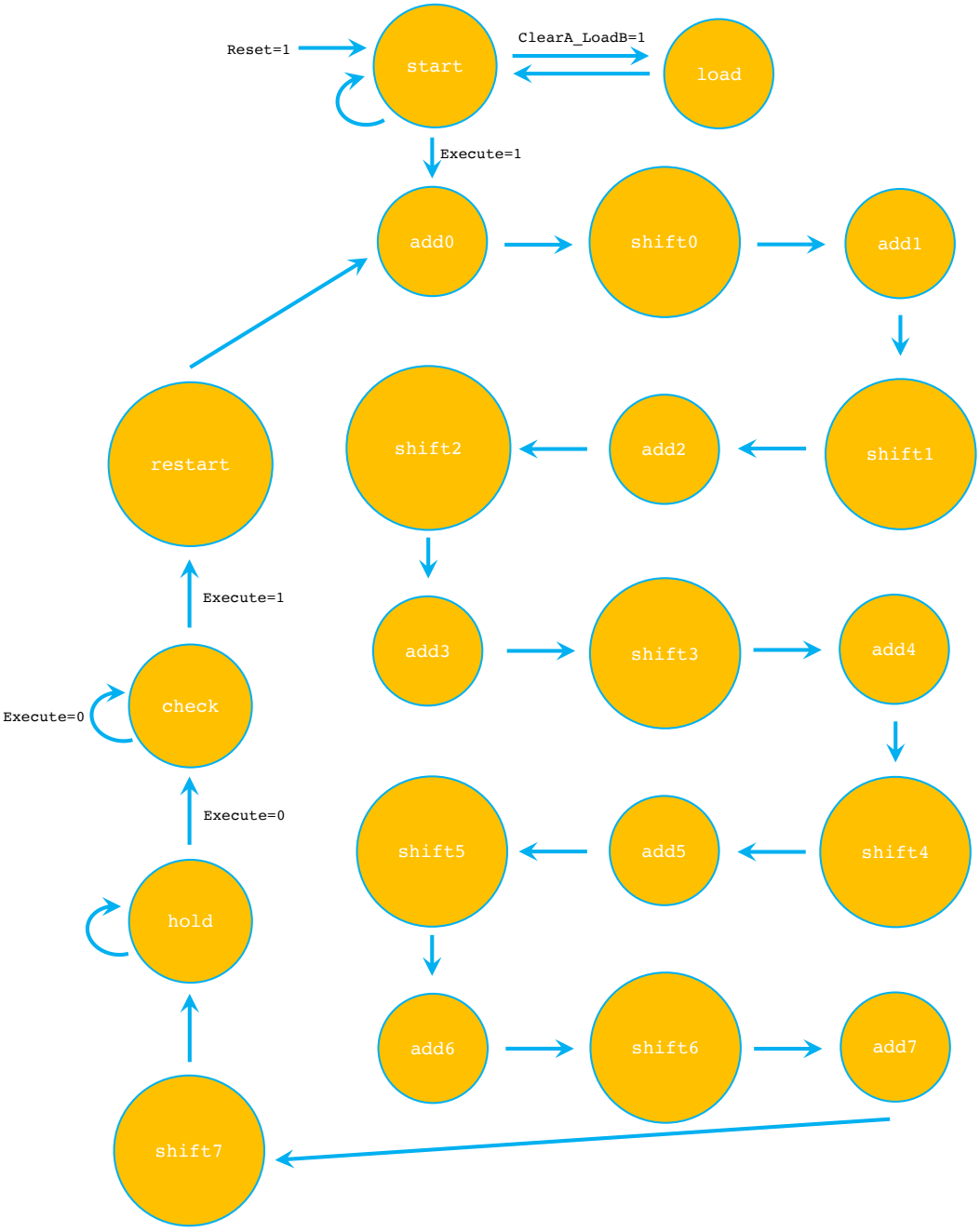
Description: This component is a 9-bit Adder/Subtractor which can switch between 2 modes: Add or Subtract based on the Subtract_Enable signal. When Subtract_Enable signal is low, it performs as a normal 9-bit ripple-carry adder. When Subtract_Enable is high, it performs as a 9-bit subtractor. The 8-bit input A and 8-bit input B are sign-extended to 9 bits. The Subtract-Enable is connected to Cin of the adder, and the 9-bit input A is directly fed into one input of the adder. However, the 9-bit input B is first XOR with Subtract_Enable and then fed into the other input of the adder. This is because the input B can be inverted and added by 1 which makes it a negative number while keeping its original magnitude when Subtract_Enable is high. Finally, the 9-bit Sum gives the result of the addition or subtraction, and the Cout is unused.

Purpose: This module is a 9-bit Adder/Subtractor which is used to add/subtract the value stored in register A with the value from the switches.

Control Unit Block Diagram



FSM Diagram


Reset=1
start
ClearA_LoadB=1
load
Execute=1
add0
shift0
add1
restart
shift2
add2
shift1
Execute=1
add3
shift3
add4
Execute=0
check
Execute=0
shift5
add5
shift4
hold
add6
shift6
add7
shift7

Module Description

Module: Control_Unit.sv

Input: CLK, Reset, ClearA_LoadB, Execute, Register_B_Shift_Out

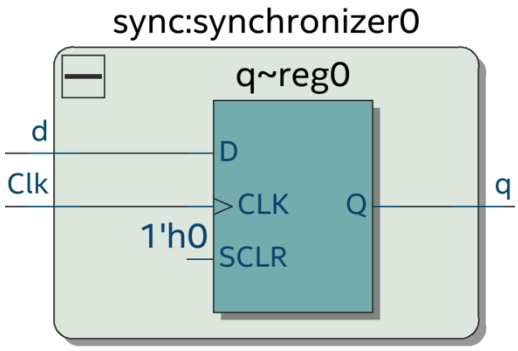
Output: Clear_Load, Shift, LoadXA, Subtract_Enable, Clear_Register

Description: This module contains both combinational logics and sequential logics. When Reset is high, the output Clear_Register will be high which clears all the registers: register X, register A, and register B. When ClearA_LoadB is high, the 8-bit [7:0] S will load into register B, and the output Clear_Load will be high which will clear register X and register A. When Execute is high, the output LoadXA will be high only if the input Register_B_Shift_Out is high in ADD state, and the output Shift will be high only in SHIFT state. If the state machine goes into the last ADD state, the output Subtract_Enable will be high in order to make the last operation a subtraction which ensures that the final result has the correct sign.

The Execute signal and ClearA_LoadB signal are finally modified to be active-low signals after connecting to the synchronizers.

Purpose: This module is the control unit of the 8-bit Multiplier. The control unit makes sure the other modules receive the correct control signals when the circuit goes through start state, load state, 8 consecutive add-shift states based on the least significant bit of the multiplier stored in register B, hold state, check state, and restart state. The last 3 states are used to perform consecutive multiplications (the result may overflow).

Synchronizer Block Diagram



Module Description

Module: Synchronizers.sv

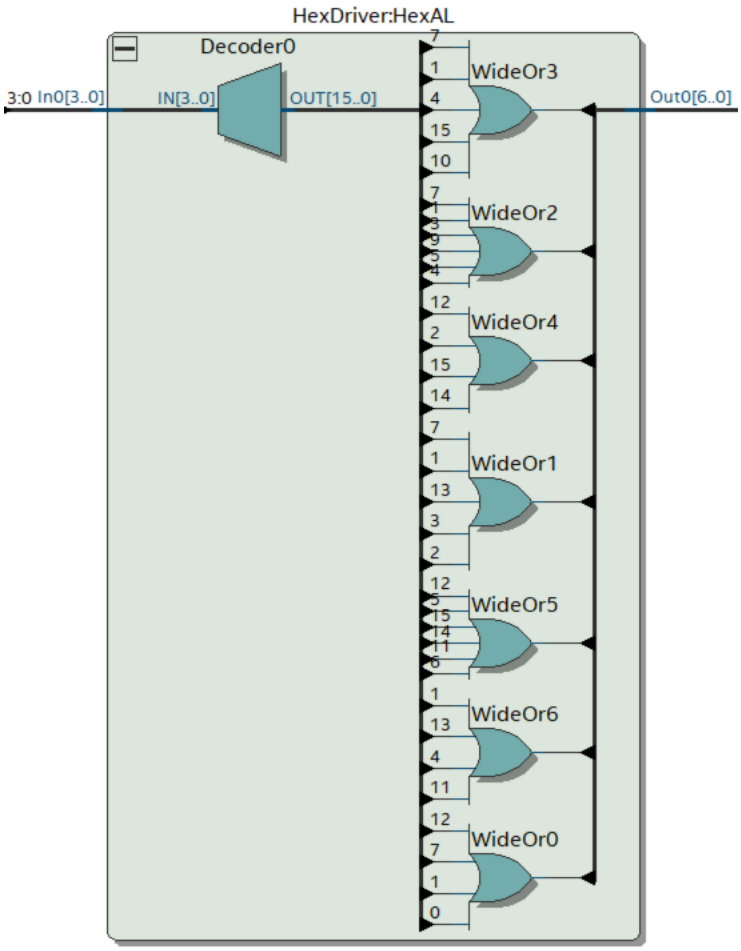
Input: Clk, d

Output: q

Description: This component contains a positive-edge triggered DFF.

Purpose: This module is used to change the asynchronous signals to synchronous signals which are only triggered on the positive edge of Clk in order to eliminate the meta-stability problem.

HexDriver Block Diagram



Module Description

Module: HexDriver.sv

Input: [3:0] In0

Output: [6:0] Out0

Description: This component contains a decoder which decodes the each 4-bit binary value to its corresponding 7-bit binary control value of LED segments. The input [3:0] In0 is translate to 7-bit control value of LED segments [6:0] Out0 which is capable of displaying the hexadecimal letter (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F) on the HEX units.

Purpose: This module is used as 7-Segment Display which can show the 4-bit hexadecimal (2-bit hexadecimal value representing 8-bit binary value in register A and 2-it hexadecimal value representing 8-bit binary value in register B) values to the user on the 4 HEX units.

Pre-Lab Question

Multiplication Example

$8'b\ 1100\ 0101 \times 8'b\ 0000\ 0111 = -59 \times 7$ (in Decimal)

Function	X	A	B	M	Comments for Next Step
ClearA_LoadB	0	0000 0000	0000 0111	1	M=1 Add S to A
ADD1	1	1100 0101	0000 0111	1	Shift XAB right by 1 bit after ADD complete
SHIFT1	1	1110 0010	1000 0011	1	M=1 Add S to A
ADD2	1	1010 0111	1000 0011	1	Shift XAB right by 1 bit after ADD complete
SHIFT2	1	1101 0011	1100 0001	1	M=1 Add S to A
ADD3	1	1001 1000	1100 0001	1	Shift XAB right by 1 bit after ADD complete
SHIFT3	1	1100 1100	0110 0000	1	M=0 Do not add S to A Shift XAB
SHIFT4	1	1110 0110	0011 0000	0	M=0 Do not add S to A Shift XAB
SHIFT5	1	1111 0011	0001 1000	0	M=0 Do not add S to A Shift XAB
SHIFT6	1	1111 1001	1000 1100	0	M=0 Do not add S to A Shift XAB
SHIFT7	1	1111 1100	1100 0110	0	M=0 Do not add S to A Shift XAB
SHIFT8	1	1111 1110	0110 0011	1	8^{th} shift done Stop 16-bit Product in AB

X ----- 1-bit binary value stored in Register X

A ----- 8-bit binary value stored in Register A

B ----- 8-bit binary value stored in Register B

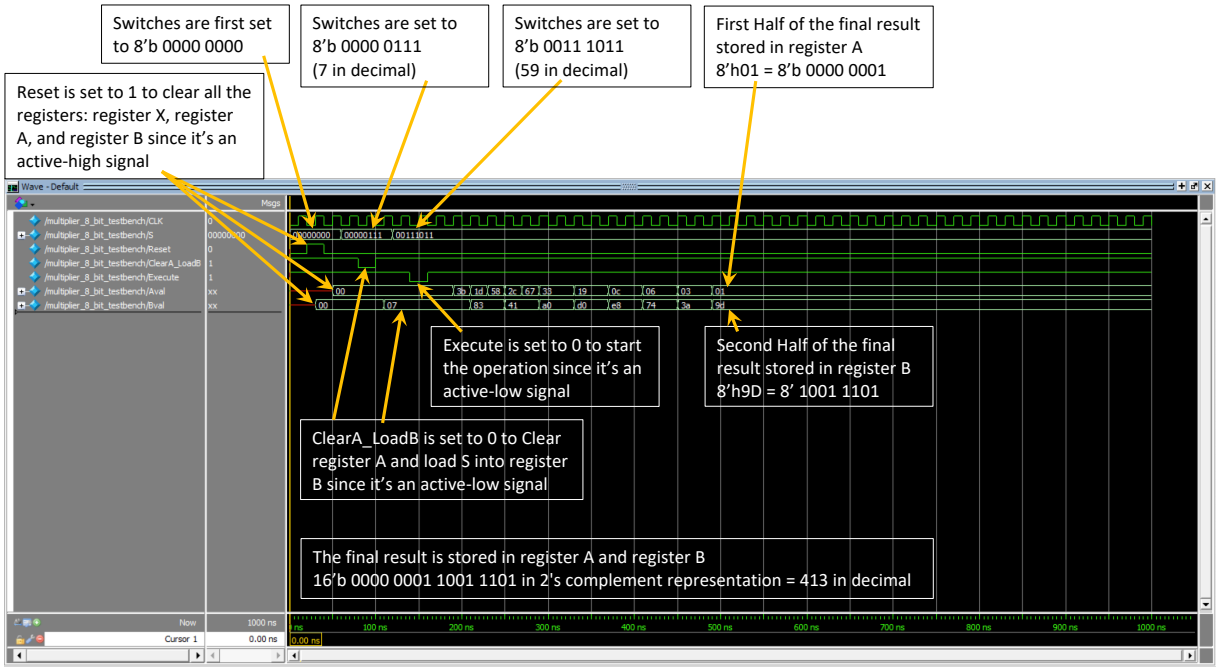
M ----- 1-bit binary value from the least significant bit of Register B (B[0])

S ----- 8-bit binary value set by Switches SW[7:0]

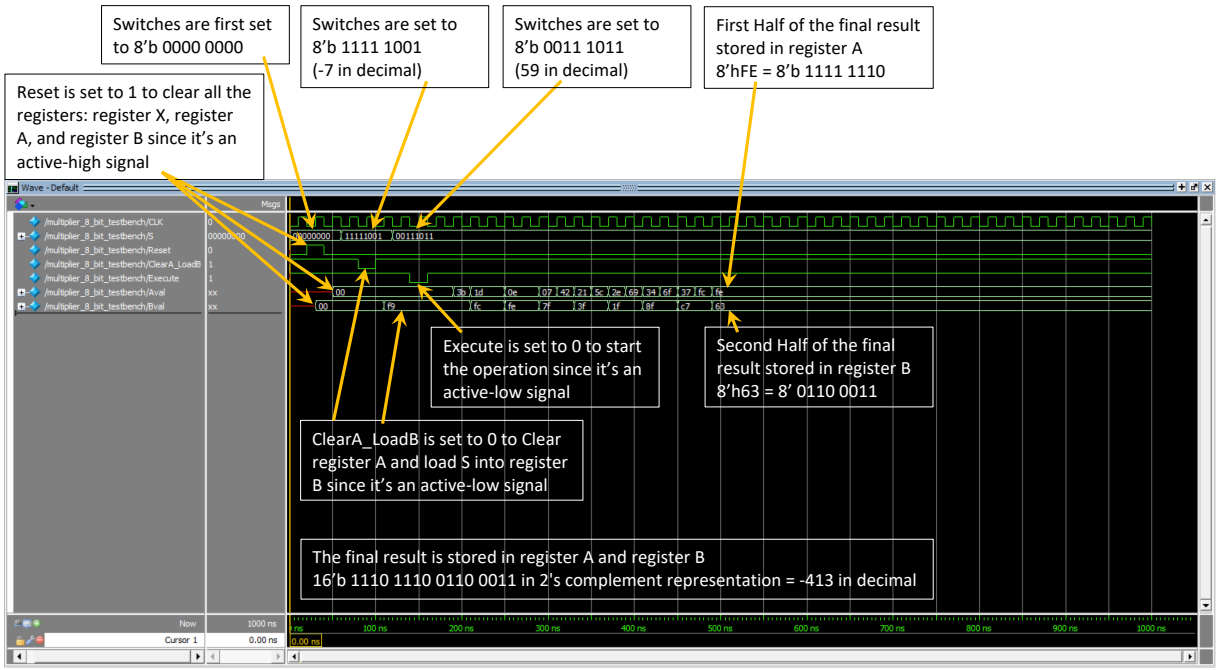
S is kept with binary value 8'b1100 0101 (-59 in decimal) during the entire operation

Quartus ModelSim Waveform

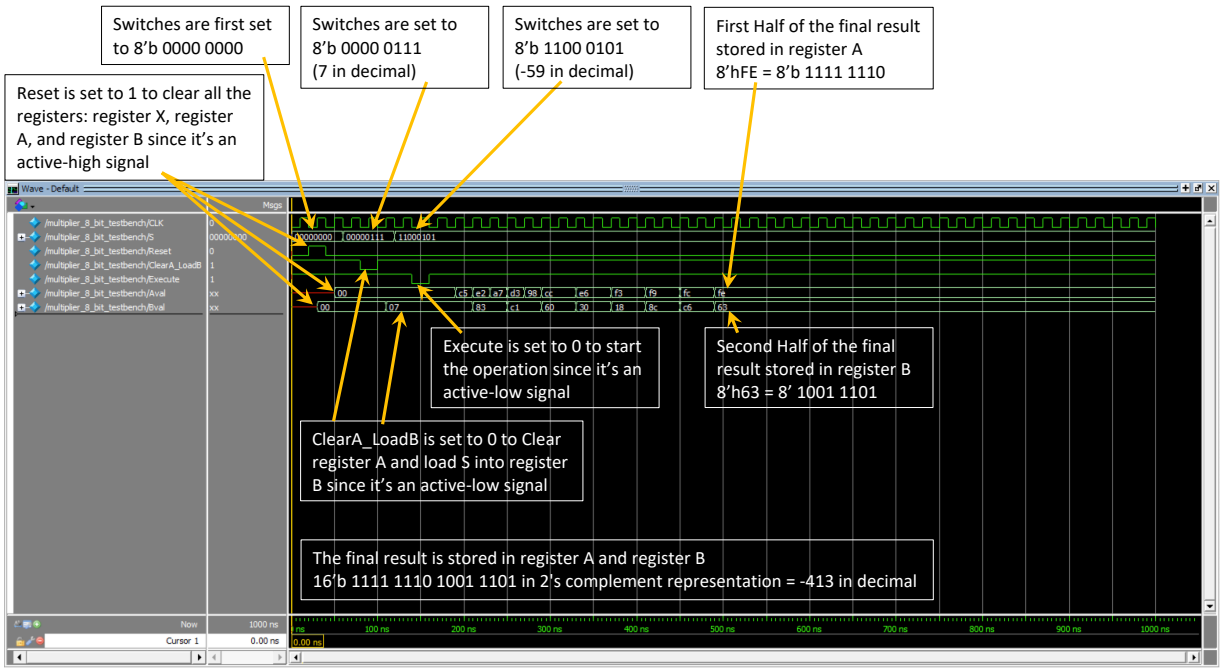
8′b 0000 0111 × 8′b 0011 1011 = 7 × 59 (in Decimal)



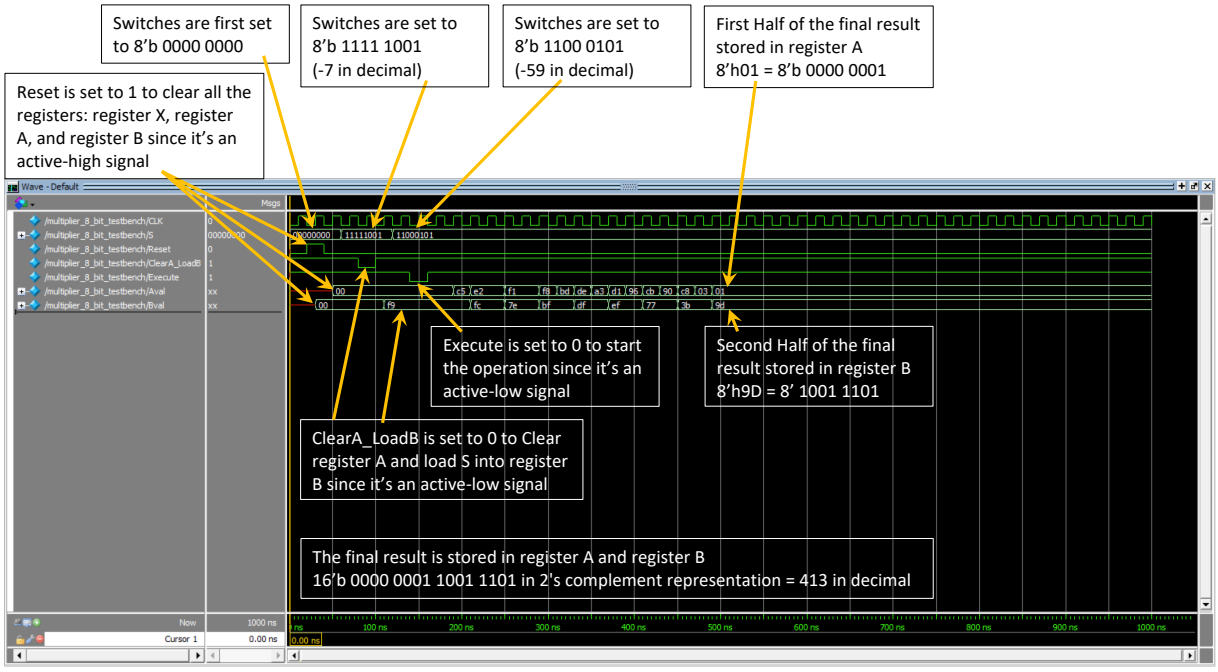
8′b 1111 1001 × 8′b 0011 1011 = -7 × 59 (in Decimal)



$8'b\ 0000\ 0111 \times 8'b\ 1100\ 0101 = 7 \times -59$ (in Decimal)



$8'b\ 1111\ 1001 \times 8'b\ 1100\ 0101 = -7 \times -59$ (in Decimal)



Post-Lab

	8-bit Multiplier
LUT	96
DSP	None
Memory (BRAM)	0
Flip-Flop	41
Frqeuency	215.84MHz
Static Power	89.94mW
Dynamic Power	0.00mW
Total Power	98.66mW

If the ripple-carry adder is replaced by the carry-lookahead adder or carry-select adder, the maximum frequency will increase since those two adders have faster computational time. However, this would also increase the total gates which is not worthy because the ripple-carry adder has the least total gates. Besides the 9-bit adder/subtractor, the control unit may be the place which can be further optimized. The state machine of the design has 8 Add states, 8 Shift states, 1 Start state, 1 Load state, 1 Hold state, 1 Check state, and 1 Restart state which are 21 states in total. There are too many states, and many states have the same output control signals, so the design is not optimal. For example, all the Shift state has Shift_Enable = '1' and other control signals = '0'. Same for Add states, all the Add state has LoadXA = '1' (possibly Subtract_Enable = '1') and other control signals = '0'. Therefore, the 8 Add states and 8 Shift states could be combined together into 1 Add state and 1 Shift state. Then, the next state will just depend on the least significant bit from register B, and a counter is added to keep track of how many add-shift operations are done. However, this design needs to be tested to see if it minimizes the total gates and increases the maximum frequency. Overall, the state machine in the control unit seems to be the best place to optimize in order to minimize the total gates and increase the maximum frequency of the design.

The purpose of register X is to store the correct sign bit (9^{th} bit) of the sum which is computed by adding two sign-extended 9-bit 2's complement numbers using the 9-bit adder/subtractor. The 8-bit Multiplier utilizes the add-shift/subtract-shift algorithm to perform the multiplication of two 8-bit 2's complement numbers. The X register is set whenever the least significant bit of register B is '1', B[0] = 1, which means the FSM is in add/subtract state. Then, the X register stores the most significant bit, also called the sign bit (9^{th} bit), of the result from the 9-bit adder/subtractor. In addition, the X register gets cleared when either ClearA_LoadB pushbutton is pressed, or the multiplier is performing consecutive multiplications (may have overflow).

When the most significant bit of the multiplier stored in Register B is '1' which means the multiplier is a negative number in 2's complement representation, the circuit needs to perform subtract-shift operation instead of add-shift in order to make sure the result has the correct sign bit after shifting. Supposed that in the last subtract-shift operation, the 8-bit subtractor is performing 126-127 which is 8'b 0111 1110 + 8'b 1000 0001 in binary. The result will be 8'b 1111 1111 with the value of carry out being '0'. If the 8-bit Multiplier circuit uses the carry out of the 8-bit adder/subtractor as the shift-in value of the final result instead of the output of the 9^{th} bit from the 9-bit adder/subtractor, the final value would be 16'b 0111 1111 1XXX XXXX which is a positive number in 2's complement representation stored in register AB after completing the last shift state. However, the final result of multiplication should be a negative number. On the other hand, performing 126-127 with a 9-bit adder/subtractor is sign-extended 9'b 0 0111 1110 + sign-

extended 9'b 1 1000 0111 in binary. The result will be 9'b 1 1111 1111 which has a negative sign bit. The final result would be 16'b 1111 1111 1XXX XXXX which is a negative number in 2's complement representation after completing the last shift state. Therefore, using the result of the 9th bit from the 9-bit adder/subtractor will ensure register A gets the correct sign bit when it performs right-shift.

8-bit Binary Subtraction		Decimal Subtraction	
	0111 1110		126
+	1000 0001	-	127
=	0 1111 1111	=	-1
Right Shift by 1 bit	0111 1111	=	127

The final result is 8'b 0111 1111 with a decimal value of +127 after shifting right by 1 bit which is incorrect.

9-bit Binary Subtraction		Decimal Subtraction	
	0 0111 1110		126
+	1 1000 0001	-	127
=	1 1111 1111	=	-1
Right Shift by 1 bit	1111 1111	=	-1

The final result is 8'b 1111 1111 with a decimal value of -1 after shifting right by 1 bit which is correct.

The limitation of continuous multiplications is that the result may be incorrect after triggering Execute several times. The 8-bit Multiplier is capable to calculate the correct product of two 8-bit 2's complement numbers without overflow issue because the final product is stored in 8-bit register A and 8-bit register B which result in a total of 16-bit. The range of 16-bit 2's complement number is -32768 to 32767, and the range of 8-bit 2's complement input is -128 to 127. Therefore, the product will never cause the overflow problem when two 8-bit 2's complement numbers are multiplied together. However, if consecutive multiplications are performed, the value stored in register X and the value stored in register A are cleared every time when Execute is triggered. This happens because the add-shift algorithm only works when register XA stores '0' and register B stores the 8-bit 2's complement Multiplier at the beginning of each computational cycle. Therefore, there will be cases that the final result is too large which overflows into register A after triggering Execute several times. Then, the 8-bit Multiplier will not give the correct result after pressing Execute again because part of the Multiplier stored in register A is cleared. Overall, the 8-bit Multiplier will only give the correct result in consecutive multiplications when the product is completely stored in register B.

The advantage of implemented multiplication algorithm is that it is easy to implement the algorithm using binary numbers in the digital system, but it is not practical to implement the pencil-and-paper method using decimal numbers in the digital system. Therefore, this algorithm can perform consecutive multiplications of 8-bit 2's complement numbers very quick in the digital system which is way faster than doing consecutive multiplications using pencil-and-paper method. However, the disadvantage is that the implemented multiplication algorithm has to take care of the situation when a positive number multiplies with a negative number. In that case, it has to change the last add-shift operation to subtract-shift operation to make sure the result has the correct sign which is complicated. Therefore, the implemented multiplication algorithm is hard to understand conceptually compared to the pencil-and-paper method because it contains lots of add-shift states which depends on the least significant bit of the multiplier. However, the pencil-and-paper method seems more straight forward while doing multiplication in decimal on paper.

Conclusion

The functionality of the design for this lab is an 8-bit Multiplier which is able to compute the product of two 8-bit 2's complement numbers and display the result through the 4 HEX displays. The 8-bit Multiplier utilizes the add-shift/subtract-shift algorithm which is controlled by the least significant bit of the multiplier, B[0], stored in register B. In addition, the sign-extended 9-bit adder/subtractor is crucial for this lab because it ensures register A gets the correct shift-in sign bit from register X which further makes sure that the final result is correct. Moreover, the 8-bit Multiplier is capable of doing consecutive multiplications only if the product is completely stored in register B because register X and register A get cleared at the beginning of each execution. The FSM could be simplified more to optimize the entire design. The lab manual contains detailed explanations of add-shift algorithm and detailed schematic of the 8-bit Multiplier which are really helpful while designing the entire circuit. Furthermore, the post-lab questions make us understand the lab more about the utilization of 9-bit adder/subtractor and the problem of overflow.