ECE 385: Digital Systems Laboratory

Fall 2022

Experiment #4

An 8-Bit Multiplier in SystemVerilog

Jialiang Xu (jx17)
Jinrui Hu (jinruih2)
TA: Neo Yuan

October 3, 2022

1 Introduction

In this report, we describe the multiplier circuit that is capable of conducting multiplication between two 8-bit 2's complement numbers.

The multiplier is consisted of two 8-bit registers A and B, one 1-bit register X, 8-bit switches S, four 7-segment Hex displays, and two buttons (Run) and (Load-Clear-Reset).

To conduct the multiplication operation, the `multiplicand` is first set on the switches, and then loaded into the register B when (Load-Clear-Reset) is pushed and released. The `multiplier` is set and stays on the switches S. When the (Run) button is pushed, the final 16-bit result will be computed. It will be stored in registers AB and displayed on the Hex displays. The multiplier also supports consecutive multiplication, which uses the previous result as the multiplicand in the next round of calculation.

In the rest of this report, multiple aspects relevant to the multiplier are covered. We start off with answering the pre-lab question in §2. Then, the written description and diagrams of the multiplier circuit are provided in §3.

2 Pre-Lab Question

In this section we answer the question asked in the pre-lab part of the Experiment #4 manual[1].

Question: 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.

Answer: Initial Values: X = 0, A = 00000000, B = 11000101 (achieved using ClearA_LoadB signal), S = 00000111, M is the least significant bit of the multiplier (Register B). The calculating process can be found in Table 1.

Function	X	A	B	M	Comments for the next step
Clear A, LoadB, Reset	0	0000 0000	*00000111*	1	Since M = 1, multiplicand (available from switches S) will be added to A.
ADD	1	1100 0101	*00000111*	1	Shift XAB by one bit after ADD complete
SHIFT	1	1110 0010	1 *0000011*	1	Add S to A since M = 1
ADD	1	1010 0111	1 *0000011*	1	Shift XAB by one bit after ADD complete
SHIFT	1	1101 0011	11 *000001*	1	Add S to A since M = 1
ADD	1	1001 1000	11 *000001*	1	Shift XAB by one bit after ADD complete
SHIFT	1	1100 1100	011 *00000*	0	Do not add S to A since M = 0. Shift XAB.
SHIFT	1	1110 0110	0011 *0000*	0	Do not add S to A since M = 0. Shift XAB.
SHIFT	1	1111 0011	00011 *000*	0	Do not add S to A since M = 0. Shift XAB.
SHIFT	1	1111 1001	100011 *00*	0	Do not add S to A since M = 0. Shift XAB.
SHIFT	1	1111 1100	1100011 *0*	0	Do not add S to A since M = 0. Shift XAB.
SHIFT	1	1111 1110	01100011	1	8^{th} shift done. Stop. 16-bit Product in AB.

Table 1: The Calculating Process of "11000101 × 00000111" by the Multiplier Circuit

[1] See https://wiki.illinois.edu/wiki/display/ECE385FA22/Lab+4

3 Written Description and Diagrams of Multiplier Circuit

3.1 Summary of Operation

The operation has three major stages: 1) the operands being loaded, 2) the multiplier computing the result, and 3) the result being stored and displayed.

To load the operands, the `multiplicand` is first set on the switches, and then loaded into the register B when (Load-Clear-Reset) is pushed and released. This is done by connecting the `Load` port of register B to the signal generated by the button. The `multiplier` is set and stays on the switches S.

To compute the result, the multiplier utilizes a `Finite State Machine`, an `Add/Subtract` module, and a (Run) button. When the button is pushed, the `Finite State Machine` starts transitioning and generating control signals according to the given multiplication algorithm. The `Add/Subtract` module takes in the control signals, carries out the algorithm by adding to / subtracting from the shift registers.

To store and display the result is a trivial process. When the final 16-bit result is computed, it resides in the concatenation of the registers AB which is connected to the Hex display digits.

The multiplier also supports consecutive multiplication, which uses the previous result as the multiplicand in the next round of calculation. This is achieved naturally with the design of the multiplier described above. Note that the result will be truncated into 8-bits when regarded as an operand, so when the result is higher than 8-bit this feature will be malfunctioning.

3.2 Top Level Block Diagram

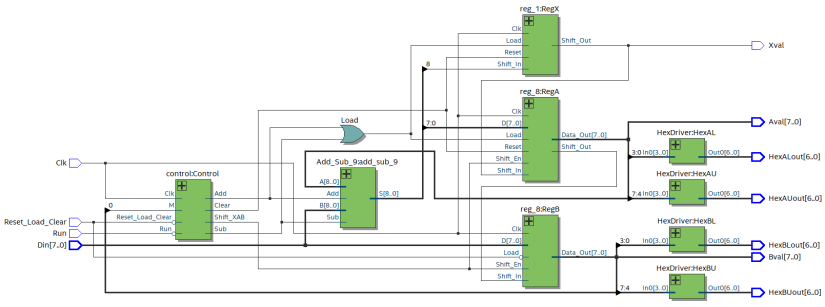

Figure 1: The Top Level Block Diagram of the Multiplier.

The top level block diagram can be found in Figure 1. From left to right there are four major components of the circuit:

- The `Control` module that generates control signals which are utilized by other components.
- The `Add/Subtract` module that takes in control signals and manipulate register values accordingly.
- The `Register` module that load and stores operands, intermediate values, and final compuation results.
- The `Display` module that consists of four *Hex* displays connected to Register A and B for displaying register values.

3.3 Written Description of .sv Modules

3.3.1 The `Control` module

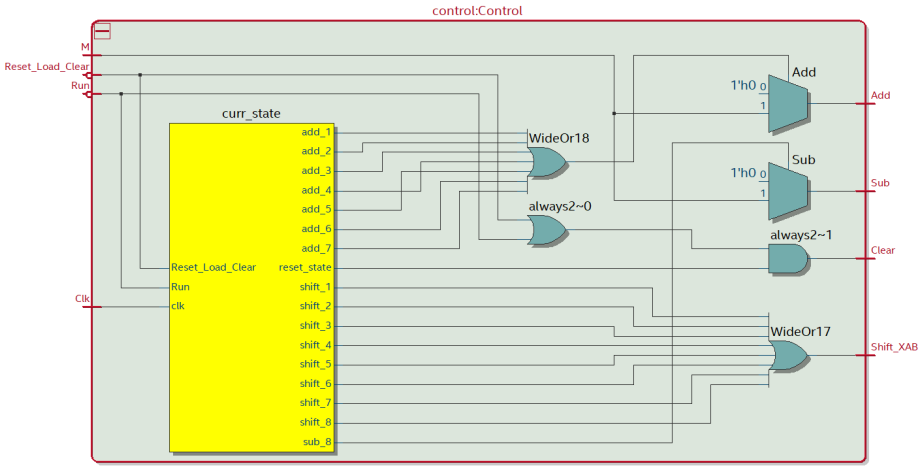

Figure 2: The Block Diagram of the Control Module

- **Module:** `control` in control.sv
- **Inputs:** Clk, Reset_Load_Clear, Run, M
- **Outputs:** Shift_XAB, Add, Sub, Clear
- **Description:** This module keeps track of the current circuit status with an `Finite State Machine`. At each state, the `control` module generates signals that will determine the behavior of the connected `Add_Sub_9` and `reg_8` modules (see Figure 1 for connection details).
- **Purpose:** to control the circuit behavior by keeping track of the circuit status and generating control signals.

3.3.2 The `FullAdder` module

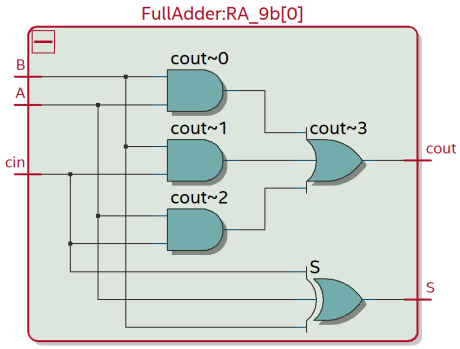

Figure 3: The Block Diagram of the FullAdder Module

- **Module:** `FullAdder` in Adder.sv
- **Inputs:** A, B, cin
- **Outputs:** S, cout
- **Description:** This module performs simple full adding functionality.
- **Purpose:** to facilitate the building of the higher-level 8-bit ripple adder.

3.3.3 The `RippleAdder` module

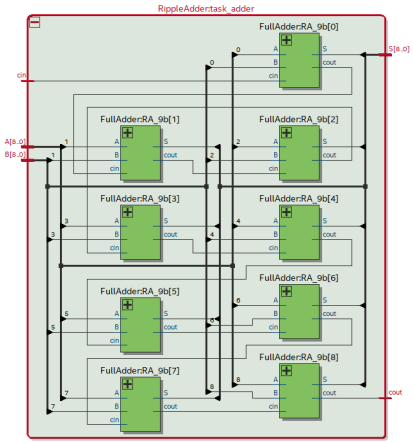

Figure 4: The Block Diagram of the RippleAdder Module

- **Module:** `RippleAdder` in Adder.sv
- **Inputs:** [8:0] A, [8:0] B, cin
- **Outputs:** [8:0] S, cout
- **Description:** This module performs adding by rippling the simply passing the carry bit across eight full adder modules, as decribed in the previous Experiment[2].
- **Purpose:** to facilitate building the higher-level `Add/Subtract` module.

3.3.4 The `Add/Subtract` module

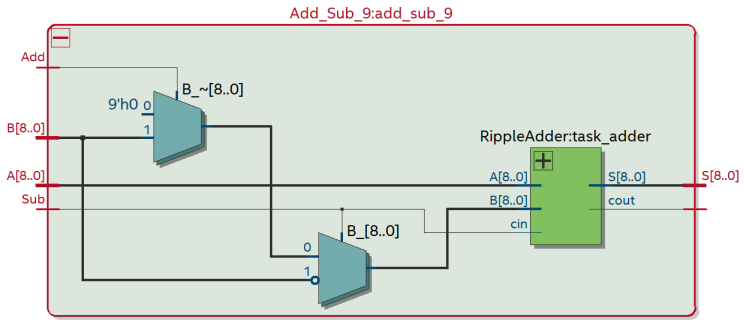

Figure 5: The Block Diagram of the Add/Subtract Module

- **Module:** `Add_Sub_9` in Adder.sv
- **Inputs:** [8:0] A, [8:0] B, Add, Sub
- **Outputs:** [8:0] S, cout
- **Description:** during the computation of the multiplication, every time the last bit of Register B (also referred to as M) is a logic one, the module adds the `Multiplicand` on S to A except when it is in the last cycle of the SHIFT-ADD/SUB repetition of the `Finite State Machine` where it subtracts S from A instead. This behavior utilizes the control signals generated by the `Control` module.
- **Purpose:** this module conducts the specific multiplication process between the Register A and the switches S. It performs adding or subtracting based on the output of the `Control` module.

[2] https://wiki.illinois.edu/wiki/display/ECE385FA22/Lab+3

3.3.5 The `reg_1` module

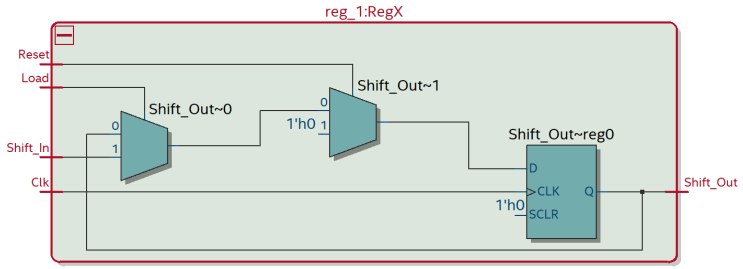

Figure 6: The Block Diagram of the reg_1 Module

- **Module:** `reg_1` in reg_8.sv
- **Inputs:** Clk, Reset, Shift_In, Load
- **Outputs:** Shift_Out
- **Description:** a simple 1-bit shift register used to load and store values participating in the compuation process.
- **Purpose:** this module is used to implement the Register X for sign extension during the arithmetic shifting in the computation process.

3.3.6 The `reg_8` module

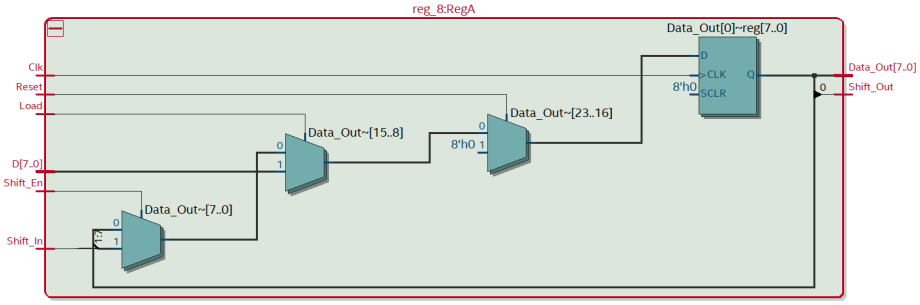

Figure 7: The Block Diagram of the reg_8 Module

- **Module:** `reg_8` in reg_8.sv
- **Inputs:** [7:0] D, Clk, Reset, Shift_In, Load, Shift_En
- **Outputs:** [7:0] Data_Out, Shift_Out

- **Description:** a simple 8-bit shift register used to load and store values participating in the compuation process.
- **Purpose:** this module is used to implement the Registers A, B for loading and storing the to-be-computed values of the `Multiplicand` and `Multiplier`. The concatenation of Registers A and B are also used to display the final computed result.

3.3.7 The `HexDriver` module

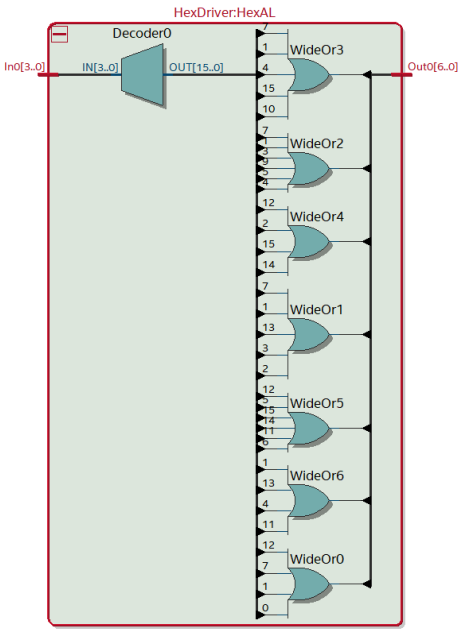

Figure 8: The Block Diagram of the HexDriver Module

- **Module:** `reg_8` in reg_8.sv
- **Inputs:** Clk, Reset, Shift_In, Load
- **Outputs:** Shift_Out
- **Description:** a simple Hex Display driver that outputs corresponding 7-bit signals that interprets a 4-bit binary value into human-readable hexadecimal outputs.
- **Purpose:** this module is used to visualize the values stored in the Registers A, B so that the operands and compuation results can be read from the display directly for human evaluation.

3.4 State Diagram for Control Unit

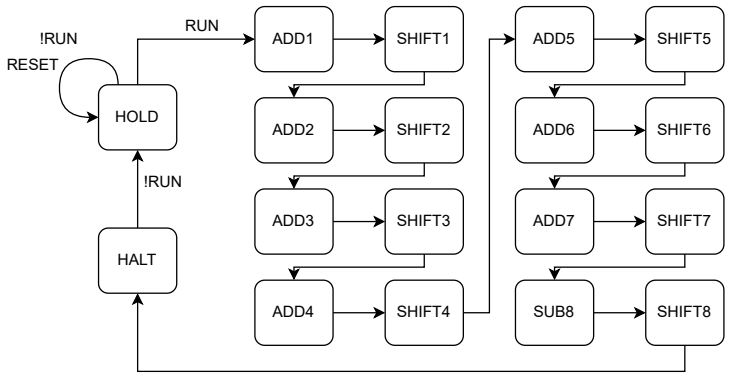

Figure 9: The State Diagram of the Finite State Machine in the Control Module.

At the state "SUB8" or states with names of the shape "ADD#", the control signals depend on the last bit of Register B (*i.e.*, M). When M is a logical one, $Shift = 1$, $Add = 1$, $Sub = 0$ for "ADD#" and $Shift = 1$, $Add = 0$, $Sub = 1$ for "SUB8", otherwise $Shift = 0$, $Add = 0$, $Sub = 0$.

At the states with names of the shape "SHIFT#", the control signals generated will be $Shift = 1$, $Add = 0$, $Sub = 0$.

4 Annotated Pre-Lab Simulation Waveforms

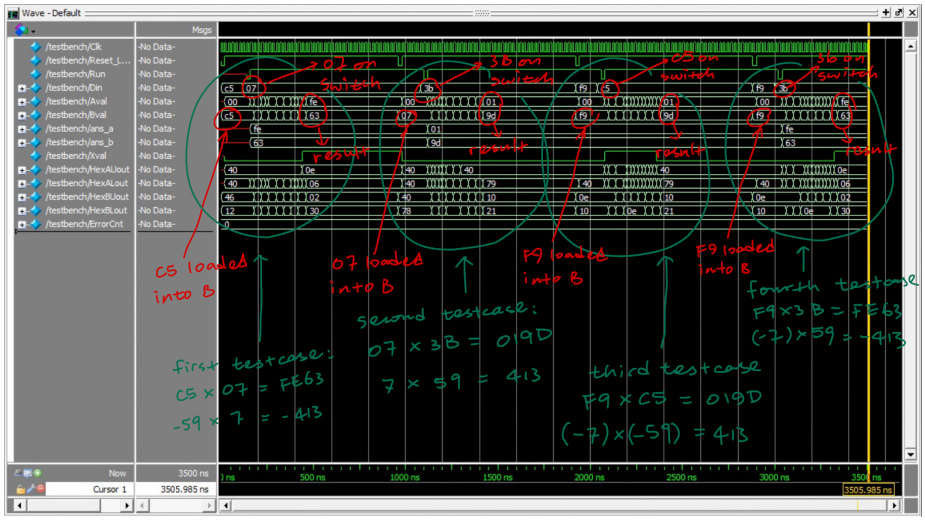

Figure 10: The Annotated Pre-Lab Simulation Waveforms

5 Answers to Post-Lab Questions

In this section we answer the questions in the Post-Lab section of the Experiment manual. The order is determined by the questions' order in the manual.

5.1 Compilation Report Table

LUT	102
DSP	N/A
Memory (BRAM)	0 / 1,677,312 (0 %)
Flip-Flop	0
Frequency	204.37 MHz
Static Power	90.05 mW
Dynamic Power	0.00 mW
Total Power	121.81 mW

Table 2: The Resource Usage Table of the Circuit

To optimize the design's maximum frequency, we can switch from the Carry Ripple Adder to a Carry Select Adder, so that the latency for the addition or subtraction can be reduced.

5.2 Written Questions

In this section we answer the questions that requires a written response proposed in the Post-Lab section.

5.2.1 What is the purpose of the X register? When does the X register get set/cleared?

The purpose of the Register X in our design is to enable arithmetic shift during the computation, which keeps the value's sign bit when conducting right-shifting. Register X is critical because it keeps track of the sign bit of Register A, which will be placed as the new most significant bit after a shift on Register A has been finished.

5.2.2 What would happen if you used the carry out of an 8-bit adder instead of output of a 9-bit adder for X?

If the carry out of an 8-bit adder is used instead, the behavior will be incorrect. Under cases where a negative result does not generate a carry out (*e.g.*, -1d + 0d), or where a positive result generates a carry out (*e.g.*, -1d + 2d), the carry out cannot correctly reflect the sign bit of the final result and will lead to erroneous behavior.

5.2.3 What are the limitations of continuous multiplications? Under what circumstances will the implemented algorithm fail?

Between two runs of the multiplication, the Register A and X will be cleared since they will be used to hold data in the compuation process. This leads to the behavior that the result will be truncated to the length same to that of Register B, which is 8 bits. This will not be an issue when the final result's value is persisted after the truncation, however when the absolute value of the number is larger than a threshold (in our case, $\geq 256d$ or $\leq -256d$), the result will be incorrect and the continuous multiplication will fail.

5.2.4 What are the advantages (and disadvantages?) of the implemented multiplication algorithm over the pencil-and-paper method discussed in the introduction?

- **Advantages**: it is making full use of the shift register where each bit of the registers is taking a piece of working information during each time step of the process, thus it avoids wasting hardware resources on the FPGA, comparing to the pencil-and-paper algorithm where the storage and summation of eight 16-bit intermediate result are needed.
- **Disadvantages**: the algorithm itself is less intuitive, and the implementation is less straightforward.

6 Conclusion

6.1 Functionality Discussion

In this experiment, we implemented the Multiplier circuit capable of conducting multiplication between two 8-bit 2's complement numbers.

To conduct the multiplication operation, the `multiplicand` is first set on the switches, and then loaded into the register B when (Load-Clear-Reset) is pushed and released. The `multiplier` is set and stays on the switches S. When the (Run) button is pushed, the final 16-bit result will be computed. It will be stored in registers AB and displayed on the Hex displays. The multiplier also supports consecutive multiplication, which uses the previous result as the multiplicand in the next round of calculation.

6.2 Lab Manual Suggestions

The lab manuals are of high-quality, they are clear, easy to follow, and have very few things that can be better. One such thing is that the indexing can be more consistent. For instance the title of the manual pdf itself says Experiment 8 which is not true, and on page 5.8 it says "fill in the table shown on 5.6 with your design's statistics", but the table is on page 4.7.