**ECE 385**

Fall 2021

Experiment #4

**8-bit Multiplier**

Steven Dimov & Owen Shin

Section ABE

TA: Abigail Wezelis

**Introduction**

This week’s lab experiment was to construct a multiplier circuit capable of multiplying two 8-bit 2’s compliment numbers using System Verilog and run it on the DE-10 Lite. The goal of the design specifically includes to be able to multiply with respect to signed numbers, so operations + with a +, - with a -, + with a -, as well as – with + must all be possible. The last couple of terms meaning that the multiplicand and multiplier must both be able to handle positive and negative numbers. The design also adds an additional 9th bit to the design that handles sign of the 16-bit output.

**Pre-lab question**

First, the multiplication operands 11000101, 00000111 are assigned to multiplicand and multiplier, accordingly.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Function | X | A | B | M | Comments |
| Clear A, Load B, Reset | 0 | 00000000 | 00000111 | 1 | M=1, so the value on the switches A, will be added to A. |
| ADD | 1 | 11000101 | 00000111 | 1 | Subsequent SHIFT after ADD |
| SHIFT | 1 | 11100010 | 10000011 | 1 | M=1, so ADD A to A. |
| ADD | 1 | 10100111 | 10000011 | 1 | Subsequent SHIFT after ADD |
| SHIFT | 1 | 11010011 | 11000001 | 1 | M=1, so ADD A to A. |
| ADD | 1 | 10011000 | 11000001 | 1 | Subsequent SHIFT after ADD |
| SHIFT | 1 | 11001100 | 01100000 | 0 | M=0, so only SHIFT |
| SHIFT | 1 | 11100110 | 00110000 | 0 | M=0, so only SHIFT |
| SHIFT | 1 | 11110011 | 00011000 | 0 | M=0, so only SHIFT |
| SHIFT | 1 | 11111001 | 10001100 | 0 | M=0, so only SHIFT |
| SHIFT | 1 | 11111100 | 11000110 | 0 | M=0, so only SHIFT |
| SHIFT | 1 | 11111110 | 01100011 | 1 | 8th SHIFT, final result in AB, no subtraction |

This method correctly outputs the correct answer in binary, 1111111001100011, -413 in decimal.

**Written Description of Circuit**

To load the operands, the switches on the FPGA are used to configure the inputs that load the sets of 8-bit registers A & B. Reset is pressed to both clear all the registers and then load the 8-bit data from the switches into the 8 least significant bits of the 16-bit line of registers, which corresponds to B. Technically speaking, only the set of registers B is loaded in from the switches. This is because after changing the switches to configure A, the next step in the sequence is to press the execute button to multiply B with A, by directly taking A from the switches when used in the addition states of the FSM.

The multiplier begins computation of the two inputs A & B when the Execute state is initiated by the button. The process of multiplication commences by stepping through a series of cycles, where in each one the least significant bit is checked for a one or a zero, where then the input A is added to the 8 most significant bits and then subsequently right shifted by one bit for when it is a one, and only right shifted when it is a zero. After this specific cycle of right shifting is done 7 times, the next state checks for whether or not the input is negative and subtracts the other factor from input A if it is negative or just skips this step if positive and simply right shifts once more. After this is done, the final result is left in the 16-bit line of registers, AB, which are visible on the HEX displays of the FPGA. Additional multiplication operations can be performed onto the result by reconfiguring the switches and reinitiating the execute state, as long as the output does not exceed 8 bits.

b. Top Level Block Diagram

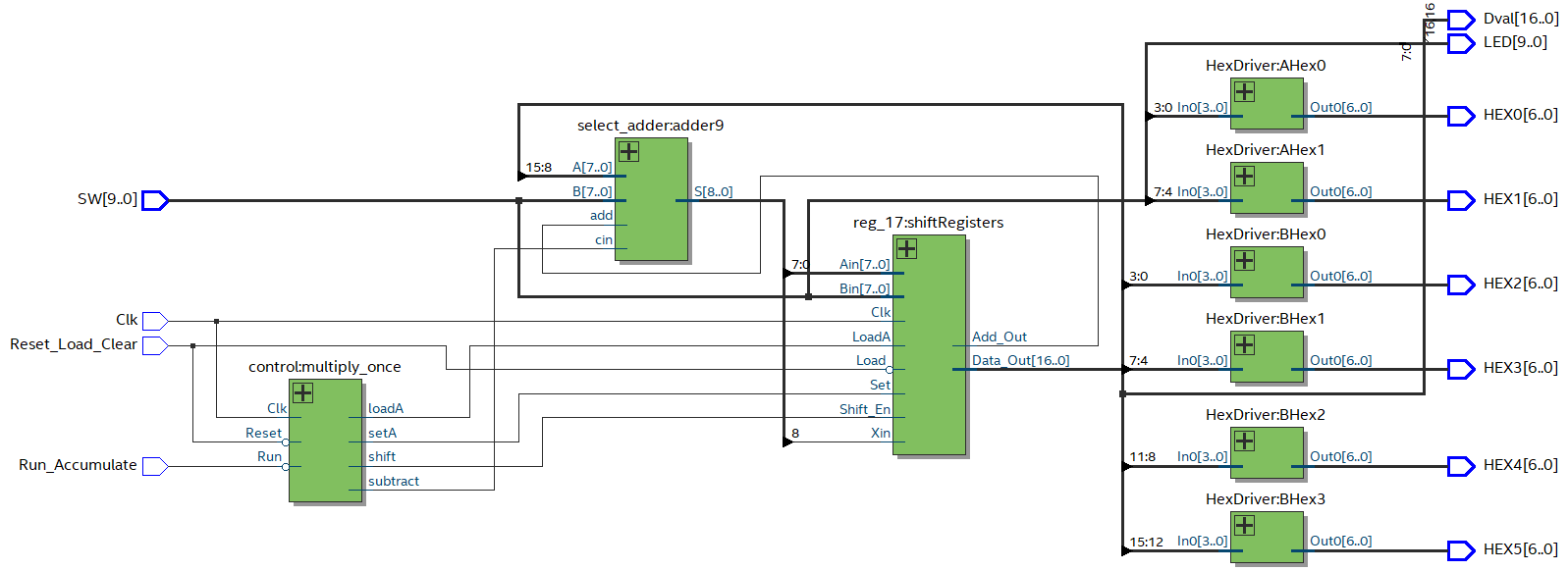


Figure 1: RTL diagram of all modules in the multiplier. The whole system includes hex decoders, a control unit, a 9-bit adder, and a 17-bit shift register.

c. Written Description of .sv Modules

i. List all modules used in a format shown in the appendix of this document.

Module: control.sv

Inputs: Clk, Reset, Run

Outputs: shift, loadA, subtract, setA

Description: This is a finite state machine with synchronous reset and run. When Reset is pressed, the machine returns to a standby state. When Run is pressed, the machine goes into an add-shift sequence. Outputs are entirely state-dependent.

Purpose: This module is used to control the register unit and the adder unit. The outputs instruct the register unit on whether to shift or parallel load from the adder unit or switches or reset. The control unit instructs the adder unit on whether to add or subtract.

Module: reg17.sv

Inputs: Clk, Load, Set, LoadA, Shift\_En, Xin, [7:0] Ain, [7:0] Bin,

Outputs: Add\_Out, [16:0] Data\_Out

Description: 17-bit shift register with synchronous functions. Registers can parallel load the most significant bits from Xin and Ain or the least significant bits from Bin. Control signals come from all other inputs. Contents are outputted as Data\_Out.

Purpose: This register holds the contents of operations and receives sums according to instructions from the control register. The Add\_Out output is the least significant bit of its contents, which determines whether or not the adder unit performs an add.

Module: select\_adder.sv

Inputs: add, [7:0] A, [7:0] B, cin

Ouputs: [8:0] S

Description: Purely combinational adder unit that utilizes two 4-bit select adders and an extra one-bit full adder at the most significant bit. The two inputs to the most significant adder are the same as the most significant inputs to the second most significant adder. One of the inputs goes through the bitwise expression ((B ^ cin) & add). The cin input is used as the least significant carry in.

Purpose: This unit conditionally adds inputs A (most significant bits of register unit) and B (contents of switches) or subtracts B from A, depending on inputs add and cin. The results of these operations are outputted to the register unit.

d. State Diagram for Control Unit

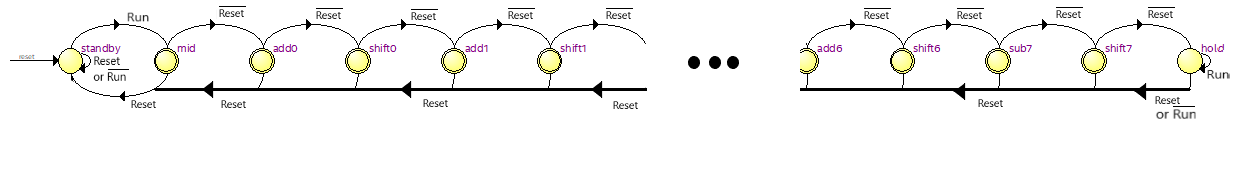
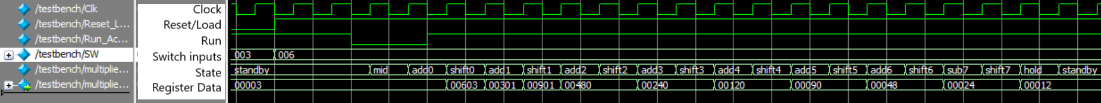
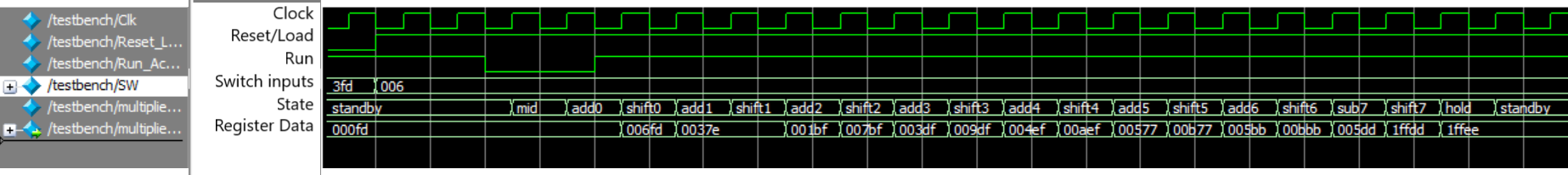


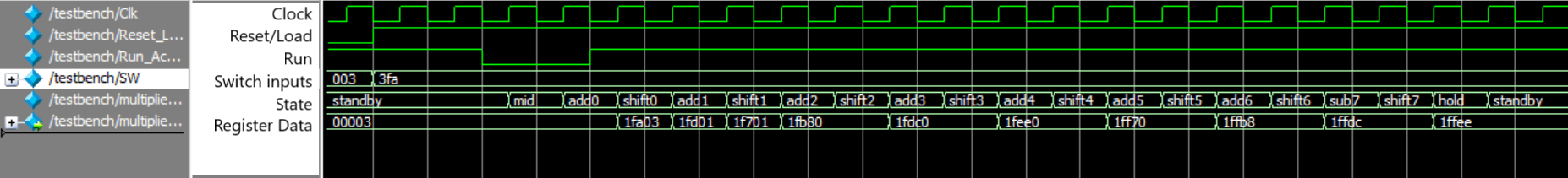
Figure 2: State machine for control unit of multiplier. The machine begins in a standby state, and begins multiplication when the run button is pressed. From there, it goes into an intermediate state (“mid”) where the most significant bits of the register are cleared, and then the contents of the switches are conditionally added based on the least significant bit of the register and the register is shifted. The previous two steps are repeated 6 times, then a conditional subtraction and another shift take place. Finally, the control unit enters a hold state, which transfers to the standby state if the run button is not pressed. The machine outputs 4 control bits which control whether the register is to take an addition, subtraction, shift, or reset (of the most significant bits).

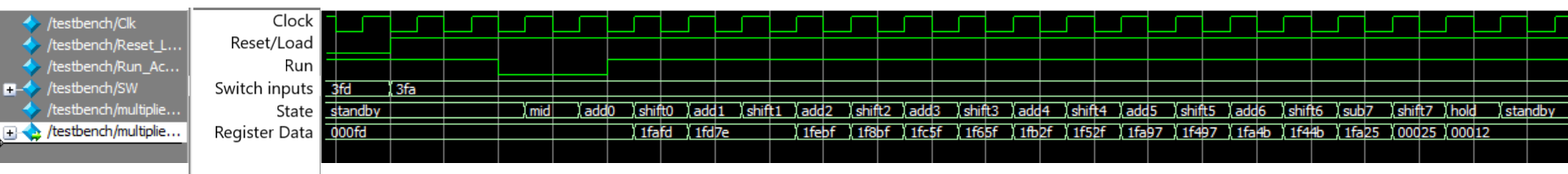
4. Annotated pre-lab simulation waveforms.

a. Must show 4 operations where operands have signs (+\*+), (+\*-), (-\*+) and (-\*-)







Figures 3-6: Modelsim outputs for different operations. From top to bottom, operations are 3x6, -3x6, 3x(-6), -3x(-6). When the machine exits an ”add” or ”sub” state, it performs a conditional add or subtract. When it exits a ”shift” state, it performs a shift. Note that the contents of the registers at the end (lower right of each picture) are equal to the products of the two numbers. Also note that the register contents begin as the previous (left) values of the switch inputs, as they were loaded earlier before changing.

**Answers to post-lab questions**

design statistics table:

|  |  |
| --- | --- |
| LUT | 677 |
| DSP | N/A |
| Memory (BRAM) | 18,432 |
| Flip-Flop | 921 |
| Frequency |  |
| Static Power |  |
| Dynamic Power |  |
| Total Power |  |

Table 1: Incomplete statistics table due to some compilation errors.

The design could probably be improved in the state machine and the adder. If the adder used a slower design it would require less gates, and if the state machine were generally improved to not use a one-state-per-operation model, some logic could probably be conserved.

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

The X register is to ensure that the sign of the number in the register is maintained as the register is being shifted. When an addition takes place, it takes the most significant bit of the sign-extended numbers summed together, and when the register shifts the X and second most significant bit (if we consider X to be most significant) exchange values.

For example, if you loaded 0x01 into the rightmost bits of the register and then multiplied by a negative number like 0xFC, 0xFC would be added to the empty most significant bits, making them 0xFC. Because the second most significant bit is high, X would also be high. Upon a shift, the 1 in X goes into the second most significant bit, and 1 goes from that bit into X. This repeats on each subsequent shift, maintaining the negative sign, until the final result 0xFFFC, which is correct. If it weren’t for the X register receiving the 1 upon the initial addition, the machine would not know what value to shift in from the left. If it had shifted in a 0 by default during this example, the number would incorrectly be positive. If it shifted in a 1 by default, other examples may be wrongly negative.

• 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 was used in place of the output of a 9-bit adder for X, it would no longer ensure that it will sign extend AB when it is right-shifted. If so, this cause errors in computation as the multiplier requires sign extension to account for upholding 2’s compliment notation. This will result in completely different outputs.

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

The implemented algorithm has limitations when foregoing continuous multiplications. Such limitations include reaching numbers so high that not enough bits are available to compensate for “overflow”. This does not mean overflow in the traditional sense where there are no physical bits left to account for the carry-out bit, but rather there are no more assigned registers left to receive the carry-out without being overwritten in the consequent multiplication. This is because of how the algorithm expects a cleared A upon adding onto the A registers. Consequently, having products greater than 8 bits will yield an incorrect result the following execution.

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

The advantages to the implemented algorithm are that it can multiply numbers, positive or negative, in either arrangement/commutation. Other advantages include the use of fewer logic gates and registers as compared to the extensiveness that the pencil paper design would require. A disadvantage is that this design does not allow for computations with inputs greater than 8 bit 2’s compliment, because of the previously limitation of an 8-bit output to correct compute the next result.

**Conclusion**

During our demo, our design’s functionality was limited to being able to only multiply two positive integers or two negative integers without respect to overflow. Efforts we made to correct it were successful to some degree. Specifically, our design had issues with adding properly in some cases where ones were being added on each cycle of the finite state machine. This was due to an oversight in the design of the 2’s compliment implementation where the conditionality of the addition of 1 was not accounted for, which caused an extra one to be added in instances where it should not have been. Ultimately, we were able to bring our design to meet the entirety of capabilities that were expected.