

**Project Report: Sequential 8-bit Signed Multiplier Implementation on Artix 7 FPGA**

**Digital Design 1 – Dr. Mohamed Shalan**

**By: Amal Fouda, Dana Alkhouri, Farida Bey, Mohamed Sabry, Nour Selim**

**Fall 2023**

**1. Introduction**

The objective of this project is to design and implement a sequential 8-bit signed multiplier using the shift-and-add algorithm. The implementation will be realized on the Artix 7 FPGA, integrated into the Basys 3 FPGA board.

**2. Purpose Overview**

**2.1. Multiplication Algorithm**

The shift-and-add algorithm, commonly known as the paper-and-pencil method, will be employed for the multiplication process. This algorithm is sequential and involves shifting and adding steps, mimicking manual multiplication.

**2.2. Signed 8-bit Multiplier**

The multiplier will accept two 8-bit signed numbers as inputs and produce a 16-bit signed product. The signed numbers will be represented using two's complement notation.

**3. Design Overview**

**3.1 Input**

* **Multiplier (SW7-SW0):** Toggle switches SW7 to SW0 are used to input the 8-bit signed multiplier.
* **Multiplicand (SW15-SW8):** Toggle switches SW15 to SW8 are used to input the 8-bit signed multiplicand.

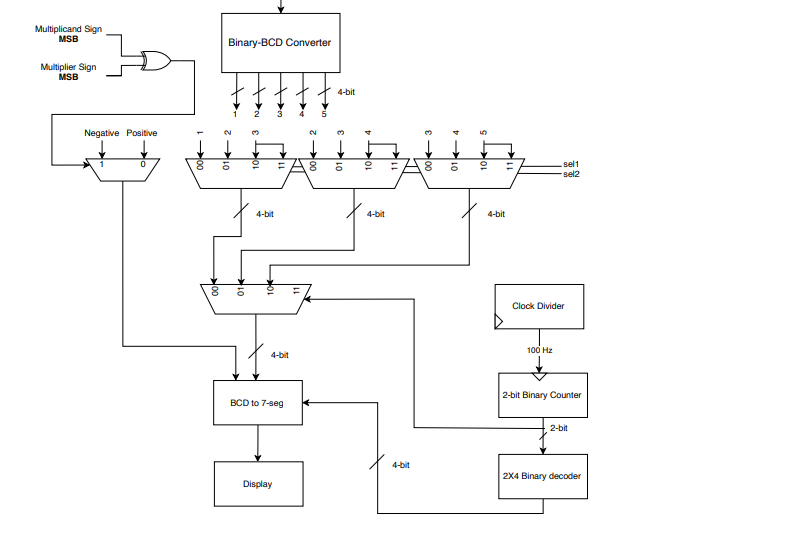
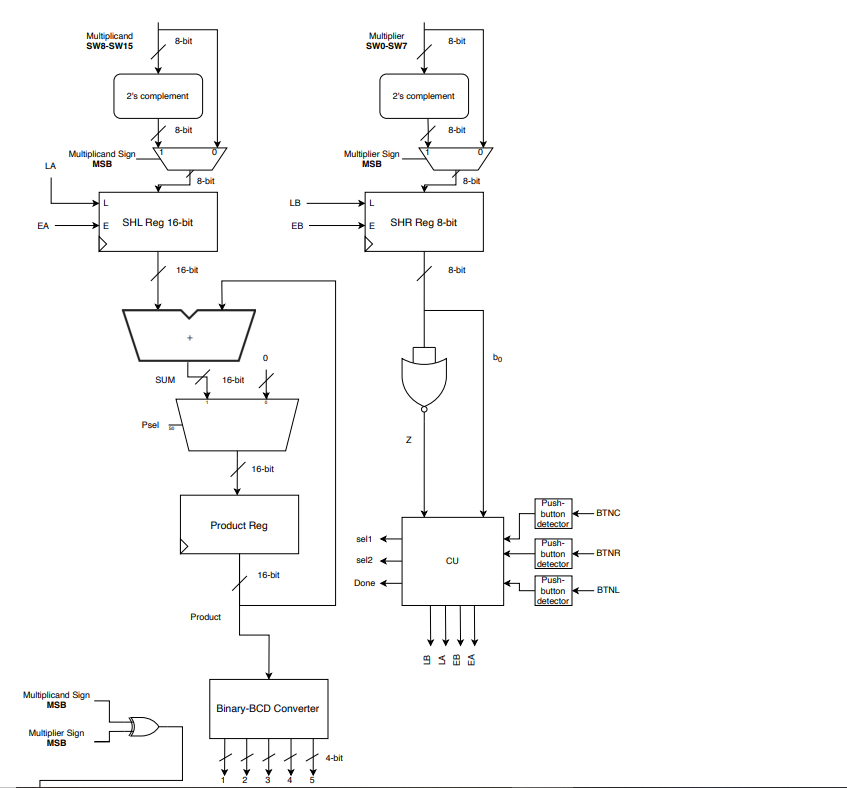
**3.2 Output**

* **Product Display:** The right three 7-segment display digits show the decimal product.
* **Sign Display:** The leftmost 7-segment digit is used to display the sign of the product.

**3.3 Control**

* **Scrolling:** Push buttons BTNL and BTNR allow users to scroll through the product digits, as the result can be up to 5 decimal digits.
* **Start multiplication:** Push button BTNC is used to start the multiplication process.
* **End of multiplication indicator:** LED LD0 indicates the end of the multiplication process.

**4. Block Diagram**



**5. Implementation Details and Verilog Modules**

**5.1 Multiplier**

The multiplier module begins with the conversion of the 8-bit signed inputs *MP* and *MC* to their two's complement form, which is necessary for proper handling of negative numbers during multiplication. The *Shift\_Left* and *Shift\_Right* registers are initialized to zero and one, respectively. These registers are used for the shifting operations during multiplication.

The multiplication logic is triggered by the rising edge of the clock (*posedge clk*). If the *load* signal is asserted (when *BTNC* is pressed), the module loads the initial values for *Shift\_Left* and *Shift\_Right* based on the multiplicand and multiplier. If the least significant bit *b0* of *Shift\_Right* is 1, the product is accumulated. The *zero\_flag* is set (becomes 0) when *Shift\_Right* becomes zero, indicating the completion of the multiplication. The *sign* output is determined by XORing the sign bits (indicated by the most significant bit of the multiplier and multiplicand).

Lastly, the *done* signal is asserted when the *zero\_flag* is true, indicating the completion of the multiplication, and LED0 is turned on.

**5.2 bin\_to\_BCD**

Bin\_to\_BCD takes a 16-bit binary input (*bin*) and converts it into Binary-Coded Decimal (BCD) representation. The BCD output is stored in the *bcd\_output* register. The module uses double dabble logic that does the following:

* 1. *BCD* register is initialized to 21 bits of zeros, and the lower 16 bits are assigned the value of *bin*
  2. For the conversion, we use two nested for loops iterate through the *BCD* register. The outer loop (*i*) iterates from 0 to 12, and the inner loop (*j*) iterates from 0 to i/3. To adjust the BCD digits, the module checks within the loops if each BCD digit is greater than 4. If true, the digit is adjusted by adding 3 to it.
  3. The expression *BCD[16 - i + 4 \* j -: 4]* is used to access and modify a specific 4-bit segment within the BCD register.
  4. The final BCD output is assigned to the *bcd\_output* register

**5.3 CU**

Our control unit has inputs for pushbuttons BTNC, BTNL and BTNR, and outputs the selection lines sel1 and sel2 responsible for shifting the numbers displayed on the seven segment, and loading onto the display.

We begin by The 2-bit counter counter is initialized to *2'b00* in the initial block. The load register is assigned the value of the BTNC button. This indicates that the counter is loaded when the *BTNC* button is pressed. The counter is incremented when the *BTNL* button is pressed, and decremented when *BTNR* is pressed. Lastly, The 2-bit counter value is assigned to the output signals *sel\_1* and *sel\_2,* that later go into the top\_module, where we use a mod3 updown counter to shift between numbers when scrolling.

**5.4 top\_module**

The top module – as the name suggests - collates all the modules together and send out the signals to the FPGA that need to be displayed (done, segments and anode\_active). The instances in the module were:

* 1. pushbutton\_detect
     1. The pushbutton\_detect instances detect rising edges of button signals.
  2. Multiplier
     1. The multiplier module performs multiplication on the inputs MP and MC based on the control signals from the CU module.
  3. CU
     1. The CU module generates control signals based on button inputs and the state of its counter.
  4. bin\_to\_BCD
     1. The result of the multiplication is converted from binary to BCD using the bin\_to\_BCD module.
  5. BCD\_to\_7Seg
     1. The BCD\_to\_7seg module converts BCD digits to 7-segment display signals (segments) based on the count from bin\_counter.
  6. clk\_divider
     1. The clock is divided by 250,000 using the clk\_divider module to produce a slower clock (clk\_out) for counter operations.
  7. binary\_counter
     1. Two binary counters (count and count\_dis) operate to control the display and digit selection.

the module also includes the case statements for multiplexing to shift from number to number via the sel1 and sel2 signals coming from the control unit. After multiplexing, the three signals then go into another multiplexer to output the signal that goes into the 7 segment module, and is then displayed.

**5.5 BCS\_to\_7segments**

*Add module here*

**5.6 pushbutton\_detector**

*Add module here*

**5.7 synchronizer**

*Add module here*

**5.8 debouncer**

*Add module here*

**5.9 risingedge\_detector**

*Add module here*

**6. Implementation Issues**

The multiplier works according to the specified description provided in the project handout with no reported errors or issues.

**7. Contributions**

Amal Fouda:

* + Main circuit on Logisim Evolution
  + Binary to BCD logic circuit on Logisim Evolution
  + Multiplier module on Verilog
  + Top\_module module on Verilog
  + Bin\_to\_BCD module on Verilog
  + Block diagram design
  + Debugging and simulating via testbenches on Verilog

Dana AlKhouri:

* + Main circuit on Logisim Evolution
  + Clock divider module on Verilog
  + Counter module on Verilog
  + Block diagram design
  + Top\_module module on Verilog
  + Constraint file
  + Debugging and simulating via testbenches on Verilog

Farida Bey:

* + Main circuit on Logisim Evolution
  + Control unit module on Verilog
  + BCD\_to\_7Seg module on Verilog
  + Top\_module module on Verilog
  + Pushbutton\_detection module on Verilog
  + Block diagram design
  + Debugging and simulating via testbenches on Verilog

Mohamed Sabry:

* + Main circuit on Logisim Evolution
  + Register circuits on Logisim Evolution
  + 2’s complementor circuit on Logisim Evolution
  + Top\_module module on Verilog
  + CU module on Verilog
  + Block diagram design
  + Debugging and simulating via testbenches on Verilog

Nour Selim:

* + Main circuit on Logisim Evolution
  + Bin\_to\_BCD module on Verilog
  + Multiplier module on Verilog
  + Control unit module on Verilog
  + Constraint file
  + Block diagram design
  + Debugging and simulating via testbenches on Verilog

**8. Acknowledgements**

We acknowledge the support and resources provided by Dr. Mohamed Shalan for the successful completion of this project.