THE AMERICAN UNIVERSITY IN CAIRO

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

This project aims 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 sequential algorithm 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 and inputs the 8-bit signed multiplier.
- **Multiplicand (SW15-SW8):** Toggle switches SW15 to SW8 input the 8-bit signed multiplicand.

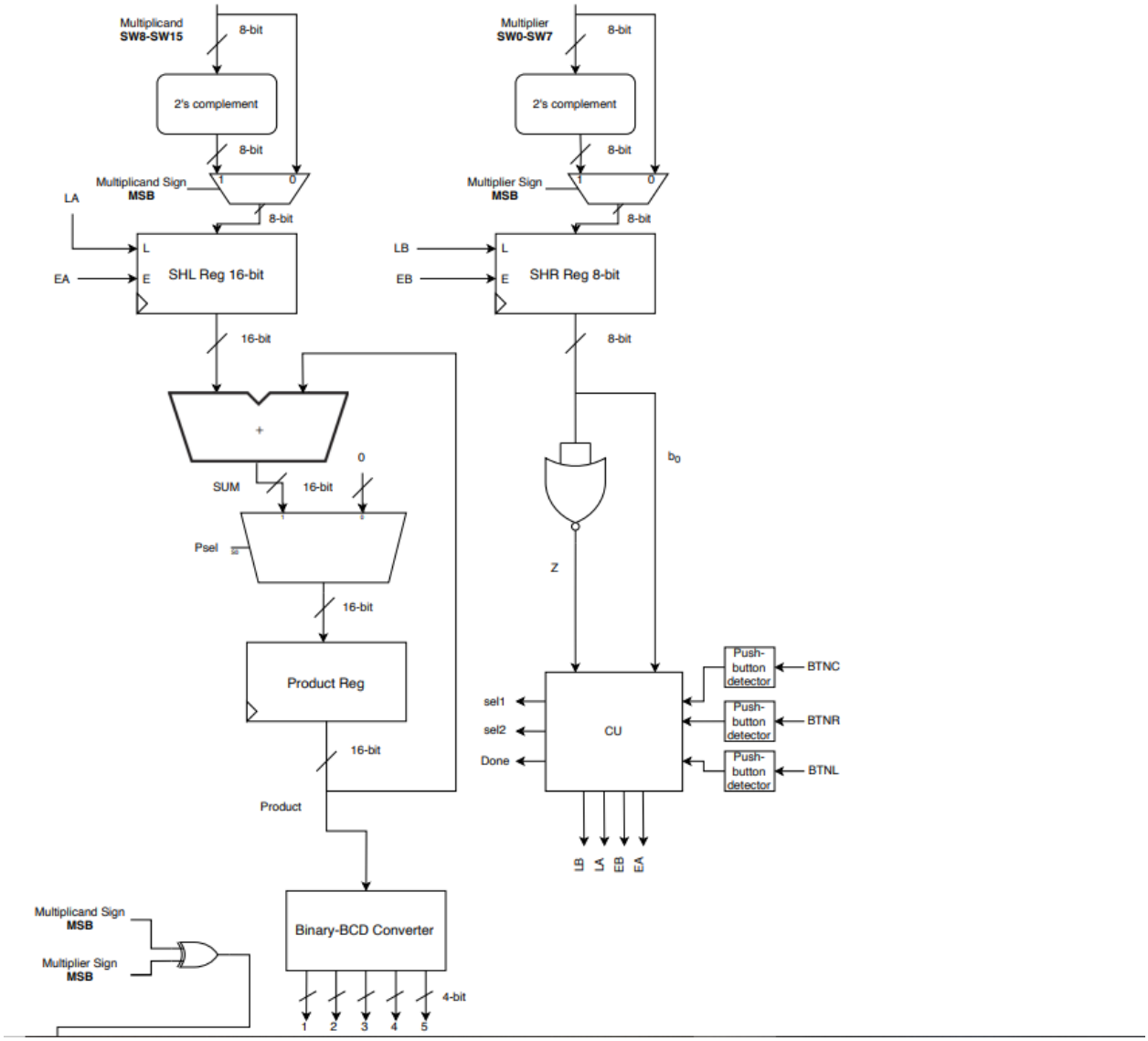
3.2 Output

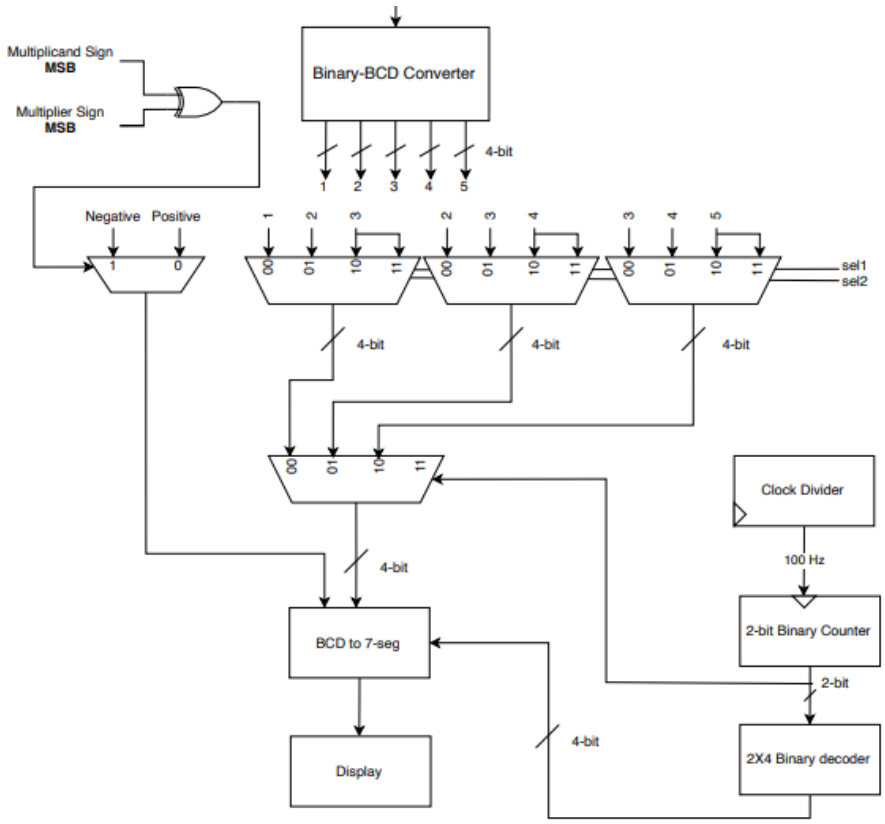
- **Product Display:** The right three 7-segment display digits show the decimal product.
- **Sign Display:** The leftmost 7-segment digit is used to display the sign of the product.

3.3 Control

- **Scrolling:** Push buttons BTNL and BTNR allow users to scroll through the product digits, resulting in up to 5 decimal digits.
- **Start multiplication:** The 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


Multiplicand
SW8-SW15
8-bit
2's complement
8-bit
Multiplicand Sign
MSB
1
0
8-bit
LA
EA
L
E
SHL Reg 16-bit
16-bit
+
0
SUM
16-bit
Psel
16-bit
Product Reg
16-bit
Product
Multiplier
SW0-SW7
8-bit
2's complement
8-bit
Multiplier Sign
MSB
1
0
8-bit
LB
EB
L
E
SHR Reg 8-bit
8-bit
b0
Z
Push-
button
detector
BTNC
Push-
button
detector
BTNR
Push-
button
detector
BTNL
sel1
sel2
Done
CU
LB
LA
EB
EA
Multiplicand Sign
MSB
Multiplier Sign
MSB
Binary-BCD Converter
4-bit
1
2
3
4
5



5. Implementation Details and Verilog Modules

5.1 Multiplier

The multiplier module begins with converting the 8-bit signed inputs *MP* and *MC* to their two complement forms, which is necessary for properly handling 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. The product is accumulated if the least significant bit b0 of Shift_Right is 1. 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 a 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. We use two nested loop iterations through the BCD register for the conversion. The outer loop (*i*) iterates from 0 to 12, and the inner loop (*j*) iterates from 0 to i/3. The module checks within the loops to adjust the BCD digits 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. It outputs the selection lines sel1 and sel2, which are responsible for shifting the numbers displayed on the seven segments and loading them onto the display.

We begin with the 2-bit counter, 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,* which later go into the top_module, where we use a mod3 up-down counter to shift between numbers when scrolling.

5.4 top_module

As the name suggests, the top module collates all the modules together and sends out the signals that need to be displayed to the FPGA (done, segments, and anode_active). The instances in the module were:

a. pushbutton_detect
    i. The pushbutton_detect instances detect rising edges of button signals.
b. Multiplier
    i. The multiplier module performs multiplication on the inputs MP and MC based on the control signals from the CU module.
c. CU
    i. The CU module generates control signals based on button inputs and the state of its counter.
d. bin_to_BCD
    i. The multiplication result is converted from binary to BCD using the bin_to_BCD module.
e. BCD_to_7Seg

    i. The BCD_to_7seg module converts BCD digits to 7-segment display signals (segments) based on the count from bin_counter.

f. clk_divider
    i. The clock is divided by 250,000 using the clk_divider module to produce a slower clock (clk_out) for counter operations.

g. binary_counter
    i. 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 from the control unit. After multiplexing, the three signals go into another multiplexer to output the signal that goes into the 7-segment module and is displayed.

5.5 BCD_to_7segments

In this module, it converts binary-coded decimals to 7 segments. It takes 4 inputs and 2 outputs, and there is no instantiation:

a. Segments (output): These represent the right number that should be displayed on the 7-segment display. The binary representation in the module represents each number on the digital display.
b. Enable (en) (input): control when the 7 segments should display when it is equal to 1.
c. Count (input): is a 2-bit input; depending on the count, we decide how many anodes are on. Another case for the count is when it is less than three because the last anode is for the sign representation (it is off when the number is positive).
d. Anode_active (output): is the display of the 4 digits.
e. Num (input): is a 4-bit number that determines the configuration of the 7 segments, so if the number that needs to be represented is 1, it will go to the second case.

5.6 Clk Divider

it uses a binary counter instantiation to divide the input clk based on the parameter n. it takes 1 input, which is clk and 1 output which clk_out. In addition, a wire that is connected to the binary counter module, which has 32 bits, is used to count the clk cycles.

a. This module's functionality is that when the reset equals 1, the clk_out is set to zero. But, when the count of the binary counter reaches a value of (n-1), the clk_out will toggle, dividing the parameter n and producing the clk_output cycles.

This module is used to generate a slower clock.

5.7 synchronizer

we used a synchronizer to change the input signal from asynchronous to synchronous. Meta is a register that catches the input signal. Meta<= SIG means that at every rising edge, the value of

SIG is captured by the register Meta to consider the delay that happens. Then, SIG1<= takes the value of meta after the delay.

5.8 debouncer

it is used in our project to deliver a signal without any noise; the output signal will be stable and better than the input signal. It takes 3 registers, each representing the state of the input signal. When reset is 1, the value of the registers is zero, which means reset all the registers. Otherwise, it will capture the signal imputed into these registers. The output of this module is either the value of the reset or the value of the result that is coming from **anding** q1,q2, and q3 (debounced version of the input).

5.9 risingedge_detector

It is based on an FSM, in which x represents the input. This module has 2-bit variables, state and next state; they present the current and next states in FSM. We have three states: A, B, and C (2 bits). If the input x is zero, the next state will always be A, and if x ==1, the next state will be B; otherwise, it will move from B to C. The output Z will equal state B at the end of this module.

6.0 pushbutton_detector

We implemented the pushbutton detector the same way in the lab. Which takes 2 inputs, 1 output, 4 wires, and 4 instantiations:

a. Clk divider: took a parameter of (50_000) to connect it to the signal port in the clk divider module. It takes clk, a constant zero representing the reset (we don't have a reset; we made it always a constant zero), and a clk out as a wire [the implementation of clk divider explained above].
b. Debouncer: another instantiation that takes the clk_out variable from the clk divider module, takes an input that captures the coming signal, and the output of this is represented by a wire to be used in the synchronizer module [the implementation of debouncer explained above].
c. Synchronizer: it takes a wire that is an output w1 from the debouncer to be an input to the synchronizer, and it takes clk out. In addition, it takes as an output signal a wire w2 [the implementation of the synchronizer explained above].
d. Rising_edge_detect: It also takes a clk_out, and it takes x as input, in which it detects any rising edges. Z is an output that represents the output signal, which means what is the output state when a rising edge happens [the implementation of the rising_edge_detect explained above].

7. Implementation Issues

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

8. 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 test benches 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 test benches 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 test benches on Verilog

Mohamed Sabry:

- The 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 test benches 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 test benches on Verilog

9. Acknowledgements

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