Sequential Signed Multiplier Report

Department of Computer Science and Engineering

The American University in Cairo

Bemen Girgis
Ziad Hassan

25.05.2023

INTRODUCTION

This report aims to understand how a sequential signed multiplier works internally.

A sequential signed multiplier performs the multiplication of signed numbers. It takes two numbers as input (The multiplier and the multiplicand) and returns their product as output.

The sequential multiplier we discuss here represents numbers in two's complement.

Design Process

The design process took a few steps.

1. First, we designed a block diagram; The purpose of this was to help convey the overall structure of the design. They also make the logic more clear and help in discovering any logic errors early on.
2. Second, we implemented the design on Logisim-evolution. Logisim-evolution is an educational software for designing and simulating digital logic circuits. Logisim-evolution allows for easier design validation and error detection. It also provides an excellent interface that helps us quickly adjust our circuits to test changes.
3. Third, we designed a Finite state machine for the control unit of the multiplier. Finite state machines are fundamental because they allow modeling and controlling systems depending on the state of the circuit. They are great for tasks that require complex control.
4. Lastly, we implemented our design using Verilog, Which entails translating the modules in the circuit into Verilog code, connecting them, and defining

their behavior.

After completing the above steps, we could finally implement our design on an FPGA, more specifically, DIGILENT BASYS3.

Specifications

The implemented design uses the four “seven-segment displays” on the board, three push buttons, and all 16 switches. The switches represent the numbers to be multiplied in two’s complement form, where the eight leftmost switches represent the multiplier, and the eight rightmost switches represent the multiplicand.

The seven-segment display is used to display the result of the multiplication. The leftmost display is used to display the multiplication sign, while the other three display the unsigned product. The leftmost display is turned off whenever the product is a positive number because it is impossible to show a positive sign. If the product is a negative number, the leftmost display displays the negative sign.

Due to the limited number of displays available (one for the sign + three for the digits), we used two push buttons to modify the position of the viewable display. The third push button was used to signal the start of multiplication.

Block Diagram

Figure 1 shows the block diagram, which explains how the different components of the circuit work together to do the multiplication.

First, the two numbers are sign checked. If either is negative, we take its two’s complement form and load it to its register. Otherwise, the number is loaded as is.
We also check if we have an odd number of negative numbers, and if we do, we assign the product a negative sign.

After everything is loaded to the shift registers, we start the following multiplication process:

1. If the multiplier is zero, we are done.
2. Else check the least significant bit of the multiplier.

3. If the bit is high, we add the multiplicand to the product.
4. If the bit is low, we keep the product as is.
5. Left shift the multiplicand, and right shift the multiplier.
6. Go to 1.

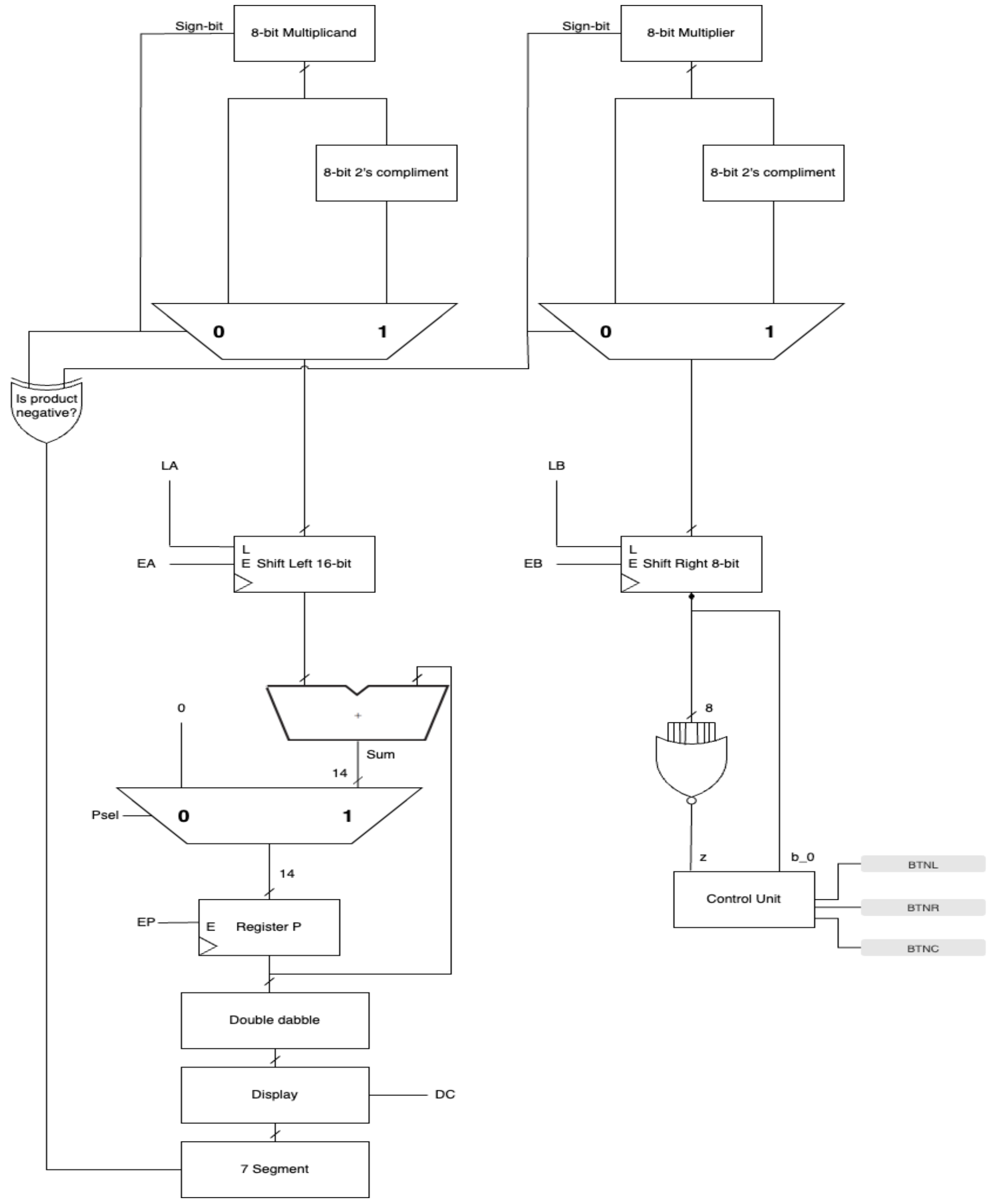

Figure 1 Block Diagram

Binary to BCD

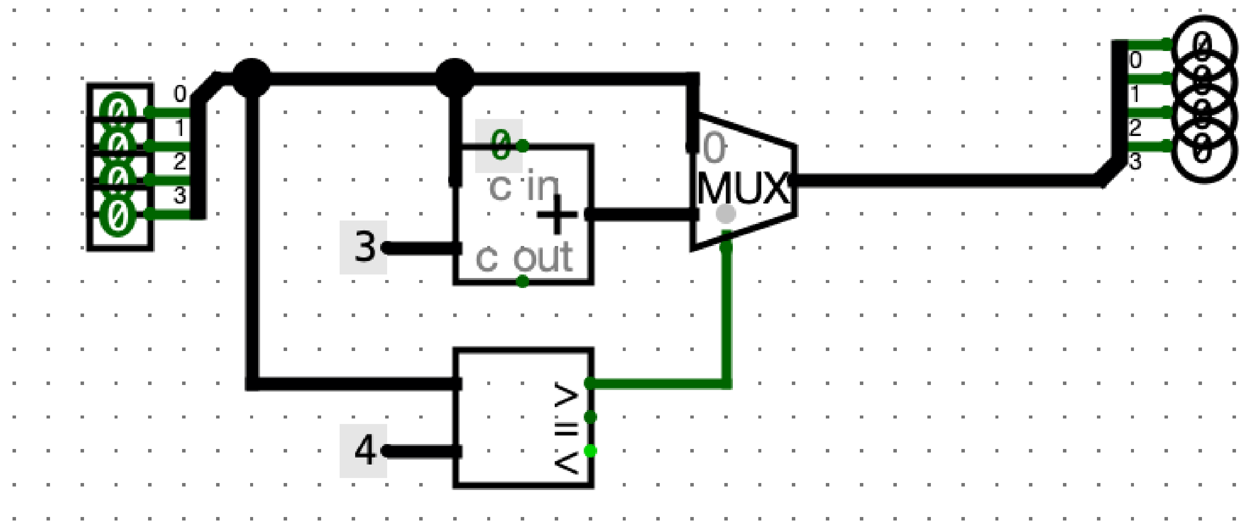

Figure 2 - one double dabble unit

We used the double dabble algorithm to convert the binary numbers to BCD. The algorithm starts from this description, If a prospective BCD digit is five or larger, add three before shifting left. The final result is the BCD value.

The ADD-3 rule is justified by the fact that the weight of a shifted out bit has been decreased from 16 to 10 whenever the shifted value is 10 or higher. Before shifting, we adjust by adding half of that number, or three. The fact that the value before shifting is 5 or above allows us to identify the 10 value after shifting. You'll see that the rule makes sure that none of the different numbers may ever contain a non-BCD value.

We highly recommend you read this paper(http://www.eprg.org/computerphile/doubledabble.pdf) to better understand how the double dabble algorithm works, provided above was a brief explanation with no examples.

Moore Diagram for the main control unit

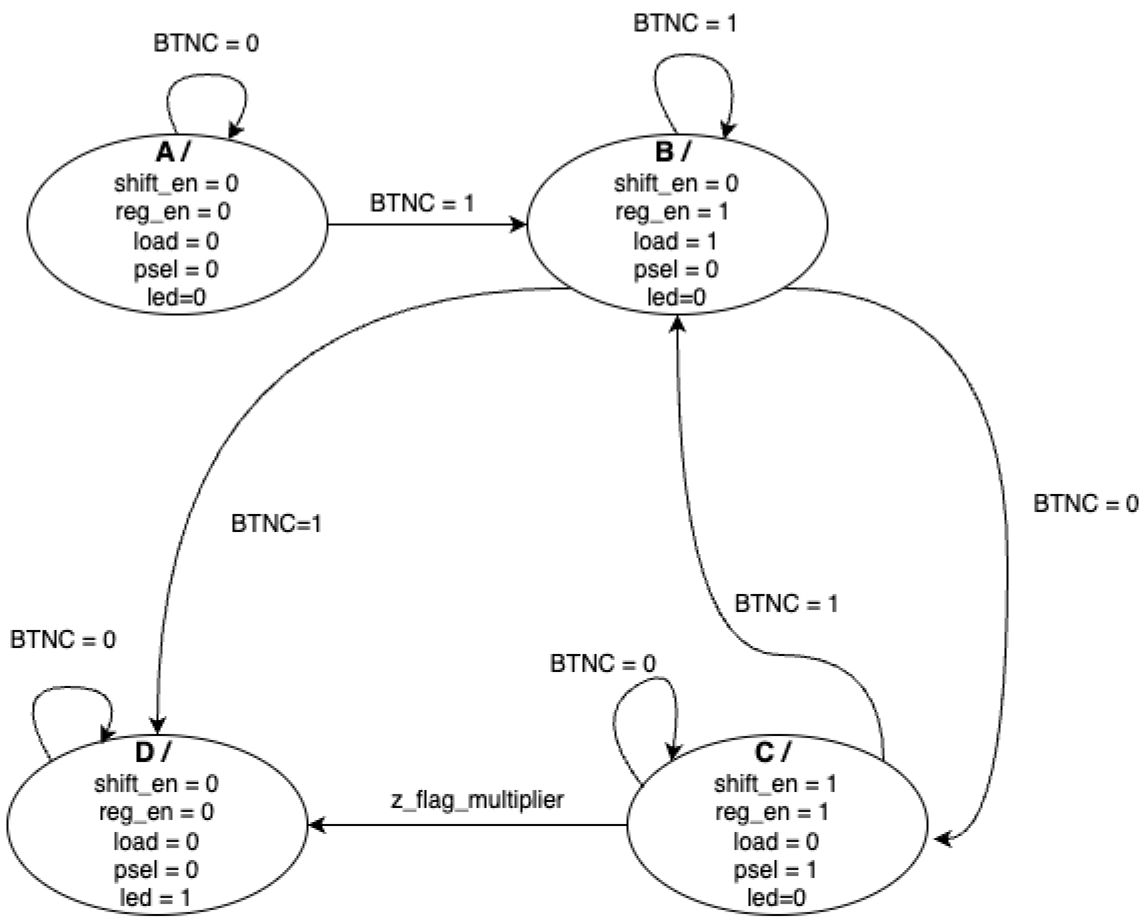

Figure 3 moore state diagram for the main control unit

This main control unit is responsible for producing all of the control signals necessary and required to perform the multiplication process correctly.

Meaning of signals with respect to the block diagram :

- **Shift_en** : The Shift_en signal serves as an enable signal for the shifting operation within the multiplication process. When activated, the numbers stored in the two registers, namely the multiplicand and the multiplier, can be shifted simultaneously. The multiplicand is shifted to the left, while the multiplier is shifted to the right. In the block diagram, this signal is represented by the components (LB, LA). By activating Shift_en, the multiplication algorithm progresses by shifting the values in the registers, aligning them for subsequent operations.

- **Reg_en** : Reg_en is a control signal responsible for enabling the loading operation of the register that holds the partial products, and eventually the final product. By activating Reg_en, the selected value, determined by the multiplexer select signal (**psel**), is loaded into the register. The value loaded into the register can either be zero or the output value from the full adder. This signal is denoted as (EP) in the block diagram, indicating its role in enabling the loading of the partial products and the final product into the respective register.

- **Load** : The Load signal acts as a control signal that triggers parallel loading of the two shift registers containing the multiplicand and multiplier. This loading operation occurs at the beginning of a new multiplication process. By activating Load, the values of the multiplicand and multiplier are simultaneously loaded into their respective shift registers. This signal corresponds to the components (EA, EB) in the block diagram, signifying its function in enabling the parallel loading operation.

- **Psel** : Psel represents a control signal involved in selecting a multiplexer. This multiplexer determines the value to be received by the register holding the partial products. The specific value selected depends on the state of the Psel signal. This signal is labeled as (Psel) within the block diagram, emphasizing its purpose in controlling the multiplexer's selection process.

- **Led** : Led is a control signal indicating the multiplication process's end. It is active only when the multiplication operation is completed, and it controls a light-emitting diode (LED) on the field-programmable gate array (FPGA). By activating Led, it triggers the LED to signal the completion of the multiplication operation. The length of this signal is typically one bit, representing an on or off state.

- **z_flag_multiplier** : z_flag_multiplier is used to signal the multiplier is zero. It is used to move from state C to D, which signals the end of multiplication. This is an optimization because it allows us to end multiplication without having to wait until all bits of the multiplier are shifted.

Moore Diagram for button control unit

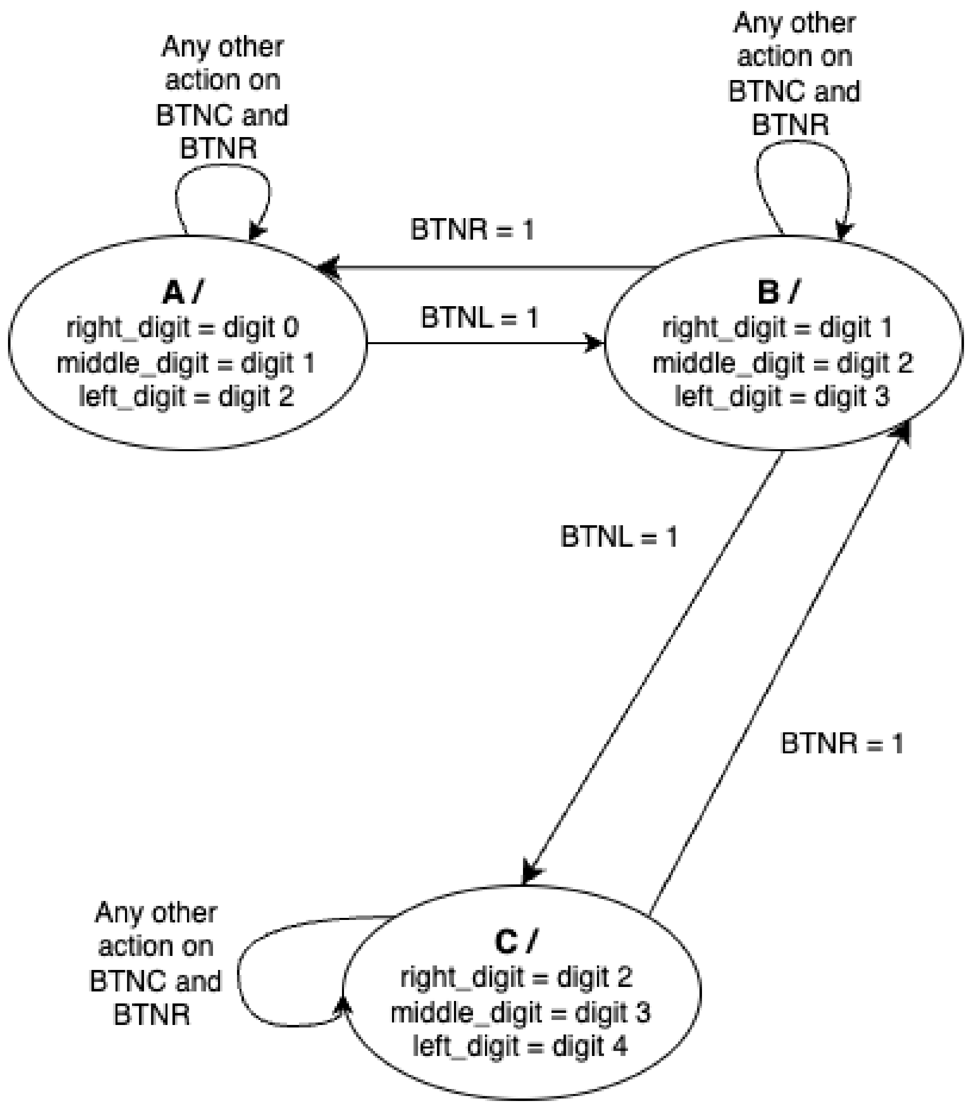
Any other action on BTNC and BTNR
Any other action on BTNC and BTNR
BTNR = 1
A / right_digit = digit 0 middle_digit = digit 1 left_digit = digit 2
BTNL = 1
B / right_digit = digit 1 middle_digit = digit 2 left_digit = digit 3
BTNL = 1
BTNR = 1
Any other action on BTNC and BTNR
C / right_digit = digit 2 middle_digit = digit 3 left_digit = digit 4

Figure 4 moore state diagram for the button control unit

After multiplication, the product can be stored in a maximum of five decimal digits, which is not possible because of the display size problem mentioned earlier in the report. That's why we used another finite state machine to move the product digits to the left or to the right on the three digits of the display we have available. This way the algorithm user can control which three digits of the product is displayed. When multiplication is done, the three least significant digits of the product are shown by default. Afterward, the user is free to display any three consecutive digits on the display based on buttons BTNL "move to the left" and BTCR "move to the right". If a new multiplication is started, the display is not reset to the default mode again ; it stays displaying the last three digit locations the user picked.

This button control unit was designed using a Moore finite state mach as the main control unitine. The state diagram for this FSM is in figure 4.

To prevent code from behaving unexpectedly, it is necessary to ensure control units and push button detectors work on the same divided clock frequency. Otherwise, buttons won't behave as expected. In our main file, this issue is considered and fixed ; all control units and their corresponding push buttons (or any other signal that is dependent on the clock) should receive the same clock frequency.

This secondary control unit could've been integrated within the main control unit. However, it is always better to separate concerns and split the codes into modules based on their functionality. As a result , it is much easier to modify a specific part of the code without worrying about other parts needing to be fixed later. Also, it makes understanding the code flow much easier when everything is organized and separated based on functionality.

Known Issues

- Our testing indicates that there are no issues with the design or logic. However, and while this is not exactly an "issue," Verilog as a hardware description language that supports parallel execution was confusing to use. Especially to people who are used to sequential software languages like C.

References

http://www.eprg.org/computerphile/doubledabble.pdf