8-Bit Signed Multiplier using the Shift-and-Add Multiplication Algorithm

Ahmed Allam
Computer Science Dept.
American University in Cairo
Cairo, Egypt
ahmedeallam@aucegypt.edu

Hussein Elazhary
Computer Science Dept.
American University in Cairo
Cairo, Egypt
helazhary@aucegypt.edu

Eslam Tawfik
Computer Science Dept.
American University in Cairo
Cairo, Egypt
eslamtawfik@aucegypt.edu

Kirollos Zikry
Computer Science Dept.
American University in Cairo
Cairo, Egypt
kirollos21@aucegypt.edu

***Abstract*—This project was focused on designing and implementing a sequential 8-bit signed multiplier using the shift and add algorithm, tailored for the Artix 7 FPGA on the Basys 3 FPGA board. This project advanced the concepts discussed in lectures. We employed toggle switches for inputting two 8-bit binary-signed values and utilized a 7-segment display for outputting the product in decimal. The design also incorporated push buttons for scrolling the product digits and an LED indicator for the completion of multiplication. This project not only demonstrated a practical application of digital design principles but also showcased our proficiency in FPGA programming and hardware interfacing. We are using the Verilog coding guidelines and best practices listed in Dr. Shalan's GitHub repository, available at *github.com/shalan/verilog_coding_guidelines*. All of our code is available at *github.com/ahmed-alllam/8-Bit-Signed-Multiplier*.**

***Index Terms*—8-bit Signed Multiplier, Shift and Add Algorithm, FPGA, Digital Design, Hardware Interfacing, Binary Arithmetic, Sequential Circuits, Decimal Display**

I. Team Members and Roles

This section highlights the contributions of each team member to the 8-bit signed multiplier project.

A. Ahmed Allam

Contributions:

- Developed the main project file in Verilog, integrating various components of the multiplier.
- Designed the core multiplier circuit in Logisim.
- Implemented bidirectional shift functions in Logisim for the multiplication algorithm.
- Created the product register in Logisim for result storage.
- Developed a rising edge detector in Verilog for enhanced push button functionality.
- Conducted extensive debugging across the project to ensure correct results in all test cases.

B. Hussein Elazhary

Contributions:

- Implemented a conditional adder in Logisim for the double dabble algorithm, which adds 3 when a bit value of 4 or larger is detected, facilitating binary to BCD conversion.
- Implemented the double dabble algorithm using these conditional adders to efficiently convert the 16-bit binary output from the multiplier into a BCD format.
- Implemented an 8-bit 2's complementer in Logisim, essential for handling negative numbers in binary arithmetic.
- Created a universal shift register capable of shifting bits left, right, loading, and retaining values, enhancing the versatility of the data handling process.
- Designed a seven-segment display decoder in Logisim, converting 4-bit binary input into signals suitable for a 7-segment display
- Focused on output formatting to BCD to display the 16-bit output on the Basys 3 board's 7-segment display, converting binary values to a binary-coded decimal format.

C. Eslam Tawfik

Contributions:

- Developed the block diagram for the multiplier, establishing a foundational design framework.
- Contributed to the creation of shift registers and Binary Coded Decimal (BCD) components in Logisim.
- Designed and implemented shift left and shift right registers modules in verilog.
- Implemented the first part of the main module, dealing with the multiplier.
- Addressed and resolved binary to BCD issues, ensuring accuracy, and helped in merging multiplying and displaying components in the main module.

D. Kirollos Zikry

Contributions:

- Orchestrated the design of the control unit for seamless integration with other components.
- Implemented and refined the codebase for the control unit, ensuring its efficient operation.
- Devised an algorithm facilitating the smooth scrolling of results displayed on the seven-segment display.
- Crafted and optimized the codebase responsible for executing the scrolling algorithm.
- Engineered the debouncer and synchronizer for the push buttons, enhancing their reliability and responsiveness.

II. Block Diagram

A. Design Overview

The following section detail the internal architecture of a shift-and-add multiplier circuit, dissecting both the computational and display elements. The top section outlines the multiplication mechanism, while the bottom section describes the output visualization process, from binary-to-BCD conversion to seven-segment display interfacing.

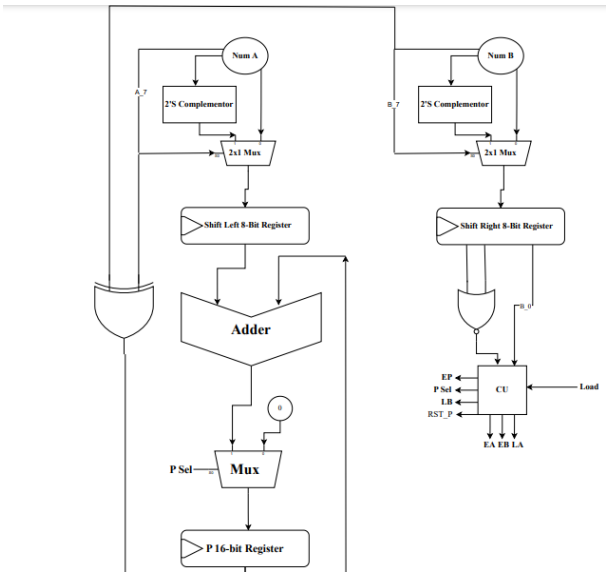

Fig. 1. Block Diagram

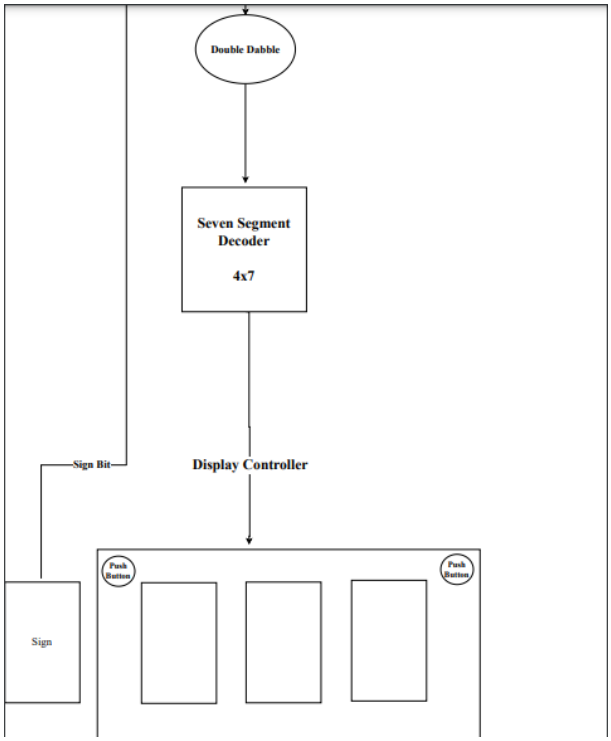

Fig. 2. Block Diagram cont.

The block diagram illustrates key components of a shift-and-add multiplier circuit. It comprises 2's complementors for inputs 'Num A' and 'Num B' to facilitate negative number multiplication. Following these, 2x1 multiplexers enable conditional selection between the direct and complemented inputs based on the multiplication logic. The shift registers, associated with each input, are essential for bitwise shifting operations intrinsic to the multiplication algorithm. The adder combines the shifted multiplicand with the accumulator's current value. A subsequent multiplexer, governed by the 'P Sel' signal, routes the adder's output or an alternate signal to the 'P 16-bit Register', which holds the intermediate and final products. The Control Unit (CU) orchestrates the synchronization of shifts, additions, and data routing via control signals such as 'Load', 'EP', 'P Sel', 'LSB', 'RST P', 'EA', 'EB', and 'LA', ensuring the correct execution of the multiplication process.

The lower portion of the block diagram delineates the output stage of the multiplier, focusing on the display mechanism. The 'Double Dabble' block is indicative of the binary to Binary-Coded Decimal (BCD) conversion algorithm, crucial for representing the binary product in a human-readable decimal format. This output feeds into a 'Seven Segment Decoder,' typically a 4x7 decoder, translating the BCD into signals that drive a seven-segment display. A 'Display Controller' manages these signals, ensuring the correct representation of numbers on the display. The 'Sign Bit' input to the Display Controller likely toggles the display between positive and negative numbers. Lastly, individual blocks, represent the segments of the display, are controlled by the Display Controller, with Push Buttons interfaced to navigate through the displayed digits.

B. Component Description

III. Logisim Simulation Breakdown

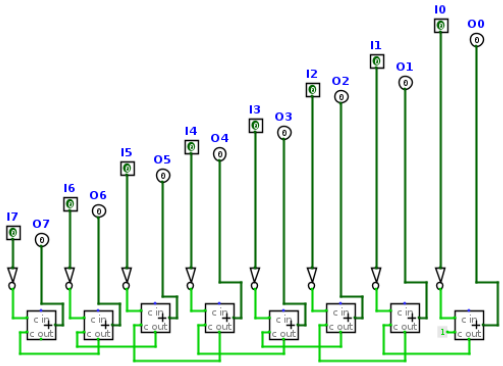

Fig. 3. 8 bit 2's complementer

This is an 8-bit 2's complementer circuit that uses full adders and NOT gates. The circuit uses NOT gates applied to each of the 8 input bits, effectively generating the 1's complement of the input number. The 1's compliment is a bitwise inversion, where all 0s are turned into 1s, and all 1s are turned into 0s.

The 1's complement output is fed into a series of full adders, which are arranged to add a binary 1 to the least significant bit (LSB). This addition of 1 to the 1's complement results in the 2's complement, which is the standard method for representing negative binary numbers. The carry-out from each full adder is connected to the carry-in of the next higher-order full adder, chaining them together in a typical binary addition fashion.

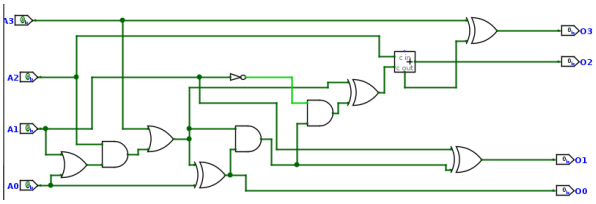

Fig. 4. Conditional Adder

This conditional adder is designed to add a value of 3 to a 4-bit binary input whenever the input value is 4 or greater. This circuit is typically used in binary to BCD conversion processes, such as the double dabble algorithm. It features standard logic gates like AND, OR, and NOT, as well as a half-adder for the conditional addition. The inputs A0 to A3 represent the 4-bit binary number, and the outputs O0 to O3 represent the resulting sum after the conditional addition.

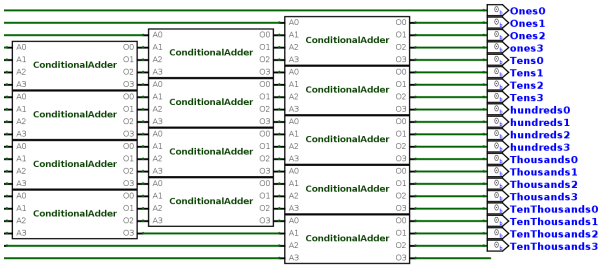

Fig. 5. Double Dabble

This double dabble algorithm implementation uses conditional adders for a 16-bit binary to BCD (Binary-Coded Decimal) conversion is a sophisticated digital logic design. In this setup, the double dabble algorithm, which is used for converting a binary number into its BCD represntation, is implemented using a series of conditional adders instead of the standard shift-and-add approach. The 16-bit input binary number is processed through these conditional adders, which are designed to add a specific value to certain parts of the binary number whenever a condition is met (when the bits exceeds a decimal value of 4).

The process involves shifting the binary number left one bit at a time and examining each part of the number to determine if an addition is necessary before the next shift. This is done for all 16 bits of the binary number. The use of conditional adders in this method enhances the efficiency of the conversion process, as these adders are tailored to perform additions only when specific conditions are met, reducing unnecessary operations. This optimized approach, a graphical tool for designing and simulating logic circuits, enables a more efficient and faster conversion from binary to BCD, crucial in applications requiring rapid and accurate decimal representation of binary data.

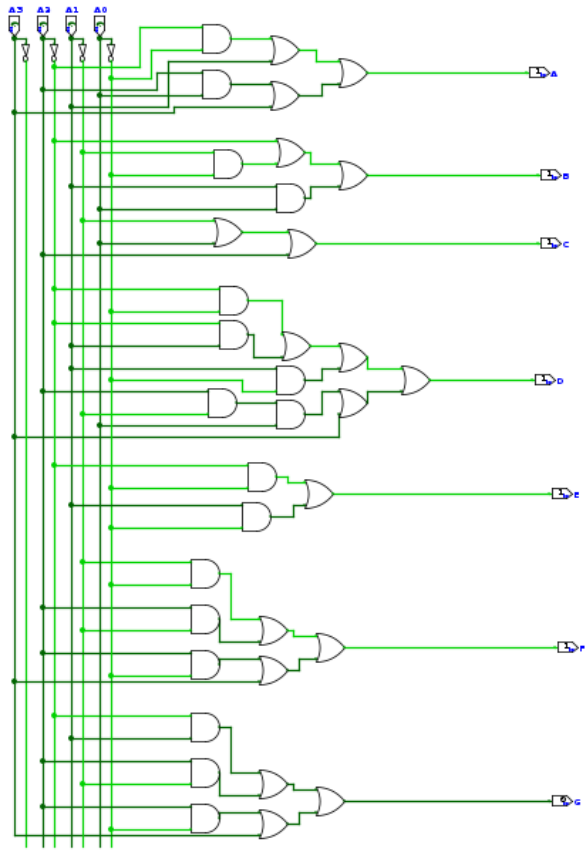

Fig. 6. SevenSegmentDecoder

This seven segment decoder circuit is designed to take a binary input and translate it to a signal that will drive a seven-segment display, which is used to represent decimal digits from 0 to 9 in a readable format. The binary input consists of 4 bits. The seven-segment decoder interprets these inputs to output seven signals, each controlling one of the seven individual segments (labeled A through G) of the display. When the appropriate segments are lit, the desired number or character is displayed.

In this specific implementation, the circuit uses a combination of logic gates (AND, OR, NOT) to create the necessary logic functions that define which segments should be on or off for any given input. Each output line corresponds to one segment of the seven-segment display. The logic gates are arranged in such a way that for each possible input combination, the correct segments are powered to display the corresponding digit or character.

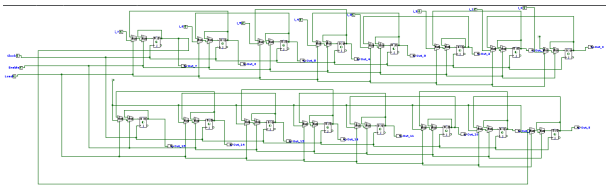

Fig. 7. ShiftLeftRegister

A shift left register is designed to move all bits within it one position to the left when a clock signal is applied. For an 8-bit shift left register, upon each clock pulse, bit 7 moves to bit 6's position, bit 6 to bit 5's, and so on, with bit 0 typically receiving a 0. The bit that in the highest order (bit 7 in this case) is usually shifted out of the register and can be used for further processing or simply discarded.

In the context of creating an 8-bit signed multiplier, a shift left register can play a crucial role. Multiplication of binary numbers can be performed through a series of shift-and-add operations. For signed numbers, the process often involves representing the numbers in 2's complement form to account for negative values. The multiplier uses the shift left register to align the multiplicand with the correct bit of the multiplier for addition. As the bits of the multiplier are scanned (typically starting from the least significant bit), the multiplicand is shifted left and conditionally added to a product register based on the value of the current bit in the multiplier. If the bit is 1, the shifted multiplicand is added; if it's 0, the addition is skipped.

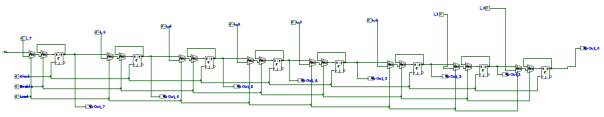

Fig. 8. ShiftRightRegister

The shift right register operates by moving all the bits in the register one position to the right upon each clock pulse. In an 8-bit register, the bit in position 0 is typically shifted out of the register, and a new bit is introduced into position 7. Depending on the type of shift right operation (logical or arithmetic), the new bit could be a zero (logical shift) or a value that maintains the sign of the number (arithmetic shift), which in the case of a signed number is the same as the most significant bit (MSB) before the shift.

The shift right register, in this case, would be part of the circuit that holds the next product of the two numbers being multiplied. As the bits of the multiplier are processed, the register shifts right to position the partial product correctly for addition or subtraction with the multiplicand, according to the multiplier's bits. By iteratively shifting and adding or subtracting the multiplicand, the final signed product is obtained in the register after processing all bits of the multiplier. This method is suitable for hardware implementations of signed multiplications in digital systems.

IV. Design and Implementation in Verilog

A. Design Approach

Our design approach for the 8-bit signed multiplier project was methodical and innovative, ensuring a seamless blend of theory and practical application. Initially, we developed a comprehensive block diagram to visualize the system architecture, ensuring clarity and structure in our design process. We utilized Logisim for detailed simulation, meticulously

testing each circuit component for accuracy and efficiency. The core design was then brought to life using Verilog in Vivado, leveraging FPGA technology for implementation. This approach, underpinned by rigorous testing and validation, ensured a robust and efficient digital multiplier, aligning with our project goals and digital design standards.

B. Implementation Details

- Main: The main module in the Verilog code serves as the central unit of the 8-bit signed multiplier project. It integrates various sub-modules like the clockDivider, debouncer, synchronizer, and CU, maneging their operations to achieve the multiplication process. Inputs include two 8-bit numbers and control buttons, with the output displayed on a 7-segment display. The module handles two's complement conversion for signed multiplication, sequential shifting of operands, and the control logic for multiplication. The debouncing of the control button input ensures stable operation, while the BCD and SevenSegmentScroller modules convert the multiplication result for display, showcasing a comprehensive design that covers both computational and user interface aspects.

- CU: The CU module functions as a Control Unit in our 8-bit signed multiplier project. This module is designed to generate control signals based on input conditions, which are instrumental in managing the data flow and operations within the multiplier circuit. Inputs like b0, z, and load determine the state of outputs such as Psel, EP, EA, EB, LA, LB, and RstP. These control signals directly influence various parts of the multiplier, like data paths and registers, ensuring accurate and efficient execution of multiplication and related operations. Essentially, the CU module acts as the decision-making heart of our digital multiplier system.

- BCD: The BCD module plays a crucial role in our 8-bit signed multiplier project. It's designed to convert binary input into Binary Coded Decimal (BCD) output, essential for displaying the binary multiplication results in a human-readable decemal format. The module initializes the BCD output with zeros and then loads the binary input. It then iteratively adjusts the BCD output, adding 3 to each 4-bit group exceeding 4, effectively translating the binary input to its BCD equivalent. This conversion is important for interfacing the multiplier's output with a 7-segment display or similar decimal display devices, making the results easily comprehensible.

- SevenSegmentDecoder: The sevenSegDecoder module is designed to drive a 7-segment display based on a 4-bit binary input. It decodes the binary number into the coresponding segments needed to display the number on a 7-segment display. Each binary input (from 0 to 9) is mapped to a specific pattern of the 7 segments to represent the decimal digits. The output is disabled, turning off all segments, when the enable signal is not asserted. This module is essential for displaying numerical information in a human-readable form on the 7-segment display.

- SevenSegmentScroller: The SevenSegmentScroller module is a dynamic display controller for a 7-segment display. It handles the scrolling and display of a 20-bit product on a 4-digit 7-segment display. The module uses a clock divider for timing, a counter for position tracking, and debouncers for button inputs. It changes the displayed digits based on left and right button presses, allowing the user to scroll through the product digits using an elegantly designed Finite State Machine (FSM). The seven-segment decoder instances convert each 4-bit segment of the product into the corresponding display output. Additionally, the module includes logic to display the sign of the product, enhancing the user interface of the digital multiplier project.

- ShiftLeftRegister: The ShiftLeftRegister module in Verilog is designed for shifting operations in digital circuits. It takes an 8-bit input number and shifts it left within a 16-bit output register. This module is triggered by a clock signal and controlled by load and enable signals. When the load signal is active, the input number is loaded into the lower 8 bits of the register, with the upper 8 bits zero-padded. If enabled, the register contents are shifted left each clock cycle, inserting a zero at the least significant bit. This module is crucial for operations requiring data manipulation or alignment in digital systems.

- ShiftRightRegister: The ShiftRightRegister module is designed to perform right shifting of an 8-bit input number. When the load signal is high, the input number is loaded into the register. If the enable signal is active, the module shifts the contents of the register to the right each clock cycle, filling the most significant bit with zero. This module is essential for bit manipulation tasks in digital circuits, particularly where data alignment or bit-wise operations are required.

- ClockDivider: The clockDivider module is integral to managing timing in our 8-bit signed multiplier project. It divides the frequency of the input clock signal, clk, by a specified factor, N, which is set to 500,000 by default. The module employs a counterModN instance for counting clock cycles, and the output clock signal, clk out, toggles its state when the counter reaches N - 1. This controlled clock pacing is crucial for synchronizing various components of the multiplier circuit, ensuring that operations are executed at a manageable and consistent rate.

- CounterModN: The counterModN module functions as a configurable counter in the digital multiplier project. It counts up to a predefined maximum value N, with the width of the counter defined by parameter X. The counter increments on each positive edge of the input clock clk when enabled by en. An asynchronous reset reset sets the counter back to zero. This modular counter is essential for creating time-controlled sequences or managing iteration processes within the larger digital system.

- Debouncer: The debouncer module is designed to stabilize the input signal, removing any noise or oscillations caused by mechanical switches or similar components. It uses a series of flip-flops (q1, q2, q3) to ensure that the input signal is stable over several clock cycles before asserting the output. This debouncing technique is crucial for reliable signal processing in digital systems, especially when interfacing with physical input devices.

- Synnchronizer: The synchronizer module is designed to synchronize an asynchronous input signal with a system clock in digital circuits. This synchronization is crucial for preventing metastability issues when integrating asynchronous signals into a clocked digital system. The module uses two flip-flops (q1 and q2) to align the input signal changes with the clock edges, effectively reducing the risk of unstable behavior in the system. The output signal reflects the input state, delayed by two clock cycles, ensuring it is properly synchronized with the system clock.

V. Validation Activities

Here are some test cases done as a sample to test the correctness of the implementation of the algorithm. All test cases are inputted to the FPGA and the result from the seven segment display are shown here.

- **12 x -15 = (00001100) x (11110001) = -180**

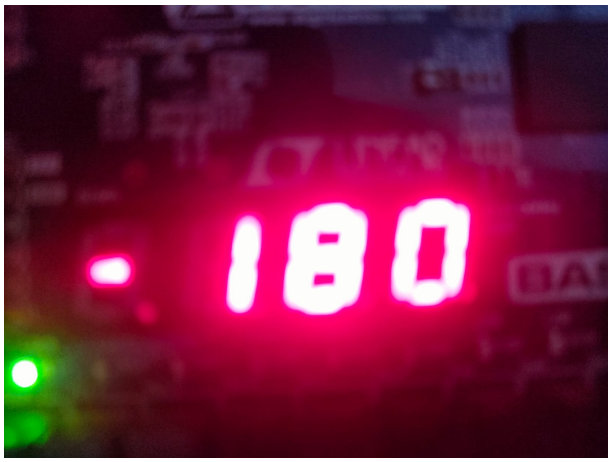

Fig. 9. FPGA Showing correct result: -180

- **64 x 64 = (01000000) x (01000000) = 4096**

Fig. 10. FPGA Showing correct result: 4096

Fig. 11. FPGA Showing correct result cont.

- **-32 x 1 = (11100000) x (00000001) = -32**

Fig. 12. FPGA Showing correct result: -32

- **-64 x -2 = (11000000) x (11111110) = 128**

Fig. 13. FPGA Showing correct result: 128

VI. Conclusion

In summary, our 8-bit signed multiplier project comprises the BCD module, converting binary input into a human-readable decimal format, The Control Unit (CU) serveing as the project's command center, managing data flow and operations, ClockDivider for timing management and control, while the Debouncer module ensures signal stability, essential for reliable input processing. The Main module acts as the project's core, handling the multiplication algorithm. The 7-segment display integration, with the SevenSegmentDecoder and SevenSegmentScroller modules, enhances result visibility and user interaction. Shift registers (ShiftLeftRegister and ShiftRightRegister) play a crucial role in bit manipulation, and the Synchronizer module handles synchronization between asynchronous signals and the system clock. Overall, the design approach we used was appropriate and well coordinated with careful and rigorus planning at the start that helped us reduce errors and complete the project much more effectively.