8x8 Signed Sequential Multiplier

Report

Adam Aberbach
Alyaman Huzaifa Massarani
Mohamed Khalil Brik

CSCE2301/230 – Digital Design I
Dr. Mohamed Shalan
FALL 2023

I. Overview

1- Objective

Efficient and accurate arithmetic processes are crucial in the ever-changing field of digital electronics and computer engineering. The focus of this project is to create and apply an 8x8 signed multiplier, which is a crucial element in the extensive field of digital computing. The main goal of this undertaking is to create a hardware module with the ability to perform multiplication on two 8-bit signed values. This job serves as both a practical application of digital design principles and a crucial learning exercise in comprehending the complexities of hardware-level arithmetic operations.

2- Significance

This endeavor holds significance that goes beyond its instructional worth. In the field of digital signal processing and computer arithmetic, the requirement for efficient and dependable multipliers is of utmost importance. These multipliers are essential components in a wide range of high-performance computing jobs, encompassing basic arithmetic operations as well as intricate algorithmic computations in modern microprocessors and embedded systems. The 8x8 signed multiplier is essential in these situations as it provides a harmonious combination of computational power and resource efficiency.

3- Functionality

The 8x8 signed multiplier functions based on the concepts of binary multiplication, which is a fundamental operation in digital systems. This project distinguishes itself by emphasizing signed integers, which introduces an added level of intricacy in managing the sign bits during multiplication. This element is vital for a diverse array of applications, particularly in systems where negative values are just as common as positive ones.

4- Application

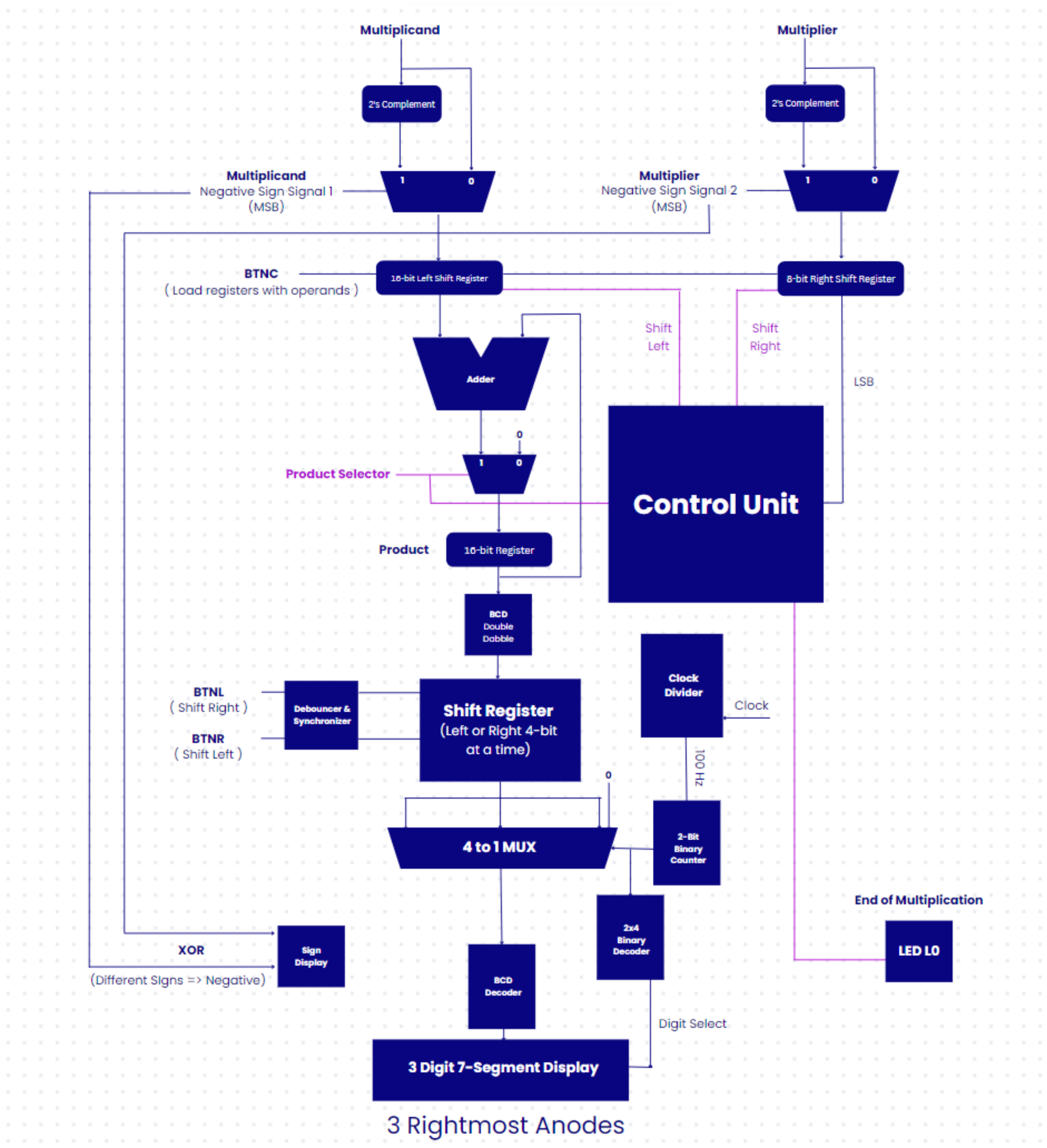
The applicability of such multipliers is extensive and diverse. They play a crucial role in the operation of microprocessors, enabling a wide range of arithmetic and logical functions. Moreover, within embedded systems, these multipliers play a crucial role in executing computing tasks, especially in applications that need signal processing, image processing, and other tasks that demand significant computational power. The design and successful implementation of this 8x8 signed multiplier not only showcase a profound comprehension of digital design concepts but also lay the groundwork for enhancements in the efficiency and capability of contemporary computing systems.

II. Project Requirements

The project requires the development of a sequential 8x8 signed multiplier using the shift and add method. This multiplier will be built on a Basys 3 FPGA board, which is equipped with an Artix 7 FPGA. Precise management of signed binary multiplication, ensuring there is no overflow or underflow, is a required assessed performance criteria. To execute the implementation, the FPGA board is detailed and its input, processing, and output capabilities are exploited. The inputs are regulated by toggle switches, while the output is displayed on seven-segment displays equipped with push buttons for scrolling and initiating the multiplication process. An LED will illuminate to indicate the completion of the multiplication procedure. The deliverables comprise a complete report and a meticulously documented Verilog source.

III. Block Diagram and System Design

1. Block Diagram Visual



2. System Design

The architecture of the 8x8 bit signed multiplier was deliberately designed to support the digital multiplication of signed binary integers efficiently. The initial step of this procedure involves two fundamental inputs, namely the multiplicand and the multiplier. Both of these inputs are 8-bit signed binary numbers. The system utilizes the 2's complement conversion to accurately handle the sign of the inputs. This guarantees that the multiplication operation can efficiently handle negative integers.

Once the inputs have been properly prepared, both the 16-bit left shift register and the 8-bit right shift register are activated. The purpose of these registers is to meticulously arrange the bits in anticipation of the multiplication procedure. The shifting process is crucial since it directly impacts the accuracy of the resulting product.

The adder component within the system is responsible for sequentially adding shifted multiplicands. This component is responsible for efficiently aggregating the partial products that are utilized to generate the ultimate multiplication outcome. Throughout this period, the product selector, under strict supervision from the Control Unit, assumes the responsibility of making pivotal decisions at various stages of the process. These judgments are made to ensure that the 16-bit register accurately stores the correct result of the multiplication.

Several BCD decoders are connected to seven-segment monitors to visually portray the outcome. The ultimate result is exhibited in a decimal format that is easily comprehensible due to these exhibits. To determine the output sign, an XOR gate is employed, which entails simultaneously examining the signs of the inputs. The result of this XOR computation is then shown on a dedicated sign display.

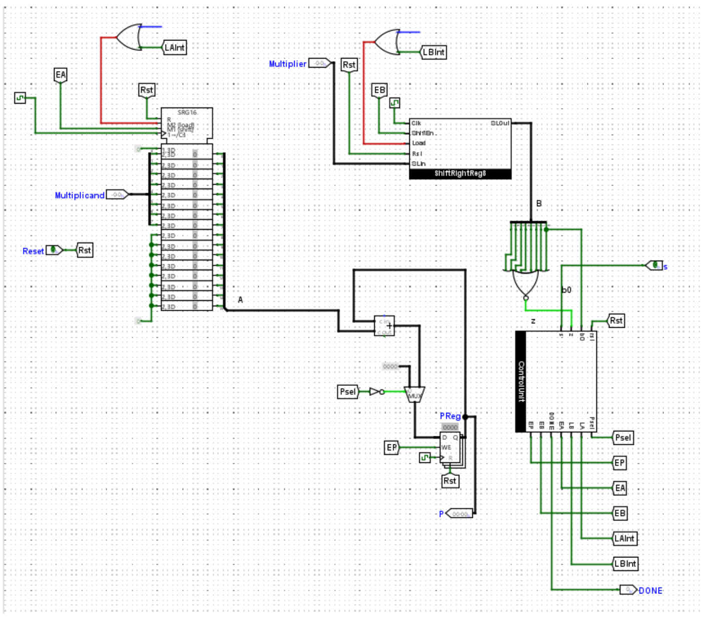
Operational control can be maintained by utilizing a sequence of buttons that oversee the load and shift activities. The buttons have the function of initiating the multiplication cycle, managing the shift operations, and ultimately signaling the completion of the multiplication process by means of LED indicators. From the user's perspective, this interactive interface enables the manipulation of the multiplication system in a comprehensible and more manageable manner.

IV. Logisim Evolution Implementation

The Logisim implementation of the 8 bit signed multiplier makes use of several components. These include the Sequential Multiplier, the Control Unit, the 2's Complement Converter… To avoid redundancy from the original Logisim implementation presented, we will be giving a glimpse of the first two components only.

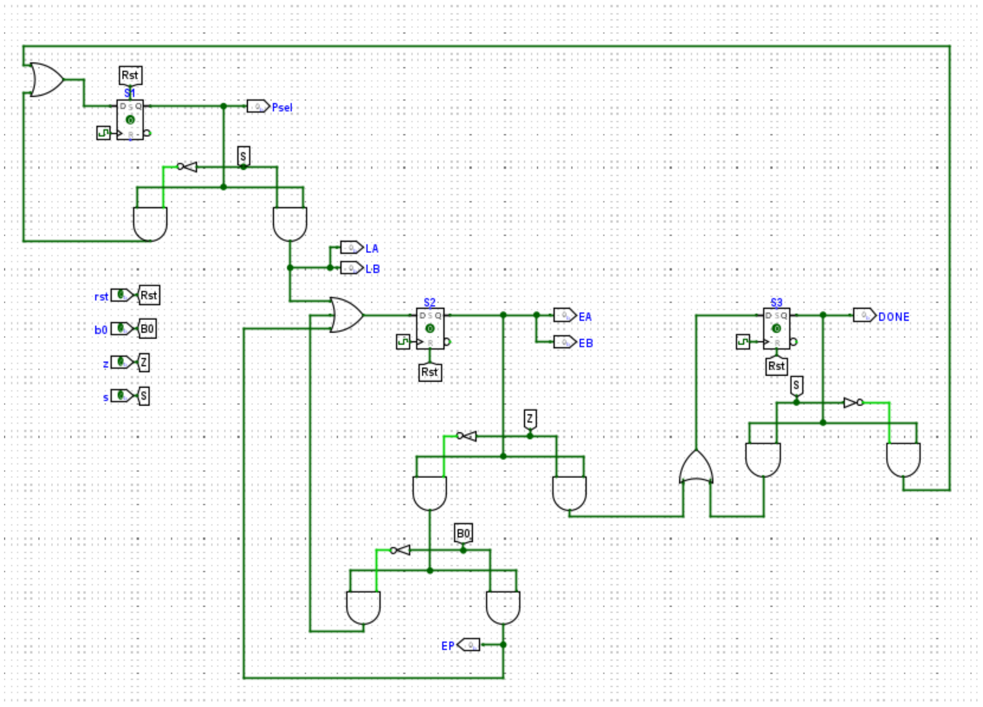
1. Sequential Multiplier

A sequential digital system utilized for a multiplier is depicted in the circuit diagram that is presented below. Two principal input buses, each of which is connected to registers, are referred to as the Multiplicand and Multiplier, respectively. The control signals known as 'EA' and 'EB' are responsible for enabling these registers. An indication of a sequential process shifting bits for multiplication is provided by the presence of a left shift register (SR16) for the multiplicand and a right shift register for the multiplier in the system. The components known as 'PReg' and 'Psel' are utilized for the purpose of product selection and storage. The circuit is equipped with reset lines for the purpose of initialization, as well as a 'DONE' signal that indicates the completion of a cycle.



2. Control Unit

The control unit seen in this circuit design utilizes a blend of logic gates and flip-flops to oversee the functioning of the 8x8 bit signed multiplier. The circuit incorporates control signals, such as 'Psel', 'EA', 'EB', and 'DONE', which serve to synchronize the data flow and operating states of the multiplier. In addition, there exist reset signals ('Rst') for system initialization, as well as multiplexers (represented by select inputs 'S') for the purpose of switching between various data pathways. The 'DONE' signal signifies the conclusion of the multiplication procedure.



V. Verilog HDL Modeling and FPGA Implementation

Module Description

This Verilog module describes a digital system that performs various operations based on input buttons. It takes two 8-bit inputs (num1 and num2) and performs a signed multiplication. Here's a breakdown of its functionality:

Input Handling: The module interfaces with several buttons (shift_right_btn, shift_left_btn, start_btn, rst_btn) and two 8-bit inputs (num1, num2). It outputs to a 7-segment display (segments, anodes), LEDs (sh_right_led, sh_left_led, rst_led, start_led, start_posedge_led), and other control signals (load, dir, en, done).

Button Edge Detection: PushButtonDetector instances detect the rising edge of button presses for reset, start, shift right, and shift left buttons. This is used to trigger various operations within the module.

MakePositive Modules: These convert num1 and num2 to positive values and also determine if they were originally negative. This is likely part of handling signed multiplication.

Multiplication and Binary to BCD Conversion: The SeqMult module performs sequential multiplication of the positive forms of num1 and num2, outputting a binary product (bin). The DoubleDabble module converts this binary number to its Binary-Coded Decimal (BCD) representation.

Shifting Operations: The module supports shifting the BCD representation left or right, controlled by the shift buttons. This is managed by the ShiftRegisterBidirectional module, which uses a clock divider (ClockDivider) for controlling shift timing.

Display: The Display3DigitsSign module drives a 7-segment display, showing the result of the multiplication. The sign of the result is determined by the XOR of the signs of the two inputs.

Start and Reset Control: The module has a start mechanism controlled by the start_btn and a reset functionality controlled by rst_btn. The start signal is toggled by the start_posedge, and reset is handled by rst_posedge.

This module is focused on signed multiplication with additional features like result display on a 7-segment display and shift operations for viewing different parts of the result.

```
module Main (
    input clk,
    shift_right_btn,
    shift_left_btn,
        start_btn,
        rst_btn,
        input [7:0] num1,
        num2,
        output [6:0] segments,
        output [3:0] anodes,
        output sh_right_led,
        sh_left_led,
        rst_led,
        start_led,
        load,
        dir,
        en,
        start_posedge_led,
        done
    );
      wire is_negative1, is_negative2;
      wire [7:0] num_positive1, num_positive2;
      wire [15:0] bin;
      wire [19:0] bcd, shifted_bcd_int;

      wire sh_right_posedge, sh_left_posedge, rst_posedge, start_posedge;
      wire clk_shr;
      reg  start = 0;

      //wire load, dir, en;

      assign sh_right_led = sh_right_posedge;
      assign sh_left_led = sh_left_posedge;
      assign rst_led = rst_posedge;
      assign start_led = start;
      assign start_posedge_led = start_posedge;

      PushButtonDetector rst_detector (
          .clk(clk),
          .rst(1'b0),
          .a  (rst_btn),
          .x  (rst_posedge)
      );
      PushButtonDetector start_detector (
          .clk(clk),
          .rst(1'b0),
          .a  (start_btn),
          .x  (start_posedge)
      );

      PushButtonDetector sh_right_detector (
        .clk(clk),
        .rst(1'b0),
        .a  (shift_right_btn),
        .x  (sh_right_posedge)
    );
    PushButtonDetector sh_left_detector (
        .clk(clk),
        .rst(1'b0),
        .a  (shift_left_btn),
        .x  (sh_left_posedge)
    );

    MakePositive #(8) mp_inst1 (
        .num(num1),
        .is_negative(is_negative1),
        .num_positive(num_positive1)
    );

    MakePositive #(8) mp_inst2 (
        .num(num2),
        .is_negative(is_negative2),
        .num_positive(num_positive2)
    );

    SeqMult seq_mult (
        .rst(rst_posedge),
        .clk(clk),
        .start(start),
        .multiplier(num_positive1),
        .multiplicand(num_positive2),
        .product(bin),
        .done(done)
    );

    // We need to load the contents of `bin` into the shift regiser module
  instance
    // when we finish the multiplication, then turn the `load` input of the
    // shift register off to allow for shifting.
    // In order to achieve this, a PushButtonDetector is used to detect the
    // positive edge of the `done` signal from the SeqMult and output it
    // to `load`.
    PushButtonDetector load_driver (
        .clk(clk),
        .rst(1'b0),
        .a  (done),
        .x  (load)
    );
```

```verilog
  DoubleDabble dd_inst (
      .bin(bin),
      .bcd(bcd)
  );

  assign en  = sh_left_posedge || sh_right_posedge;
  assign dir = sh_right_posedge ? 1 : 0;
  // This is to sync the timing of the ShiftRegister with
  // the timing of the shift_posedge signals.
  ClockDivider #(500_000) clkdiv_for_shr (
      .clk(clk),
      .rst(rst),
      .clk_out(clk_shr)
  );
  ShiftRegisterBidirectional shift_reg20 (
      .clk (clk_shr),
      .num (bcd),
      .load(load),
      .en  (en),
      .dir (dir),
      .out (shifted_bcd_int)
  );

  Display3DigitsSign display_unit (
      .clk(clk),
      .rst(rst_posedge),
      .negative_sign(is_negative1 ^ is_negative2),
      .dig0(shifted_bcd_int[3:0]),
      .dig1(shifted_bcd_int[7:4]),
      .dig2(shifted_bcd_int[11:8]),
      .en(1'b1),
      .segments(segments),
      .anodes(anodes)
  );

    always @(posedge start_posedge or posedge rst_posedge)
        if (rst_posedge) start <= 0;
        else if (start_posedge) start <= 1;
endmodule
```

VI. Testing and Validation

Multiple testbenches were used at the start of the development process in order to test and validate the modules. The testbenches used two main strategies, (a) $display("varName: %d", varname); and $monitor("varName: %d", varname); to display a variable's value at times of interest to debug the

modules, and (b) the Vivado simulation waveforms to get an overview of the whole process with all modules and their inputs and outputs and internal wires and registers.
In addition to the testbenches, multiple signals of the top-level module have been connected to some of the LEDs on the FPGA for debugging purposes during the hardware development process.
A wide range of numbers have been used for validation of the inputs and outputs, with diverse properties like different combinations of positive and negative numbers, and numbers varying in length from very short to very long (pushing the limits of 5 decimal digits).

VII. Discussion of Implementation Issues

The main issue of the implementation has to do with the reset button. Ideally, only a start button would be required where the user would change the numbers by toggling the switches, then press start and the new output would be calculated and displayed. Unfortunately due to some issues with internal registers of some modules, a reset button is also required to be pressed before the star button in order to reset the state of multiple modules (including the sequential multiplier itself).

VIII. Member Contributions:

Adam Aberbach:
- Implementation of the button functionality logic (Debouncer, Synchronizer, Rising Edge Detector).
- Implementation of the Clock Divider (100Hz) for the buttons, and the display on the FPGA.
- Designed the 8x8 bit signed Multiplier Block Diagram.
- Contributed to the writing of the project report.

Alyaman Huzaifa Massarani:
- Contributed to the writing of the project report.
- Implemented the display logic for displaying the digits on the 7-segment display.
- Implemented the bidirectional shift register for shifting the BCD representation of the product.
- Responsible for the final debugging process and making sure everything is working (in addition to the testbenches).
- Implemented the sequential multiplier, including both the datapath module (which is the top-level module of the multiplier), in addition to the internal control unit which uses a FSM.

Mohamed Khalil Brik:
- Contributed to the writing of the project report.
- Contributed to the Logisim design of the project (Double Dabble, shifting operations).
- Contributed to the Verilog code (Double Dabble, shifting operations).
- Contributed to the logic flow in the main module.