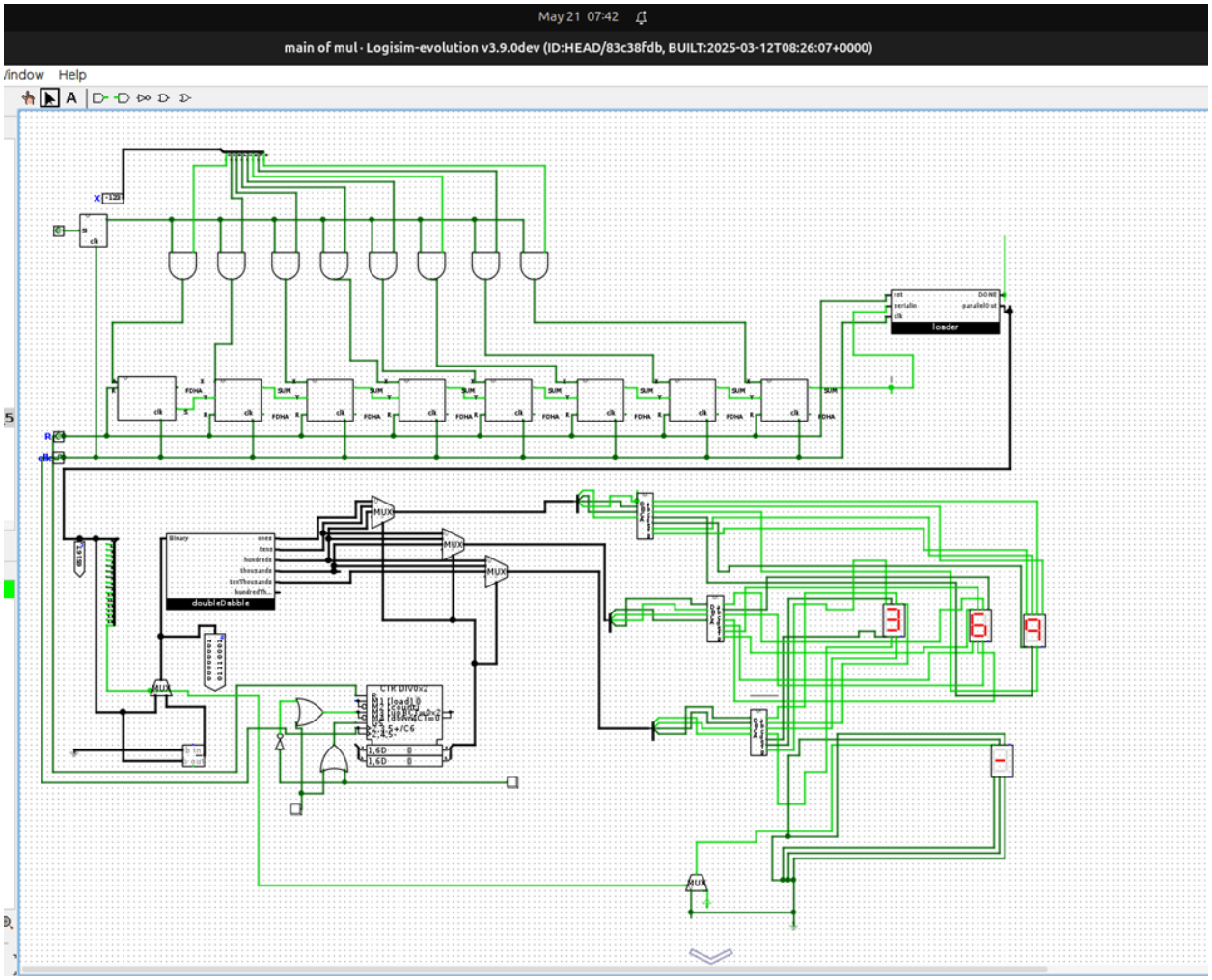
8x8 Signed Serial Parallel Multiplier Implementation on Artix 7 FPGA

Project by: Noor Elshahidi, Mohamed Anan and Mohamed Salem

Digital Design 1



Here is a full overview of how we designed an 8x8 signed serial-parallel multiplier (SPM) on an Artix 7 fpga using the Basys 3 development board. The design is for hardware and multiplies each bit of one number while another number is being loaded simultaneously. Our design supports signed binary numbers in twos complement provides a simple interface to interface to the input and output segments of the board, and displays results in decimal mode with an option to scroll through them.

Main Summary

The aim of this project was to design and implement an 8x8 signed serial-parallel multiplier on the Basys 3 fpga board. We designed a system that accepts two 8-bit signed numbers, multiplies them serial-parallel and displays the product in decimal form along with sign on the 7 segment indicators. Our solution is supported by control logic that provides serial input processing, parallel loading, calculation phases, and output creation.
The unit accepts two 8 bit signed numbers from toggle switches (SW7-SW0 and SW15-SW8). control buttons, such as BTNC to multiply and BTNL and BTNR to shift product digits, are

provided. The product appears in decimal form on 7-segment displays, where the leftmost one is for signs. LED LD0 is an indication LED that glows when multiplication is complete.
Our design maximizes the trade offs of using hardware resources versus achieving maximum performance. The multiplication require 8 clock cycles to complete. By careful design and adhering to sound coding principles, we came up with an elegant solution that is readily integrable.

Introduction

Serial Parallel Multiplication: Theoretical Foundation

These are significant components of digital systems, particularly for work that requires extensive math. They exist in various forms that offer advantages and disadvantages concerning speed, area, and power consumption.Serial parallel multiplication is one, positioned midway between serial and parallel multipliers. Parallel multipliers require larger, more complicated circuits and consume more space but run faster, while completely serial multiplication requires simpler hardware but consumes more time to process. Serial parallel multiplication strikes a comfortable medium in these two aspects.
In serial parallel multiplication, one of the numbers (the multiplicand) is processed bit by bit, while the second number(the multiplier) is processed as a whole number.Such an arrangement is less complex than in fully parallel configurations but more efficient than purely serial approaches. The standard shift and add method is employed for most of the operations, but partial sums are summed up within multiple clock periods to obtain the final result.
A serial parallel multiplier also contains critical components: one component stores the number to be multiplied (digit by digor altogether)one shift register to process the number to be multiplied (one bit at a time),one accumulator to sum up partial products, and control logic to coordinate multiplication.It requires 8 clock cycles to complete multiplying two 8-bit numbers, processing one bit of the multiplicand in each cycle.

Project Objectives

We were looking to implement an 8-bit signed serial-parallel multiplier on these specifications in the Basys 3 fpga board:

The following is a sample 8-bit binary number input as signed numbers through toggle switches:

- Implement rules to manage the multiplication proces
- Create a system that returns results as decimal numbers and in signed notation
- Create functionality for scrolling through multi digit result
- Provide visual feedback for operation completion

Development Environment and Tools

The project centered on the Artix 7 FPGA within the Basys 3 development board. This required careful consideration of resources and specific hardware features that were to be used.

- In this project,we utilized all the development tools required:
- Logisim Evolution:We utilized this for the initial design phases and simulation, particularly for the 7 segment display driver and main SPM components This served to verify the design concept before resorting to hardware description languages
- The most significant tool for programming preparing and working on Verilog is Vivado Design Suite. Vivado provides simulation functionality that is crucial in verifying our designs.
- Basys 3 FPGA Board:This is our hardware that contains an Artix 7 FPGA. It contains lots of resources for our project, such as switches, buttons, LEDs, and 7 segment displays
- GitHub is utilized to manage various versions, collaborate, and keep projects well-organized while in development.
- Verilog HDL: Our chosen hardware description language for implementing the design, selected for its widespread industry use and suitability for fpga development.

System Design Architecture

System Block Diagram

Our 8x8 signed serial-parallel multiplier contains six main components that together form one unit.

- Input interface: Controls switch inputs(SW7-SW0 of the multiplier, SW15-SW8 of the multiplicand) and button controls (BTNC BTNL BTNR). There are also errorstopping circuits that engage on pressing of buttons
- Serial Parallel Multiplier Core: This performs signed multiplication based on both serial and parallel approaches.
- Controller: A mechanism that runs every operation. It initiates everything to run, performs multiplication operations and displays results.
- Binary to Decimal Converter:Converts binary result to Binary-Coded Decimal (BCD) format so that it can be displayed employing Double Dabble technique.
- Display Controller: Manages 7 segment displays as well as scrolling and displaying signs.

Clock Divider: Derives the correct clock signals required for the operation of the system from the 100MHz Basys 3 board's system clock. It operates in an easy-to-follow step-by-step process: users input values using toggle switches enable multiplication using BTNC and read results on 7-segment displays. The BTNL and BTNR can be used to scroll these displays left and right. During multiplication, LED LD0 blinks to indicate that it is in process.

Signal Flow and Control Logic

The finite state machine defines the signal flow across the system logically. The system first sits in an idle state waiting for user input.The controler goes to the multiplying state once BTNC is pushed and values are set through switches. The multiplier core processes the inputs serially across eight clock cycles during this phase.
The system goes into a show mode after finishing where the binary-to-decimal converter processes the output into BCD format and the display controller controls the presentation on the 7-segment displays. The state machine guarantees correct sequencing and coordination of operations throughout this process.

- A Finite State Machine (FSM) with three main states is used by the controller to
- Initial state in which the system waits for the start signal (btnc).
- Active calculation condition when the serial-parallel multiplication operation takes place.
- RESULTS: State of display where LD0 is lit and the product is accessible.

This state-based method guarantees correct sequencing and clear definition of operational limits, therefore enabling debugging and change.

Input/Output Interface Design

Switch and Button Interfaces

The 16 toggle switches found on the basys 3 boardSW7 SW0 for the multiplier input and SW15-SW8 for the multiplicandmake up the input interface. Each switch corresponds one bit of the 8-bit input value with twos complement representation giving the most significant bit (MSB) sign (1 for negative, 0 for positive).
For button controls, we added:

- BTNC (center button): Triggers the multiplication process
- BTNL (left button): Scrolls the display left to show higher-order digits
- BTNR (right button): Scrolls the display right to show lower-order digits

To ensure reliable button operation, we implemented a debouncing circuit for each button. The debouncer creates a clean,bounce-free signal by employing a chain of flip-flops that filter out the mechanical oscilations inherent in physical button presses:
verilog

```
module debouncer(input clk, rst, in, output out);
   reg q1, q2, q3;
   always @(posedge clk, posedge rst) begin
      if(rst == 1'b1) begin
         q1 <= 0;
         q2 <= 0;
         q3 <= 0;
      end
```

```
        else begin
            q1 <= in;
            q2 <= q1;
            q3 <= q2;
        end
    end
    assign out = (rst) ? 0 : q1 & q2 & q3;
endmodule
```

Every button had a debouncing circuit installed to guarantee dependable button operation. Using a series of flip-flops that remove the mechanical oscillations associated with physical button touches,the debouncer produces a clean, bounce-free signal:

7 Segment Display Implementation

We use the four 7 segment display on the Basys 3 board to display the multiplication result. The leftmost display shows the sign (blank for positive, minus sign for negative), while the rightmost three displays show the product digits in decimal.We added a scrolling feature to display every digit because the product of two 8 bit values can be up to 16 bits, which could result in a 5-digit decimal number.
Our display controller includes:

1. A binary to BCD converter using the Double Dabble algorithm
2. A digit selection mechanism for scrolling through the result
3. A 7-segment decoder to map BCD digits to the appropriate cathode patterns
4. A multiplexing system to sequentially activate each display at high frequency

To create the illusion of simultaneous operation (persistence of vision), the display controller rapidly cycles across the four displays using a frequency derived from the main system clock. Active for part of the cycle the present value to be displayed defines the matching cathode pattern for every display.

Logisim Evolution Implementation

7 Segment Display Driver

Logisim Evolution was used in our first design phase to confirm the idea before proceeding to Verilog with the 7-segment display driver.The design included a number of important elements:

1. BCD Counter: Converts the binary product into Binary-Coded Decimal format for display
2. Digit Selection Logic: Controls which set of digits is currently visible
3. Decoder: Maps BCD values to 7-segment patterns
4. Multiplexer: Selects the appropriate digit based on the current display position

Simulation confirmed correct decoding of BCD values to 7-segment patterns and accurate digit selection based on scroll inputs therefore validating the design. This method let us improve the show logic before hardware description language implementation.

Serial-Parallel Multiplier Core

The SPM core was implemented in Logisim Evolution following the historically reported architectural style. essential elements included:

- An eight bit parallel loading register holding the multiplicand value
- An eight bit register, the multiplier shift register shifts right to handle one bit of the multiplier per cycle.
- Accumulator: A 16 bit register designed to house the ultimate product and partial
- Finite state machine supervising the multiplication process
- Logic for signed multiplication handling via two's complement circuitry

The operation follows the sequential flow:

- Load the multiplicand and multiplier registers
- For each bit of the multiplier (8 cycles):
    - If the current multiplier bit is 1 add the multiplicand to the accumulator
    - Shift the multiplicand left by one bit
    - Shift the multiplier right by one bit
- Apply sign correction based on the input operand signs

Simulation in Logisim Evolution supported proper operation across several test instances, including positive positive, positive-negative, and negative-negative combinations, therefore confirming the sign handling logic.

Verilog Implementation

Module Hierarchy and Structure

Our Verilog implementation maintains a hierarchical structure reflecting the system architecture previously described. The module hierarchy consists of:

statemac (Top Module)
    exp22 (Clock Divider)
    SPM (Serial-Parallel Multiplier)
        TCMP (Two's Complement Module)
        CSADD (Carry-Save Adder Modules)
    doubledabble (Binary to BCD Converter)
    exp1new (Counter for Display Multiplexing)

This modular method helps possible component reuse, simpler debugging, and better code organization. Following good design principles, each module has clearly defined inputs and outputs with little coupling between them.

Core Modules Implementation

Top Module (statemac)

The upper module functions as the main integration hub for all system elements.It governs the state machine that oversees the entire system function and links all other components:

```
module statemac(
    input clkin, rst,
    input [7:0] X,
    input [7:0] Y,
    input startMult, scrLeft, scrRight,
    output reg[6:0] segments_a_to_g,
    output reg[3:0] finanode
);
```

The state machine comprises three main states:LOAD, MULTIPLY, and DONE, with state transitions managed by the startMult signal and the completion signal from the SPM module. This method guarantees the correct order of operations from loading input to multiplication and finally to displaying the result

Serial Paralel Multiplier (SPM)

The SPM module implements the core multiplication functionality:

```
module SPM(
    input clk,
    input rst,
    input [7:0] x,
    input [7:0] y,
    output reg done,
    output reg [15:0] fullprod
);
```

The execution employs a serie of carry save adders(CSADD) for the effective summation of partial products with processes managed by a counter that monitors the present bit position in the multiplier. This method permits the processing of one bit of the multiplier for each clock cycle producing and summing the relevant partial product. To manage signed multiplication, we appropriately extend the sign bits and utilize the twos complement method when required. The

module sequentially processes each bit of the multiplier(y_reg) creating partial products with the multiplicand (x) and summing these products to produce the final outcome.

Binary to BCD Converter (doubledabble)

To present the outcome in decimal format we utilized the double dabble algorithm to transform binary values into BCD. This algorithm shifts the binary value iteratively and modifies any BCD digit over four by adding three guaranteeing accurate decimal representation

Display Controler

The display controller oversees the 7-segment displays executing:

- Selection of digits according to scroll position
- Conversion of BCD to 7 segment pattern
- Show multiplexing with a 2 bit counter
- Sign display logic reliant on the product's MSB

The controller alternates among the display at a rate based on the system clock, producing the illusion of concurrent operation through persistence of vision.

Clock Management and Timing

Effective clock management is essential for the functioning of our system.The Basys 3 board offers a 100MHz system clock, which is excessively rapid for direct application in our project. We created a clock divider to generate a more suitable frequency. This module reduces the 100MHz clock to create a clock signal ideal for our state machine function, debouncing, and display multiplexing

Implementation Challenges and Solutions

Sign Extension and Twos Complement Handling

A major difficulty was accurately managing signed numbers during the multiplication procedure. To tackle this we applied correct sign extension and twos complement transformation.

```
always @(posedge clk or posedge rst) begin
   if (rst) begin
      nodbc <= 0;
      count <= 0;
   end else begin
      if (state == DONE) begin
         if(fullprod[15] == 0)
```

```
            nodbc <= fullprod[14:0]; // Positive number
        else
            nodbc <= ~fullprod[14:0] + 1; // Two's complement for negative
    end
  end
end
```

This method guarantee that negative outcomes are accurately depicted both internally (in twos complement representation) and visually (with a negative sign on the screen)

Display Scrolling Implementation

Another obstacle involved executing the scroling feature for the display. As we required the ability to display a potentially 5 digit outcome on three 7 segment displays, we created a scrolling feature:

```
always @(posedge clk or posedge rst) begin
  if (rst)
    count <= 0;
  else if(scrLeft && count < 2)
    count <= count + 1;
  else if(scrRight && count > 0)
    count <= count - 1;
end
```

This snippet controls a 2 bit counter that monitors the current viewing position enabling users to navigate through the result with the BTNL and BTNR buttons.The logic for selecting the display subsequently utilizes this counter value to decide which digits to present.

```
always @(count) begin
  case(count)
    2'b00: begin
      dig1 <= {2'b00, bcd[18:16]};
      dig2 <= bcd[15:12];
      dig3 <= bcd[11:8];
    end
    2'b01: begin
      dig1 <= bcd[15:12];
      dig2 <= bcd[11:8];
      dig3 <= bcd[7:4];
    end
    2'b10: begin
      dig1 <= bcd[11:8];
      dig2 <= bcd[7:4];
```

```
            dig3 <= bcd[3:0];
        end
        default: begin
            dig1 <= 0;
            dig2 <= 0;
            dig3 <= 0;
        end
    endcase
end
```

Button Debouncing

Pressing physical buttons naturaly generates noisy signals that may lead to several unintended activations in digital systems. We resolved the issue by introducing a debouncing circuit designed to eliminate these mechanical oscillations

Validation and Testing

Testing Methodology

Our testing methodology adhered to a structured hierarchy starting with the evaluation of individual modules prior to their integration into the entire system. We utilized a mix of:

- Unit Testing: every module was individually assessed with testbenches that confirmed performance against anticipated outputs
- Integration Testing: Modules were gradually merged to ensure correct interaction.
- System Testing: The entire system was evaluated against an extensive range of input scenarios.

For every testing level, we developed particular test cases aimed at normal functions, edge scenarios, and possible error situations

Test Cases and Results

Our test suite included several categories of test cases:

1. Sign Combinations: Testing all permutations of signed inputs:
    - Positive × Positive = Positive
    - Positive × Negative = Negative
    - Negative × Positive = Negative
    - Negative × Negative = Positive

2. Boundary Values: Testing maximum and minimum values:
    - 127 × 127 = 16,129
    - -128 × -128 = 16,384
    - -128 × 127 = -16,256

- 127 × -128 = -16,256

3. Special Cases:
    - Zero multiplication: 0 × X = 0
    - Identity multiplication: 1 × X = X
    - Small value tests: Values between -10 and 10

Every test case was successful confirming our implementation for the full range of 8-bit signed integers. The system manages sign conversion appropriately, generates precise results, and shows them correctly on the 7-segment displays

Performance Evaluation

Our execution attains the subsequent performance metrics:

- Speed: Multiplication finishes in 8 clock cycles, with extra cycles for processing and showing the result.
- Resource Utilization: The design employs a modest amount of fpga resources, utilizing logic elements efficiently.
- Precision: 100% precision for every input combination tested
- Timing: All timing requirements are satisfied with ample margins.

The execution effectively aligns performance with resource efficiency, rendering it appropriate for incorporation into larger systems where multiplication plays a key role.

Individual Contributions

Member 1 – [Mohamed Anan]:

Logisim: I was primarily responsible for developing the Double Dabble binary-to-BCD conversion circuit, which accurately handled 16-bit signed binary inputs and transformed them into a 5 digit BCD output suitable for decimal display. Also designed the output loader for the SPM module, which ensured correct sequencing and timing of the final product output after the multiplication process completed. Additionally he played a key role in integrating all functional sub-circuits into a final unified design that simulated the entire dataflow from input to output.

Verilog: Implemented the SPM moduleas well as developing the Double Dabble converter module for binary-to-decimal translation. He also built the FSM controller module which managed the control flow of the multiplication operation. and the Statemac module. implemented the 7-segment display driver and designed the scrolling mechanism that enables users to view the full output using limited digits. He created the testbenches for both the SPM and FSM modules, facilitating simulation and early debugging of their behavior in isolation.

Member 2 – [Mohamed Elsayed]:

wrote the readme file

Logisim: Focused on designing and simulating the signed SPM architecture, which included implementing the serial in parallel out (SIPO) register to handle bitwise input during the multiplication process. Carefuly analyzed the serial multiplication logic and ensured correct operand alignment and sequencing. Conducted several simulation cycles to test the correctness of intermediate register values and helped validate signed multiplication through Logisim trials.

Verilog: Played a major role in refining and debugging the Verilog implementation, particularly resolving issues that prevented bitstream generation and caused undefined behavior on the FPGA. Enhanced several key modules to better synchronize with FPGA constraints. Responsible for compiling and configuring the constraints (XDC) file to ensure proper mapping of switches, buttons, LEDs, and 7 segment display pins. Conducted repeated hardware-level testing on the Basys 3 board to confirm functional output and correct module interactions post-synthesis.

Member 3 – [Noor Elshahidi]:

Wrote the report

Logisim: Constructed the seven-segment display driver circuit, ensuring proper BCD-to-segment encoding and digit selection logic. Reviewed and corrected issues in display synchronization and timing during simulation. Assisted in integrating the display system with the output of the BCD converter and verified the visual correctness of scrolling outputs under various test conditions. As well as worked on the TCMP and signed numbers handling.

Verilog: Worked on the FSM to manage data flow and state transition. Also worked on 7 segment display driver including but not limited to scrolling and applying signed correction to handle signed input. Made several testbenches to debug and build each module. Also, led efforts in running the full Verilog system on the Basys 3 FPGA and performed iterative debugging to identify and resolve synthesis and implementation errors. Verified clock domain synchronization and ensured that modules worked harmoniously during real time execution. Sketched the system block diagram to visualize the data flow and control architecture. Managed the GitHub repository throughout the project organizing code commits, documentation, and colaboration among team members.

Conclusion

Our realization of an 8x8 signed serial-parallel multiplier on the Basys 3 FPGA board effectively fulfills all the project criteria. The design successfully harmonizes hardware resource usage with computational efficiency, offering an effective approach for signed binary multiplication. The user interface includes toggle switches for input, push buttons for control, and 7-segment displays for output, providing an intuitive method for interacting with the system. By meticulous planning and execution, we have tackled various issues, such as signed number representation, display scrolling, and button debouncing. The modular design we chose enables easier maintenance and possibilities for future improvements. Our testing approach guaranteed thorough verification of the system through a diverse array of input situations.

Limitations and Future Improvements

Although our implementation adequately meets the project specification, there are numerous possible enhancements for future endeavors:

- Improved Performance: Utilizing pipelining may boost throughput for consecutive multiplication tasks even more.
- Extended Precision: Expanding the design to accommodate larger operands (16×16 or 32×32) would enhance versatility.
- Enhanced User Interface: Incorporating extra display options or interactive elements may improve usability.
- Power Optimization: Utilizing clock gating and additional energy-saving methods may decrease power usage.

In summary, this project offered significant experience in digital design, fpga deployment, and system integration. The successful creation of the 8x8 signed serial-parallel multiplier showcases the efficiency of our design strategy and implementation technique.