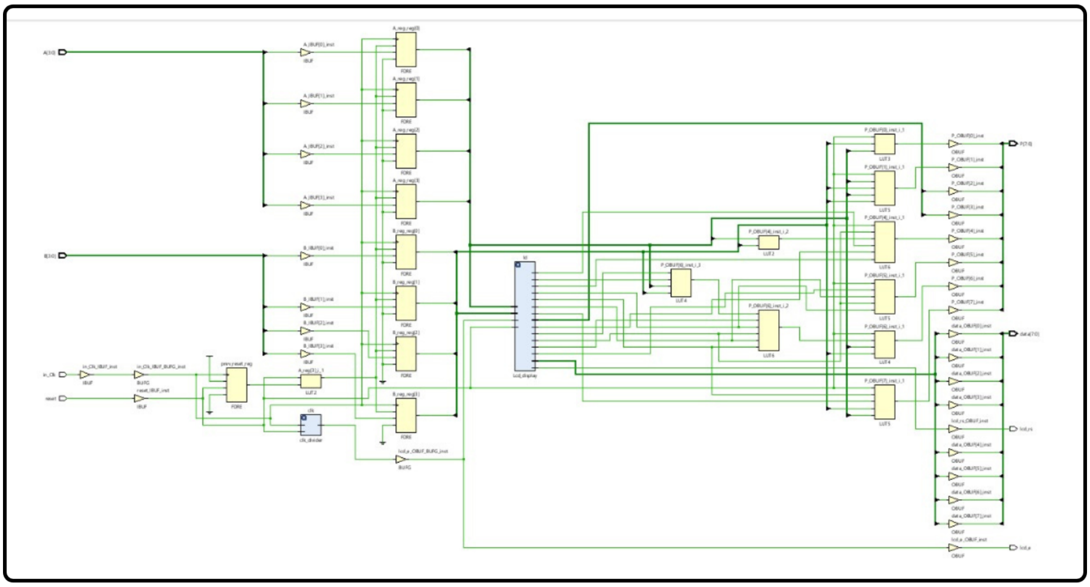


EE2016: Assignment-3

By Siva(EE23B151),
Bhavesh(EE23B017)
and Naveen(EE23B113)

Group 28

Date: 30/8/2024

Indian Institute of Technology Madras

Contents

1 Experiment Objective

The objective of this experiment is to use the full adders and half adders created during Experiment 1, along with the clock divider from Experiment 2, to implement the Wallace algorithm for the multiplication of two 4-bit binary numbers. The output will be displayed on both the LCD display module and LEDs.

1.1 Overview

- In Experiments 1 and 2, we have designed essential components of a microprocessor and implemented them on the FPGA board. These components include full adders, half adders, and a clock divider, which will be reused in this experiment.
- In this experiment, we will design a multiplier (termed Wallace multiplier) using Verilog for a pair of four-bit unsigned numbers. This design will utilize the previously developed components to optimize the multiplication process.
- The experiment will include performing a simulation to verify the functionality of the Wallace multiplier. The simulation will help in checking the accuracy and timing of the multiplication process.
- We will demonstrate the results of the Wallace multiplication using the LEDs on the FPGA board. This provides a visual representation of the binary multiplication output.
- Additionally, we will learn how to write Verilog code to output a set of characters on the LCD that is accessory to the FPGA board. This skill is crucial for displaying textual or numerical information to the user in a more readable format.
- An optional part of this experiment is to show the product of two unsigned numbers directly on the LCD. This involves converting the binary output of the Wallace multiplier into a suitable format for display, thereby enhancing the user interface.

2 Wallace Algorithm

2.1 What is Wallace Algorithm

Wallace multiplication, commonly referred to as a Wallace Tree multiplier, is an efficient method for performing binary multiplication using a specific type of adder tree to reduce the partial products.

2.2 Steps Involved

- **Partial Product Generation:**
  Each bit of the multiplicand is multiplied with each bit of the multiplier. This process generates multiple rows of partial products, similar to the long multiplication method taught in basic arithmetic. For an $n \times n$ multiplication, n rows of n bits each are generated.
- **Partial Product Reduction:**
  The key feature of Wallace multiplication is how it reduces these partial products. Instead of using a straightforward sequential approach, it uses a hierarchical tree structure of adders (half adders and full adders) to sum these products in parallel.
- **Final Addition:**
  Once all partial products are reduced to just two rows, a conventional adder (such as a carry-propagate adder) is used to add these two rows and obtain the final product.

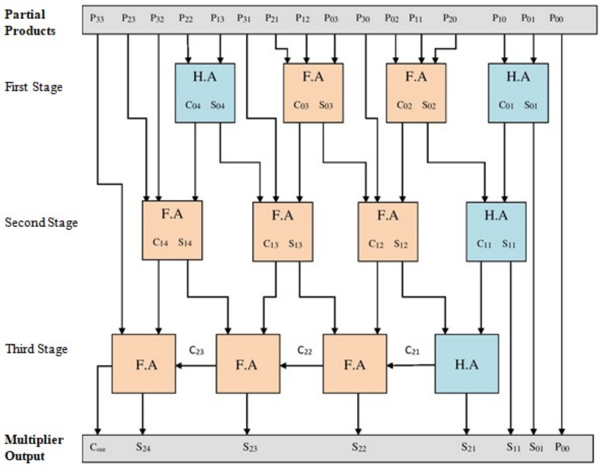


Figure 1: Tree Diagram for 4-bit × 4-bit Multiplier

2.3 Advantages of Wallace Multiplication

- **High Speed:** Wallace multipliers are significantly faster than traditional array multipliers because of their parallel processing nature. The tree structure allows for simultaneous reduction of partial products.
- **Reduced Delay:** By reducing the height of the adder tree, the critical path (the longest path delay through the circuit) is minimized. This results in lower overall computation time.
- **Scalability:** Wallace multipliers can be easily scaled to handle higher-bit multiplication, making them suitable for a wide range of digital applications.

3 LCD Interface

The objective is to use the LCD display to provide a clear visual representation of processed data.

3.1 Steps for Interfacing

1. **Set Up LCD Connections:** Connect the LCD display to the FPGA board using appropriate GPIO pins. Ensure that data lines, control lines (RS, RW, EN), and power supply are connected correctly. Proper wiring is critical for reliable communication between the FPGA and LCD.

2. **Clock Signal Generation:** Utilize a clock divider to generate a suitable clock signal for the LCD interface. This clock signal must align with the timing specifications required by the LCD for data processing and display.

3. **Initialization of LCD:** Write initialization routines to configure the LCD for operation. This includes sending specific commands to set up display settings, such as the cursor position, display on/off control, and clearing the display. Initialization is crucial for the LCD to function as intended.

4. **Data Conversion for Display:** Convert binary or numerical data into a format suitable for the LCD. This often involves converting raw binary data into ASCII characters that the LCD can display. Proper data formatting ensures that the information is readable and correctly represented.

5. **Data Transmission to LCD:** Implement logic to transmit the converted data to the LCD. Each character or data unit must be sent in the correct sequence with appropriate timing to ensure that the LCD displays the information accurately.

6. **Testing and Debugging:** Verify the correct operation of the LCD by testing with known input values. Use the debugging process to ensure that the LCD displays the intended output accurately and consistently. This step helps identify and fix any issues with data transmission or display settings.

3.2 Important Commands for Controlling the LCD

To effectively interface and control the LCD display with the FPGA, specific commands need to be sent. These commands help initialize the LCD, configure its settings, and manage the display of characters. Some key commands include:

- **0x01 - Clear Display:** Clears all characters on the LCD screen and returns the cursor to the home position (top-left corner).

- **0x02 - Return Home:** Moves the cursor to the home position without clearing the display. This is useful for resetting the cursor after writing data.

- **0x06 - Entry Mode Set:** Configures the cursor movement direction (typically increment) and display shift (typically no shift). This setting is crucial for sequential writing.

- **0xC0 - Force cursor :** To begin from starting of 2nd line .

- **0x80 - Set to begin :** Sets the starting address of the display data RAM (DDRAM). This command is used to move the cursor to start from the 1st line of display

- **0x38 - Function Set:** Configures the LCD for 8-bit mode, 2-line display, and 5x7 character font. This command sets the basic operation mode of the LCD.

3.3 Control Signals: LCD_RS and LCD_E

- **LCD_RS (Register Select):**
  - The 'LCD-RS' signal is used to select between command mode and data mode.
  - When 'LCD-RS' is low (0), the LCD is in command mode, and instructions are sent to the LCD to control its operation (e.g., clear display, set cursor position).
  - When 'LCD-RS' is high (1), the LCD is in data mode, and the data sent to the LCD is treated as display data to be shown on the screen.
- **LCD_E (Enable):**
  - The 'LCD-E' signal is used to enable the LCD to read the data or command from the data lines.
  - The 'LCD-E' signal is pulsed high (1) to latch the data or command present on the data lines into the LCD.
  - A typical sequence involves setting 'LCD-E' high, waiting a short period, and then setting 'LCD-E' low. This pulse ensures that the LCD registers the data or command correctly.

3.4 Considerations

- **Timing Constraints:** Ensure that timing constraints between the FPGA clock and LCD are met. Proper timing is crucial to avoid data corruption and ensure stable display.
- **Power Supply:** Verify that both the FPGA and LCD have stable and appropriate power supplies to avoid malfunction. Fluctuations in power can lead to unreliable operation.
- **Initialization Sequence:** Follow the correct initialization sequence for the LCD to ensure it starts up in the desired mode. Incorrect initialization can result in improper display or failure to display data.

3.5 Conclusion

Interfacing an FPGA with an LCD display provides an effective way to visually present data in real time. We have encountered many issues in making the LCD display work in the way we wanted .

4 Tasks

4.1 Task 1: Implementing Wallace multiplier using Verilog

Code explanation

Firstly, we define the module for wallace multiplier with a reset and two 4-bit inputs for numbers A and B, and an 8-bit output ‘m’ which contains the product of A and B.

Now we define wires of size ‘4’ for the ‘partial products’. We also define wires for taking the carry as well as the sum as seen in the tree diagram.

We implement Half adders and Full adders to perform Wallace multiplication as seen in the tree diagram.

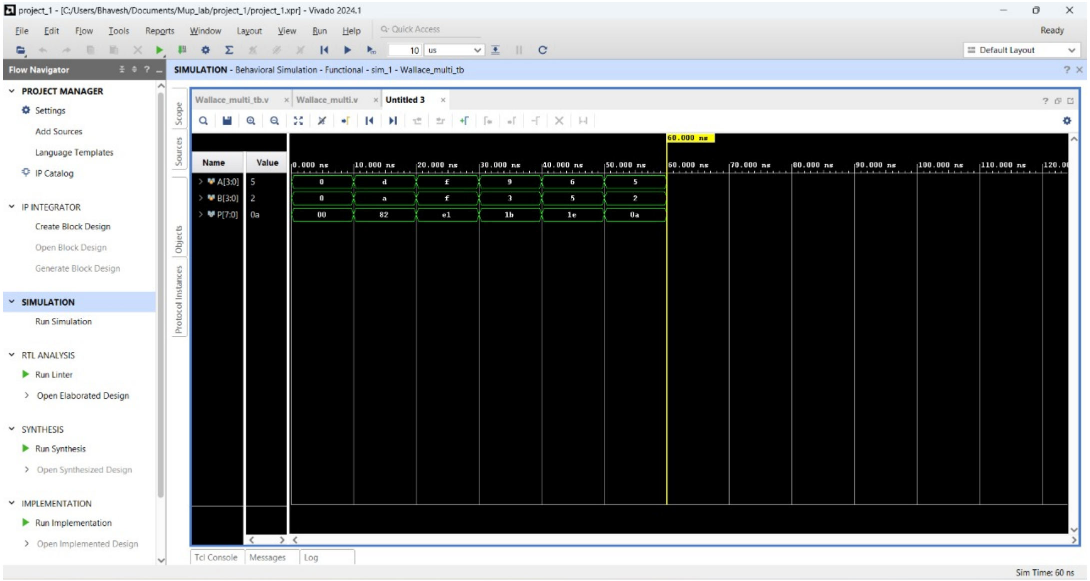
4.2 Task 2: Simulate Wallace Multiplier using Testbench

We created a module for testbench which contains 4-bit registers for ‘A’ and ‘B’ (inputs). We then declare wire ‘m’ for getting outputs from the Wallace multiplier module which is instantiated in the testbench module.

Verilog Codes for Task 1 and Task 2

Click on the blue hyperlinks:

- **Half Adder**: contains the Half-Adder module.
- **Full Adder**: contains the Full-Adder module.
- **Wallace Multiplier**: contains the main module.
- **Testbench**: used for testing the Wallace multiplier module.



project_1 - [C:/Users/Bhavesh/Documents/Mup_lab/project_1/project_1.xpr] - Vivado 2024.1

SIMULATION - Behavioral Simulation - Functional - sim_1 - Wallace_multi_tb

Name	Value	0.000 ns	10.000 ns	20.000 ns	30.000 ns	40.000 ns	50.000 ns
A[3:0]	5	0	d	f	9	6	5
B[3:0]	2	0	a	f	3	5	2
P[7:0]	0a	00	82	e1	1b	1e	0a

Figure 2: Simulation graph

4.3 Task 3: Showing outputs of multiplier on LEDs in FPGA

This is done in the **bonus task** along with LCD display. Instead of hard-coding inputs A and B in the code itself, we made the code such that it accepts inputs from user via the switches in FPGA board. **Constraint file**: for LED display alone.

4.4 Task 4: To display characters in LCD screen

The hex values of ‘1’ to ‘9’ are 8’h31 to 8’h39 and for ‘A’ to ‘G’, they are from 8’h41 to 8’h47. (excluding 0x40 which is ‘@’)

The commands are as follows:

- 0: 8’h38, control signal to display on two lines.
- 1: 8’h0C, keeps display on but cursor off.
- 2: 8’h06, moves the cursor to the right.
- 3: 8’h01, clears the display.
- 4: 8’hC0, chooses the second line.

Verilog code: contains the complete module LCD display.

4.5 Task 5: Constraint file for LCD module

```
#  Package pins for LCD
set_property -dict { PACKAGE_PIN P3 IOSTANDARD LVCMOS33 } [get_ports {data[7]}];
set_property -dict { PACKAGE_PIN M5 IOSTANDARD LVCMOS33 } [get_ports {data[6]}];
set_property -dict { PACKAGE_PIN N4 IOSTANDARD LVCMOS33 } [get_ports {data[5]}];
set_property -dict { PACKAGE_PIN R2 IOSTANDARD LVCMOS33 } [get_ports {data[4]}];
set_property -dict { PACKAGE_PIN R1 IOSTANDARD LVCMOS33 } [get_ports {data[3]}];
set_property -dict { PACKAGE_PIN R3 IOSTANDARD LVCMOS33 } [get_ports {data[2]}];
set_property -dict { PACKAGE_PIN T2 IOSTANDARD LVCMOS33 } [get_ports {data[1]}];
set_property -dict { PACKAGE_PIN T4 IOSTANDARD LVCMOS33 } [get_ports {data[0]}];

# Package pins for Enable, RS and input clk
set_property -dict { PACKAGE_PIN T3 IOSTANDARD LVCMOS33 } [get_ports {lcd_e}];
set_property -dict { PACKAGE_PIN P5 IOSTANDARD LVCMOS33 } [get_ports {lcd_rs}];
set_property -dict { PACKAGE_PIN N11 IOSTANDARD LVCMOS33 } [get_ports { in_Clk }];
```

4.6 Task 6 (Bonus): Printing the outputs of Wallace Multiplier on LCD

The ‘Wallace_main‘ module is designed to perform multiplication using a Wallace tree multiplier and display the results on an LCD as well as through LEDs. The module integrates various submodules to achieve these functionalities.

Key Components

- **Inputs**:
  - ‘A‘: 4-bit input representing the first number to be multiplied.
  - ‘B‘: 4-bit input representing the second number to be multiplied.
  - ‘reset‘: Signal used to reset the system.
  - ‘in_Clk‘: Main clock signal.

- **Outputs**:
  - ‘P‘: 8-bit output representing the product of ‘A‘ and ‘B‘.
  - ‘data‘, ‘lcd_rs‘, ‘lcd_e‘: Signals used to control and send data to the LCD.
  - **LEDs**: Display the product ‘P‘ for visual output.

Submodules

- **Clock Divider (‘clk_divider‘)**:
  - Generates a slower clock (‘out_clk‘) from the main clock (‘in_Clk‘).
  - This slower clock is used to synchronize and operate the LCD.
- **Wallace Multiplier (‘Wallace_multi‘)**:
  - Computes the product of ‘A‘ and ‘B‘ using the Wallace tree multiplier technique.
  - Outputs the result as ‘P‘.
- **LCD Display (‘Lcd_display‘)**:
  - Utilizes the slower clock (‘out_clk‘) to control the LCD.
  - Displays the result in the format “PR.OF {B} & {A} = {P}“, where ‘P‘ is the product.
- **ASCII Converter (‘Ascii8‘)**:
  - Converts a binary input signal into its corresponding ASCII representation.
  - Outputs ‘8’b00110000‘ for a binary input of ‘0‘ and ‘8’b00110001‘ for a binary input of ‘1‘.

Operation

- **Clock and Reset Handling**:
  - ⋆ On each rising edge of ‘in_Clk‘, the module checks for a reset signal.
  - ⋆ When a reset is detected (transition from low to high), the input values ‘A‘ and ‘B‘ are stored in registers (‘A_reg‘ and ‘B_reg‘).
- **Multiplication**:
  - ⋆ The Wallace multiplier module computes the product of ‘A‘ and ‘B‘, outputting the result as ‘P‘.
- **ASCII Conversion**:
  - ⋆ The ‘Ascii8‘ module converts binary inputs into their ASCII representations for ’0’ and ’1’.
- **Display and Output**:
  - ⋆ The LCD is updated with the multiplication result, formatted as “PR.OF B & A = m“, using the slower clock.
  - ⋆ The product ‘P‘ is displayed on LEDs, providing visual feedback of the result.

4.7 Verilog Codes and Constraint files

Click on the blue hyperlinks:

- **Main module**: contains the module which connects all the individual modules.
- **LCD display**: used for writing on the LCD display.
- **ASCII module**: used for converting binary to ASCII for LCD display.
- **Clock divider**: used for generating a slower clock by using the input clock on FPGA.
- **Wallace multiplier**: module that implements wallace multiplication to find product.
- **Constraint file**: used for mapping the variables in main module to respective i/o ports in FGPA board.

4.8 Outputs

CLICK HERE to see full utilization report.
CLICK HERE to see video of circuit testing.

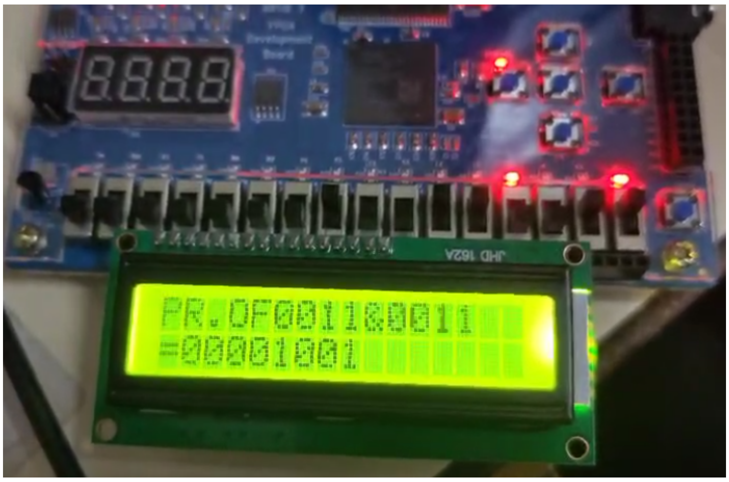
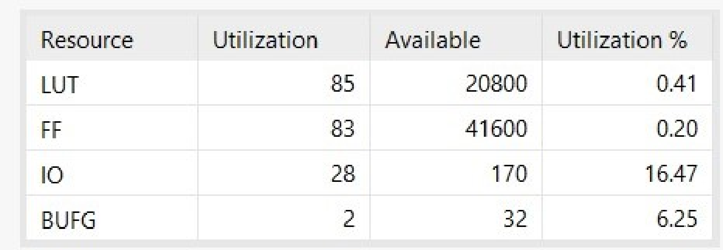


Figure 3: $A = (0011)_2, B = (0011)_2, P = (00001001)_2$



Resource	Utilization	Available	Utilization %
LUT	85	20800	0.41
FF	83	41600	0.20
IO	28	170	16.47
BUFG	2	32	6.25

Figure 4: No. of LUTs used

5 Problems Faced

We initially encountered issues with the LCD display, which only showed "0000" for all multiplication results. After adjusting the Verilog code to support various 4-bit inputs, the LCD functionality improved. Despite this, we faced difficulties when trying to load and display different inputs within a single reset cycle on the LCD, although we successfully managed to handle multiple inputs with the LEDs.