# Lab 6: Profiling  Adding a Custom IP to a Zynq Processor

## ENGG3050: Reconfigurable Computing Systems

**Instructor:**
Dr. Shawki Areibi

**Group 42: Tuesday-3:30 Section**

**Bilal Ayyache: 0988616**
**Anthony Granic: 0994824**

**Lab Start Date:** November 4, 2019
**Lab End Date:** November 12, 2019

# Contents



# List of Figures

# 1 Project Implementation

## 1.1 Problem Statement

The main objective of Lab 6 is to gain knowledge and experience in the process of designing embedded systems using Hardware-Software Co-design techniques. Techniques include developing IPs,soft cores, and hard cores using block design, VHDL, and C Programming. The three requirements of Lab 6 include:

- Control LEDs on FPGA board using Switches
- Run Matrix Multiplication Using ZED Board. Find bottlenecks.
- Add two numbers using ZEDBoard

## 1.2 System Overview, Justification Of Design reason behind design

### 1.2.1 Designed System Overview

The first requirement of the lab was met using hardware-software Co-Design. a Micro Blaze Soft core Processor was created on the NEXYS A7 FPGA Board in Vivado using block design to control LED lights on the FPGA board. The MicroBlaze was tested using C programming. using the switches on the NEXYS FPGA board, LEDs on the board were controlled using a C-program software application coded for the specific Hardware system implementation designed.

The second requirement of the lab was met through designing an embedded processor system on the ZED board. The design was capable of performing matrix multiplication using C programming. The algorithm performance was analyzed and profiled to find bottle necks in design.

The third requirement of the lab was met by creating a custom IP block in Vicado using the ZEDBoard to add two inputs and store result in a register. By designing a Multiplier IP and connecting it to the ZED board development kit, A software application was capable of adding two numbers using C-Programming.

## 1.3 Assumptions and Constraints

The following assumptions and constraints were made:

- Assumption: Both Zedboard, and Nexys-A7 boards are functional
- Assumption: The Analysis results extracted from Vivado are accurate
- Assumption: Vivado 2019 version can be used to complete the requirements
- Constraint: Vivado,VHDL, and the assigned boards must be used to complete the requirements.

# 2 Software Implementation

## 2.1 Requirement 1

```c
#include "xparameters.h"
#include "xgpio.h"
#include "xstatus.h"
#include "xil_printf.h"

/* Definitions */
#define SwitchesInput XPAR_AXI_GPIO_0_DEVICE_ID
#define LEDOutput XPAR_AXI_GPIO_1_DEVICE_ID /* GPIO device */
#define LED_CHANNEL 1 /* GPIO port for LEDs */
#define printf xil_printf /* smaller, optimized printf */

XGpio ledD; /* GPIO Device driver instance */
XGpio switchD;

int LEDOutputExample(void) {

volatile int Delay;
int Status1;
int Status2;
int data;


Status1 = XGpio_Initialize(&ledD, LEDOutput);
Status2 = XGpio_Initialize(&switchD, SwitchesInput);

if (Status1 != XST_SUCCESS || Status2 != XST_SUCCESS) {
    return XST_FAILURE;
}

XGpio_SetDataDirection(&ledD, LED_CHANNEL, 0x0000);
XGpio_SetDataDirection(&switchD, LED_CHANNEL, 0xFFFF);

    while (1) {
    /* Write output to the LEDs. */
        data = XGpio_DiscreteRead(&switchD, 1);
        XGpio_DiscreteWrite(&ledD,LED_CHANNEL,data);
    }

    return XST_SUCCESS; /* Should be unreachable */
}

int main(void) {
    int Status;
    /* Execute the LED output. */
    Status = LEDOutputExample();
    if (Status != XST_SUCCESS) {
        xil_printf("GPIO output to the LEDs failed!\r\n");
    }
    return 0;
}
```

Figure 1: Requirement 1: C Code

After implementing the hardware design presented in figure 7, The following C code was used to test the system. The GPIOs were first initialized as input and output. The LEDs are all turned off as initial condition and all 16 switches were connected as input. THE LED CHANNEL is set to 1 which chooses the 1st bus in the GPIO selected. In a while loop the switches value is saved to a data variable which is used to drive the LED output.

## 2.2 Requirement 3

```c
#include <stdio.h>
#include "xbasic_types.h"
#include "xparameters.h"
#include "platform.h"
#include "xil_printf.h"

Xuint32 *baseaddr_p = (Xuint32 *)XPAR_ADDERV2IP_0_S00_AXI_BASEADDR;

int main()
{
    init_platform();

    *(baseaddr_p + 0)= 0x00020003;
    xil_printf("Input: 0x%08x \n\r", *(baseaddr_p+0));
    xil_printf("Result: 0x%08x \n\r", *(baseaddr_p+1));

    return 0;
}
```

Figure 2: C Code to Test Hardware Design

Figure 2 presents the C code used to test the hardware system. 2 and 3 are added to register 0 using the pointer assigned to the registers

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity adder16b is
port(
    clk: in std_logic;
    a: in std_logic_vector(15 downto 0);
    b: in std_logic_vector(15 downto 0);
    p: out std_logic_vector(15 downto 0)
    );
end adder16b;

architecture Behavioral of adder16b is
  begin
   process (clk)
   begin
      if clk'event and clk = '1' then
         p <= a + b;
     end if;
  end process;
end Behavioral;
```

Figure 3: VHDL Code of Adder

The Following code in figure 3 highlight the VHDL code used to run the adder in the IP created

```
signal adderOut     : std_logic_vector(15 downto 0);

component adder16b
port(
    clk: in std_logic;
    a: in std_logic_vector(15 downto 0);
    b: in std_logic_vector(15 downto 0);
    p: out std_logic_vector(15 downto 0)
);
end component;
```

Figure 4: Declaring the Adder Component in the IP Editor

```
-- Add user logic here
adder: adder16b
port map(
    clk => S_AXI_ACLK,
    a => slv_reg0(31 downto 16),
    b => slv_reg0(15 downto 0),
    p => adderOut
);
-- User logic ends
```

Figure 5: Declaring and Porting of Adder

The VHDL code in figure 4 and 5 demonstrates how the adder was declared as a component and ported to the hierarchy.

```
begin
    -- Address decoding for reading registers
    loc_addr := axi_araddr(ADDR_LSB + OPT_MEM_ADDR_BITS downto ADDR_LSB);
    case loc_addr is
      when b"00" =>
        reg_data_out <= slv_reg0;
      when b"01" =>
        reg_data_out <= X"0000" & adderOut;
      when b"10" =>
        reg_data_out <= slv_reg2;
      when b"11" =>
        reg_data_out <= slv_reg3;
      when others =>
        reg_data_out  <= (others => '0');
    end case;
end process;
```

Figure 6: Setting output to register

The output of the VHDL application was set to register 2.

# 3 Block Diagrams

## 3.1 Requirements Block Diagram Description

The designed system for the first requirement includes the MicroBlaze processing system, a processor system reset, a local memory, debug module, clocking wizard, AXI interconnect module and finally an AXI GPIO to use LEDs as output and switches as input(refer to figure: 7). Two AXI GPIOs where used in this design. One GPIO was used to declare switches as input and LEDs as output.

The designed system for the second requirement includes the Zynq processing system, a Processor system reset, an AXI BRAM Controller, an AXI GPIO module, a Block Memory Generator, and an AXI Interconnect. These together make the program able to be run on the ZedBoard(refer to figure: 8).

The designed system for the third requirement includes the Zynq processing system, a Processor system reset, an AXI BRAM Controller, an AXI GPIO module, a Block Memory Generator, and an AXI Interconnect. These together make the program able to be run on the ZedBoard(refer to figure: 9).

## 3.2 Requirement 1

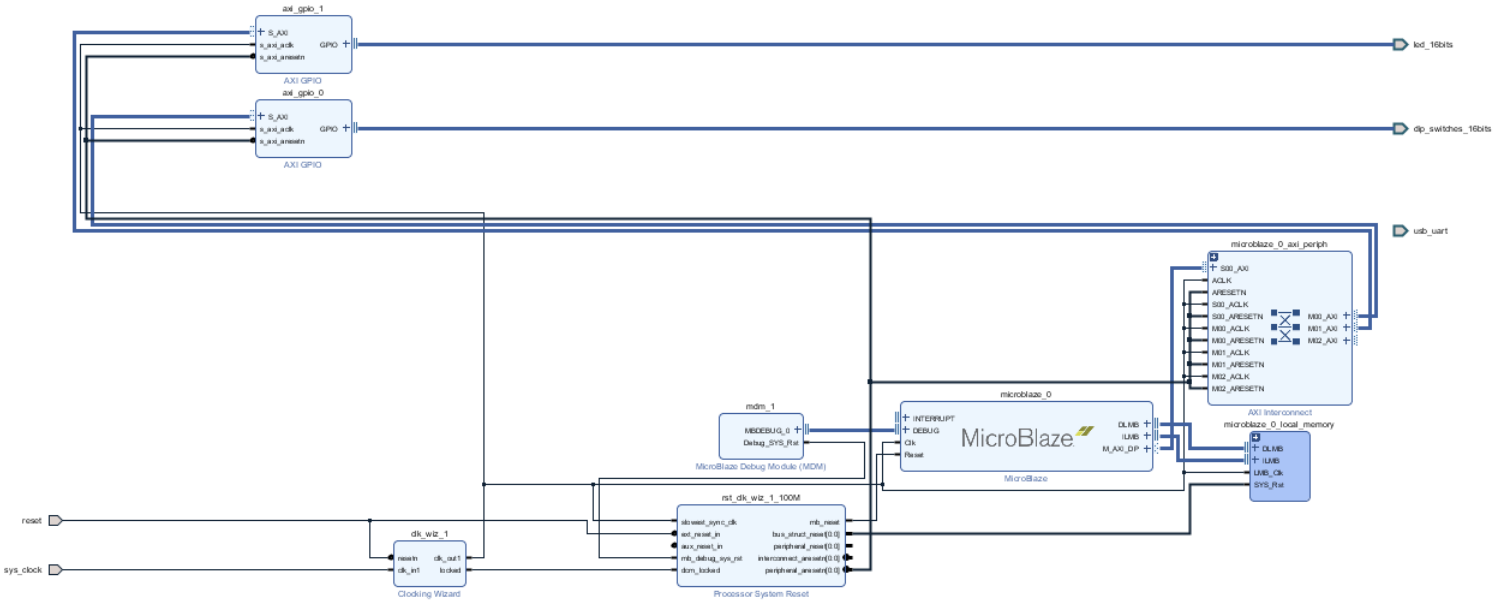


Figure 7: Requirement 1 Block Diagram

### 3.3 Requirement 2

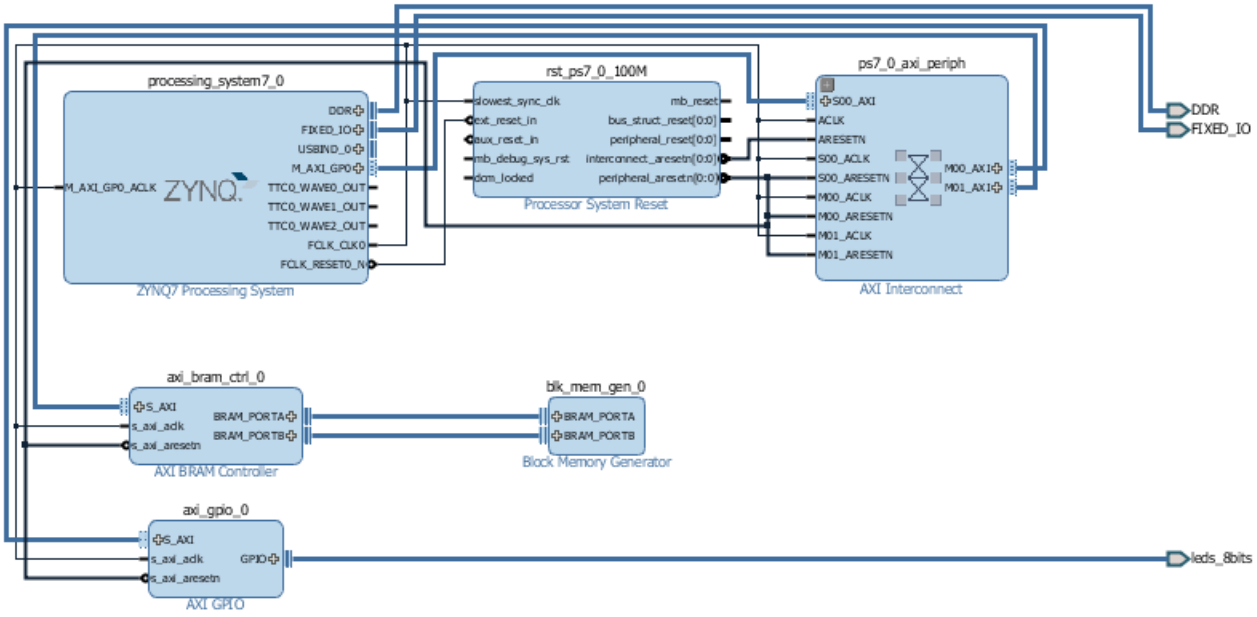


Figure 8: Requirement 2 Block Diagram

### 3.4 Requirement 3

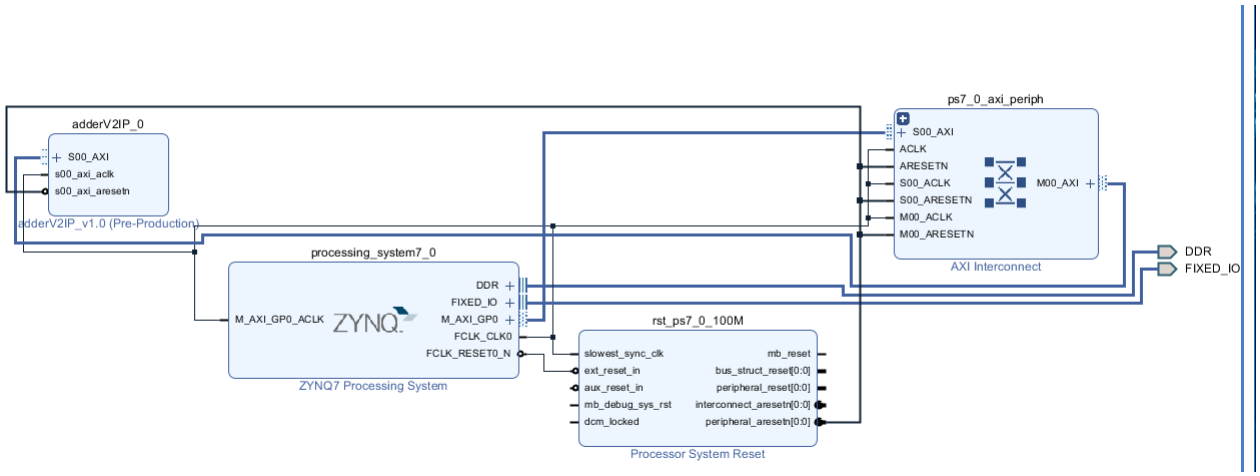


Figure 9: Requirement 3 Block Diagram

# 4 Utilized Resource Analysis

## 4.1 LED interface with ZED board

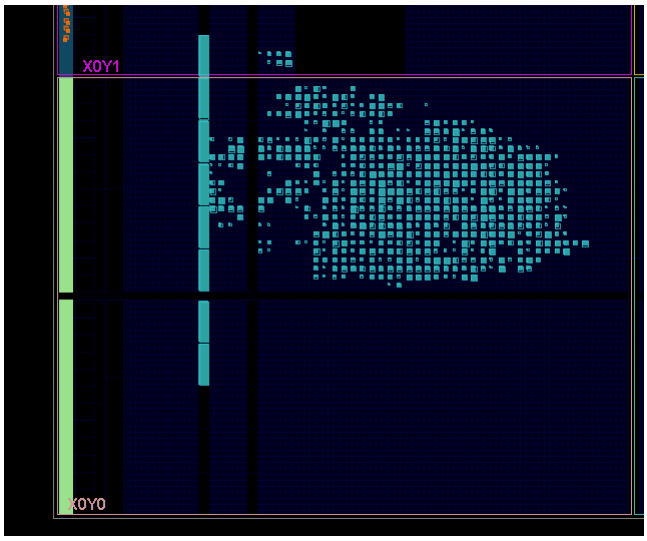


Figure 10: Visual Representation of Usage

**Design Timing Summary**

| Setup | | Hold | | Pulse Width | |
|---|---|---|---|---|---|
| Worst Negative Slack (WNS): | 2.519 ns | Worst Hold Slack (WHS): | 0.106 ns | Worst Pulse Width Slack (WPWS): | 3.000 ns |
| Total Negative Slack (TNS): | 0.000 ns | Total Hold Slack (THS): | 0.000 ns | Total Pulse Width Negative Slack (TPWS): | 0.000 ns |
| Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 |
| Total Number of Endpoints: | 4483 | Total Number of Endpoints: | 4483 | Total Number of Endpoints: | 1740 |

**All user specified timing constraints are met.**

Figure 11: Timing Analysis of Resources

| Name | Slice LUTs (63400) | Slice Registers (126800) | F7 Muxes (31700) | Slice (15850) | LUT as Logic (63400) | LUT as Memory (19000) | Block RAM Tile (135) | DSPs (240) | Bonded IPADs (2) |
|---|---|---|---|---|---|---|---|---|---|
| N microblaze_design_wrapper | 1475 | 1489 | 111 | 584 | 1347 | 128 | 1489 | 8 | 34 |
| microblaze_design_i (microblaze_design) | 1475 | 1489 | 111 | 584 | 1347 | 128 | 8 | 0 | 0 |

Figure 12: Utilization On FPGA Board

## 4.2 Profiling on ZED Board

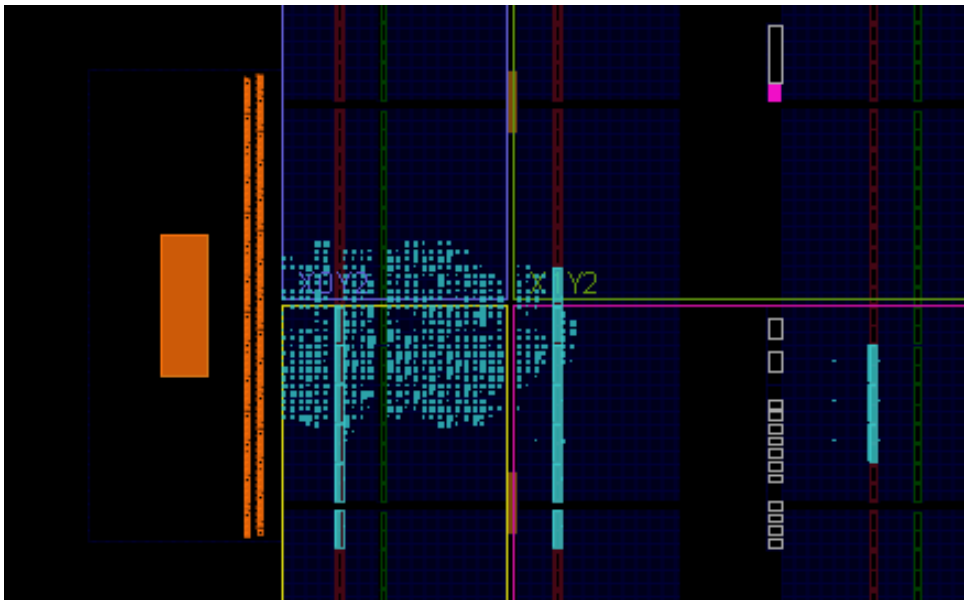


Figure 13: Visual Representation of Usage

| Setup | | Hold | | Pulse Width | |
|---|---|---|---|---|---|
| Worst Negative Slack (WNS): | 1.107 ns | Worst Hold Slack (WHS): | 0.041 ns | Worst Pulse Width Slack (WPWS): | 4.020 ns |
| Total Negative Slack (TNS): | 0.000 ns | Total Hold Slack (THS): | 0.000 ns | Total Pulse Width Negative Slack (TPWS): | 0.000 ns |
| Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 |
| Total Number of Endpoints: | 4610 | Total Number of Endpoints: | 4610 | Total Number of Endpoints: | 1797 |

Figure 14: Timing Analysis of Resources

| Name | Slice LUTs (53200) | Slice Registers (106400) | Slice (13300) | LUT as Logic (53200) | LUT as Memory (17400) | LUT Flip Flop Pairs (53200) | Block RAM Tile (140) | Bonded IOB (200) | Bonded IOPADs (130) |
|---|---|---|---|---|---|---|---|---|---|
| zynq_design_1_wrapper | 1372 | 1680 | 563 | 1294 | 78 | 574 | 16 | 8 | 130 |
| zynq_design_1_i (zynq_des... | 1372 | 1680 | 563 | 1294 | 78 | 574 | 16 | 0 | 0 |

Figure 15: Utilization On FPGA Board

### 4.3 Adder Interface

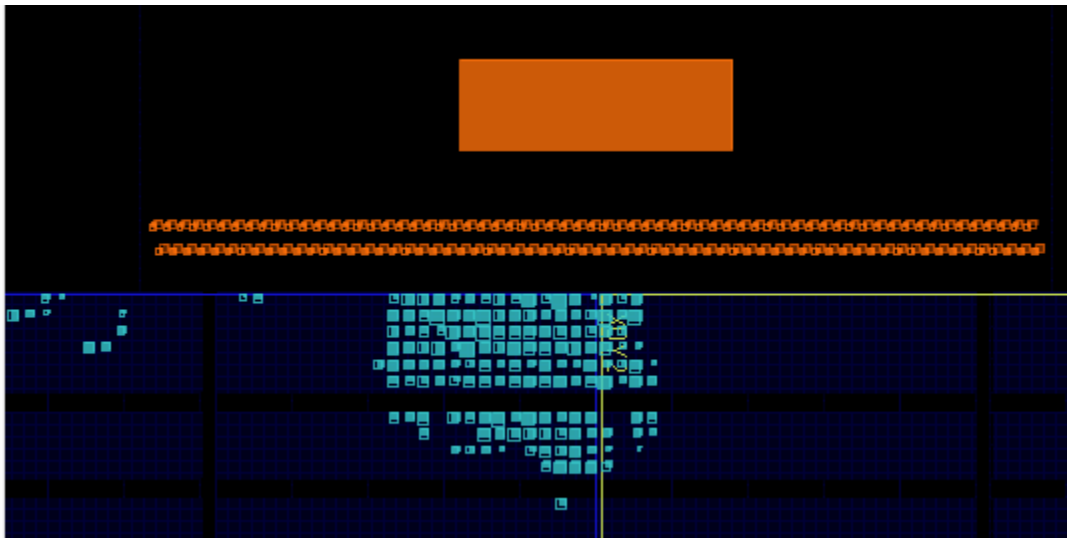

Figure 16: Equation 2 Resources

| Setup | | Hold | | Pulse Width | |
|---|---|---|---|---|---|
| Worst Negative Slack (WNS): | 3.662 ns | Worst Hold Slack (WHS): | 0.042 ns | Worst Pulse Width Slack (WPWS): | 4.020 ns |
| Total Negative Slack (TNS): | 0.000 ns | Total Hold Slack (THS): | 0.000 ns | Total Pulse Width Negative Slack (TPWS): | 0.000 ns |
| Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 |
| Total Number of Endpoints: | 1398 | Total Number of Endpoints: | 1398 | Total Number of Endpoints: | 652 |

**All user specified timing constraints are met.**

Figure 17: Timing Analysis of Resources

| Name | Slice LUTs (53200) | Slice Registers (106400) | Slice (13300) | LUT as Logic (53200) | LUT as Memory (17400) | Block RAM Tile (140) | PHY_CONTROL (4) | BUFIO (16) |
|---|---|---|---|---|---|---|---|---|
| adderBlock_wrapper | 420 | 587 | 159 | 360 | 60 | 587 | 130 | 1 |
| adderBlock_i (adderBlock) | 420 | 587 | 159 | 360 | 60 | 0 | 0 | 0 |

Figure 18: Utilization On FPGA Board

### 4.4 Utilization Summary

The ZedBoard profiling used the most resources, which was expected due to the size of the processor logic needed. The LED interface used more resources than the adder. All of these designs were fairly simple and small so not all of the FPGA board was needed.

## 5 Lab Results

### 5.1 Requirement 1: LED Interface With ZED Board

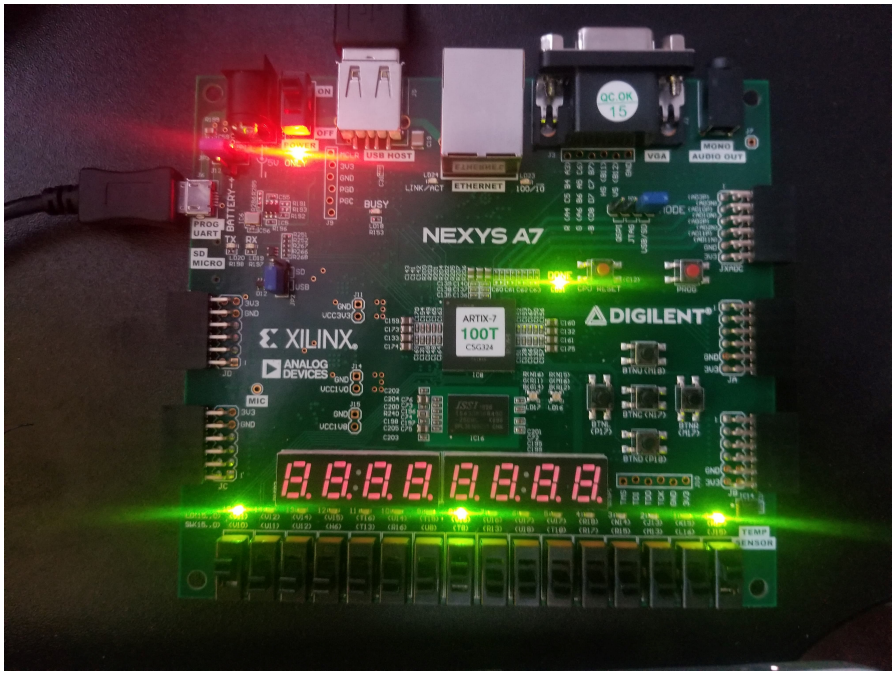


Figure 19: Requirement 1 results

Figure 19 displays the output of the LEDs the switches are turned high.(ie: The LED turns on as the switch on the FPGA is switched from off to on).

## 5.2 Requirement 2: Profiling On ZED Board

The profiling of the program on the ZedBoard was successful. It showed the following results for usage of different functions.

| Name (location) | Samples | Calls | Time/Call | % Time |
|---|---|---|---|---|
| Summary | 76551 | | | 100.0% |
| Xil_ICacheDisable | 50971 | | | 66.58% |
| matrix_multiply | 17404 | 1 | 17.404ms | 22.74% |
| sad | 5881 | 24 | 245.041us | 7.68% |
| XUartPs_SendByte | 1431 | 185 | 7.735us | 1.87% |
| matrix_add | 282 | 1 | 282.000us | 0.37% |
| init_candidate_image | 178 | 1 | 178.000us | 0.23% |
| init_ref_image | 178 | 1 | 178.000us | 0.23% |
| Xil_L2CacheDisable.part.1 | 163 | | | 0.21% |
| mcount | 20 | | | 0.03% |
| xil_printf | 17 | 0 | | 0.02% |
| print | 12 | 0 | | 0.02% |
| FIQInterrupt | 4 | | | 0.01% |
| main | 3 | 0 | | 0.0% |
| __gnu_mcount_nc | 2 | | | 0.0% |
| outnum | 2 | 0 | | 0.0% |
| register_tm_clones | 2 | | | 0.0% |
| strlen | 1 | | | 0.0% |
| XScuGic_DeviceInitialize | 0 | 1 | 0ns | 0.0% |
| XScuGic_GetPriTrigTypeByDistAddr | 0 | | | 0.0% |
| XScuGic_RegisterHandler | 0 | 1 | 0ns | 0.0% |
| Xil_Assert | 0 | | | 0.0% |
| Xil_L1DCacheFlush | 0 | | | 0.0% |
| Xil_L1DCacheInvalidateRange | 0 | | | 0.0% |
| cleanup_platform | 0 | 1 | 0ns | 0.0% |
| cortexa9_init | 0 | 0 | | 0.0% |
| disable_caches | 0 | 1 | 0ns | 0.0% |
| enable_caches | 0 | 1 | 0ns | 0.0% |
| init_platform | 0 | 1 | 0ns | 0.0% |
| init_uart | 0 | 1 | 0ns | 0.0% |
| outbyte | 0 | | | 0.0% |

Figure 20: Requirement 2 results

This table shows that accessing the cache is the biggest bottleneck of the program. This is likely due to the cache being accessed constantly by the program. The function used most by the program is matrix-multiply. This was expected as it is the function with the heaviest processor usage in the program. To speed up the program a hardware accelerator should be used for matrix-multiply, as it is the part of the code that bottlenecks the speed.

### 5.3 Requirement 3: Adder Interface

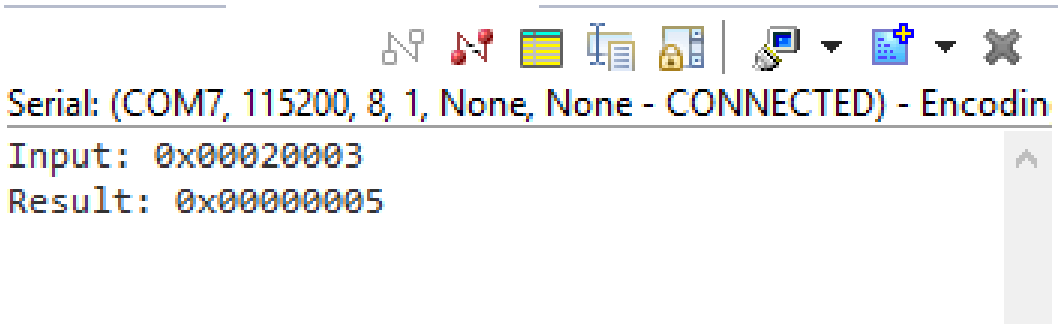


Figure 21: Requirement 3 results

Figure 21 displays the output of 2 and 3 being added. By analyzing the results, it seems that the design is fully functional.

## 6 Error Analysis

During the development process of requirement 1, localization of file was an issue faced multiple times. Vivado could not locate generated bit file for an unknown reason. The solution to such error was restarting the design process from scatch. Error was not clear.

During the development process of requirement 2, there were 2 errors in this portion of the lab, one minor and one major. The minor problem was running the code on the ZedBoard processor multiple times, as the previous code would only terminate if the program was restarted. The major problem was not being able to run and profile the Linux code due to insufficient knowledge of the software.

During the development process of requirement 3, connecting the board to the terminal was an issue faced due to unknown reasons. Although the device was connected as directed by the lab manual, the output of the program would not print on the terminal. Issue faced was solved by restarting the computer.