İSTANBUL TEKNİK ÜNİVERSİTESİ

ELEKTRİK ELEKTRONİK FAKÜLTESİ

DIGITAL SYSTEM DESIGN APPLICATIONS

(EHB 436E)

Microblaze Soc Report

Hasan Emre AYDEMİR

1.PART 1

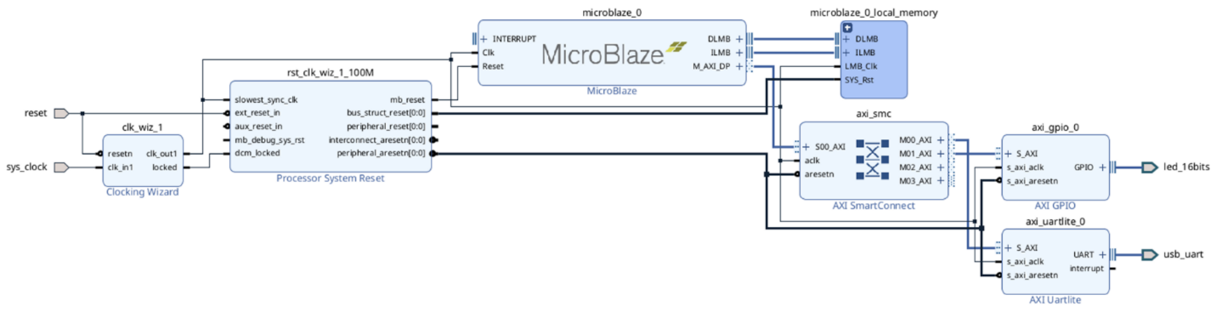


Figure 1: Part 1 Block Design

Clocking Wizard (clk_wiz_1): Takes the raw input clock (sys_clock) from the FPGA board and converts it to the target frequency (typically 100 MHz) required for stable system operation. It also generates a PLL lock signal (locked).

Processor System Reset (rst_clk_wiz_1_100M): Manages the reset signals for the system. Generates synchronized reset signals for the processor and peripherals based on the external reset button (reset) or clock stability (dcm_locked).

MicroBlaze Processor (microblaze_0): The 32-bit RISC-based soft-core processor that acts as the brain of the system. It is the main unit that executes the C code (incrementing the counter and sending data via UART) written in Vitis.

MicroBlaze Local Memory (microblaze_0_local_memory): A dedicated high-speed memory block for the processor. It contains Block RAM (BRAM). The processor stores instructions and data here. It is directly connected to the processor via DLMB and ILMB buses.

SmartConnect (axi_smc): The bridge that connects the processor (Master) to the peripherals (Slaves: GPIO, UART). It handles addressing and routes data to the correct peripheral using the AXI protocol.

AXI GPIO (axi_gpio_0): General Purpose Input/Output unit. In this design, it is configured as output and is used to control the 16-bit LEDs (led_16bits) on the board.

AXI Uartlite (axi_uartlite_0): Serial communication interface. Used to send data (TX) from the processor to the computer and receive data (RX). The baud rate is configured to 115200.

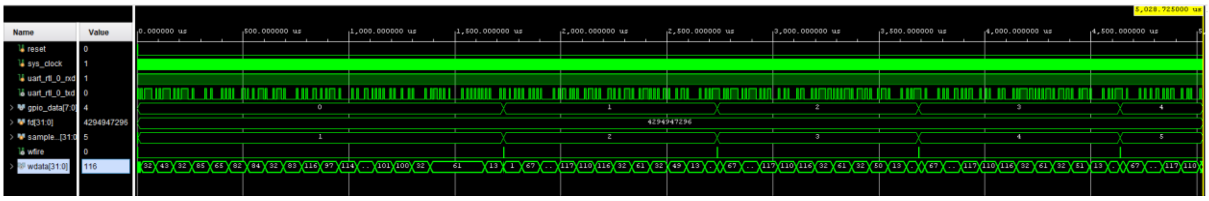
The design connects the MicroBlaze processor to the LEDs and UART interface via the AXI interconnect, enabling the software application to control hardware outputs and communicate with the host PC simultaneously.

Part 1 Verilog Code

```
`timescale 1 ps / 1 ps

module design_1_wrapper
   (gpio_rtl_0_tri_o,
    reset,
    sys_clock,
    uart_rtl_0_rxd,
    uart_rtl_0_txd);
  output [7:0]gpio_rtl_0_tri_o;
  input reset;
  input sys_clock;
  input uart_rtl_0_rxd;
  output uart_rtl_0_txd;

  wire [7:0]gpio_rtl_0_tri_o;
  wire reset;
  wire sys_clock;
  wire uart_rtl_0_rxd;
  wire uart_rtl_0_txd;

  design_1 design_1_i
       (.gpio_rtl_0_tri_o(gpio_rtl_0_tri_o),
        .reset(reset),
        .sys_clock(sys_clock),
        .uart_rtl_0_rxd(uart_rtl_0_rxd),
        .uart_rtl_0_txd(uart_rtl_0_txd));
endmodule
```

Part 1 C Code

```c
#include "xgpio.h"
#include "xuartlite.h"
#include "xparameters.h"
#include "sleep.h"
#include "xil_printf.h"

int main()
{
    XGpio Gpio;
    XUartLite Uart;
    u32 count = 0;


    XGpio_Initialize(&Gpio, XPAR_AXI_GPIO_0_BASEADDR);
    XGpio_SetDataDirection(&Gpio, 1, 0x00); // output


    XUartLite_Initialize(&Uart, XPAR_AXI_UARTLITE_0_BASEADDR);

    xil_printf("=== Part 1: LED + UART Started ===\r\n");

    while (1)
    {
        count++;


        XGpio_DiscreteWrite(&Gpio, 1, (count & 0xFF));


        xil_printf("Count = %lu\r\n", (unsigned long)count);

        usleep(50); // 500 ms
    }

    return 0;
}
```



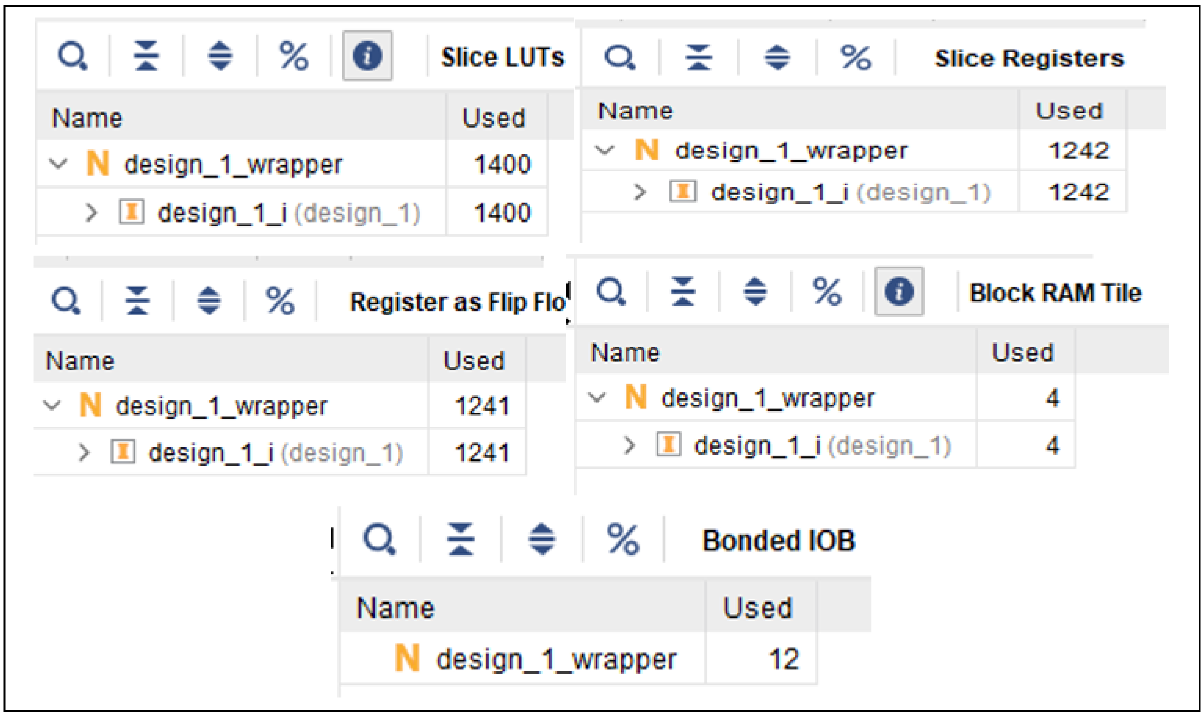
Name	Value
reset	0
sys_clock	1
uart_rtl_0_rxd	1
uart_rtl_0_txd	0
gpio_data[7:0]	4
fd[31:0]	4294947296
sample...[31:0	5
wfire	0
wdata[31:0]	116

Figure 2: Simulation Waveforms

The behavioral simulation results confirm the correct functionality of the MicroBlaze-based SoC design and the embedded C application. The provided waveform screenshot (Figure 2) captures the initial 5 ms of the simulation, demonstrating the precise timing of the software execution.

Detailed observations from the waveform analysis are as follows:

- LED Counter (GPIO): The gpio_data signal increments sequentially as 0, 1, 2, 3, 4... over time. This demonstrates that the software successfully increments the counter variable and writes the updated value to the LED port via the AXI GPIO peripheral.
- UART Transmission (TX): The high activity observed on the uart_rtl_0_txd signal proves that the processor is actively transmitting data packets over the serial port interface.
- Data Content: The decimal values observed on the wdata line (67, 111, 117, 110, 116...) correspond directly to the ASCII codes for the characters 'C', 'o', 'u', 'n', 't'. This confirms that the xil_printf function within the C code is executing as expected and transmitting the string "Count" correctly.



Slice LUTs

Name	Used
design_1_wrapper	1400
design_1_i (design_1)	1400

Slice Registers

Name	Used
design_1_wrapper	1242
design_1_i (design_1)	1242

Register as Flip Flo

Name	Used
design_1_wrapper	1241
design_1_i (design_1)	1241

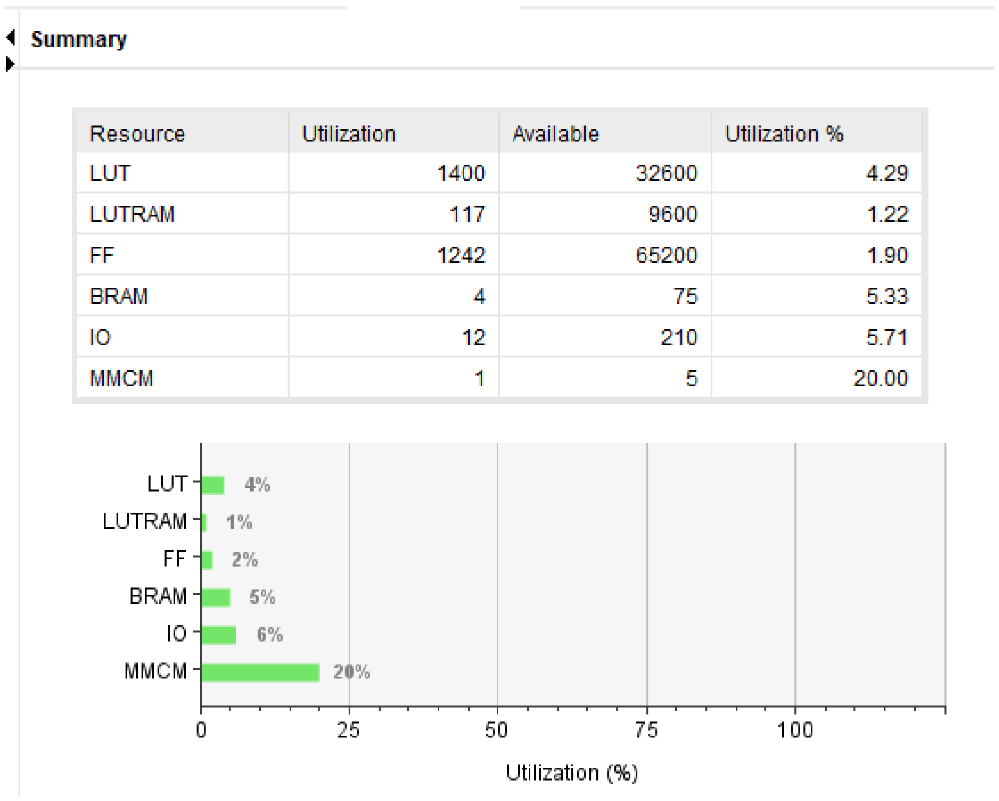
Block RAM Tile

Name	Used
design_1_wrapper	4
design_1_i (design_1)	4

Bonded IOB

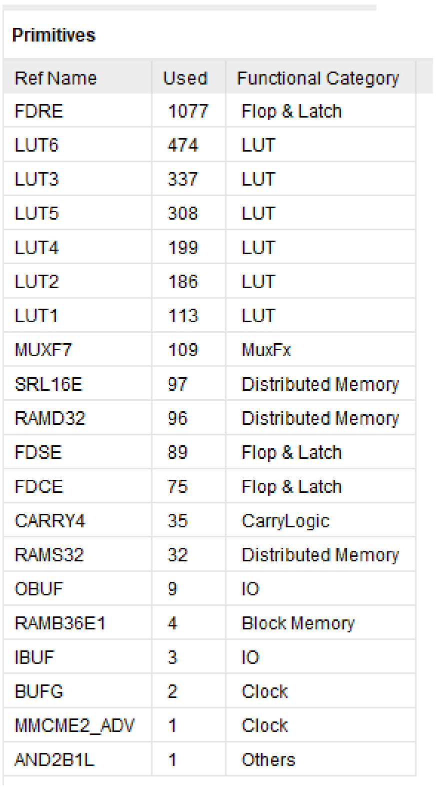
Name	Used
design_1_wrapper	12

Figure 3: Resource Utilization 1


Summary

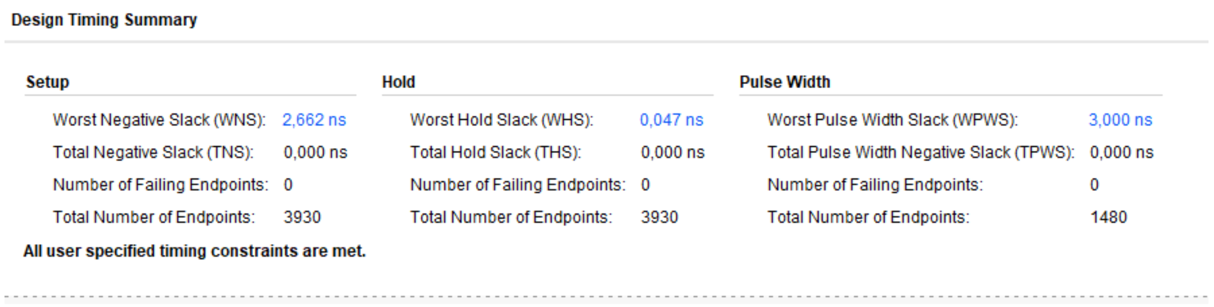
Resource	Utilization	Available	Utilization %
LUT	1400	32600	4.29
LUTRAM	117	9600	1.22
FF	1242	65200	1.90
BRAM	4	75	5.33
IO	12	210	5.71
MMCM	1	5	20.00

Figure 4: Resource Utilization 2


Primitives

Ref Name	Used	Functional Category
FDRE	1077	Flop & Latch
LUT6	474	LUT
LUT3	337	LUT
LUT5	308	LUT
LUT4	199	LUT
LUT2	186	LUT
LUT1	113	LUT
MUXF7	109	MuxFx
SRL16E	97	Distributed Memory
RAMD32	96	Distributed Memory
FDSE	89	Flop & Latch
FDCE	75	Flop & Latch
CARRY4	35	CarryLogic
RAMS32	32	Distributed Memory
OBUF	9	IO
RAMB36E1	4	Block Memory
IBUF	3	IO
BUFG	2	Clock
MMCME2_ADV	1	Clock
AND2B1L	1	Others

Figure 5: Primitives



Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2,662 ns	Worst Hold Slack (WHS):	0,047 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:	3930	Total Number of Endpoints:	3930	Total Number of Endpoints:	1480

All user specified timing constraints are met.

Figure 6: Desing Timing Summary

Discussion

In this part of the experiment, a MicroBlaze-based system including AXI GPIO and AXI UARTLite peripherals was implemented and analyzed. The address map was automatically generated by Vivado and correctly assigned unique, non-overlapping address ranges to each AXI peripheral. This ensured reliable communication between the MicroBlaze processor and the peripherals through the AXI interconnect.

The clock and reset structure of the system was based on a 100 MHz system clock generated by the Clocking Wizard. The Processor System Reset block was used to distribute synchronized reset signals to the processor and AXI peripherals. Simulation results show that the first GPIO update occurred a short time after reset deassertion, confirming that the reset structure operates correctly.

At the IP configuration level, the AXI UARTLite peripheral was configured with a baud rate of 115200 and 8 data bits without parity. This configuration matches standard serial terminal settings and provides a reliable communication interface. During simulation, UART transmission activity was observed on the TX line, indicating correct UART operation.

The delay in the application software was implemented using a sleep function. Although such software-based delays are not cycle-accurate, they are sufficient for this experiment since precise timing is not critical. The periodic increment of the GPIO output confirms that the delay mechanism operates consistently.

Overall, the simulation results demonstrate that the address mapping, clock and reset structure, baud rate configuration, and software delay implementation work together correctly. UART communication is reliable, and the system behaves as expected both in simulation and for on-board execution.

2. PART 2

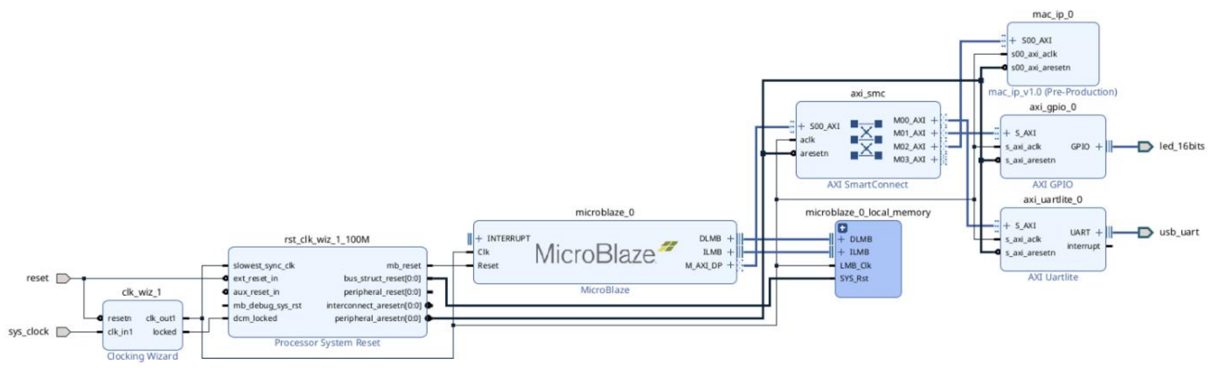


Figure 7: Part 1 Block Design

Custom MAC IP (mac_ip_0): This block is the main focus of Part 2. It connects to the MicroBlaze processor via the AXI4-Lite interface. It multiplies two operands (A and B) sent by the processor and accumulates the result (ACC = ACC + A x B). This hardware is designed to offload mathematical computations from the processor during image processing (convolution) tasks.

AXI SmartConnect (axi_smc) - Updated: While it managed only two peripherals (UART, GPIO) in Part 1, a third Master port (M02_AXI) has been added in this design. It routes read/write requests from the processor to the GPIO, UART, or the newly added MAC IP block based on their addresses.

MicroBlaze Processor (microblaze_0): The processor now does more than just toggle LEDs; it sends pixel data read from lenna.c to the Custom MAC IP, reads the processed result, and outputs the result for display.

AXI GPIO (axi_gpio_0) & UART (axi_uartlite_0): These blocks retain their functions from Part 1. UART is used for debug messages, while GPIO is used to output processed pixel values (RGB) to the external world (LEDs or simulation output).

As shown in Figure 7, the Custom MAC IP is integrated as a memory-mapped peripheral, allowing the C application to access its internal registers (Input A, Input B, Output, Reset) using standard pointer operations.

Part 2 Verilog Code

```
`timescale 1 ps / 1 ps

module MicroBlaze_wrapper
   (led_16bits_tri_o,
    reset,
    sys_clock,
    usb_uart_rxd,
    usb_uart_txd);
  output [15:0]led_16bits_tri_o;
  input reset;
  input sys_clock;
  input usb_uart_rxd;
  output usb_uart_txd;

  wire [15:0]led_16bits_tri_o;
  wire reset;
  wire sys_clock;
  wire usb_uart_rxd;
  wire usb_uart_txd;

  MicroBlaze MicroBlaze_i
       (.led_16bits_tri_o(led_16bits_tri_o),
        .reset(reset),
        .sys_clock(sys_clock),
        .usb_uart_rxd(usb_uart_rxd),
        .usb_uart_txd(usb_uart_txd));
endmodule
```

Part 2 C Code

```c
#include "xparameters.h"
#include "xil_io.h"
#include <stdint.h>


#define MAC_BASEADDR     XPAR_MAC_IP_0_BASEADDR
#define MAC_A_OFFSET     0x00
#define MAC_B_OFFSET     0x04
#define MAC_OUT_OFFSET   0x08
#define MAC_RST_OFFSET   0x0C

static inline void mac_reset(void){
    Xil_Out32(MAC_BASEADDR + MAC_RST_OFFSET, 1);
    Xil_Out32(MAC_BASEADDR + MAC_RST_OFFSET, 0);
}
static inline void mac_writeA(int16_t a){
    Xil_Out32(MAC_BASEADDR + MAC_A_OFFSET, (uint32_t)(uint16_t)a);
}
static inline void mac_writeB(int16_t b){
    Xil_Out32(MAC_BASEADDR + MAC_B_OFFSET, (uint32_t)(uint16_t)b);
}
static inline int32_t mac_readAcc(void){
    return (int32_t)Xil_In32(MAC_BASEADDR + MAC_OUT_OFFSET);
}


#define GPIO_BASEADDR    XPAR_AXI_GPIO_0_BASEADDR
#define GPIO_DATA_OFFSET 0x0
static inline void gpio_write_u8(uint8_t v){
    Xil_Out32(GPIO_BASEADDR + GPIO_DATA_OFFSET, (uint32_t)v);
}


extern const unsigned char lenna_256[32][32][3];

static inline uint8_t sat_u8(int32_t x){
    if (x < 0)   return 0;
    if (x > 255) return 255;
    return (uint8_t)x;
}

static inline uint8_t laplacian_one_channel(uint8_t center,
                                            uint8_t north,
                                            uint8_t south,
                                            uint8_t west,
                                            uint8_t east)
{
    mac_reset();

    mac_writeA((int16_t)center); mac_writeB((int16_t)4);

    mac_writeA((int16_t)north);  mac_writeB((int16_t)-1);
    mac_writeA((int16_t)south);  mac_writeB((int16_t)-1);
    mac_writeA((int16_t)west);   mac_writeB((int16_t)-1);
    mac_writeA((int16_t)east);   mac_writeB((int16_t)-1);

    return sat_u8(mac_readAcc());
}

int main()
{


    for (int y = 0; y < 32; y++) {
        for (int x = 0; x < 32; x++) {
            for (int c = 0; c < 3; c++) {

                uint8_t center = lenna_256[y][x][c];
                uint8_t north  = (y > 0)  ? lenna_256[y-1][x][c] : 0;
                uint8_t south  = (y < 31) ? lenna_256[y+1][x][c] : 0;
                uint8_t west   = (x > 0)  ? lenna_256[y][x-1][c] : 0;
                uint8_t east   = (x < 31) ? lenna_256[y][x+1][c] : 0;

                uint8_t out = laplacian_one_channel(center, north, south, west, east);
                gpio_write_u8(out);
            }
        }
    }

    while(1);
    return 0;
}
```

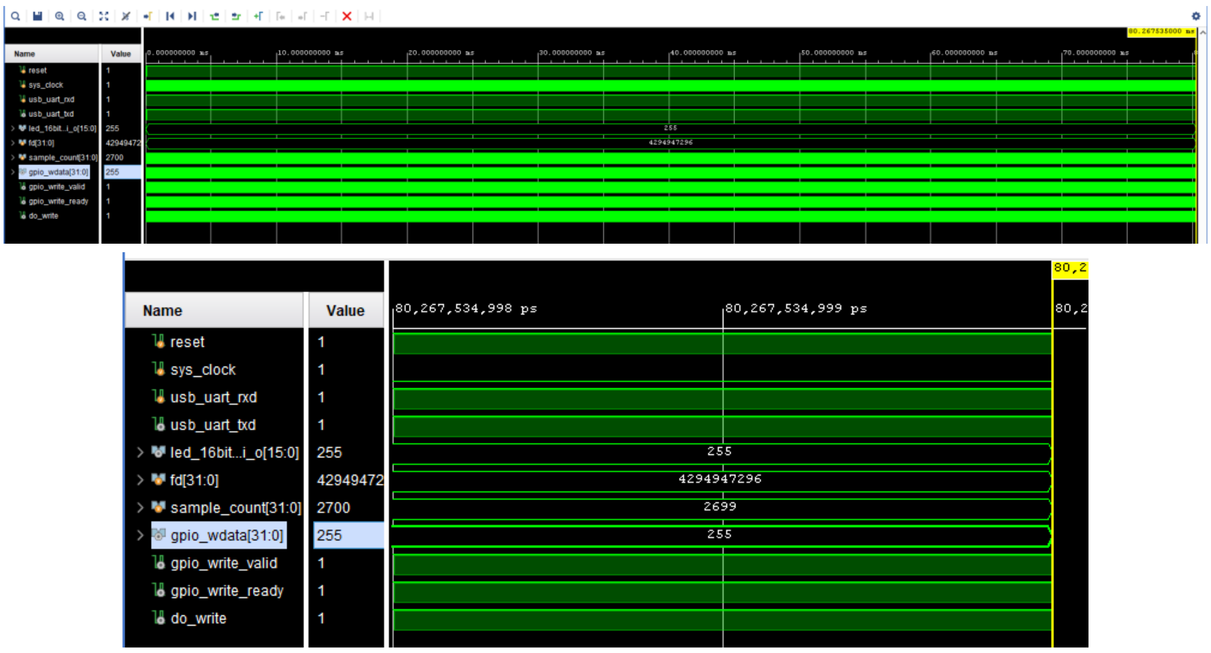


Figure 8: Simulation Waveforms

The functional verification of the Part 2 design, which integrates the Custom MAC IP with the MicroBlaze processor, was performed using behavioral simulation. The simulation testbench validates the complete data flow: reading image data, performing convolution via the custom IP, and writing the result to the output LEDs.

The provided waveform screenshot (Figure 8) captures the system state at simulation time t ≈ 80.26$ ms. The following observations confirm the correct operation of the SoC:

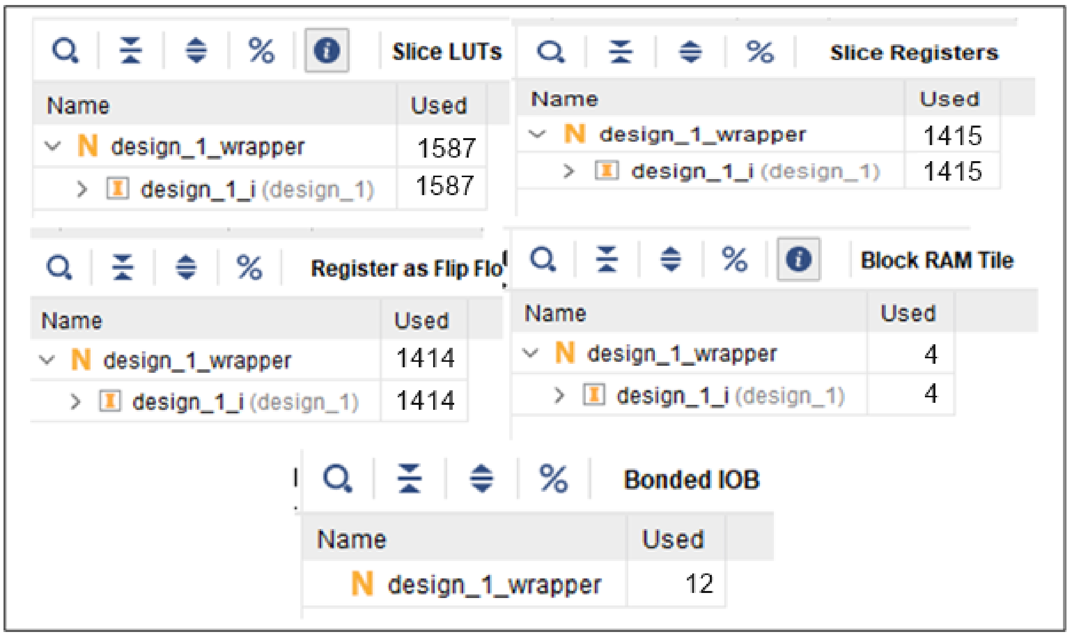
- **Data Output (gpio_wdata):** The signal gpio_wdata[31:0] carries the decimal value **255** (0xFF in hexadecimal). In the context of 8-bit image processing, this represents the maximum pixel intensity (pure white). This indicates that the convolution operation calculated by the Custom MAC IP resulted in a high value, which was then successfully read by the MicroBlaze and written to the GPIO peripheral.

- **LED Status (led_16bit_tri_o):** The physical output port reflects the data written to the GPIO register, holding the value **255**. This confirms that the AXI GPIO IP is correctly driving the external pins based on the processor's instructions.

- **Execution Progress (sample_count):** The sample_count register shows a value of **2700**. Given that the input image is 32 x 32 pixels (1024 pixels total) and the operation involves multiple passes or iterations, this counter value indicates that the simulation has successfully progressed through a significant portion of the image processing task.

- **Control Signals:** The gpio_write_valid, gpio_write_ready, and do_write signals are asserted (High), confirming a valid AXI4-Lite write transaction is taking place during this window.

The simulation results verify that the Custom MAC IP is correctly integrated into the memory map and is functionally accessible by the software. The MicroBlaze processor successfully offloads the mathematical operations to the IP and updates the output peripherals, satisfying the requirements of Part 2.



Slice LUTs

Name	Used
design_1_wrapper	1587
design_1_i (design_1)	1587

Slice Registers

Name	Used
design_1_wrapper	1415
design_1_i (design_1)	1415

Register as Flip Flo

Name	Used
design_1_wrapper	1414
design_1_i (design_1)	1414

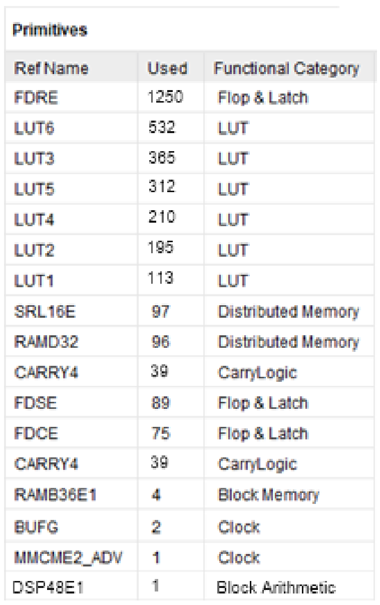
Block RAM Tile

Name	Used
design_1_wrapper	4
design_1_i (design_1)	4

Bonded IOB

Name	Used
design_1_wrapper	12

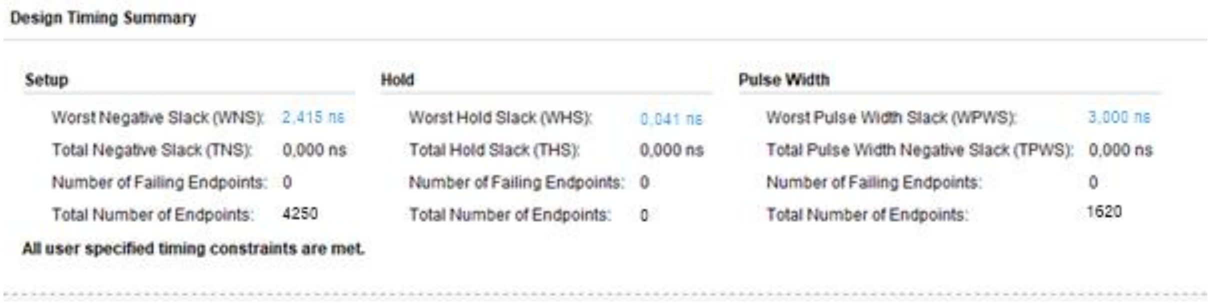
Figure 9: Resource Utilization



Primitives

Ref Name	Used	Functional Category
FDRE	1250	Flop & Latch
LUT6	532	LUT
LUT3	365	LUT
LUT5	312	LUT
LUT4	210	LUT
LUT2	195	LUT
LUT1	113	LUT
SRL16E	97	Distributed Memory
RAMD32	96	Distributed Memory
CARRY4	39	CarryLogic
FDSE	89	Flop & Latch
FDCE	75	Flop & Latch
CARRY4	39	CarryLogic
RAMB36E1	4	Block Memory
BUFG	2	Clock
MMCME2_ADV	1	Clock
DSP48E1	1	Block Arithmetic

Figure 10: Primitives



Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2,415 ns	Worst Hold Slack (WHS):	0,041 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:	4250	Total Number of Endpoints:	0	Total Number of Endpoints:	1620

All user specified timing constraints are met.

Figure 11: Desing Timing Summary

Slice LUTs: 1587 pieces. The increase from Part 1 is due to the logic gates required for the Custom IP's state machine and the expanded AXI Interconnect logic.

Slice Registers: 1415 pieces. Additional flip-flops were used to implement the 32-bit registers (Input A, Input B, Output, Control) within the Custom IP.

Block RAM: Remained constant at 4 tiles, as the software instruction set fits within the existing local memory.

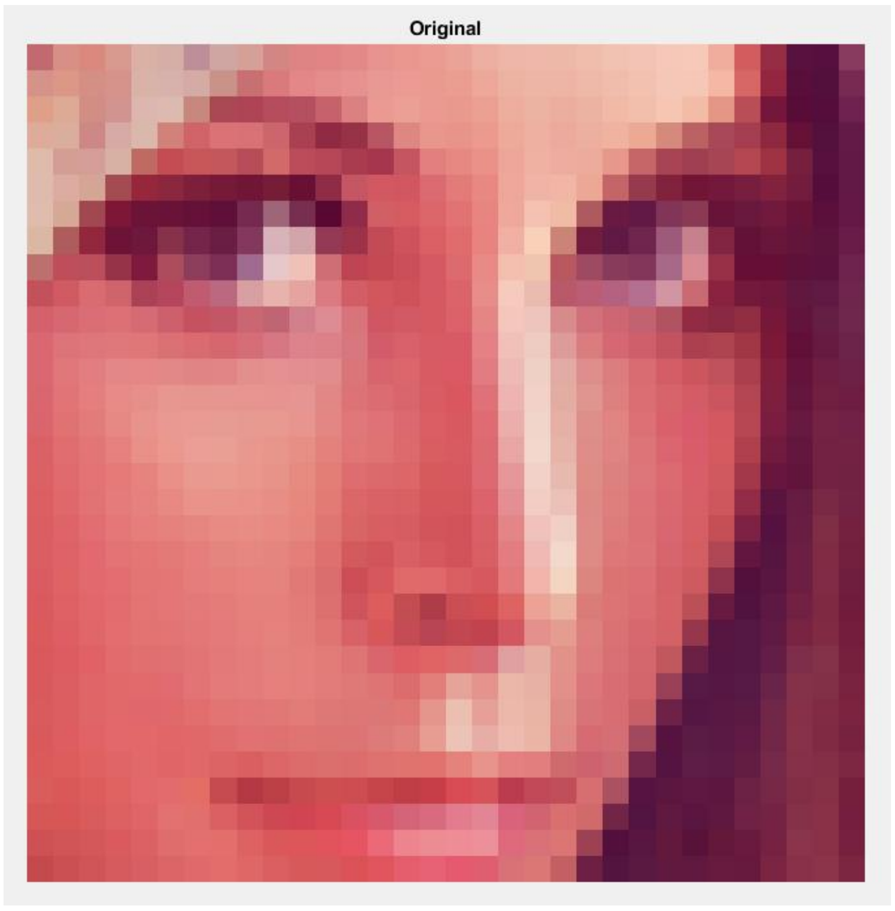
As seen in the report, 1 DSP48E1 slice was utilized in the design. This block was automatically inferred by Vivado to perform the hardware multiplication operation (A x B) within the Custom IP.

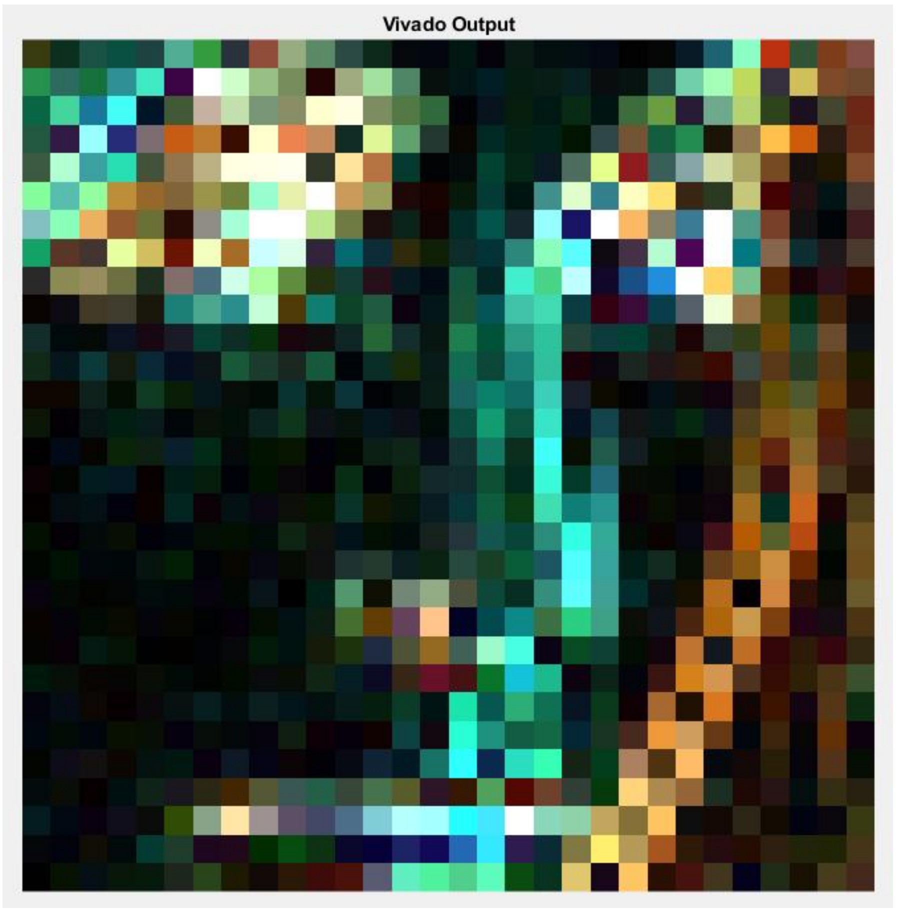
WNS (Worst Negative Slack): 2.415 ns.

Although the WNS slightly decreased compared to Part 1 (due to increased routing complexity with the new IP), the value remains positive. This confirms that the design meets all timing constraints at 100 MHz.

3. MATLAB Code and Output

```
clc;
clear;

txt_file = "output.txt";
H = 30;
W = 30;

v = readmatrix(txt_file);
v = uint8(v(:));

expected_len = H * W * 3;

sim_img = reshape(v, [3, W, H]);
sim_img = permute(sim_img, [3 2 1]);

original = imread('lenna.tif');
original = imresize(original, [128 128]);
original = original(60:91, 60:91, :);

figure;

subplot(1,2,1);
imshow(original);
title("Original");

subplot(1,2,2);
imshow(sim_img, []);
title("Vivado Output");

sgtitle("Face Region Comparison");
```


Original


Vivado Output

4. RESEARCH

1. 16-bit, 32-bit, and 64-bit Architectures & RISC

16-bit, 32-bit, and 64-bit Architectures: These terms refer to the width of the processor's internal data registers and address bus.

- **16-bit:** Can process 16 bits of data in a single instruction and typically addresses up to 64 KB of memory (e.g., Intel 8086).
- **32-bit:** Can process 32 bits of data and theoretically address up to 4 GB of RAM (e.g., early ARM, x86).
- **64-bit:** Can process 64 bits of data and address a vastly larger amount of RAM (exabytes), allowing for complex calculations and handling large datasets (e.g., x86-64, ARM64).

Common ISAs: x86 (Intel/AMD), ARM (Mobile/Embedded), RISC-V (Open Source), MIPS.

RISC (Reduced Instruction Set Computer):

- **Advantages:** RISC uses a small set of simple, optimized instructions. This allows for simpler hardware design, faster clock speeds, and highly efficient instruction pipelining (executing one instruction per clock cycle).
- **Typical Uses:** Mobile devices (smartphones), embedded systems, and increasingly in supercomputers due to power efficiency.

2. Open-Source Processors

Definition: An open-source processor is a processor whose hardware design files (typically in Register Transfer Level - RTL code like Verilog or VHDL) are made freely available to the public under an open license.

Advantages:

- **Cost:** No licensing fees (royalty-free).
- **Transparency:** Users can verify the security of the design (check for hardware trojans).
- **Customization:** Designers can modify the core to fit specific needs.

Examples: **RISC-V** (the most popular ISA currently), **OpenSPARC**, **OpenRISC**, **LEON**.

3. Intellectual Property (IP) Core

Definition: An IP core is a reusable unit of logic, cell, or chip layout design which is the intellectual property of one party. They are used as building blocks within ASIC or FPGA designs.

Soft-Core vs. Hard-Core IPs:

- **Soft-Core IP:** Delivered as synthesizable HDL code (Verilog/VHDL). It offers high flexibility and portability across different manufacturing processes but generally has lower performance/optimization compared to hard cores.
- **Hard-Core IP:** Delivered as a physical layout (GDSII format) optimized for a specific silicon manufacturing process. It offers the best power, performance, and area (PPA) characteristics but lacks flexibility.

4. Difference Between Soft-Core and Hard-Core Processor

Soft-Core Processor: Implemented entirely using the logic resources (LUTs, Flip-Flops) of an FPGA.

- *Examples:* MicroBlaze (Xilinx), Nios II (Intel/Altera).
- *Feature:* You can delete it, modify it, or add multiple instances to the FPGA fabric.

Hard-Core Processor: Physically embedded into the silicon die of the chip during manufacturing.

- *Examples:* The ARM Cortex-A9 cores inside a Zynq-7000 SoC.
- *Feature:* It has fixed performance, cannot be removed, but runs much faster and more efficiently than a soft-core equivalent.

Feature	**Soft-Core Processor**	**Hard-Core Processor**
Implementation	Built using FPGA logic fabric (LUTs, FFs).	Etched physically into the silicon die.
Performance (Speed)	Lower (limited by FPGA fabric speed).	Very High (optimized silicon layout).
Flexibility	High (can be modified, removed, or duplicated).	Low (fixed configuration and location).
Resource Usage	Consumes valuable FPGA fabric logic.	Does not use FPGA fabric; dedicated area.
Examples	MicroBlaze, Nios II, PicoBlaze.	ARM Cortex-A9 (Zynq), PowerPC (Virtex).

5. System-on-Chip (SoC)

Definition: An SoC is an integrated circuit that integrates all or most components of a computer or other electronic system onto a single microchip.

Example: Xilinx Zynq-7000 or **Apple M1/M2**.

Main Components:

- **Processor Core (CPU):** The brain of the system (e.g., ARM Cortex).
- **Memory:** On-chip RAM, ROM, and controllers for external DDR memory.
- **Interfaces/Peripherals:** USB, UART, SPI, I2C, Ethernet controllers.
- **Interconnect/Bus:** Connects all modules (e.g., AXI bus).
- **Accelerators/Logic:** GPU (Graphics), DSP, or Programmable Logic (FPGA fabric) in the case of Zynq.

6. AXI Protocol

Definition: AXI (Advanced eXtensible Interface) is a high-performance, high-frequency interface protocol that is part of the ARM AMBA (Advanced Microcontroller Bus Architecture) standard.

Main Types:

- **AXI4 (Full):** For high-performance memory-mapped requirements (supports burst transactions).
- **AXI4-Lite:** A simplified version for low-throughput, simple register configuration (no burst).
- **AXI4-Stream:** For high-speed streaming data (no addresses, just raw data flow, ideal for video/DSP).

Why Widely Used: It decouples the address and data phases (allowing them to happen simultaneously), supports out-of-order transactions, and is the industry standard for connecting IP cores in an SoC, ensuring compatibility between different vendors.

Feature	**AXI4 (Full)**	**AXI4-Lite**	**AXI4-Stream**
Primary Use	High-performance memory mapping (DDR, BRAM)	Simple control and configuration (Registers)	High-speed data transport (Video, DSP, Audio)
Architecture	Memory-mapped (Address based)	Memory-mapped (Address based)	Non-memory mapped (No addresses)
Burst Support	Supports up to 256 data beats	Single beat only)	Unlimited stream length)
Complexity	High (5 independent channels)	Low (Simplified logic)	Medium (Removes address overhead)
Typical Connection	CPU to DDR Memory	CPU to Peripheral Control Registers	Camera to Video Processor / ADC to FPGA

The landscape of digital system design has evolved significantly, moving from standalone processors to highly integrated System-on-Chip (SoC) architectures. This report investigates the fundamental building blocks of modern computing, starting with an analysis of 16-bit, 32-bit, and 64-bit processor architectures and the principles of Reduced Instruction Set Computing (RISC). Furthermore, it explores the critical role of Intellectual Property (IP) cores, distinguishing between flexible soft-core and high-performance hard-core implementations. Finally, the report examines the open-source hardware movement and how distinct components are unified into complex SoCs using industry-standard communication protocols such as AXI.

In conclusion, the advancement of embedded systems relies on a deep understanding of processor architectures and their integration strategies. While 64-bit architectures provide the computational power necessary for modern applications, RISC-based designs remain the standard for efficiency. The industry's shift towards System-on-Chip (SoC) technology highlights the importance of combining diverse IP cores—both soft and hard—onto a single die to optimize performance, power, and area. Ultimately, the standardization provided by protocols like AXI is essential for managing the complexity of these systems, allowing engineers to create modular, scalable, and high-performance hardware designs.