Report of the 8th Experiment

Name, Surname:
Armin Asgharifard
Student No: 040190912

Report Completion Date

20.12.2022

Course title

Digital System Design Applications

Prof. Dr. Berna Örs Yalçın

Res. Assis. Serdar Duran

Name, Surname: Armin Asgharifard
Student No: 040190912

Implementation of an Image Processing System

Design Codes

Mac_Normalize.v

If the input of this module exceeds 255, the output will be equal to 255. If the input gets below 0, the output will be 0. In other conditions, the output equals the input.

```
`timescale 1ns / 1ps

module MAC_Normalize(
    input signed [19:0] data,
    output reg signed [7:0] result
    );

    always @(*)
    begin
        if (data < 0)
            result = 0;
        else if (data > 255)
            result = 255;
        else
            result = data[7:0];
    end
endmodule
```

CONV.v

A concatenation of a MAC block and a MAC normalizer block is a convolution block, which can multiply three 8-bit data and three 8-bit weights, add them and give us the 8-bit result.

```
`timescale 1ns / 1ps

module CONV(
    input clk,
    input reset,
    input [23:0] data,
    input [23:0] weight,
    output [7:0] result
    );

    wire signed [19:0] result_reg;

    MAC mac(.clk(clk), .reset(reset), .data(data), .weight(weight), .result(result_reg));
    MAC_Normalize norm(.data(result_reg), .result(result));
endmodule
```

CONV128.v

This block consists of 128 convolution blocks, where each convolution block gets a portion of the big input data, with the same weight for all blocks, and the results of each block are concatenated to give us the bigger result.

Name, Surname: Armin Asgharifard
Student No: 040190912

```
`timescale 1ns / 1ps

module CONV128(
    input clk,
    input reset,
    input [1039:0] data,
    input [23:0] weight,
    output [1023:0] result
    );

    genvar i;
    generate
        for (i = 128; i >= 1; i = i - 1)
        begin
            CONV conv(
                .clk(clk),
                .reset(reset),
                .data(data[(i + 2) * 8 - 1 : (i + 2) * 8 - 24]),
                .weight(weight),
                .result(result[i * 8 - 1 : i * 8 - 8])
                );
        end
    endgenerate
endmodule
```

input_control.v

This module provides the RAM address to the BRAM block, and the weights to the CONV128 block. With a 2-bit counter, this module divides the input kernel to three, and gives them to the weight output, and transmits each of them to CONV128 at each count of the counter. So, it will take three clock cycles to transmit the whole kernel, and at each of these cycles, the BRAM address increments by one. Therefore, in one round of the counter, three rows of the image and the whole kernel is loaded into CONV128.

```
`timescale 1ns / 1ps

module input_control(
    input clk,
    input reset,
    input conv_run,
    input [71:0] kernel,
    output reg enable_ram,
    output reg [7:0] address_ram,
    output reg [23:0] weight
    );

    reg [1:0] count = 0;
    integer last_addr = 8'b0;

    always @(posedge clk, posedge reset)
    begin
        if (reset) begin
            enable_ram <= 0;
            address_ram <= 0;
```

```
            count <= 0;
        end
        else begin
            count <= count + 1;
            if (conv_run) begin
                enable_ram <= 1;
                case (count)
                    0: begin
                        weight <= kernel[71:48];
                        address_ram <= last_addr;
                    end
                    1: begin
                        weight <= kernel[47:24];
                        address_ram <= last_addr + 1;
                    end
                    2: begin
                        weight <= kernel[23:0];
                        address_ram <= last_addr + 2;
                    end
                    3: begin
                        last_addr <= last_addr + 1;
                    end
                endcase
            end
            else
                enable_ram <= 0;
        end
    end
endmodule
```

output_control.v

This module holds the same counter as in input_control.v, and when the counter reaches the last number, the incoming result from CONV128, is written into a certain address in the second BRAM block. This address is incremented by one for every counter round.

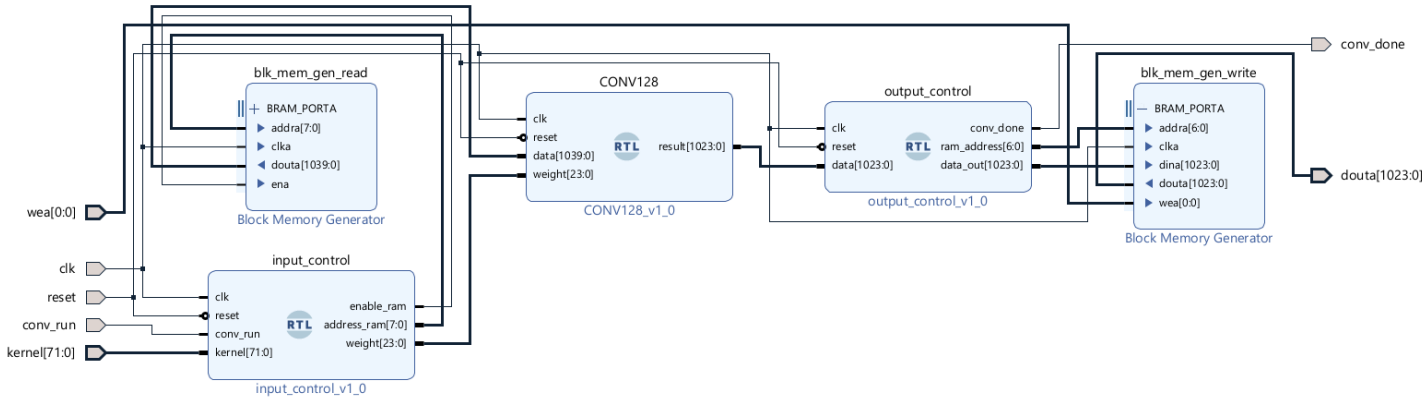
```
`timescale 1ns / 1ps

module output_control(
    input clk,
    input reset,
    input [1023:0] data,
    output reg conv_done,
    output reg [6:0] ram_address,
    output reg [1023:0] data_out
    );

    reg [1:0] count = 0;
    integer last_addr = 8'b0;

    always @(posedge clk, posedge reset)
    begin
        if (reset) begin
            conv_done <= 0;
            ram_address <= 0;
```

Name, Surname: Armin Asgharifard
Student No: 040190912

```
            data_out <= 0;
            count = 0;
        end
        else begin
            count <= count + 1;
            if (count == 3) begin
                ram_address <= last_addr;
                data_out <= data;
                last_addr <= last_addr + 1;
            end
            if (last_addr == 128) begin
                conv_done <= 1;
            end
        end
    end
endmodule
```

Block Design

The block design is given below. Note that the BRAM block intended for write operations is set as *always enabled*.



Testbench and Simulation

After wrapping the design, a test code, which is given below, is written.

```
`timescale 1ns / 1ps

module img_proc_design_tb();
    reg clk, reset, run, wea;
    wire done;
    wire [1023:0] douta;
    reg [71:0] kernel;

    reg signed [7:0] a = -1;
    reg signed [7:0] b = 8;

    reg [1023:0] result [0:127];
    integer i, file;

    img_proc_design_wrapper wrapper (
```

```
        clk,
        done,
        run,
        douta,
        kernel,
        reset,
        wea
        );

    always @(*)
        #5 clk <= ~clk;
    initial
    begin
        clk <= 0;
        wea <= 1;
        reset <= 1;
        #10 reset <= 0;
        run <= 1;
        kernel [71:64] <= a;
        kernel [63:56] <= a;
        kernel [55:48] <= a;
        kernel [47:40] <= a;
        kernel [39:32] <= b;
        kernel [31:24] <= a;
        kernel [23:16] <= a;
        kernel [15:8] <= a;
        kernel [7:0] <= a;
        #100;
        for (i = 0; i < 128; i = i + 1) begin
            result[i] = douta;
            #40;
        end
        file = $fopen("D:\output_image.txt", "a");
        for (i = 0; i < 128; i = i + 1) begin
                $fwrite(file, "%h\n", result[i]);
        end
        $fclose(file);
    end
endmodule
```

The hex result of the convolution operation is saved in a text file in D drive of the computer. This file can be opened as an image using the given MATLAB code. Check out the following results.

Name, Surname: Armin Asgharifard
Student No: 040190912

Original Image

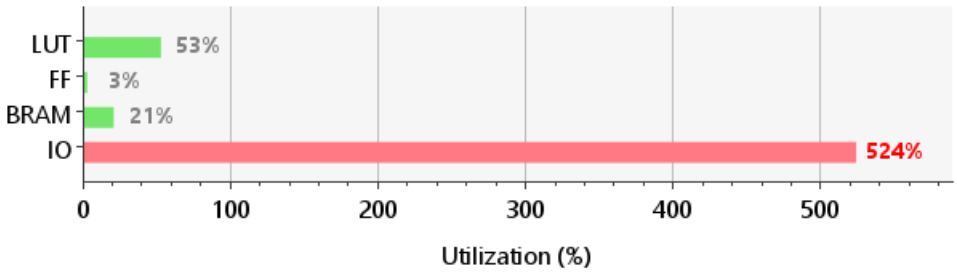
MATLAB Result

Vivado Result

Resource Utilization

Let's take a look at the post-synthesis resource utilization report of the design.

Resource	Utilization	Available	Utilization %
LUT	33583	63400	52.97
FF	3925	126800	3.10
BRAM	29	135	21.48
IO	1101	210	524.29



Setup Delays and Maximum Clock Frequency

By looking at the setup delays, we can calculate the maximum clock frequency.

Name, Surname: Armin Asgharifard
Student No: 040190912

Name	Slack ^1	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	S
Path 1	∞	18	18	57	img_proc_d.../CLKBWRCLK	img_proc_des...t_reg[19]/D	10.661	6.564	4.097	∞	
Path 2	∞	18	18	57	img_proc_d.../CLKBWRCLK	img_proc_des...t_reg[19]/D	10.661	6.564	4.097	∞	
Path 3	∞	18	18	57	img_proc_d.../CLKBWRCLK	img_proc_des...t_reg[19]/D	10.661	6.564	4.097	∞	
Path 4	∞	18	18	57	img_proc_d.../CLKBWRCLK	img_proc_des...t_reg[19]/D	10.661	6.564	4.097	∞	
Path 5	∞	18	18	57	img_proc_d.../CLKBWRCLK	img_proc_des...t_reg[19]/D	10.661	6.564	4.097	∞	
Path 6	∞	18	18	57	img_proc_d.../CLKARDCLK	img_proc_des...t_reg[19]/D	10.661	6.564	4.097	∞	
Path 7	∞	18	18	57	img_proc_d.../CLKARDCLK	img_proc_des...t_reg[19]/D	10.661	6.564	4.097	∞	
Path 8	∞	18	18	57	img_proc_d.../CLKARDCLK	img_proc_des...t_reg[19]/D	10.661	6.564	4.097	∞	
Path 9	∞	18	18	57	img_proc_d.../CLKARDCLK	img_proc_des...t_reg[19]/D	10.661	6.564	4.097	∞	
Path 10	∞	18	18	57	img_proc_d.../CLKBWRCLK	img_proc_des...t_reg[19]/D	10.661	6.564	4.097	∞	

Since the highest delay is 10.661 ns, then the maximum clock frequency is $\frac{1}{10.661\ ns} = 93.7\ MHz$.