DIGITAL SYSTEM DESIGN APPLICATION

HOMEWORK 8

CODES

```
module CONV (
 input clk,
 input reset,
 input [23:0] data,
 input [23:0] weight,
 output [7:0] result
);
 wire [19:0] mac_result;

 MAC mac_unit (.data(data),.weight(weight),.result(mac_result));
 MAC_Normalize mac_normalize_unit (.data(mac_result),.result(result));

endmodule

module CONV128 (
 input clk,
 input reset,
 input [1039:0] data,
 input [23:0] weight,
 output [1023:0] result
);
 genvar i;
 generate
   for (i = 0; i < 128; i=i+1)
   begin
     wire [7:0] conv_result;
     CONV conv_unit (.clk(clk),.reset(reset),.data(data[(i+1)*8-1:i*8]),.weight(weight),.result(conv_result));
     assign result[(i+1)*8-1:i*8] = conv_result;
   end
 endgenerate
endmodule
```

```
module control_input (
 input clk,
 input reset,
 input conv_run,
 input   [71:0] kernel,
```

```
assign kernel= kernela;
 always @(posedge clk) begin
  if (reset) begin
    counter <= 0;
    address_ram <= 8'h00;
  end
 end
 always @(posedge clk) begin
  if (conv_run) begin
    counter <= counter + 1'b1;
    address_ram <= address_ram + 1;
    weight <= kernela[47:24];
    kernela <= kernela[71:48];
  end
 end
 always @(posedge clk) begin
  if (counter == 3) begin
    counter <= 0;
  end
 end
 assign enable_ram = conv_run;

endmodule

module output_control (
 input clk,
 input reset,
 input [1023:0] data,
 output reg conv_done,
 output [6:0] ram_address,
 output [1023:0] data_out
);
 reg [1:0] counter;
 reg [6:0] current_address;

 always @(posedge clk) begin
  if (reset) begin
    counter <= 0;
    current_address <= 0;
  end
  else begin
    counter <= counter + 1'b1;
  end
```

TOP.V

```
`timescale 1ns / 1ps

module TOP (
  input clk,
  input reset,
  input conv_run,
  input [71:0] kernel,
  output conv_done,
  output [1023:0] data_out

);
  wire [7:0] ram_address;
  wire [23:0] weight;
  wire [1023:0] data;
  wire  result_conv128;
  wire [7:0] address_ram_2;
  wire [1023:0] data_2;
  wire enable_ram;
  control_input control_input_unit (
    .clk(clk),.reset(reset),.conv_run(conv_run),.kernel(kernel),.enable_ram(enable_ram),.address_ram(ram_address),.weight(weight));
  blk_mem_gen_0_0 design1
      (.addra_0(ram_address),
       .clka_0(clk),
       .dina_0(kernel),
       .douta_0(data),
       .ena_0(enable_ram),
       .wea_0(conv_done));
  CONV128 conv128_unit (
    .clk(clk), .reset(reset), .data(data), .weight(weight), .result(result_conv128));
  output_control output_control_unit (
    .clk(clk),  .reset(reset),  .data(result_conv128),  .conv_done(conv_done),  .ram_address(address_ram_2), .data_out(data_2) );
  blk_mem_gen_1_1 design_
```

PROCESS OF SUBMODULES

Convolution Unit : There are 3 stages here. Mac normalize, convolution and 128 convolution. In the Mac normalization phase, we are adjusting the size of our data. For this example, we convert a 20-bit input to 8 bits. Then we apply the 2 d convolution process that we did in experiment 7 with the convolution module. However, at this stage, we do not scroll, we just do a multiplication of digits. With 128 convolutions, we do this by shifting 128 times.

Block RAM #1 :

A block tasked with hosting the image file named input image.coe. We get my image from this block with data wire. The address cable from the input controller decides which data will go.

Input Controller :

A block that we call the kernel, which contains the lablassian formula, starts the process with this block and we send our kernel to the conv 128 block. In this block, the scrolling process of the kernel that will go is also done. It also sends address information to block ram.

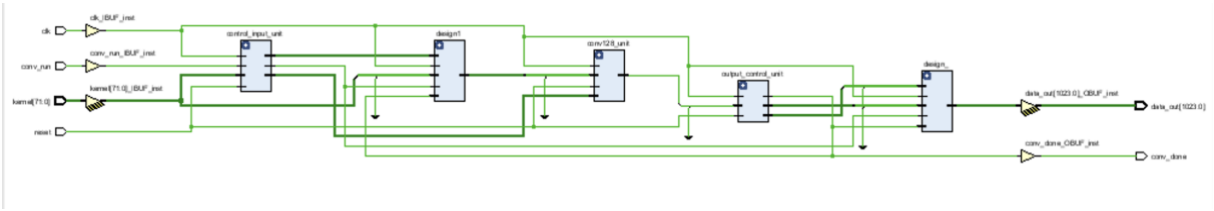
Block RAM #2 :

A block responsible for generating the image file named output image.code. We define the address information for the record with the address cable coming from the output controller block.

Output Controller .

It takes the sum of each row that has been convoluted and records it, and then takes and records the 128 transactions of the next row. we numerically obtain our output subjected to the operations we want to generate

RTL SCHEMA



UTILIZATION REPORT

Reports | Design Runs | **Utilization** ×

Hierarchy

Name	Slice LUTs (63400)	Slice Registers (126800)	Block RAM Tile (135)	Bonded IOB (210)	BUFGCTRL (32)
TOP	13	88	58	1100	1
control_input_unit (control_input)	5	79	0	0	0
design1 (blk_mem_gen_0_0)	0	0	29	0	0
design_ (blk_mem_gen_1_1)	0	0	29	0	0
output_control_unit (output_control)	8	9	0	0	0

TESTBENCH CODE

```
`timescale 1ns / 1ps

module testbench;
 reg clk;
 reg reset;
 reg conv_run;
 reg [71:0] kernel;

 wire conv_done;
 wire [1023:0] data_out;

 TOP top_unit (
   .clk(clk),
   .reset(reset),
   .conv_run(conv_run),
   .kernel(kernel),
   .conv_done(conv_done),
   .data_out(data_out)
 );

 initial begin
   clk=1'b0;
   kernel[7:0] = -8'd1;kernel[15:8] = -8'd1;kernel[23:16] = -8'd1;kernel[31:24] = -8'd1;kernel[39:32] = 8'd8;kernel[47:40] = -
8'd1;kernel[55:48] = -8'd1;kernel[63:56] = -8'd1;kernel[71:64] = -8'd1;
   $fwrite("input_image.txt", kernel);
   #10 reset = 1;
   #10 reset = 0;
   #10 conv_run = 1;
   #10 conv_run = 0;
   #10 conv_run = 1;
 end
```

MATLAB

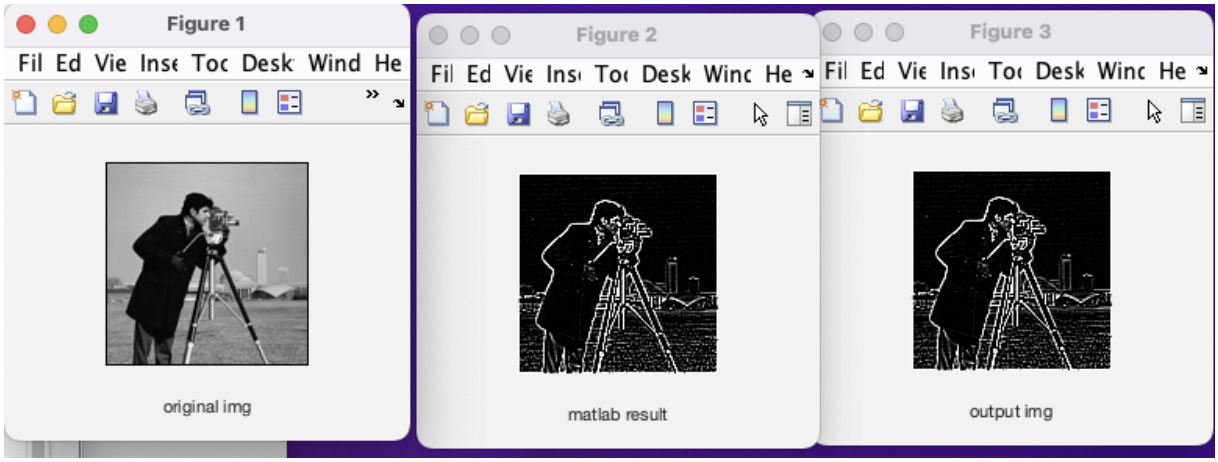


Figure 1
original img
Figure 2
matlab result
Figure 3
output img