Istanbul Technical University

Digital System Design Applications

Experiment VIII
Image Processing System

In this experiment, students will implement an image processing system. A zero padded input image will be loaded up to a RAM row by row. Using a given kernel, 2-D convolution will be applied on the input image. Output image will be recorded to a text file, then will be examined in MATLAB.

Objectives

- Creating an image processing system from scratch.
- Testing the created system by using a sample grayscale image.
- Program testbenches to generate output text files.
- Verify the circuit operation by examining the output image.

Requirements

Students are expected to know;

- The general concept of 2-D convlution,
- How to read state diagrams and express them in Verilog.
- How to set and instantiate a Xilinx Block RAM IP.
- How to record circuit outputs to a text file using a testbench.

Experiment Report Checklist

1. Codes for newly created modules (do not include the modules written in previous experiments).
2. Clear explanations for designing the processes of submodules.
3. Show post-sytnhesis FPGA resource utilization of the full design (number of LUTs, flip flops and I/O).
4. Determine the maximum clock frequency for the full system.
5. Testbench code, if edited or re-written.
6. Waveform of the simulation for one iteration, especially show that BRAM addresses and state registers are changing according to the algorithmic state machine shown in Figure 4.
7. MATLAB screenshots of input and output images.

- **Projects and reports are to be done INDIVIDUALLY. High amounts of points will be deducted from similar works.**
- **Reports must be written in a proper manner. Divide your text to sections and subsections if needed, label your figures and connect your sections with proper explanations of your works. Reports filled with imprecisely placed tables and figures, with no verbal explanations in workflow, will not fare well.**

Implementation of an Image Processing System

Starting Up

- Create a new Vivado project.
- Add the source files from the previous experiments.
  1. The 2-D convolution module named CWODSP from Experiment 7 will be used in this design. The file "convolution_circuits.v" from Experiment 7 for module MULTB and CWODSP must be added to the project.
  2. The adders from Experiment 4: Half adder, full adder and parametric ripple carry adder.
- After including these files to the project, be sure to check your source hierarchy and confirm all modules are set.
- An image processing system with the top level schematic given at Figure 1 will be realized in the scope of this experiment.
  - BRAM will be an instance of Xilinx Block RAM IP.
  - 2-D CONV will be an instance of CWODSP.
  - Other modules will be created from scratch.

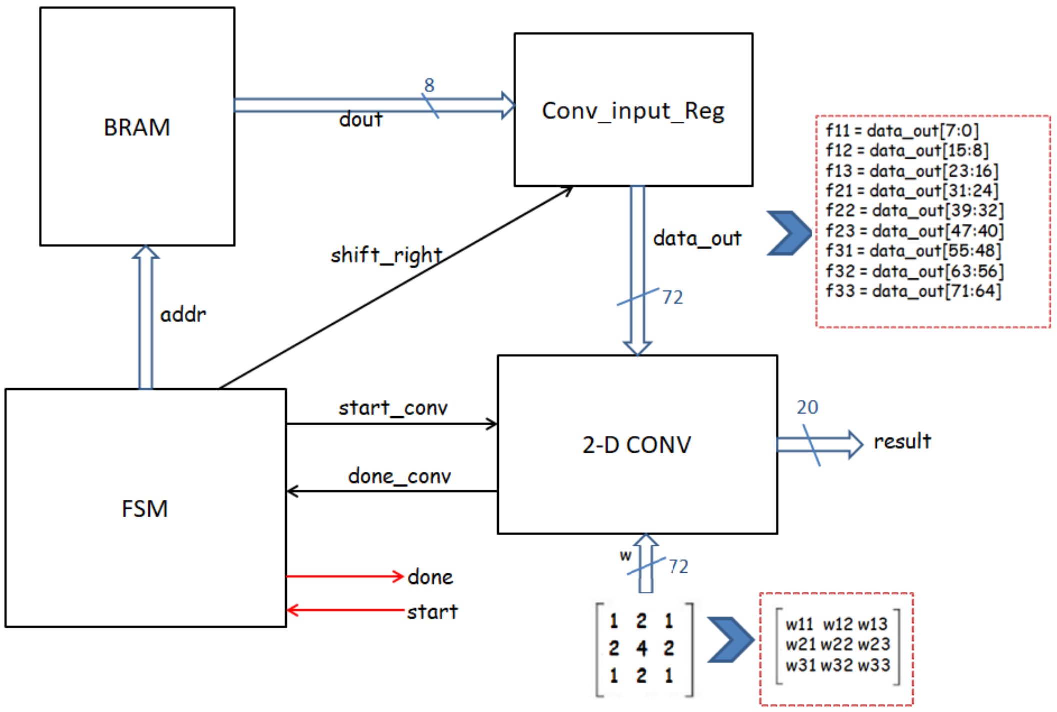


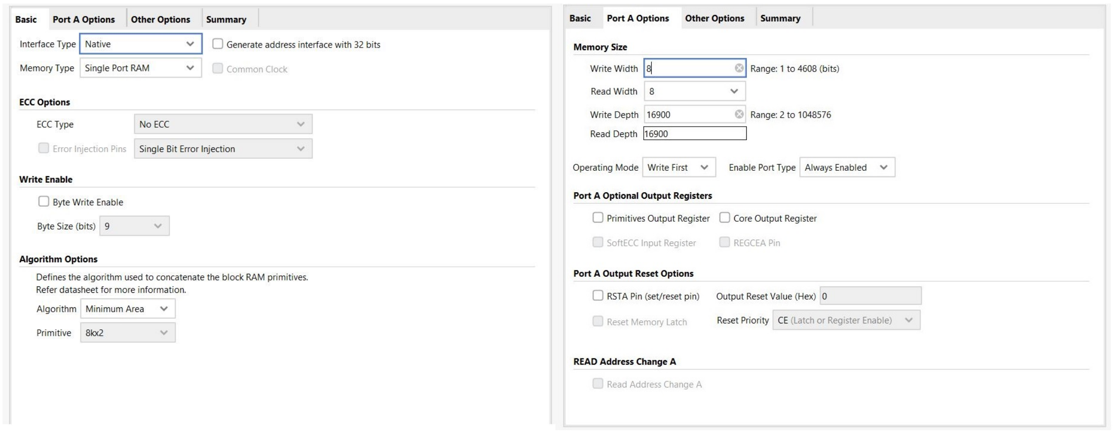
Figure 1: Top level schematic of the image processing system.

Creating Submodules

Block RAM

- Block RAM will contain the input image, which is given as a .coe file (see Ninova Course Files, under /Experiment_Files/Experiment_8/image.coe).

- Create a block BRAM IP using the settings given at Figure 2. Leave unspecified settings as default.
- Load up the "image.coe" as your BRAM's initialization file. This file has 8-bit values for each pixel, read row by row starting at top left. The original image is a 128×128 grayscale image. It's edges were padded with zeros, thus the file is $130 \times 130 = 16900$ pixels large.
- An illustration of a zero padded 5×5 image is shown in Figure 3, with BRAM addresses shown in red to indicate pixel order specified in "*.coe" file. If this example figure is considered, it will be seen that the original image given for this experiment will start at address decimal 131.
- The BRAM will only be used for reading data.



Basic | Port A Options | Other Options | Summary

Interface Type: Native
☐ Generate address interface with 32 bits
Memory Type: Single Port RAM
☐ Common Clock

ECC Options
ECC Type: No ECC
☐ Error Injection Pins: Single Bit Error Injection

Write Enable
☐ Byte Write Enable
Byte Size (bits): 9

Algorithm Options
Defines the algorithm used to concatenate the block RAM primitives.
Refer datasheet for more information.
Algorithm: Minimum Area
Primitive: 8kx2

Basic | Port A Options | Other Options | Summary

Memory Size
Write Width: 8 Range: 1 to 4608 (bits)
Read Width: 8
Write Depth: 16900 Range: 2 to 1048576
Read Depth: 16900

Operating Mode: Write First
Enable Port Type: Always Enabled

Port A Optional Output Registers
☐ Primitives Output Register ☐ Core Output Register
☐ SoftECC Input Register ☐ REGCEA Pin

Port A Output Reset Options
☐ RSTA Pin (set/reset pin) Output Reset Value (Hex): 0
☐ Reset Memory Latch Reset Priority: CE (Latch or Register Enable)

READ Address Change A
☐ Read Address Change A

Figure 2: Settings for Block RAM IP.

Convolution Input Register

- Convolution input register will hold a 3×3 part of the input image to be delivered to the 2-D Convolution block.
- Create a new source Verilog file named "conv_input_reg.v" and create a module here with the same name.
- The module "conv_input_reg" should have two 1-bit inputs, "clk" and "enable"; an 8-bit input named "data_in" and a 72-bit output reg named "data_out".
- With each clock cycle, if "enable" input is set, "data_out" will be shifted right by one byte, with it's most significant byte replaced by "data_in".
- It will stay unchanged when "enable" input is zero.

Control Finite State Machine

- Create a new Verilog file named "control_FSM.v" and create a module here with the same name.
- Set the module inputs and outputs as seen in Figure 1. Declare outputs as regs.

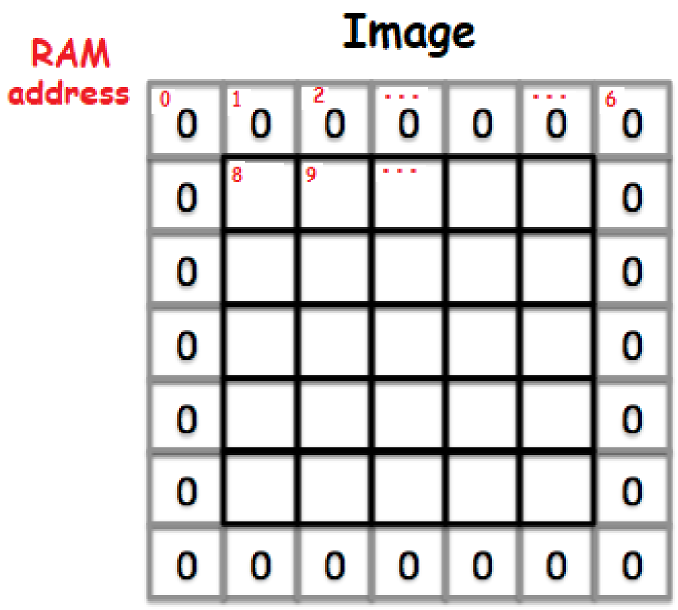


Figure 3: A zero padded image, reading order is specified with RAM addresses shown in red.

- This circuit should implement the algorithmic state machine shown in Figure 4. The white rectangles in Figure 4 indicates the states. The blue soft corner rectangles in Figure 4 indicates the actions realized during the state above. The brown diamond in Figure 4 indicates the decisions taken in the above state.
- Coding can be done in any desired style, but using case structure with state names as shown in Figure 4 pre-assigned to parameters is recommended.
- Students must come up with a method to correctly specify BRAM addresses for reading the required pixel to be used in convolution calculations (Refer to Figure 3 to understand how the image is stored in BRAM). Values i and j may be used as pixel counters. You may also define a reg to hold current pixel address that is currently being read for convolution.

Forming the Top Module

- When all submodule designs are completed, the system must be gathered up in a top level module as seen at Figure 1. Create a new Verilog source file named "TOP.v" and instantiate all the blocks seen at Figure 1 here.
- Top module will have 1-bit input signals named "clk" and "start", 1-bit output signal "done" and 20-bit output signal named result.
- Make all module connections here using wires.
- Figure 5 shows the application that is going to be performed on this top module, and it specifies the kernel for convolution to be used. Use these coefficients as inputs for your 2-D convolution module.

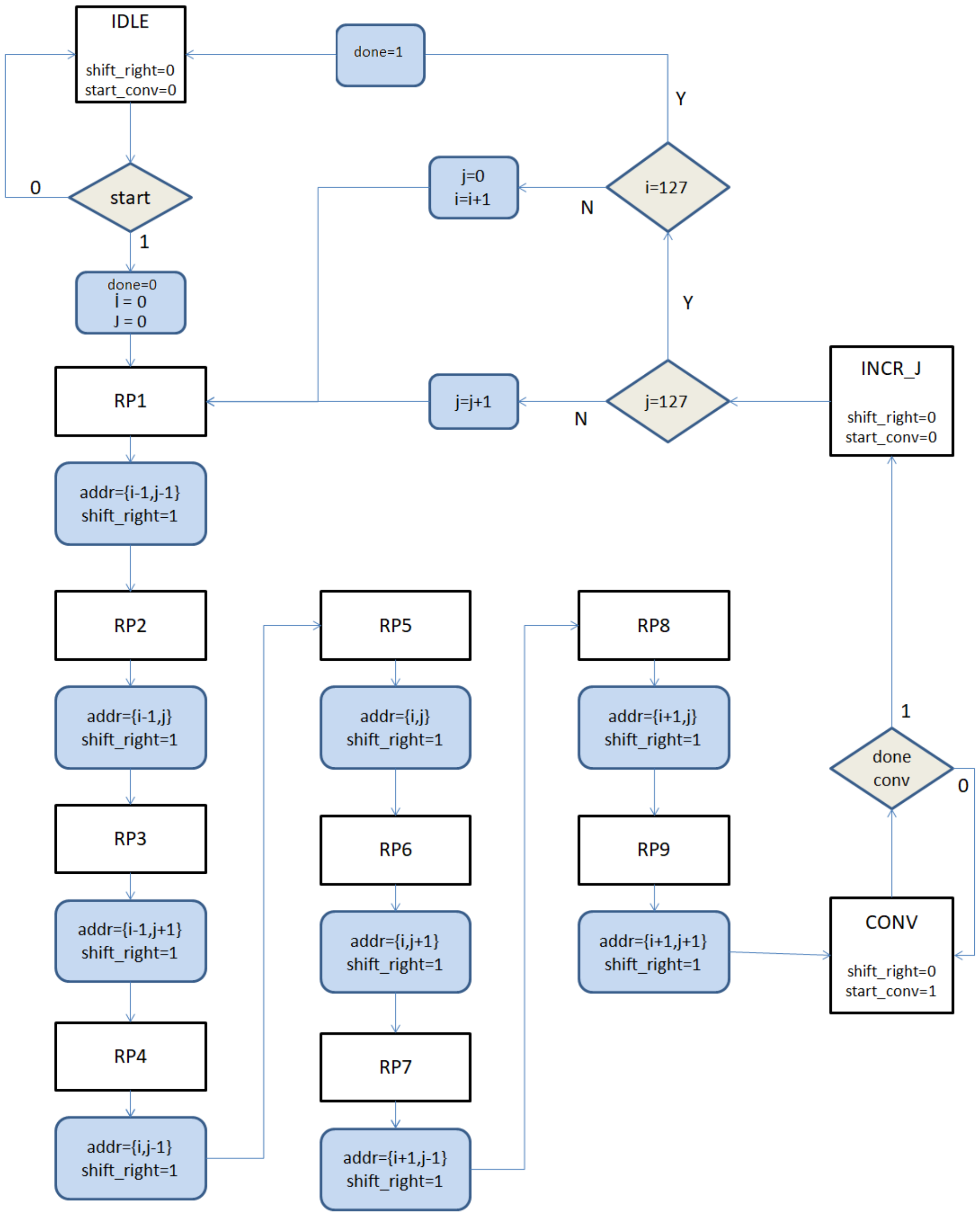


Figure 4: Algorithmic state machine for module control_FSM.

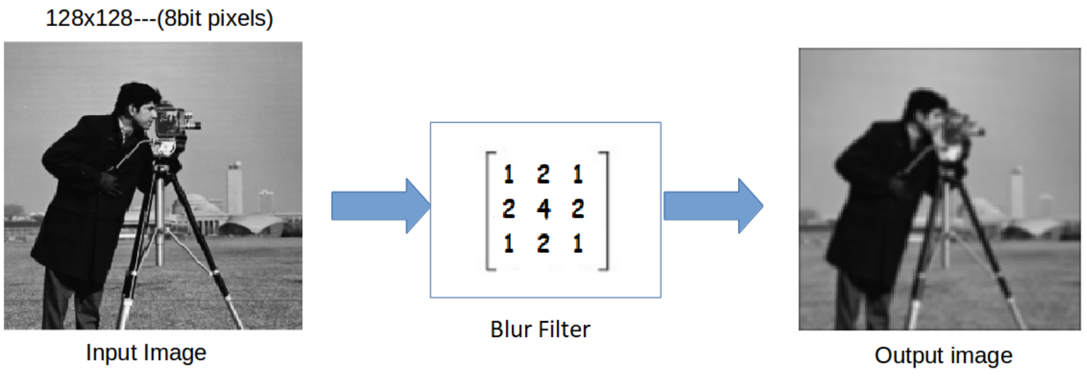


Figure 5: Operation performed by the overall image processing system in this experiment.

Testing the Image Processing System

- Download the sample testbench at Ninova course files, at "/Experiment_Files/Experiment_8/exp8_tb.v". You may also write your own testbench or edit the one that's provided. If you use different signal names or instantiation names, edit them accordingly in the sample testbench. In either way, the output "result" should be written to a text file in the testbench.
- Simulate your design using your testbench and obtain the output text file, containing pixel values of your output image.
- You must clear all lines with X in the output text file, if it contains any. If you're using the sample testbench, your output file should have $128 \times 128 = 16384$ lines, each representing a pixel of the output image. The output text file will be located under "your_project_folder/project_name.sim/sim_1/xsim/behav".
- Download the sample MATLAB script, "test_image.m" for viewing your output text file as an image by using Matlab.
- You can also download the input image in "image.coe" as a text file, named "input_image.txt".
- View the input and output images by using the corresponding text files.
- MATLAB script must be located in the same folder with input and output text images.
- Once the script is ran, you should be seeing the blurred version of the input image, just like at Figure 5.

References:

1. http://www.songho.ca/dsp/convolution/convolution.html
2. Xilinx ISim User Guide (UG660)
3. Xilinx Vivado Design Suite Tutorial: Using Constraints (UG945)
4. Vivado Design Suite Properties Reference Guide (UG912)