FPGA Based Processor Design

Down Sample an Image

Dias, K. M. G. D. A. W.	140125B
Fernando, H. D.	140154L
Fernando, W. A. S. C.	140163M
Rasanka, D. V. Y.	140517E

This is submitted as a partial fulfillment for the module
EN3030: Electronic Circuits and System Design
Department of Electronic and Telecommunication Engineering
University of Moratuwa

05st of April, 2017

1. INTRODUCTION

The objective of this project is to design a task specific microprocessor and CPU (Central Processing Unit) model to down-sample an image after eliminating its high frequency components and simulate the processor design using the Verilog hardware description language (HDL), and finally to implement it in hardware using programmable logic device such as Field Programmable Gate Array (FPGA). As an approach, design of an Instruction Set Architecture (ISA) and devising a suitable algorithm to down-sample the image are undertaken. The document describes the algorithm used to eliminate the high frequency components of the image, down-sample of the image and the ISA of the processor to be implemented on FPGA.

2. PROBLEM STATEMENT

Problem state identifies the problem to be addressed in detail.

The main requirement is to design a CPU and a microprocessor for filtering and down sampling a given image converting the 512 x 512 pixel image to a 256 x 256 pixel image. The CPU should communicate with the computer in order to receive the image. Then it should save the received image in the main memory, as no such need for a secondary memory and process the image in order to down-sample. Afterwards, the CPU should send back the pixel values of the down-sampled image back to the computer Matlab programme to display the image. The task can be achieved by dividing the task into sub tasks.

Take input and store in the RAM

Matlab software generates a 1D array using the 2D image so that it can be transmitted to the RAM in FPGA board by Universal Asynchronous Receive and Transmit (UART) protocol byte by byte. UART protocol should be implemented inside the processor to receive and transmit data. RAM memory should be accessed and the pixel values should be stored one after the other in the memory. Transmission and receive of the pixel values is a major concern in the project. Apart from transmitting the image to the designed processor, there may be a need to transfer the instructions written in assembly language equivalent programme to down-sample the image.

Filter the image

Simplest way to down-sample the image is to take every other pixel from the received image and then transmit the pixel values. The issue with this method is that it cannot preserve the qualities of the original image. High frequency components in are the cause of the mentioned issue. This issue can be slightly overcome by averaging the near pixel value and then down-sampling the image but still it would not yield a good outcome. Therefore, in order to obtain a reduced image with the features identical to the original image, the image should be filtered in order to eliminate the high frequency components from the image. This task can be done by using Gaussian filter implantation in the FPGA. After processing the raw data with Gaussian filter, processed data can be stored in the same primary memory.

Down sample the filtered image

After filtering the original image, in this section we consider down sampling the filtered image. As given in the requirements in this process image has to be down sampled into half of the size. Down-sampling process also should be implemented on the FPGA and after doing the down sampling process; data can be stored in the primary memory by overwriting the data written to the memory.

Send back the image and display

After doing the given tasks to the original image the processed data should be returned serially to back to the computer and the down-sampled image should be displayed. UART transmitter should be implemented on the FPGA in order to transmit the processed data from FPGA to the computer. Serial communication software (or MatLab) can be used to collect the data.

Down-sample image using Matlab

Apart from receiving the image, Matlab software should be used to down-sample image so that the results can be compared to access the quality of the image down-sampled inside the designed processor on FPGA board.

3. OVERVIEW OF THE SOLUTION

Field Programmable Gate Array can be used to design a processor capable of down-sampling an image after eliminating the high frequency components of the image. Atlys Spartan-6 FPGA board is used to implement the designed processor.

The first and foremost task is to design the UART receiving model so that the image can be transferred to the RAM inside the FPGA using serial communication. Then, a processor is specifically designed to handle the arithmetic and logical operations needed to perform the required operations. ISA is capable of addressing the operations required for the implementation of the algorithm.

Design can be done by dividing the complex solution into smaller sub modules.

The procedure can be identified as design of ISA and processor architecture, model the designed architecture using Verilog HDL, test the correctness of the implementation by simulation results, and implement the design in the FPGA board.

The solution includes receive the image through UART and writing the data in the memory, filter and down-sample the image using the designed processor, and send the down-sampled image back to the computer.

The effectiveness of the solution can be verified by carrying out an error analysis using MATLAB.

4. INSTRUCTION SET ARCHITECTURE (ISA)

General Architecture

The processor is specifically designed to down sample an image of the size 512 x 512. The processor ISA designed so that it adheres to the requirement of the storage of all the pixel values and constraints of the FPGA board elements.

Data Memory - A data RAM which consists of 512 x 512, i.e. 218 memory locations with a width of 8 bits (1 Byte) to store the pixels of the image. The selection of the width of the memory was based on the pixel value range 0-255 of the image and depth of the memory was based on the number of pixel values in the image.

Instruction Memory - An instruction RAM which consists of instructions to be executed with 65536 memory locations and a width of 1 Byte same as in the DRAM. This basically contains the assembly code of the algorithm for filtering and down sampling the image.

Memory Address Register (MAR) – A 24 bit address register to select an address of the data RAM so that the data can be either read or write to the selected location of the data RAM.

Programme Counter (PC) – A 16 bit program counter which keeps the address of the next instruction in the instruction memory. PC selects the memory location of the instruction memory so that the instruction can be loaded to the Instruction Register.

Instruction Register (IR) – An 8 bit register to store the instructions that are read from the Instruction Memory. Instructions then are taken to the control store or state machine to generate the control signals to the units.

Accumulator (AC) – A 24 bit accumulator has direct access to the ALU via A bus. AC is the most widely used register in the ISA.

R, R1, R2, R3, R4 - Four 24 bit General Purpose Registers. These registers are used to store the intermediate variables used in the processing of the image.

Arithmetic and Logic Unit (ALU) –A 16 bit ALU performs different arithmetic and logical operations needed to filter and down-sample the image. A, B busses are used to input data in to the ALU and C bus gives the output. AC is directly connected to the ALU.

State Machine / Control Store –It generates all the control signals for the processor and makes the decision based on the given instruction. (From IR)

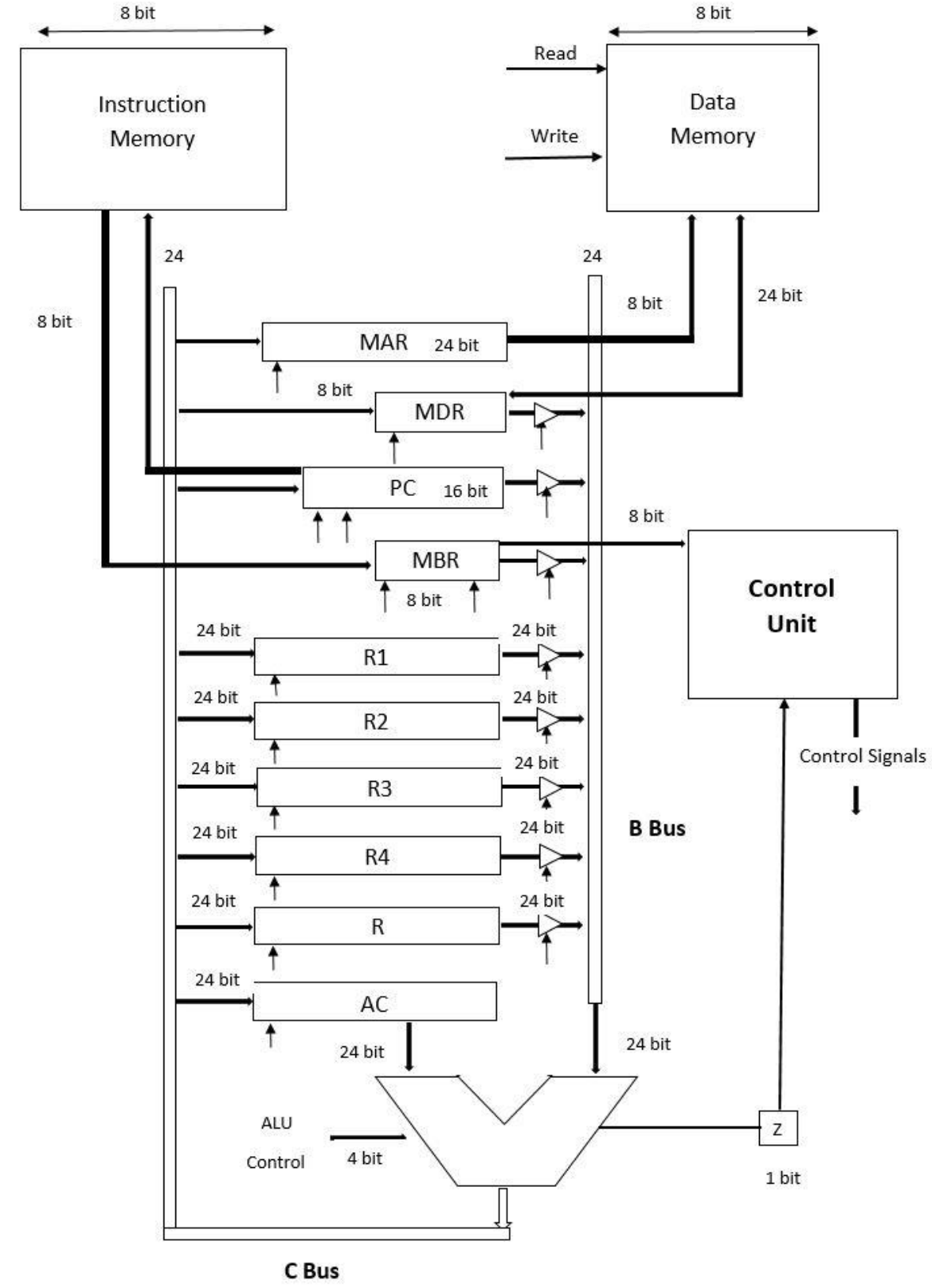
BUS – 24 bit wire which carries the data parallel from registers, Data memory, Instruction Memory, and ALU.

Instruction Set

Instruction	Instruction Code	Operation
NOP	8	No operation
LDAC	8 Ґ	AC ← DRAM [Ґ]; r is a 24 bit location in MAR
LDACV	8 V (16 bit)	AC ← V, V is loaded from IRAM
STAC	8 Ґ	DRAM [Ґ] ← AC; r is a 24 bit location in MAR
MVACMAR	8	MAR ← AC
MVACR	8	R ← AC
MVACR1	8	R1 ← AC
MVACR2	8	R2 ← AC
MVACR3	8	R3 ← AC
MVACR4	8	R4 ← AC
MOVR	8	AC ← R
MOVR1	8	AC ← R1
MOVR2	8	AC ← R2
MOVR3	8	AC ← R3
MOVR4	8	AC ← R4
JUMP	8 Ґ (16 bit)	GOTO IRAM [Ґ], Ґ is 16 bits
JMPZ	8 Ґ (16 bit)	IF (Z=1) THEN GOTO IRAM [Ґ], Ґ is 16 bits
JMNZ	8 Ґ (16 bit)	IF (Z=0) THEN GOTO IRAM [Ґ], Ґ is 16 bits
ADD	8	AC ← AC + R
ADDV	8 V(16 bit)	AC ← AC + V, V is loaded from IRAM
SUB	8	AC ← AC – R IF (AC – R = 0), THEN Z = 1, ELSE Z = 0
SUBV	8 V(16 bit)	AC ← AC – V, V is loaded from IRAM IF (AC – V = 0), THEN Z = 1, ELSE Z = 0
INAC	8	AC ← AC + 1
DEAC	8	AC ← AC – 1
MUL	8	AC ← AC << R
MULV	8 V (8 bit)	AC ← AC << V
DIV	8	AC ← AC >> 4
INR1	8	R1 ← R1 + 1
INR2	8	R2 ← R2 + 1
CLAC	8	AC ← 0; Z ← 1
AND	8	AC ← AC and R
OR	8	AC ← AC or R
XOR	8	AC ← AC xor R
NOT	8	AC ← AC’

Design of the Data Path

The data path was designed taking the ISA and Mic-1 architecture into consideration.



```
% Read and map the 2D image to a 1D memory array
close all;
image = imread('C:\Users\adhit\Desktop\Processor Design\Matlab Code\Emacs_512.png'); %
Read the image and save 2D array
image = rgb2gray(image);
memory_array = uint16(image(:)); % Make 1D array, convert to 16 bit since registers are 16
imshow(image);                  % Display Image
imwrite(image, 'C:\Users\adhit\Desktop\Processor Design\Matlab Code\gray.png');     %
Write gray scale image
% Gaussian filter the image
total = 0;              % Initialize the totoal variable

for j = 1:1:510          % Loop through rows
   for i = 2:1:511       % Loop through columns
      x = j*512+i;
      % Convolve with the gaussian filter
      total = total + memory_array(x-1)*2;
      total = total + memory_array(x)*4;
      total = total + memory_array(x+1)*2;
      total = total + memory_array(x+513);
      total = total + memory_array(x+512)*2;
      total = total + memory_array(x+511);
      total = total + memory_array(x-513);
      total = total + memory_array(x-512)*2;
      total = total + memory_array(x-511);
      total = total/16;                  % Normalize the total value
      memory_array(x-513) = total;          % Store the filtered value in the memory
      total=0;
   end
end

figure, memory_array = uint8(memory_array);
filtered_image = memory_array; % Convert to uint8 format
c = reshape(filtered_image,512,512);  % Generate 2d array from vector
imshow(c);                      % Display the filtered image

% Downsample the filtered image
k = 1;                              % Set writing memory address
for j = 0:1:255                        % loop going through rows
   for i = 0:1:255                     % loop going through columns
      y = 2*512*j + 2*i + 1;              % Map every other pixel to RAM address in memory
      memory_array(k) = memory_array(y);      % Overwrite the pixel values with every other
      k = k+1;                         % Increment writing memory address
   end
end

figure, downsampled_image = memory_array(1:65536);      % Bytes relevant to 256*256
c = reshape(downsampled_image,256,256); %         Reshape the 1D array to an image
imshow(c); % Display the filted image
```

Original Image 512 x 512 size

Filtered Image 512 x 512 size

Down-sampled image 256 x 256 size

Published with MATLAB® R2014b