FYP Final Presentation
“Hardware-Based Face Detection”

Supervisor:
Dr. Benjamin C. Schafer
Project ID: schaferb_20130322183957

<table>
<thead>
<tr>
<th>Student</th>
<th>Qixuan ZHANG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student ID</td>
<td>10802747D</td>
</tr>
<tr>
<td>Venue</td>
<td>Room DE304</td>
</tr>
<tr>
<td>Time</td>
<td>12:40 - 1:00 pm</td>
</tr>
<tr>
<td>Date</td>
<td>May 11st, 2015</td>
</tr>
</tbody>
</table>
Outline

• Recall memory

• Project Schedule & Milestones

• Methodology

• Hardware Implementation

• Result & Conclusion

• Further Development

• Q&A
Recall your memory

=> Face Detection

- Wide applications
  - Auto Focusing for Digital Cameras
  - Face Recognition
  - Video surveillance, and etc…

- Skin-Color-Based Face Detection
  - Simple implementation
  - Using fewer FPGA resources
Recall your memory  =>  FPGA

- Reconfigurable “Versatile Chips”

Field-Programmable Gate Array (FPGA)

Objectives

• To study on the image & video processing techniques, especially ones related to face detection;

• To be familiar with the FPGA system design and develop deeper understanding on its characteristics:
  • Compared to sequential micro-processors
    ✓ Parallel Execution
      => Hardware acceleration
      => Faster processing for specific applications

  • Compared to custom ASIC design
    ✓ Reconfigurable
      => Faster prototype
      ⇒ Low-cost verification

• To implement an FPGA-Based Face Detection System
<table>
<thead>
<tr>
<th>Month/Year</th>
<th>Task</th>
<th>Progress</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/2014</td>
<td>Background Learning and Basic Understanding Establishment</td>
<td>✔️</td>
</tr>
<tr>
<td>10/2014</td>
<td>Trials on C Sobel Filter and get familiar with CWB &amp; QuartusII Tools</td>
<td>✔️</td>
</tr>
<tr>
<td>11/2014</td>
<td>DE2-115 Board Experiments and Basic Function Implementation like Camera, VGA Display and Simple Image Operations</td>
<td>✔️</td>
</tr>
<tr>
<td>12/2014</td>
<td>Edge Detection Operation and HLS of the C Sobel Filter Interim Presentation &amp; Report</td>
<td>✔️</td>
</tr>
<tr>
<td>01/2015</td>
<td>System Design and Operation Flow Trials in Software Approach</td>
<td>✔️</td>
</tr>
<tr>
<td>02/2015</td>
<td>Hardware Single Modules Design and Functionality Verification</td>
<td>✔️</td>
</tr>
<tr>
<td>03/2015</td>
<td>System Module Integration and Board Verification</td>
<td>✔️</td>
</tr>
<tr>
<td>04/2015</td>
<td>Testing, Improvement, Final Report and Presentation</td>
<td>✔️</td>
</tr>
</tbody>
</table>
Major Steps of Project Implementation

- Edge Detection Trial
- MATLAB Prototype
- Adapted into hardware suitable method
- Face Detection System
MATLAB Working Flow

1. Input Frame Images
2. Skin Segmentation/Thresholding
3. Morphological Filtering
4. Labeling Connect Components
5. Area Calculation
6. Spatial Filtering
7. Centroid Computation
8. Detection Result
Skin-Color-Based Face Detection System

- One PLL (Phase-Locked Loop) manages the clock utilization;
- One Reset_Delay Module manages the RESET functions;
CCD Camera Configuration By I2C

- Multi-master, multi-slave, single-ended, serial bus
- Used for attaching lower-speed peripherals to processors on computer motherboards or embedded systems
- Two critical bus lines
  - a serial data line (SDA)
  - a serial clock line (SCL)
CCD Camera Configuration By I2C

- Look-Up Table (LUT) built according to Hardware Specifications

```hls```
case(LUT_INDEX)
0 : LUT_DATA <= 24'h000000; // Mirror Row and Columns
1 : LUT_DATA <= 24'h200000; // Exposure
2 : LUT_DATA <= {8'h09, senosr_exposure}; // Exposure
3 : LUT_DATA <= 24'h050000; // H_Blanking
4 : LUT_DATA <= 24'h060019; // V_Blanking
5 : LUT_DATA <= 24'h0A8000; // change latch
6 : LUT_DATA <= 24'h2B0013; // Green 1 Gain
7 : LUT_DATA <= 24'h2C09A; // Blue Gain
8 : LUT_DATA <= 24'h2D019C; // Red Gain
9 : LUT_DATA <= 24'h2E0013; // Green 2 Gain
10 : LUT_DATA <= 24'h100051; // set up PLL power on

`ifdef VGA_640x480p60
11 : LUT_DATA <= 24'h111f04; // PLL_m_Factor<<8+PLL_n_Divider
12 : LUT_DATA <= 24'h120001; // PLL_p1_Divider
`else
11 : LUT_DATA <= 24'h111805; // PLL_m_Factor<<8+PLL_n_Divider
12 : LUT_DATA <= 24'h120001; // PLL_p1_Divider
`endif

13 : LUT_DATA <= 24'h100053; // set USE PLL
14 : LUT_DATA <= 24'h980000; // disable calibration
15 : LUT_DATA <= 24'hA00000; // Test pattern control
16 : LUT_DATA <= 24'hA10000; // Test green pattern value
17 : LUT_DATA <= 24'hA20FFF; // Test red pattern value
18 : LUT_DATA <= sensor_start_row ; // set start row
19 : LUT_DATA <= sensor_start_column ; // set start column
20 : LUT_DATA <= sensor_row_size; // set row size
21 : LUT_DATA <= sensor_column_size; // set column size
22 : LUT_DATA <= sensor_row_mode; // set row mode in bin mode
23 : LUT_DATA <= sensor_column_mode; // set column mode in bin mode
24 : LUT_DATA <= 24'h4901A8; // row black target

default:LUT_DATA <= 24'h000000;
```hls```
Pre-processing

--- Raw image data processing

- Bayer Pattern => RGB

The Bayer arrangement of color filters on the pixel array of an image sensor

Interpolation to get RGB components
Frame Buffer (Multi-Port SDRAM Controller)

- Multi-Ports
  - 2 writing ports + 2 reading ports

- FIFO Control
  - The video frames are captured real-time and buffered in FIFO

- Critical issues
  - R/W Bandwidth (16-bit each port)
  - Read/Write Synchronization
  - Memory Utilization Efficiency

```plaintext
// FIFO Write Side 1
WR1_DATA,
WR1,
WR1_ADDR,
WR1_MAX_ADDR,
WR1_LENGTH,
WR1_LOAD,
WR1_CLK,
// FIFO Write Side 2
WR2_DATA,
WR2,
WR2_ADDR,
WR2_MAX_ADDR,
WR2_LENGTH,
WR2_LOAD,
WR2_CLK,
// FIFO Read Side 1
RD1_DATA,
RD1,
RD1_ADDR,
RD1_MAX_ADDR,
RD1_LENGTH,
RD1_LOAD,
RD1_CLK,
// FIFO Read Side 2
RD2_DATA,
RD2,
RD2_ADDR,
RD2_MAX_ADDR,
RD2_LENGTH,
RD2_LOAD,
RD2_CLK,
```
Color Space Conversion

✓ RGB $\leftrightarrow$ YCbCr

=>$\textbf{Luminance}$ component is separated from $\textbf{chrominance}$ component

=>$\text{used in skin segmentation}$

$$
\begin{bmatrix}
Y \\
C_b \\
C_r
\end{bmatrix} = 
\begin{bmatrix}
0.299 & 0.587 & 0.114 \\
-0.169 & -0.331 & 0.500 \\
0.500 & -0.419 & -0.081
\end{bmatrix} \cdot 
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix} + 
\begin{bmatrix}
0 \\
128 \\
128
\end{bmatrix}
\text{ }\text{ }Y \in [0, 255] \text{ }\text{ }C_b \in [0, 255] \text{ }\text{ }C_r \in [0, 255]
$$

(F.15)

• Techniques to deal with FP
  • Binary representation
  • Shift 10 bits to left => shift back!
  • MAC MegaCores (Multiplication & Addition)
Skin Segmentation

- YCbCr Chrominance Component Ranges
  - Theoretical range proposed by previous scholars, D. Chai & K. N. Ngan,
    "Face segmentation using skin color map in videophone applications"
    \[ 77 < Cb < 127 \]
    \[ 133 < Cr < 177 \]
  - Value range should be adjusted according to the real environment setting for the reason of lighting noise
Post-Processing

=>

Spatial Filtering + Temporal Filtering + Centroid Computation
Spatial Filtering

- **Window Size & Threshold** should be adjusted for better performance
  - 9 Rows * 9 Columns = 81 pixels
  - Threshold = 78
Temporal Filtering

- Time-Weighted Average Module
  (learnt from the averaging module by Prof. Bruce Lund, Cornell University)
- Weighting for the old average and later pixel data

\[
\text{out\_avg} = \frac{3}{4} \text{in\_avg} + \frac{1}{4} \text{pixel\_data}
\]

**Multiplication**
\[
\text{out\_avg} = \text{in\_avg} - \frac{1}{4} \text{in\_avg} + \frac{1}{4} \text{pixel\_data}
\]

**Bit Shifting**
\[
\text{out\_avg} = \text{in\_avg} - \text{in\_avg} >> 2 + \text{pixel\_data} >> 2
\]

<table>
<thead>
<tr>
<th>Frame Pixel</th>
<th>t</th>
<th>t+1</th>
<th>t+2</th>
<th>t+3</th>
<th>t+4</th>
<th>t+5</th>
<th>t+6</th>
<th>t+7</th>
<th>t+8</th>
<th>t+9</th>
<th>t+10</th>
<th>t+11</th>
<th>t+12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before Filtering</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Average</td>
<td>0.00</td>
<td>0.25</td>
<td>0.44</td>
<td>0.58</td>
<td>0.43</td>
<td>0.58</td>
<td>0.68</td>
<td>0.76</td>
<td>0.82</td>
<td>0.62</td>
<td>0.46</td>
<td>0.60</td>
<td>0.70</td>
</tr>
<tr>
<td>After Filtering</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Centroid Computation

--- To mark the faces
--- For further facial feature verification

- Calculate the centroids of the candidate regions
  - By averaging the sum of X,Y Coordinates

- At most two faces can be detected!
  - Assume the two faces can only be aligned horizontally
VGA Control by FPGA

// Horizontal Parameter
parameter H_FRONT = 16;
parameter H_SYNC = 96;
parameter H_BACK = 48;
parameter H_ACT = 640;
parameter H_BLKBACK = H_FRONT+H_SYNC+H_BACK;
parameter H_TOTAL = H_FRONT+H_SYNC+H_BACK+H_ACT;

// Vertical Parameter
parameter V_FRONT = 11;
parameter V_SYNC = 2;
parameter V_BACK = 31;
parameter V_ACT = 480;
parameter V_BLKBACK = V_FRONT+V_SYNC+V_BACK;
parameter V_TOTAL = V_FRONT+V_SYNC+V_BACK+V_ACT;
VGA Display + Pixel Coordinates Synchronization

- Used to record the **corresponding coordinates** for each operating pixels;
- Post-processing causes certain cycles **delay**
  
  \[ \Rightarrow \text{Coordinate signals are delayed for specific cycles;} \]
Hardware Implementation Result
## Hardware Synthesis Report

<table>
<thead>
<tr>
<th>Module</th>
<th>Total logic elements</th>
<th>Total combinational functions</th>
<th>Dedicated logic registers</th>
<th>Total registers</th>
<th>Total memory bits</th>
<th>Fmax (Slow 1200mV 85C)</th>
<th>Fmax (Slow 1200mV 0C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Post_Processing</td>
<td>18,606 / 114,480 (16%)</td>
<td>18,520 / 114,480 (16%)</td>
<td>6,865 / 114,480 (6%)</td>
<td>6865</td>
<td>0 / 3,981,312 (0%)</td>
<td>9.74 MHz</td>
<td>10.8 MHz</td>
</tr>
<tr>
<td>skin_seg</td>
<td>41 / 114,480 (&lt; 1%)</td>
<td>11 / 114,480 (&lt; 1%)</td>
<td>31 / 114,480 (&lt; 1%)</td>
<td>31</td>
<td>0 / 3,981,312 (0%)</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>RAW2RGB</td>
<td>102 / 114,480 (&lt; 1%)</td>
<td>93 / 114,480 (&lt; 1%)</td>
<td>72 / 114,480 (&lt; 1%)</td>
<td>72</td>
<td>30,672 / 3,981,312 (&lt; 1%)</td>
<td>249.63 MHz</td>
<td>272.7 MHz</td>
</tr>
<tr>
<td>Reset_Delay</td>
<td>54 / 114,480 (&lt; 1%)</td>
<td>54 / 114,480 (&lt; 1%)</td>
<td>37 / 114,480 (&lt; 1%)</td>
<td>37</td>
<td>0 / 3,981,312 (0%)</td>
<td>218.05 MHz</td>
<td>239.98 MHz</td>
</tr>
<tr>
<td>CCD_Capture</td>
<td>89 / 114,480 (&lt; 1%)</td>
<td>88 / 114,480 (&lt; 1%)</td>
<td>80 / 114,480 (&lt; 1%)</td>
<td>80</td>
<td>0 / 3,981,312 (0%)</td>
<td>272.03 MHz</td>
<td>295.86 MHz</td>
</tr>
<tr>
<td>RGB2YCbCr</td>
<td>481 / 114,480 (&lt; 1%)</td>
<td>457 / 114,480 (&lt; 1%)</td>
<td>195 / 114,480 (&lt; 1%)</td>
<td>195</td>
<td>0 / 3,981,312 (0%)</td>
<td>185.98 MHz</td>
<td>208.55 MHz</td>
</tr>
<tr>
<td>Sdram_Control</td>
<td>1,645 / 114,480 (1%)</td>
<td>1,394 / 114,480 (1%)</td>
<td>745 / 114,480 (&lt; 1%)</td>
<td>745</td>
<td>32,768 / 3,981,312 (&lt; 1%)</td>
<td>144.7 MHz</td>
<td>160.05 MHz</td>
</tr>
<tr>
<td>VGA_Ctrl</td>
<td>104 / 114,480 (&lt; 1%)</td>
<td>104 / 114,480 (&lt; 1%)</td>
<td>26 / 114,480 (&lt; 1%)</td>
<td>26</td>
<td>0 / 3,981,312 (0%)</td>
<td>408.0 MHz</td>
<td>447.43 MHz</td>
</tr>
<tr>
<td>I2C_CCD_Config</td>
<td>280 / 114,480 (&lt; 1%)</td>
<td>266 / 114,480 (&lt; 1%)</td>
<td>131 / 114,480 (&lt; 1%)</td>
<td>131</td>
<td>0 / 3,981,312 (0%)</td>
<td>229.15 MHz</td>
<td>247.4 MHz</td>
</tr>
<tr>
<td><strong>Complete Design</strong></td>
<td><strong>20,728 / 114,480 (18%)</strong></td>
<td><strong>20,391 / 114,480 (18%)</strong></td>
<td><strong>7,955 / 114,480 (7%)</strong></td>
<td><strong>7955</strong></td>
<td><strong>59,404 / 3,981,312 (1%)</strong></td>
<td><strong>166.0 MHz</strong></td>
<td><strong>183.42 MHz</strong></td>
</tr>
</tbody>
</table>
Conclusion

--- Performance Analysis

- **Functionality Achieved**
  - Good Functionality in Ideal Environment
    (lighting conditions, simple background)

- **Hardware or FPGA Resources Optimized**
  - Use optimized MegaCores for Altera Devices
  - Use fewer logic elements (18%)

- **Drawbacks**
  --- For the limitation of the algorithms adopted
  - Luminance Conditions
  - Facial Views
  - Skin-Color Objects

⇒ *Finding good thresholds for different environments is hard.*
⇒ *Picking bad thresholds yields many false positives.*
Further Issues To Be Considered

- **Aspect I – Algorithm View**
  Advanced Machine Learning Algorithms can be adopted for higher accuracy;

- **Aspect II – Hardware View**
  Customized hardware may be used, such as camera module, memory and DSPs.

- **Aspect III – Design Flow View**
  MATLAB to HDL Coder, Complete HLS Design Flow

- **Aspect IV – Performance Comparison View**
  Use the same algorithm for both software and hardware implementations and make comparisons.

- **Aspect V – Application View**
  Face Detection System with real applications, like security check and etc.
References

- Ramsri Goutham Golla, “Real-time Face Detection and Tracking”, Arizona State University
- K. Wakabayashi and Benjamin C. Schafer, ““All-in-C” Behavioral Synthesis and Verification with CyberWorkBench, From C to Tape-Out with No Pain and A Lot of Gain”, High-Level Synthesis from Algorithm to Digital Circuit, Springer, Chapter 7, 2008
References

- Prof. Bruce Land, “ECE5760 Advanced Micro-controllers Course Portal – Final Projects”, Cornell University, [Online], http://people.ece.cornell.edu/land/courses/ece5760/FinalProjects/
Thank you for your listening!

If no questions.....

Have a nice day! 😊
Recall memory => Face Detection

- Wide applications
  - Auto Focusing for Digital Cameras
  - Face Recognition
  - Video surveillance, and etc…

- Viola-Jones AdaBoost Face Detection, Neural Network Based Face Detection, PCA-Based Face Detection
  - Complicated
  - Resource-demanding

- Skin-Color-Based Face Detection
  - Simple implementation
  - Using fewer FPGA resources (Memory & LEs)
## Hardware Summary

<table>
<thead>
<tr>
<th>Function</th>
<th>Hardware Components</th>
<th>Core Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Processing and Control</strong></td>
<td><strong>Terasi FPGA DE2-115 Development and Education Board (Altera Cyclone IV 4CE115 FPGA device)</strong></td>
<td><strong>Control Unit, Processing Unit, Memory Unit and Communication Unit</strong></td>
</tr>
<tr>
<td><strong>Picture Capture</strong></td>
<td><strong>Terasi TRDB_D5M - 5 Mega Pixel Digital Camera Package</strong></td>
<td><strong>Picture Data Collection Unit</strong></td>
</tr>
<tr>
<td><strong>Display</strong></td>
<td><strong>PC Display with VGA Port</strong></td>
<td><strong>VGA Display Unit</strong></td>
</tr>
</tbody>
</table>

*Table 3 – Hardware list and corresponding tasks*
EDA Tools Summary

- Altera Quartus II 13.1
  => Verilog HDL Coding
- ModelSim-Altera Starter Edition
  => Simulation
- NEC CyberWorkBench (CWB)
  => HLS + Preliminary Simulation
  => From C to Verilog HDL
Edge Detection Trial

- **Sobel Filter**
  
  - Spatial Derivative
  
  (Horizontal + Vertical)

\[ \nabla_1 = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \quad \nabla_2 = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \]

\[
\frac{\partial I}{\partial x} \rightarrow \text{horizontal} \quad \frac{\partial I}{\partial y} \rightarrow \text{vertical}
\]

\[ |\nabla| = \sqrt{\nabla_x^2 + \nabla_y^2} \quad \text{gradient magnitude} \]
MATLAB Result

Pure Skin-Color-Based Face Detection

Combined-two-path method
Hardware Implementation

=> Detection Method was adjusted for hardware consideration.

<table>
<thead>
<tr>
<th>Issues to consider</th>
<th>Software (MATLAB)</th>
<th>Hardware (Verilog)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Resources &amp; Access</td>
<td>The whole frame image are stored and any pixel can be accessed whenever we want;</td>
<td>Specific pixel data in a frame cannot always be stored so they cannot be accessed whenever we want;</td>
</tr>
<tr>
<td>Code Implementation &amp; Timing</td>
<td>Sequential execution;</td>
<td>Parallel execution;</td>
</tr>
<tr>
<td></td>
<td>Delay---Wait for the preceding executions;</td>
<td>Lower CLK frequency but tasks can be divided into several cycles and many tasks can be handled at the same time (Pipelines);</td>
</tr>
</tbody>
</table>
**Basic System Considerations**

- **Tradeoff**
  - Store 10-bit R, G, B data
  - Color Space Conversion after Frame Buffer
    - More bits and more memory
  - Higher video acquisition rate
  - Better synchronization of the pixel coordinates
always @(posedge fCLK or begin
    if (first_N)
        begin
            H_Cnt <= OvGA_H;
        end
    else
        begin
            if (H_Cnt == OvGA_H)
                H_Cnt <= 0;
            if (f_CLK)
                H_Cnt <= H_Cnt + 1;
        end
    if (H_Cnt == H_FRONT)
        H_Cnt <= 0;
    if (H_Cnt == H_SYNC)
        H_Cnt <= H_Cnt + 1;
    if (H_Cnt == H_BACK)
        H_Cnt <= H_Cnt + 1;
    if (H_Cnt == H_ACT)
        H_Cnt <= H_Cnt + 1;
    if (H_Cnt == H_BLANK)
        H_Cnt <= H_Cnt + 1;
end

always @(posedge OVGA_H)
begin
    if (first_N)
        begin
            V_Cnt <= OVGA_V_SYNC_CC1K;
        end
    else
        begin
            if (V_Cnt == V_FRONT)
                V_Cnt <= V_Cnt + 1;
            else
                V_Cnt <= 0;
                if (V_Cnt == V_FRONT)
                    OVGA_V_SYNC_CC1K <= 1'b0;
                if (V_Cnt == V_SYNC)
                    OVGA_V_SYNC_CC1K <= 1'b1;
        end
end

assign oRequest = ((H_Cnt >= H_BLANK && H_Cnt <= H_TOTAL) && (V_Cnt >= V_BLANK && V_Cnt <= V_TOTAL));
assign oCurrent_X = (H_Cnt >= H_BLANK) ? H_Cnt - H_BLANK : 11'h0;
assign oCurrent_Y = (V_Cnt >= V_BLANK) ? V_Cnt - V_BLANK : 11'h0;
Active Image

2592x1944 Pixels

Figure 1.1 Pixel Array Description

Figure 1.2 Pixel Color Pattern Detail (Top Right Corner)
Image Enhancement

- Can be added for better detection performance
- Luminance Enhancement
  - linear lighting correction
  - nonlinear lighting processing
  - Proposed enhancement uses nonlinear transfer function based on a local approach in HSV color space
- Contrast Enhancement
  - Histogram Equalization
  - Gaussian convolution in HSV color space
Noise Removal

- Low-pass filters and median filters are used most often for noise suppression or smoothing, while high-pass filters are typically used for image enhancement.

Low-Pass Filter
  => Remove High-Frequency Noise

Median Filter
  => A nonlinear process
  - to reduce impulsive, or salt-and-pepper noise
  - to preserve edges in an image
CWB HLS Flow of Design

(1) Describe ANSI-C and modify for HW.
   [Steps]
   A. Add input and output variables.
   B. Modify descriptions which cannot be synthesized in HW.
      e.g. ‘recursive call’, ‘dynamic memory allocation’, 'system call’, etc.
   C. Add bit width declaration (although not necessary)
      The bit width declaration of inputs and outputs is recommended to
      achieve smaller area designs. Other internal variables are optimized
      automatically.

Behavioral Synthesis

(1) Specify clock frequency
(2) Specify scheduling mode
(3) Specify CWB libraries, FLIB and BLIB (Refer to the Section. 1-2-1)
(4) Synthesize the design

Verification

Verification in CWB
(1) Simulation based verification (behavioral level, cycle accurate level)
(2) Formal verification (Property checker, C-RTL equivalence prover)
module sobel (input_row_a00, input_row_a01, input_row_a02, output_row_a00, output_row_a01, output_row_a02, output_row, CLOCK, RESET);

#define C

#include "stdio.h"
#include "stdlib.h"

unsigned char input_row[SIZE_BUFFER];
unsigned char output_row;

/* Global variables */
unsigned char line_buffer[SIZE_BUFFER][SIZE_BUFFER];

#else

/* Entity declaration : Inputs and outputs with their bitwidths */
//defined for CWB HW-C
int in[0:8] input_row[SIZE_BUFFER]; //8-bit for each pixel-> 0-255
int out[0:8] output_row; //8-bit for each pixel-> 0-255
#endif

/* Global variables */
var[0:8] line_buffer[SIZE_BUFFER][SIZE_BUFFER];
#endif

#endif

void sobel()
{
#endif

#endif

//local variables declaration
unsigned int X, Y;
int sumX, sumY;
int SUM, rowOffset, colOffset;

char Gx[3][3] = { -1, 0, 1 },
               { -2, 0, 2 },
               { -1, 0, 1 },
               { 0, 0, 0 };

char Gy[3][3] = { -1, 0, 1 },
               { -2, 0, 2 },
               { -1, 0, 1 };

}
## Synthesis Report By Quartus II

<table>
<thead>
<tr>
<th>Flow Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Flow Status</strong></td>
</tr>
<tr>
<td><strong>Quartus II 32-bit Version</strong></td>
</tr>
<tr>
<td><strong>Revision Name</strong></td>
</tr>
<tr>
<td><strong>Top-level Entity Name</strong></td>
</tr>
<tr>
<td><strong>Family</strong></td>
</tr>
<tr>
<td><strong>Device</strong></td>
</tr>
<tr>
<td><strong>Timing Models</strong></td>
</tr>
<tr>
<td><strong>Total logic elements</strong></td>
</tr>
<tr>
<td><strong>Total combinational functions</strong></td>
</tr>
<tr>
<td><strong>Dedicated logic registers</strong></td>
</tr>
<tr>
<td><strong>Total registers</strong></td>
</tr>
<tr>
<td><strong>Total pins</strong></td>
</tr>
<tr>
<td><strong>Total virtual pins</strong></td>
</tr>
<tr>
<td><strong>Total memory bits</strong></td>
</tr>
<tr>
<td><strong>Embedded Multiplier 9-bit elements</strong></td>
</tr>
<tr>
<td><strong>Total PLLs</strong></td>
</tr>
</tbody>
</table>
### Altera IP or Megafunctons Used in The Project

<table>
<thead>
<tr>
<th>Entity</th>
<th>IP Component Name</th>
<th>Version</th>
<th>IP File</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line_Buffer1</td>
<td>Shift register (RAM-based)</td>
<td>9.1</td>
<td>v/Line_Buffer1.qip</td>
<td>Altera</td>
</tr>
<tr>
<td>sdram_pll</td>
<td>ALTPLL</td>
<td>9.1</td>
<td>v/sdram_pll.qip</td>
<td>Altera</td>
</tr>
<tr>
<td>LineBuffer_3</td>
<td>Shift register (RAM-based)</td>
<td>13.0</td>
<td>LineBuffer_3.qip</td>
<td>Altera</td>
</tr>
<tr>
<td>PA_3</td>
<td>PARALLEL_ADD</td>
<td>N/A</td>
<td>H:/Final Year Project/Kevin Codes/FPGA/DE2_115_CAMERA_Sobel/PA_3.v</td>
<td>Altera</td>
</tr>
<tr>
<td>SQRT</td>
<td>ALTSQRT</td>
<td>N/A</td>
<td>H:/Final Year Project/Kevin Codes/FPGA/DE2_115_CAMERA_Sobel/SQRT.v</td>
<td>Altera</td>
</tr>
<tr>
<td>MAC_3</td>
<td>ALTMULT_ACCUM (MAC)</td>
<td>13.0</td>
<td>MAC_3.qip</td>
<td>Altera</td>
</tr>
<tr>
<td>Sdram_RD_FIFO</td>
<td>FIFO</td>
<td>N/A</td>
<td>H:/Final Year Project/Kevin Codes/FPGA/DE2_115_CAMERA_Sobel/Sdram_Control/Sdram_RD_FIFO.v</td>
<td>Altera</td>
</tr>
<tr>
<td>Sdram_WR_FIFO</td>
<td>FIFO</td>
<td>N/A</td>
<td>H:/Final Year Project/Kevin Codes/FPGA/DE2_115_CAMERA_Sobel/Sdram_Control/Sdram_WR_FIFO.v</td>
<td>Altera</td>
</tr>
<tr>
<td>Line_Buffer3</td>
<td>Shift register (RAM-based)</td>
<td>13.0</td>
<td>Line_Buffer3.qip</td>
<td>N/A</td>
</tr>
<tr>
<td>LineBuffer3</td>
<td>Shift register (RAM-based)</td>
<td>13.0</td>
<td>LineBuffer3.qip</td>
<td>N/A</td>
</tr>
</tbody>
</table>
High-Level Synthesis
By NEC CyberWorkBench (CWB)

- From C to Verilog HDL
- Preliminary simulation & verification
Three steps to achieve breakthrough performance (From Xilinx)

1. Utilize the dedicated resources
   - Dedicated resources are faster than a LUT/Flip-Flop implementation and consume less power
   - Typically built with the CORE Generator tool and instantiated
   - DSP48E, FIFO, Block RAM, ISERDES, OSERDES, EMAC, and MGT, for example

2. Write the code for performance
   - Use synchronous design methodology
   - Ensure the code is written optimally for critical paths
   - Pipeline when necessary

3. Drive your synthesis tool
   - Try different optimization techniques
   - Add critical timing constraints in synthesis
   - Preserve hierarchy
   - Apply full and correct constraints
   - Use high effort
| Post_Processing | 18,606 / 114,480 (16%) | 18,520 / 114,480 (16%) | 6,865 / 114,480 (6%) | 6865 | 0 / 3,981,312 (0%) |