

CE2003 Laboratory 4

Lu Shengliang

SLU001

School of Computer Engineering

Nanyang Technological University

SLU001@e.ntu.edu.sg

November 2, 2013

Abstract

The aim of this lab is to build a project to process pixel input from a camera and then output on the screen after a series of process in real time. Several stages of pipelines were employed during this lab session to increase clock frequency, which results a better performance than non-pipelined.

1 Introduction

1.1 Design Modules

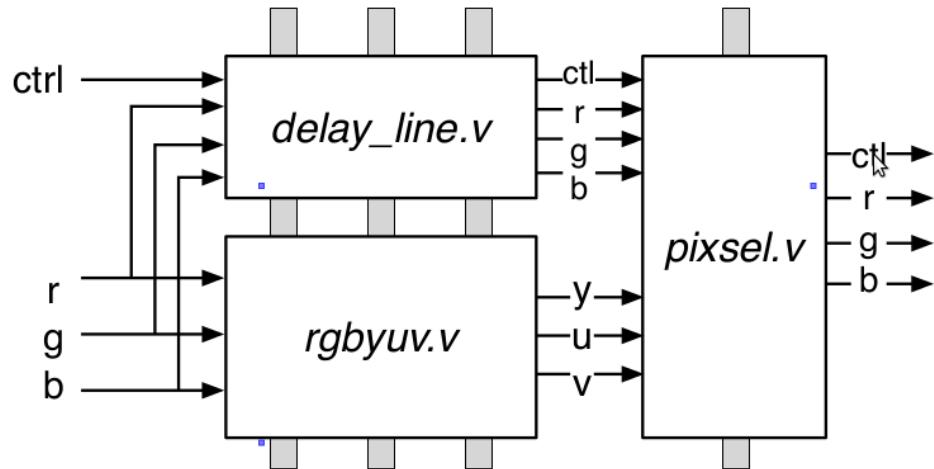


Figure 1: Design Diagram

There are 3 modules in this design. They are *delay_line*, *rgbyuv* and *pixel*. Basically, *rgbyuv* is used to convert RGB signals from a camera to YUV, which is introduced in lab manual; *Delay_line* module is used to align signal in order to synchronize related input and control signal; Module *pixsel* is used to select wanted signal from the previous two module according to control signal and output it to the top module then display on the screen.

1.2 Structure of the rest of the paper

The rest of the paper first describes the Verilog implementation of *Task 1: rgbyuv Module* in *Section 2*; *Task 2: delay_line Module* in *Section 3*; And *Section 4* presents the experimental results pixels selections and color effects. Lastly, *Section 5* presents our conclusions and discussions.

2 Task 1: rgbyuv Module

2.1 Verilog Code rgbyuv.v Non-pipelined

```
1 ry<= ( (RY_COEF * red_r + GY_COEF * grn_r + BY_COEF * blu_r) >>>8 ) +16;
2 ru<= ( (RU_COEF * red_r + GU_COEF * grn_r + BU_COEF * blu_r) >>>8 ) +128;
3 rv<= ( (RV_COEF * red_r + GV_COEF * grn_r + BV_COEF * blu_r) >>>8 ) +128;
```

Synthesize this top module and check the synthesis report. Under the synthesis report, there is a device utilization summary. It shows that, 9 DSP blocks are in utilization.

Number of bonded IOBs	80
Number of BUFG/BUFGCTRLs	1
Number of DSP48A1s	9

Figure 2: DSP Blocks

In the synthesis report, the maximum frequency is 77.873MHz.

```
Minimum period: 12.841ns (Maximum Frequency: 77.873MHz)
Minimum input arrival time before clock: 2.957ns
Maximum output required time after clock: 3.597ns
Maximum combinational path delay: No path found
```

Figure 3: Frequency 77.873MHz

2.2 Verilog Code rgbyuv.v with 2-stage pipelines

If we simply add one more stage, which will result in a 2-stage pipelined computation steps. The pipeline diagram is shown below.

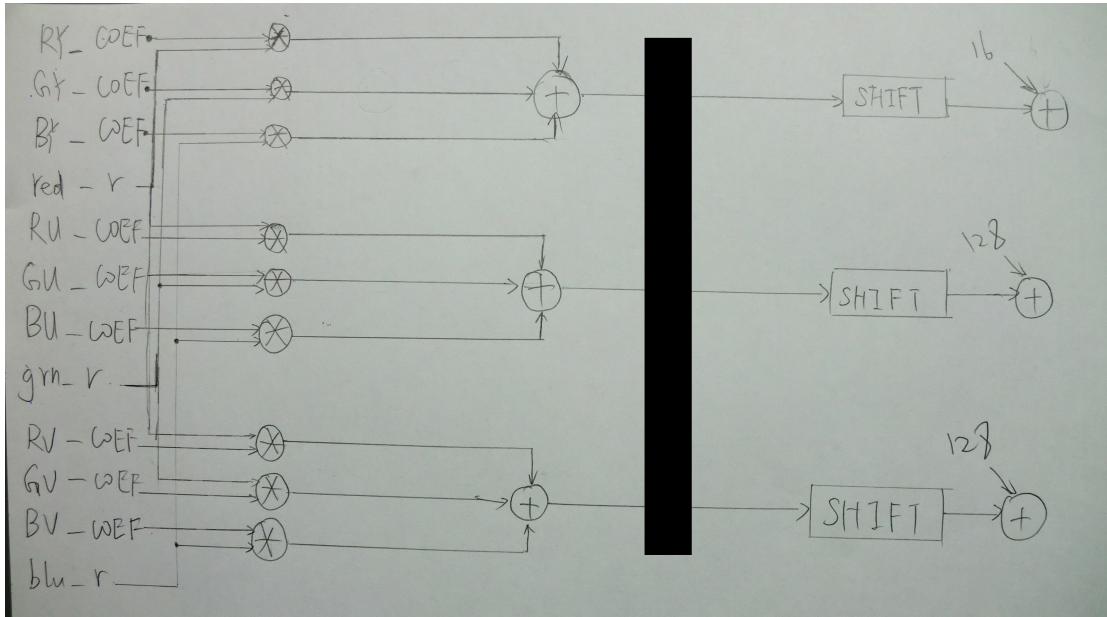


Figure 4: 2-stage pipeline

```

1 ry <= RY_COEF * red_r + GY_COEF * grn_r + BY_COEF * blu_r;
2 ru <= RU_COEF * red_r + GU_COEF * grn_r + BU_COEF * blu_r;
3 rv <= RV_COEF * red_r + GV_COEF * grn_r + BV_COEF * blu_r;
4 o_y <= (ry >>> 8) + 16;
5 o_u <= (ru >>> 8) + 128;
6 o_v <= (rv >>> 8) + 128;

```

In the synthesis report, the maximum frequency is 96.034MHz.

```

Minimum period: 10.413ns (Maximum Frequency: 96.034MHz)
Minimum input arrival time before clock: 3.428ns
Maximum output required time after clock: 3.597ns
Maximum combinational path delay: No path found

```

Figure 5: Frequency 96.034MHz

Compared to the previous two, the last implementation, which has 3 pipeline stages, is much better and manage to achieve 108MHz.

2.3 Verilog Code rgbyuv.v with 3-stage pipelines

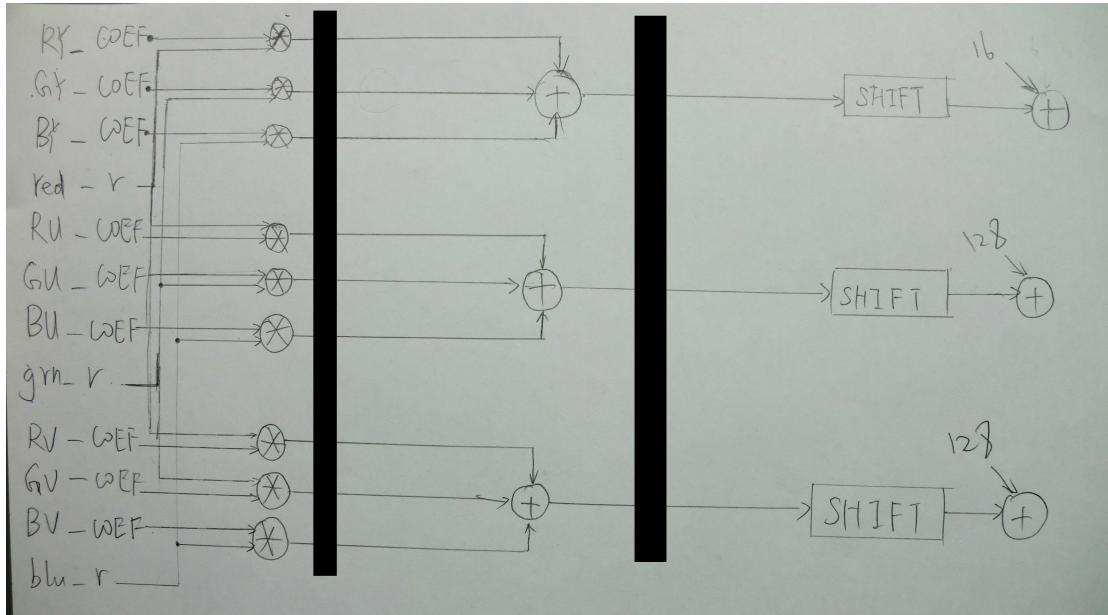


Figure 6: 3-stage pipeline

Because all the mutiplications at the first stage will be the critical path for the whole pipelines. So if they are separated like the code above, the critical path is split into 3 similar part, which results in a high frequency achievement.

```

1  ry1 <= RY_COEF * red_r;
2  ry2 <= GY_COEF * grn_r;
3  ry3 <= BY_COEF * blu_r;
4  ru1 <= RU_COEF * red_r;
5  ru2 <= GU_COEF * grn_r;
6  ru3 <= BU_COEF * blu_r;
7  rv1 <= RV_COEF * red_r;
8  rv2 <= GV_COEF * grn_r;
9  rv3 <= BV_COEF * blu_r;
10 ry <= ry1 + ry2 + ry3;
11 ru <= ru1 + ru2 + ru3;
12 rv <= rv1 + rv2 + rv3;
13 o_y <= (ry >>> 8) + 16;
14 o_u <= (ru >>> 8) + 128;
15 o_v <= (rv >>> 8) + 128;

```

This time, the frequency is 209.249MHz.

```

Minimum period: 4.779ns (Maximum Frequency: 209.249MHz)
Minimum input arrival time before clock: 4.432ns
Maximum output required time after clock: 3.597ns
Maximum combinational path delay: No path found

```

Figure 7: Frequency 209.249MHz

One last improvement, if we put all of the operations into pipelines, the maximum frequency will reach about 456MHz. However, too many pipeline stages will cause more complicated signal alignment later on.

```

Minimum period: 2.190ns (Maximum Frequency: 456.621MHz)
Minimum input arrival time before clock: 4.460ns
Maximum output required time after clock: 3.597ns
Maximum combinational path delay: No path found

```

Figure 8: Frequency 456.621MHz

2.4 skin pixel detection with a 4-stage pipeline

```

1      //add one more stage
2      p_y <= (ry >>> 8) + 16;
3      p_u <= (ru >>> 8) + 128;
4      p_v <= (rv >>> 8) + 128;
5      o_y <= p_y; o_u <= p_u; o_v <= p_v;
6      end
7  end // always @ (posedge clk)
8
9  always @(posedge clk) begin
10     if (rst) begin
11         skind <= 1'b0;
12     end else begin
13         if (73 <= p_u && p_u <= 122 && 132 <= p_v && p_v <= 173) begin
14             skind <= 1'b1;
15         end else begin
16             skind <= 1'b0;
17         end
18     end
19  end // always @ (posedge clk)

```

If a pixel is detected having a skin like color by using YUV analysis, a 1-bit register will be asserted, otherwise 0. At the end of final synthesis, the maximum frequency is still 209.249MHz.

3 Task 2: delay_line Module

3.1 Verilog Code delay_line.v

```
1  always @(posedge clk) begin          27
2    if(rst) begin                      28
3      in_r_p1 <= 8'd0;                  29
4      in_r_p2 <= 8'd0;                  30
5      in_r_p3 <= 8'd0;                  31
6      in_r_p4 <= 8'd0;                  32
7      out_r <= 8'd0;                  33
8
9      in_g_p1 <= 8'd0;                  35
10     in_g_p2 <= 8'd0;                 36
11     in_g_p3 <= 8'd0;                 37
12     in_g_p4 <= 8'd0;                 38
13     out_g <= 8'd0;                  39
14
15     in_b_p1 <= 8'd0;                  41
16     in_b_p2 <= 8'd0;                  42
17     in_b_p3 <= 8'd0;                  43
18     in_b_p4 <= 8'd0;                  44
19     out_b <= 8'd0;                  45
20
21     in_c_p1 <= 8'd0;                  47
22     in_c_p2 <= 8'd0;                  48
23     in_c_p3 <= 8'd0;                  49
24     in_c_p4 <= 8'd0;                  50
25     out_c <= 8'd0;                  51
26   end else begin                  52
27
28   in_r_p1 <= in_r;                  27
29   in_r_p2 <= in_r_p1;                28
30   in_r_p3 <= in_r_p2;                29
31   in_r_p4 <= in_r_p3;                30
32   out_r <= in_r_p3;                31
33
34   in_g_p1 <= in_g;                  34
35   in_g_p2 <= in_g_p1;                35
36   in_g_p3 <= in_g_p2;                36
37   in_g_p4 <= in_g_p3;                37
38   out_g <= in_g_p4;                38
39
40   in_b_p1 <= in_b;                  40
41   in_b_p2 <= in_b_p1;                41
42   in_b_p3 <= in_b_p2;                42
43   in_b_p4 <= in_b_p3;                43
44   out_b <= in_b_p4;                44
45
46   in_c_p1 <= in_c;                  46
47   in_c_p2 <= in_c_p1;                47
48   in_c_p3 <= in_c_p2;                48
49   in_c_p4 <= in_c_p3;                49
50   out_c <= in_c_p4;                50
51
52   end // else: !if(rst)
53   end // always @ (posedge clk)
54 endmodule // delay_line
```

As we can see from the module name, *delayline* is used to delay the input signal by using pipelines. Because we are currently using a 5-stage pipelined datapath in *rgbyuv.v*, *delayline* are supposed to have the same stage delay as it.

4 Task 3 & 4: Logically Display

Task 3 and task 4 are combined together, because task 4 is roughly improvement of task 3 with more pixel operations.

4.1 Verilog Code

```
1  always @(posedge clk) begin
2    if(rst) begin
3      out_r <= 8'd0;
4      out_g <= 8'd0;
```

```

5   out_b <= 8'd0;
6   out_ctrl <= 3'd0;
7   end else begin
8     case(in_swt)
9       1 : begin
10      out_r <= in_y; out_g <= in_y; out_b <= in_y;
11    end
12    2 : begin
13      out_r <= in_u; out_g <= in_u; out_b <= in_u;
14    end
15    4 : begin
16      out_r <= in_v; out_g <= in_v; out_b <= in_v;
17    end
18   /*8 : begin
19     if (in_skin) begin
20       out_r <= in_y; out_g <= in_y; out_b <= in_y;
21     end else begin
22       out_r <= 1'b0; out_g <= 1'b0; out_b <= 1'b0;
23     end
24   end*/
25   //display with maximum green
26   8 : begin
27     if (in_skin) begin
28       out_r <= min_r; out_g <= max_g; out_b <= min_b;
29     end else begin
30       out_r <= in_r; out_g <= in_g; out_b <= in_b;
31     end
32   end
33   //display with maximum red
34   16 : begin
35     if (in_skin) begin
36       out_r <= max_r; out_g <= min_g; out_b <= min_b;
37     end else begin
38       out_r <= in_r; out_g <= in_g; out_b <= in_b;
39     end
40   end
41   //display with maximum blue
42   32 : begin
43     if (in_skin) begin
44       out_r <= min_r; out_g <= min_g; out_b <= max_b;
45     end else begin
46       out_r <= in_r; out_g <= in_g; out_b <= in_b;
47     end
48   end
49   //display with maximum red, green and blue
50   64 : begin
51     if (in_skin) begin
52       out_r <= max_r; out_g <= max_g; out_b <= max_b;
53     end else begin
54       out_r <= in_r; out_g <= in_g; out_b <= in_b;
55     end
56   end
57   //display original skin color. otherwise max r & g & b
58   128 : begin
59     if (in_skin) begin
60       out_r <= in_r; out_g <= in_g; out_b <= in_b;
61     end else begin
62       out_r <= max_r; out_g <= max_g; out_b <= max_b;
63     end
64   end
65   //for fun
66   255 : begin
67     if (in_skin) begin
68       out_r <= in_r; out_g <= in_g; out_b <= in_b;
69     end else begin
70       out_r <= min_r; out_g <= min_g; out_b <= min_b;

```

```

71         end
72     end
73     default : begin
74         out_r <= in_r; out_g <= in_g; out_b <= in_b;
75     end
76     endcase // case (in_swt)
77     out_ctrl <= in_c;
78   end // else: !if(rst)
79 end // always @ (posedge clk)

```

A one-hot stratage was employed to control the pixel selections. More detailed results will be given. *Case 1, 2, 4*:

The screen turned black and white in case 1, because there is no color output from *pixel.v*, except *y*, which is the luminance or brightness of a pixel; The screen turned inversed darkness and grey in switch 2 and turning palely grey in switch 4.

Case 8, 16, 32:

In task 3, fpga is supposed to show those pixels detected as skin ones. The others will be blocked, which results that only skin parts will be shown on the screen.

But the result will not be very good: there will be many incorrect outputs, such like fluorescent lamp, white wall, since we are using a basic skin color detection algorithm. In task 4, instead of blocking other pixels and displaying skin ones, fpga is supposed to change skin color by replace original pixels' data with different color and output the other pixel like their original colors.

```

1  wire [8:0]      max_r = ((in_r + 64)<255) ? in_r + 64 : 255;
2  wire [8:0]      min_r = ((in_r - 32)>0)   ? in_r - 32 : 0;
3  wire [8:0]      max_g = ((in_g + 64)<255) ? in_g + 64 : 255;
4  wire [8:0]      min_g = ((in_g - 32)>0)   ? in_g - 32 : 0;
5  wire [8:0]      max_b = ((in_b + 64)<255) ? in_b + 64 : 255;
6  wire [8:0]      min_b = ((in_b - 32)>0)   ? in_b - 32 : 0;

```

```

1 out_r <= min_r; out_g <= max_g; out_b <= min_b;

```

If *max_g* was used to replace *out_g*, the skin pixels will be replaced by green pixels.

```

1 out_r <= max_r; out_g <= min_g; out_b <= min_b;

```

If *max_r* was used to replace *out_r*, the skin pixels will be replaced by red pixels.

```

1 out_r <= min_r; out_g <= min_g; out_b <= max_b;

```

If *max_b* was used to replace *out_b*, the skin pixels will be replaced by blue pixels.

Case 64, 128:

```
1 if (in_skin)
2     out_r <= max_r; out_g <= max_g; out_b <= max_b;
3 else
4     out_r <= in_r; out_g <= in_g; out_b <= in_b;
```

If all out_ were replaced by max_, the skin pixels will become pale instead of original flesh or pink color.

```
1 if (in_skin)
2     out_r <= in_r; out_g <= in_g; out_b <= in_b;
3 else
4     out_r <= max_r; out_g <= max_g; out_b <= max_b;
```

Doing fun, after exchange these two line unblocking assignment, the skin pixels went out directly, and other pixels become greyer than before. But there is a side effect: some pixels on skin may not be detected as a skin pixel, which results that the whole skin looks darker than before.

Case 255:

```
1 if (in_skin)
2     out_r <= in_r; out_g <= in_g; out_b <= in_b;
3 else
4     out_r <= min_r; out_g <= min_g; out_b <= min_b;
```

This time, all the black color shined, and skin color output directly. It because we set that, if a pixel is not a skin pixel, then it will be replaced by min_r, min_g and min_b.

5 Conclusions and Future Work

In this lab session, pipelining and signal aligning techniques were employed to process input signals from a camera. In addition, selection and logical color conversion were applied to produce several special visual effects.

5.1 *Problems occurred in our approach*

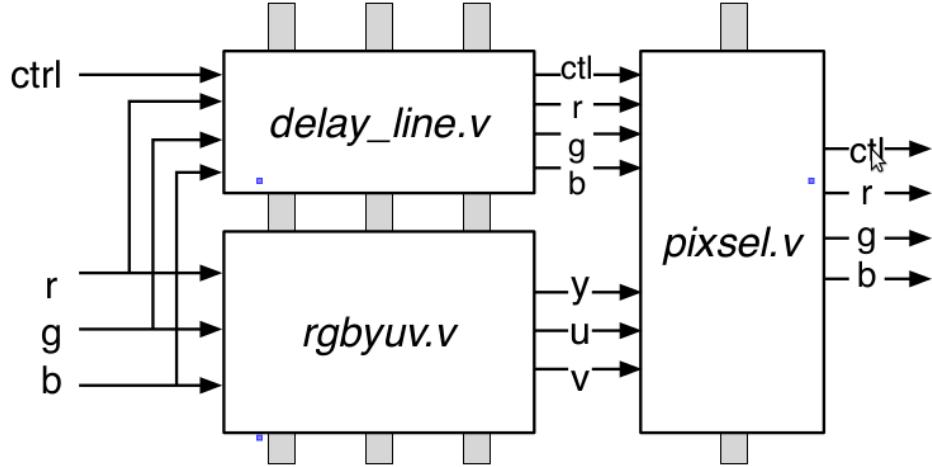


Figure 9: Design Diagram

From the figure above, we can see that the output signals (ctl, r, g, b) of *delay_line* module need to be synchronous with the output signals (y, u, v) of *rgbyuv* module. Since a 4-stage pipeline was employed in *rgbyuv* module, the *delay_line* module is also required to have a 4-stage pipeline.

If the number of stages in *delay_line* module is different from *rgbyuv*, fpges would not generate video output to screen.

Because all the multiplications at the first stage in *rgbyuv.v* will be the critical path for the whole pipelines. So if the most slow part in a pipeline is separated like the code above, it will result in a significant improvement on maximum frequency.

5.2 *Available Improvement*

10 kinds of color effect were conducted based on skin pixel detection and YUV colorspace. Actually, more achievements are Available. For example, one way is that we can change the skin color detect block to other color detect. Another way is that instead of using one-hot strategy, which will only have 8 features, we can actually add more features by changing the output with different color combinations like what was done in this report.

6 Appendix

Please ignore the time stamp on the right corner of each picture. There was something wrong with my camera.

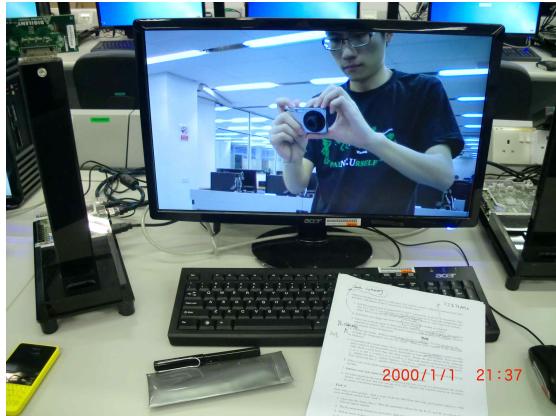


Figure 10: Case default



Figure 12: Case 2

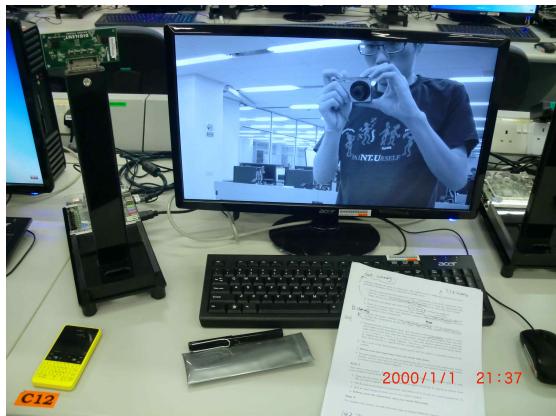


Figure 11: Case 1

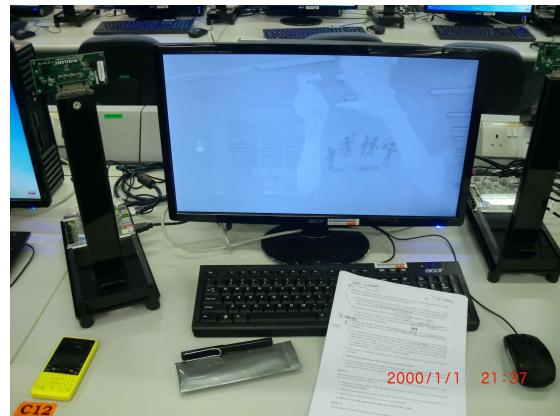


Figure 13: Case 4

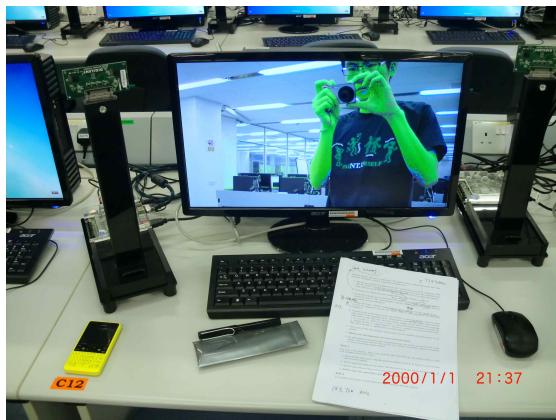


Figure 14: Case 8



Figure 17: Case 64



Figure 15: Case 16



Figure 18: Case 128



Figure 16: Case 32



Figure 19: Case 255