VIETNAM NATIONAL UNIVERSITY OF HO CHI MINH CITY
UNIVERSITY OF INFORMATION TECHNOLOGY
FACULTY OF COMPUTER ENGINEERING

FINAL PROJECT REPORT

IMAGE PROCESSING : FILTER FOR IMAGE

Lecturer: Trần Thị Điểm

NGUYỄN TUẤN DŨNG - 21520746

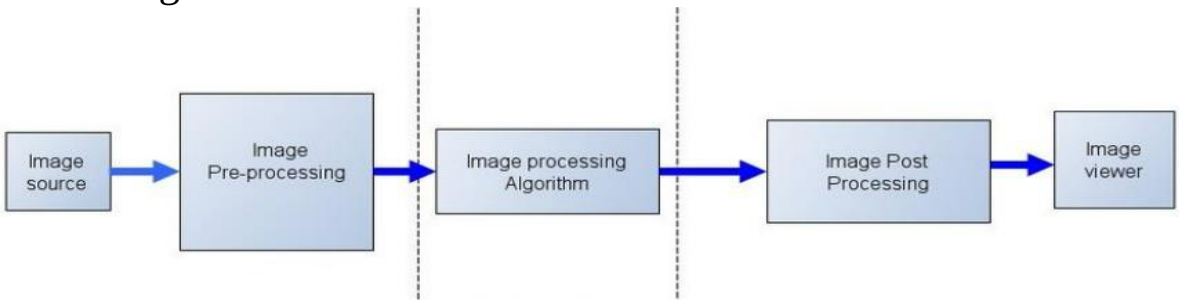
Table of Contents

CHAPTER 1: GENERAL INTRODUCTION

1.1 What is image embossing?

Image embossing is a computer graphics technique in which each pixel of an image is replaced either by a highlight or a shadow. Low contrast areas are replaced by a high color. The filtered image will represent the rate of color change at each location of the original image.

1.2 Design flow:



1.3 Technical detail:

To apply the embossing effect, a convolution operation is performed between the image and an embossing mask or kernel.

• Mask: The embossing mask is a matrix or a small filter that specifies how the pixels in the image should be adjusted to achieve the desired effect.

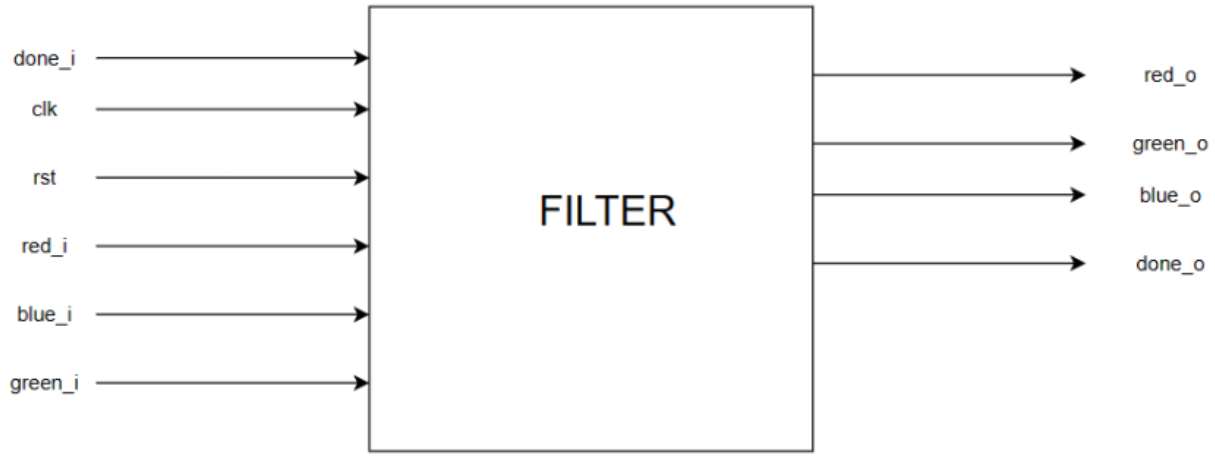
The emboss filter, also called a directional difference filter, will enhance edges in the direction of the selected convolution masks. When the emboss filter is applied, the filter matrix is in convolution calculation with the same square area on the original image.

A few basic masks:

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & -1 \\ 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

CHAPTER 2: THE STEPS TO IMPLEMENTING THE SYSTEM

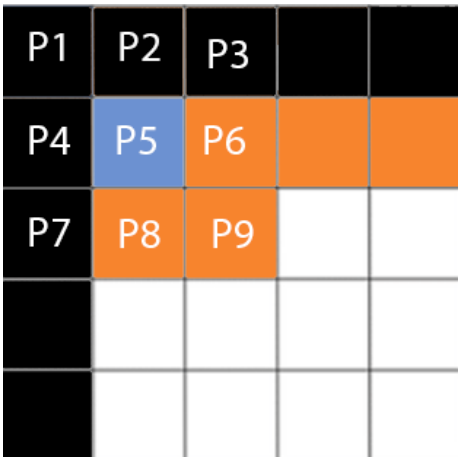
2.1 Design modules:



STT	Signals	Width	I/O	Describe
1	clk	1 bit	input	The clk signal is used to synchronize operations
2	rst	1 bit	input	Reset the module to its initial state
3	done_i	1 bit	input	Start creating the data area to calculate matrix
4	red_i	8 bit	input	The value of 1 pixel is from 0 to 255
5	blue_i	8 bit	input	The value of 1 pixel is from 0 to 255
6	green_i	8 bit	input	The value of 1 pixel is from 0 to 255
7	red_o	8 bit	output	The value of 1 pixel is from 0 to 255 after being calculated

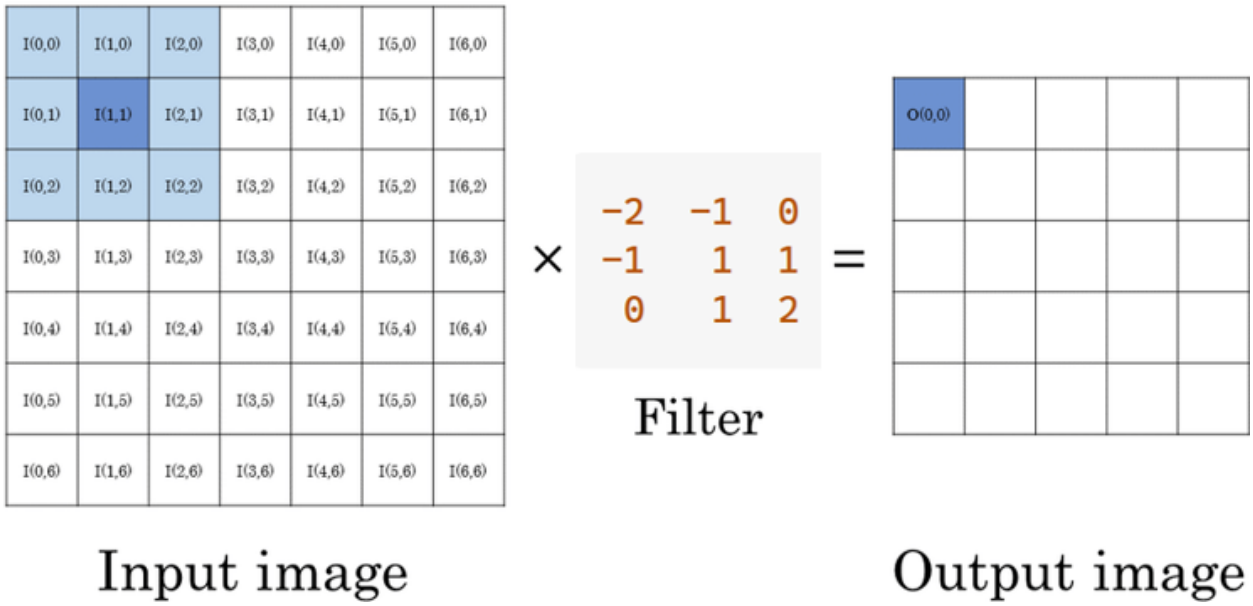
8	green_o	8 bit	output	The value of 1 pixel is from 0 to 255 after being calculated
9	blue_o	8 bit	output	The value of 1 pixel is from 0 to 255 after being calculated

2.2 Trigger condition for calculation:



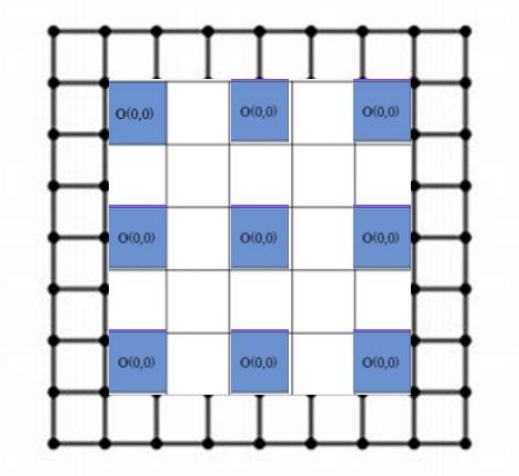
ROW2BUFFER: Use 2 fifo and counter to load data into the first row and 2 pixels of the second row and black cells default to the value 0.

2.3 Demonstration of convolution between the original image and kernel matrix:



2.4 Working with pixels:

Checking pixels in every position:

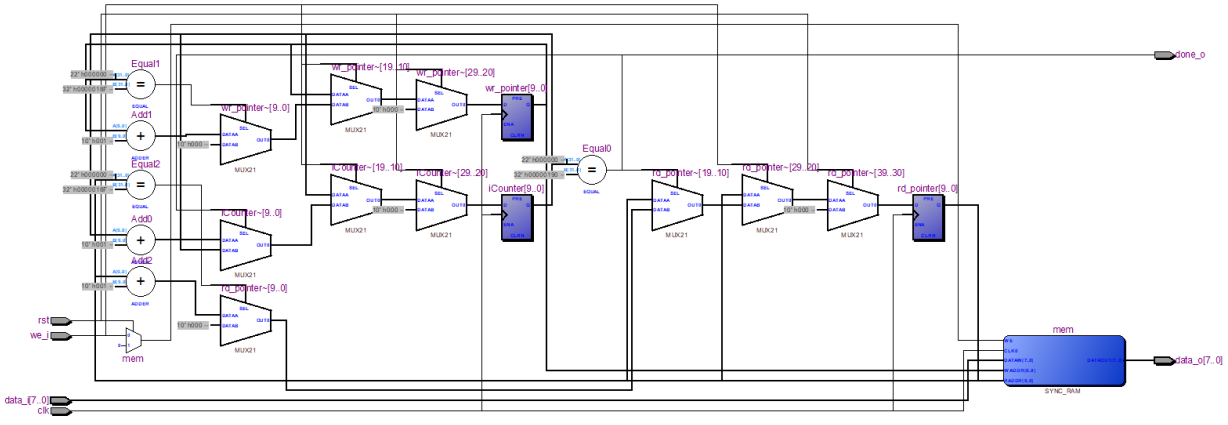


To emboss an image, we need different kernel matrices for different pixel of the image since the corner pixels and edge pixels have less neighboring pixels than those inside.

In this project, we make the embossing filter will enhance edges in diagonal direction from left to right and downward. There are a total of 9 cases where the matrix will be calculated differently, and those 9-case range squares will default to the value 0

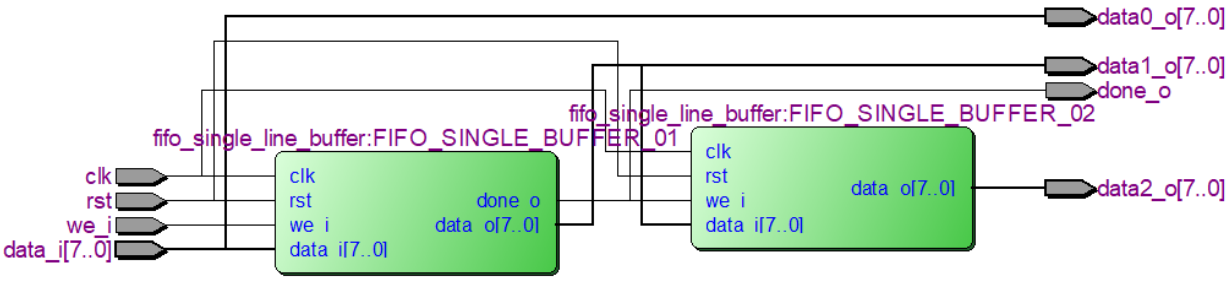
2.5 System overview:

FIFO:



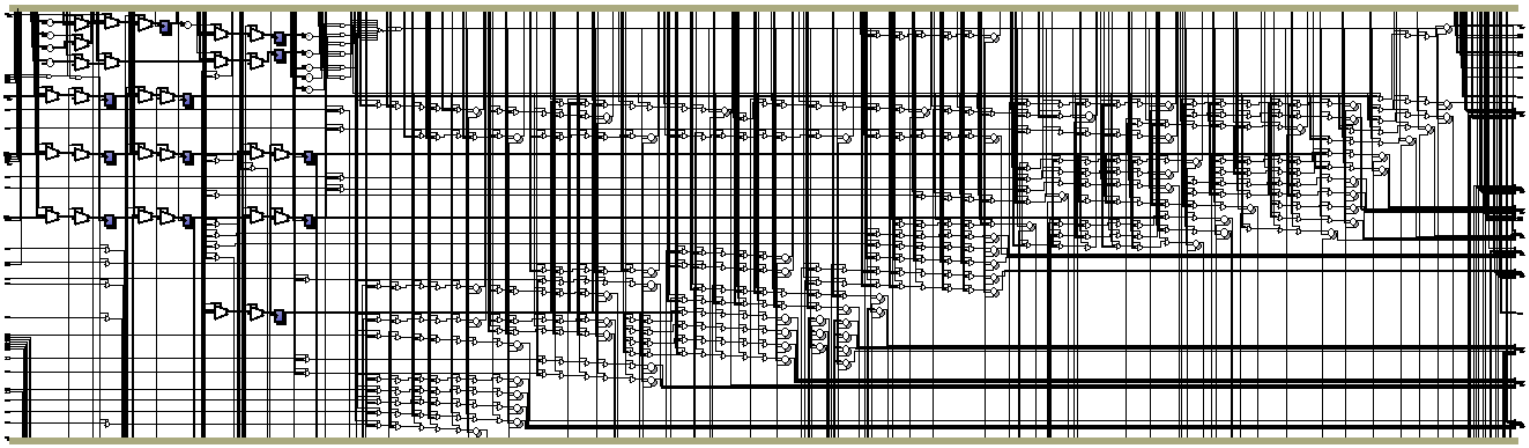
```
1  module fifo_double_line_buffer(
2      input clk,
3      input rst,
4
5      input we_i,
6      input [7:0] data_i,
7
8      output [7:0] data0_o,
9      output [7:0] data1_o,
10     output [7:0] data2_o,
11
12     output done_o
13 );
14
15  wire [7:0] fifo1_data,fifo2_data;
16  wire fifo1_done, fifo_done;
17
18  assign data0_o = data_i;
19  assign data1_o = fifo1_data;
20  assign data2_o = fifo2_data;
21
22  assign done_o = fifo1_done;
23
24 fifo_single_line_buffer  FIFO_SINGLE_BUFFER_01(
25     .clk(clk),
26     .rst(rst),
27
28     .we_i(we_i),
29     .data_i(data_i),
30
31     .data_o(fifo1_data),
32     .done_o(fifo1_done)
33 );
34
35 fifo_single_line_buffer  FIFO_SINGLE_BUFFER_02(
36     .clk(clk),
37     .rst(rst),
38
39     .we_i(fifo1_done),
40     .data_i(fifo1_data),
41
42     .data_o(fifo2_data),
43     .done_o(fifo2_done)
44 );
45
46
47  endmodule
```

FIFO_DOUBLE_LINE_BUFFER:



```v
module fifo_double_line_buffer(
    input clk,
    input rst,

    input we_i,
    input [7:0] data_i,

    output [7:0] data0_o,
    output [7:0] data1_o,
    output [7:0] data2_o,

    output done_o
);

 wire [7:0] fifo1_data,fifo2_data;
 wire fifo1_done, fifo_done;

 assign data0_o = data_i;
 assign data1_o = fifo1_data;
 assign data2_o = fifo2_data;

 assign done_o = fifo1_done;

fifo_single_line_buffer  FIFO_SINGLE_BUFFER_01(
    .clk(clk),
    .rst(rst),

    .we_i(we_i),
    .data_i(data_i),

    .data_o(fifo1_data),
    .done_o(fifo1_done)
);

fifo_single_line_buffer  FIFO_SINGLE_BUFFER_02(
    .clk(clk),
    .rst(rst),

    .we_i(fifo1_done),
    .data_i(fifo1_data),

    .data_o(fifo2_data),
    .done_o(fifo2_done)
);


 endmodule
```

Data_modulate :



```
module data_modulate(
    input clk,
    input rst,

    input [7:0] d0_i,
    input [7:0] d1_i,
    input [7:0] d2_i,

    input done_i,

    output reg [7:0] d0_o,
    output reg [7:0] d1_o,
    output reg [7:0] d2_o,
    output reg [7:0] d3_o,
    output reg [7:0] d4_o,
    output reg [7:0] d5_o,
    output reg [7:0] d6_o,
    output reg [7:0] d7_o,
    output reg [7:0] d8_o,

    output done_o
);
localparam ROWS = 400;
localparam COLS = 400;
reg [9:0] iRows,iCols;
reg [7:0] iCounter;
reg [7:0] data0,data1,data2,data3,data4,data5,data6,data7,data8;

assign done_o = (iCounter == 2) ? 1:0;

//ROW AND COL
always@(posedge clk) begin
```

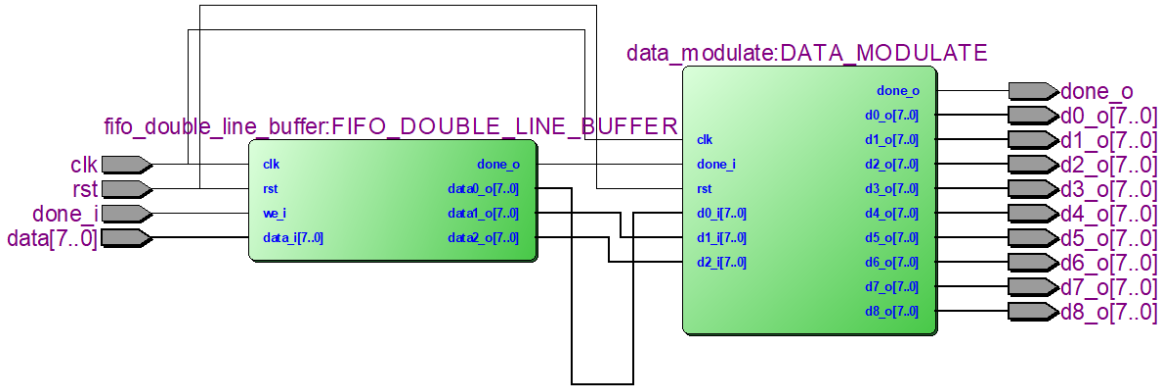
```
33      if(rst) begin
34         iRows <=0;
35         iCols <= 0;
36      end else begin
37         if(done_o == 1) begin
38            iCols <= (iCols == COLS - 1) ? 0 : iCols + 1;
39            if(iCols == COLS -1)
40               iRows <= (iRows == ROWS - 1) ? 0 : iRows + 1;
41         end
42      end
43   end
44
45   always@(*) begin
46      if(rst) begin
47         d0_o <= 0;
48         d1_o <= 0;
49         d2_o <= 0;
50         d3_o <= 0;
51         d4_o <= 0;
52         d5_o <= 0;
53         d6_o <= 0;
54         d7_o <= 0;
55         d8_o <= 0;
56      end else begin
57      //pos 1
58      if(done_o == 1) begin
59      if (iRows == 0 && iCols ==0) begin
60         d0_o <= 0;
61         d1_o <= 0;
62         d2_o <= 0;
```

```
63         d3_o <= 0;
64         d4_o <= data4;
65         d5_o <= data5;
66         d6_o <= 0;
67         d7_o <= data7;
68         d8_o <= data8;
69      end
70      //pos 2
71      else if(iRows == 0 && iCols > 0 && iCols < COLS - 1) begin
72         d0_o <= 0;
73         d1_o <= 0;
74         d2_o <= 0;
75         d3_o <= data3;
76         d4_o <= data4;
77         d5_o <= data5;
78         d6_o <= data6;
79         d7_o <= data7;
80         d8_o <= data8;
81      end
82      //pos 3
83      else if (iRows ==0 && iCols == COLS - 1) begin
84         d0_o <= 0;
85         d1_o <= 0;
86         d2_o <= 0;
87         d3_o <= data3;
88         d4_o <= data4;
89         d5_o <= 0;
90         d6_o <= data6;
91         d7_o <= data7;
92         d8_o <= 0;
```

```
94      //pos 4
95      else if (iRows > 0 && iRows < ROWS - 1 && iCols == 0) begin
96         d0_o <= 0;
97         d1_o <= data1;
98         d2_o <= data2;
99         d3_o <= 0;
100        d4_o <= data4;
101        d5_o <= data5;
102        d6_o <= 0;
103        d7_o <= data7;
104        d8_o <= data8;
105     end
106     //pos 5
107     else if (iRows > 0 && iRows < ROWS -1 && iCols > 0 && iCols < COLS -1 ) begin
108        d0_o <= data0;
109        d1_o <= data1;
110        d2_o <= data2;
111        d3_o <= data3;
112        d4_o <= data4;
113        d5_o <= data5;
114        d6_o <= data6;
115        d7_o <= data7;
116        d8_o <= data8;
117     end
118     //pos 6
119     else if (iRows > 0 && iRows < ROWS - 1 && iCols == COLS - 1) begin
120        d0_o <= data0;
121        d1_o <= data1;
122        d2_o <= 0;
123        d3_o <= data3;
124        d4_o <= data4;
125        d5_o <= 0;
126        d6_o <= data6;
127        d7_o <= data7;
128        d8_o <= 0;
129     end
130     //pos 7
131     else if ( iRows == ROWS - 1 && iCols == 0) begin
132        d0_o <= 0;
133        d1_o <= data1;
134        d2_o <= data2;
135        d3_o <= 0;
136        d4_o <= data4;
137        d5_o <= data5;
138        d6_o <= 0;
139        d7_o <= 0;
140        d8_o <= 0;
141     end
142     //pos 8
143     else if (iRows == ROWS -1 && iCols > 0 && iCols < COLS -1) begin
144        d0_o <= data0;
145        d1_o <= data1;
146        d2_o <= data2;
147        d3_o <= data3;
148        d4_o <= data4;
149        d5_o <= data5;
150        d6_o <= 0;
151        d7_o <= 0;
152        d8_o <= 0;
```

```
153     end
154     //pos 9
155     else if (iRows == ROWS - 1 && iCols == COLS - 1) begin
156        d0_o <= data0;
157        d1_o <= data1;
158        d2_o <= 0;
159        d3_o <= data3;
160        d4_o <= data4;
161        d5_o <= 0;
162        d6_o <= 0;
163        d7_o <= 0;
164        d8_o <= 0;
165     end
166     end
167     end
168  end
169
170
171  always@(posedge clk) begin
172     if(rst) begin
173        iCounter <= 0;
174     end else begin
175        if(done_i ==1) begin
176           iCounter <= (iCounter == 2) ? iCounter : iCounter +1 ;
177        end
178     end
179  end
180
181  always@( posedge clk ) begin
182     if(rst) begin
183        data0 <= 0;
184        data1 <= 0;
185        data2 <= 0;
186        data3 <= 0;
187        data4 <= 0;
188        data5 <= 0;
189        data6 <= 0;
190        data7 <= 0;
191        data8 <= 0;
192     end else begin
193        if(done_i) begin
194           data0 <= data1;
195           data1 <= data2;
196           data2 <= d2_i;
197
198           data3 <= data4;
199           data4 <= data5;
200           data5 <= d1_i;
201
202           data6 <= data7;
203           data7 <= data8;
204           data8 <= d0_i;
205        end
206     end
207  end
208
209  endmodule
```

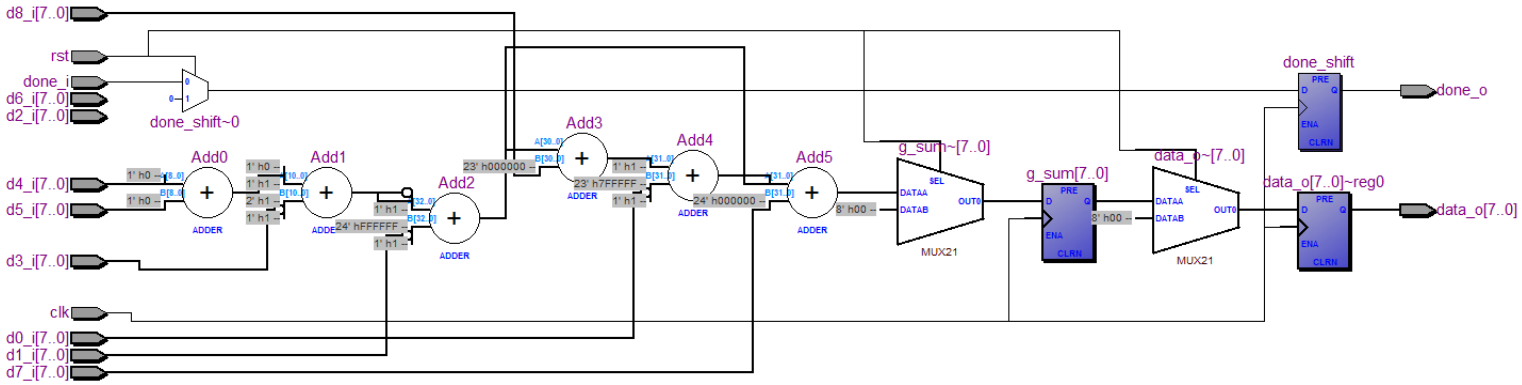
DATA_BUFFER:




```
module data_buffer(
    input clk,
    input rst,
    input [7:0] data,
    input done_i,

    output [7:0] d0_o,
    output [7:0] d1_o,
    output [7:0] d2_o,
    output [7:0] d3_o,
    output [7:0] d4_o,
    output [7:0] d5_o,
    output [7:0] d6_o,
    output [7:0] d7_o,
    output [7:0] d8_o,

    output done_o

);
 wire [7:0] double_line_fifo_data0;
 wire [7:0] double_line_fifo_data1;
 wire [7:0] double_line_fifo_data2;
 wire double_line_fifo_done;


fifo_double_line_buffer FIFO_DOUBLE_LINE_BUFFER(
    .clk(clk),
    .rst(rst),

    .we_i(done_i),
    .data_i(data),

```

```
33      .data0_o(double_line_fifo_data0),
34      .data1_o(double_line_fifo_data1),
35      .data2_o(double_line_fifo_data2),
36
37      .done_o(double_line_fifo_done)
38  );
39
40  data_modulate DATA_MODULATE(
41      .clk(clk),
42      .rst(rst),
43
44      .d0_i(double_line_fifo_data0),
45      .d1_i(double_line_fifo_data1),
46      .d2_i(double_line_fifo_data2),
47
48      .done_i(double_line_fifo_done),
49
50      .d0_o(d0_o),
51      .d1_o(d1_o),
52      .d2_o(d2_o),
53      .d3_o(d3_o),
54      .d4_o(d4_o),
55      .d5_o(d5_o),
56      .d6_o(d6_o),
57      .d7_o(d7_o),
58      .d8_o(d8_o),
59
60      .done_o(done_o)
61  );
62  endmodule
```

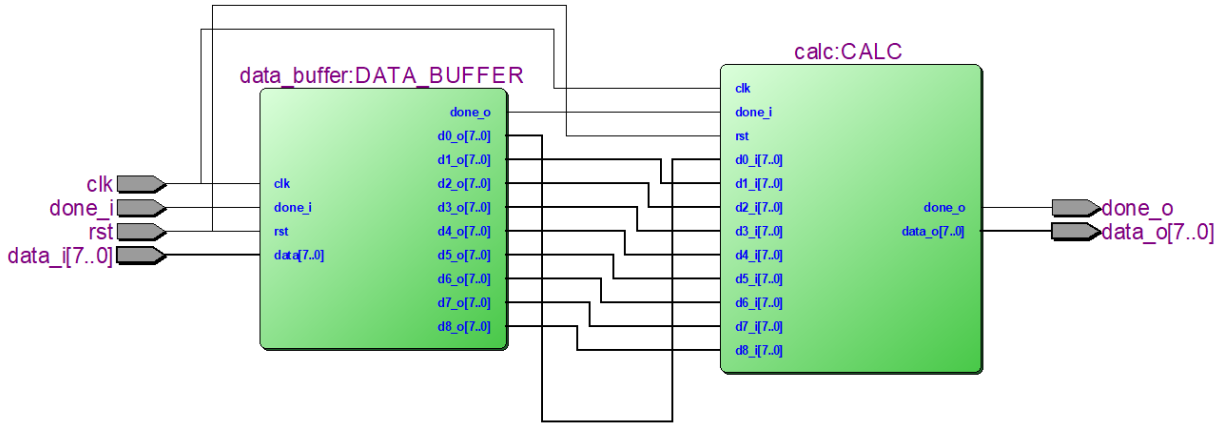
Calculation:



```
1  module calc(
2      input clk,
3      input rst,
4      input [7:0] d0_i,
5      input [7:0] d1_i,
6      input [7:0] d2_i,
7      input [7:0] d3_i,
8      input [7:0] d4_i,
9      input [7:0] d5_i,
10     input [7:0] d6_i,
11     input [7:0] d7_i,
12     input [7:0] d8_i,
13     input done_i,
14
15     output reg [7:0] data_o,
16     output done_o
17
18 );
19  reg [9:0] g_sum;
20  reg done_shift;
21
22 always@(posedge clk) begin
23     if(rst) begin
24     g_sum <=0;
25     end
26     else begin
27        g_sum <= d4_i + d5_i -d3_i -d1_i + 2*d8_i - 2*d0_i + d7_i;
28     end
29 end
30  //
31
```

```
32 always@(posedge clk) begin
33     if(rst) begin
34        data_o <= 0;
35     end
36     else begin
37        data_o <=  g_sum[7:0];
38     end
39 end
40
41 always@(posedge clk) begin
42     if(rst) begin
43        done_shift <= 0;
44     end
45     else begin
46        done_shift <= done_i;
47     end
48 end
49
50  assign done_o = done_shift;
51
52  endmodule
```

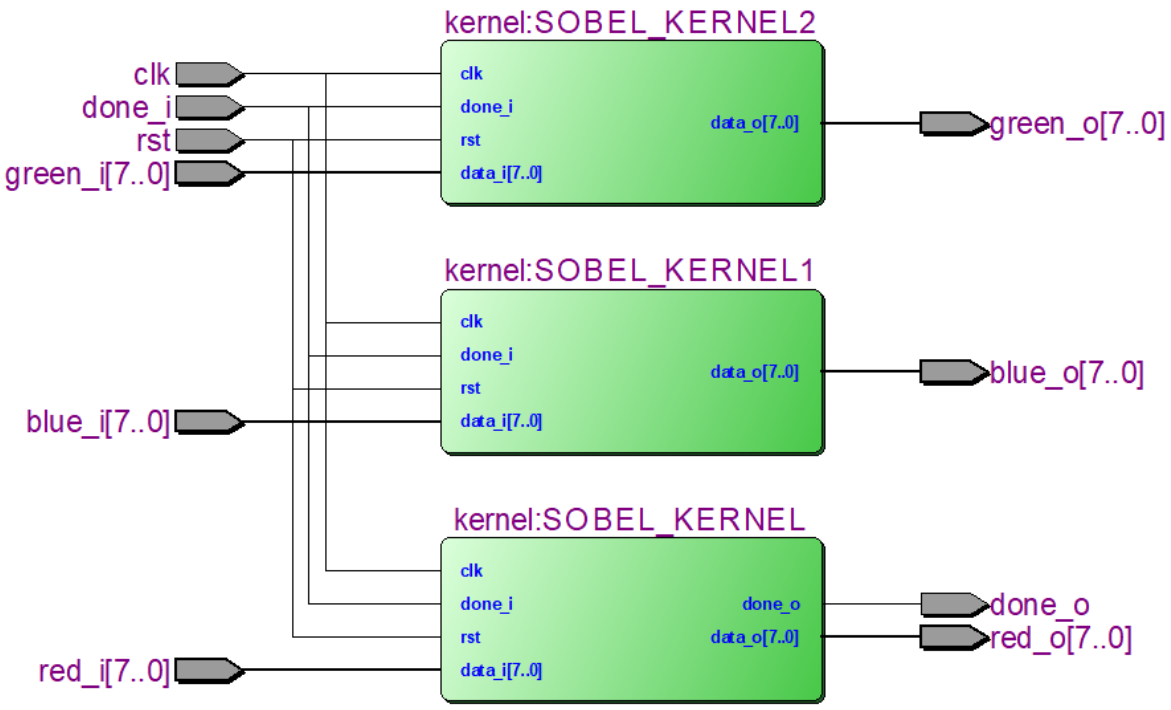
Kernel:




```
1  module kernel(
2      input clk,
3      input rst,
4
5      input [7:0] data_i,
6      input done_i,
7
8      output [7:0] data_o,
9      output done_o
10 );
11
12 wire [7:0] data[0:8];
13 wire data_buffer_done;
14
15 data_buffer DATA_BUFFER(
16     .clk(clk),
17     .rst(rst),
18     .data(data_i),
19     .done_i(done_i),
20
21     .d0_o(data[0]),
22     .d1_o(data[1]),
23     .d2_o(data[2]),
24     .d3_o(data[3]),
25     .d4_o(data[4]),
26     .d5_o(data[5]),
27     .d6_o(data[6]),
28     .d7_o(data[7]),
29     .d8_o(data[8]),
30
31     .done_o(data_buffer_done)
32
```

```
33  );
34
35  calc CALC(
36      .clk(clk),
37      .rst(rst),
38      .d0_i(data[0]),
39      .d1_i(data[1]),
40      .d2_i(data[2]),
41      .d3_i(data[3]),
42      .d4_i(data[4]),
43      .d5_i(data[5]),
44      .d6_i(data[6]),
45      .d7_i(data[7]),
46      .d8_i(data[8]),
47      .done_i(data_buffer_done),
48
49      .data_o(data_o),
50      .done_o(done_o)
51  );
52
53  endmodule
```

Filter:



```
module filter(
    input clk,
    input rst,

    input [7:0] red_i,
    input [7:0] green_i,
    input [7:0] blue_i,
    input done_i,

    output reg [7:0] red_o,
    output reg [7:0] green_o,
    output reg [7:0] blue_o,

    output done_o
);
 wire [7:0] red,blue,green;
 wire done_o1,done_o2;
kernel SOBEL_KERNEL(
    .clk(clk),
    .rst(rst),

    .data_i(red_i),
    .done_i(done_i),

    .data_o(red),
    .done_o(done_o)
);
 kernel SOBEL_KERNEL1(
    .clk(clk),
    .rst(rst),

    .data_i(blue_i),
    .done_i(done_i),

    .data_o(blue),
    .done_o(done_o1)
);
kernel SOBEL_KERNEL2(
    .clk(clk),
    .rst(rst),

    .data_i(green_i),
    .done_i(done_i),

    .data_o(green),
    .done_o(done_o2)
);
always@(*)begin
      red_o <= red;
      green_o <= green ;
      blue_o <= blue;
 end


 endmodule
```

TESTBENCH:

```
1  `define clk_period 10
2  module filter_tb();
3
4  reg clk,rst;
5  reg [7:0] red_i,green_i,blue_i;
6  wire [7:0] red_o,green_o,blue_o;
7  wire done_o;
8  reg done_i;
9
10 filter FILTER(
11    .clk(clk),
12    .rst(rst),
13
14    .red_i(red_i),
15    .green_i(green_i),
16    .blue_i(blue_i),
17    .done_i(done_i),
18
19
20    .red_o(red_o),
21    .green_o(green_o),
22    .blue_o(blue_o),
23
24    .done_o(done_o)
25 );
26
27 initial clk = 1'b1;
28 always #(`clk_period /2) clk = ~clk;
29
30 integer i,j;
31 localparam RESULT_ARRAY_LEN = 500*1024;
32 reg [7:0] result [0:RESULT ARRAY LEN - 1];
```

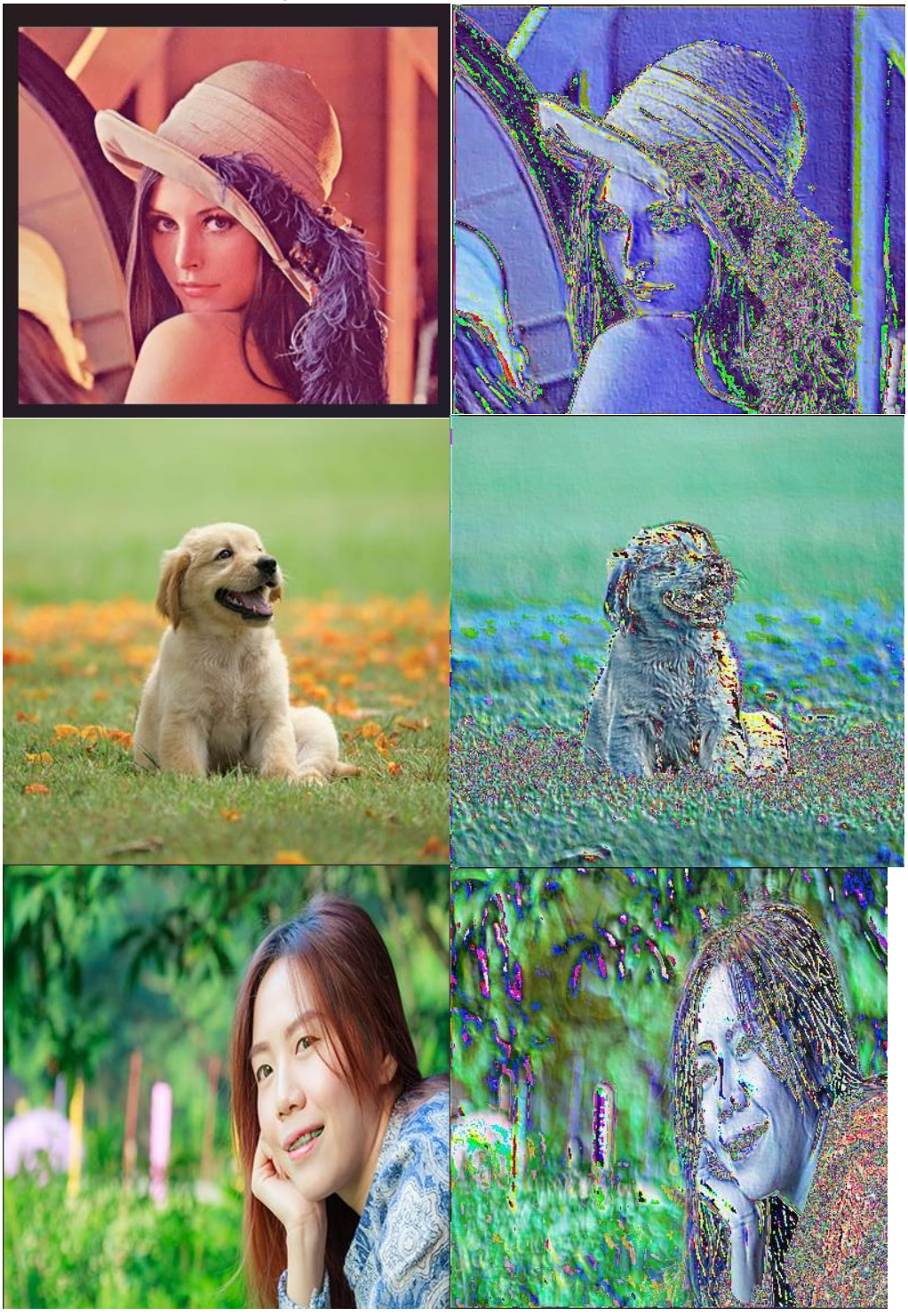
```
33
34
35 always@(posedge clk) begin
36    if(rst) begin
37       j <= 8'd0;
38    end
39    else begin
40       if(done_o == 1 ) begin
41          result[j] <= red_o;
42          result[j +1] <= green_o;
43          result[j +2] <= blue_o;
44          j <= j + 3;
45       end
46    end
47 end
48
49 `define read_fileName "E:/SOC_FINAL/girl.bmp"
50 localparam BMP_ARRAY_LEN = 500*1024;
51 reg [7:0] bmp_data [0:BMP_ARRAY_LEN - 1];
52 integer bmp_size, bmp_start_pos,bmp_width,bmp_height,biBitCount;
53
54 `define  text "E:/NO/file.txt"
55 task readBMP;
56    integer fileId,i;
57    begin
58       fileId = $fopen(`read_fileName,"rb");
59       if(fileId == 0) begin
60          $display("Open BMP error!\n");
61          $finish;
62       end
```

```
63        else begin
64           $fread(bmp_data,fileId);
65           $fclose(fileId);
66
67           bmp_size = {bmp_data[5],bmp_data[4],bmp_data[3],bmp_data[2]};
68           $display("bmp_size = %d!\n",bmp_size);
69
70           bmp_start_pos = {bmp_data[13],bmp_data[12],bmp_data[11],bmp_data[10]};
71           $display("bmp_start_pos = %d!\n",bmp_start_pos);
72
73           bmp_width = {bmp_data[21],bmp_data[20],bmp_data[19],bmp_data[18]};
74           $display("bmp_width = %d!\n",bmp_width);
75
76           bmp_height = {bmp_data[25],bmp_data[24],bmp_data[23],bmp_data[22]};
77           $display("bmp_height = %d!\n",bmp_height);
78
79           biBitCount = {bmp_data[29],bmp_data[28]};
80           $display("biBitCount = %d!\n",biBitCount);
81
82           if(biBitCount != 24) begin
83              $display("biBitCount need to be 24bit\n",biBitCount);
84              $finish;
85           end
86
87           if(bmp_width % 4) begin
88              $display("bmp_width div 4 need to be zero, \n");
89              $finish;
90           end
91
92           /*for(i = bmp_start_pos; i < bmp_size ; i = i+1) begin
93              //$display("%h",bmp_data[i]);
94              $fwrite(text,"%c", bmp_data[i]);
95           end*/
96
97        end
98     end
99  endtask
100
101
102
103  `define  write_fileName "E:/SOC_FINAL/result.bmp"
104
105 task writeBMP;
106    integer fileId ,i;
107    begin
108       fileId = $fopen(`write_fileName,"wb");
109
110       for(i = 0;i < bmp_start_pos ; i= i+1) begin
111          $fwrite(fileId , "%c",bmp_data[i]);
112       end
113
114       for(i = bmp_start_pos;i < bmp_size ; i= i+1) begin
115          $fwrite(fileId , "%c",result[i-bmp_start_pos]);
116       end
117
118       $fclose(fileId);
119       $display("writeBMP: done!\n");
120    end
121 endtask
122
```

```
123  initial begin
124      rst = 1'b1;
125      done_i = 1'b0;
126
127      red_i = 8'd0;
128      green_i = 8'd0;
129      blue_i = 8'd0;
130
131      readBMP;
132
133      #(`clk_period);
134      rst = 1'b0;
135
136      for(i = bmp_start_pos; i < bmp_size ; i = i+3)begin
137         red_i = bmp_data[i + 2];
138         green_i = bmp_data[i + 1];
139         blue_i = bmp_data[i];
140
141         #(`clk_period);
142         done_i = 1'b1;
143      end
144
145      #(`clk_period);
146      done_i = 1'b0;
147
148      #(`clk_period);
149      #(`clk_period);
150      #(`clk_period);
151      #(`clk_period);
152      #(`clk_period);
153      #(`clk_period);
154      #(`clk_period);
155      #(`clk_period);
156      #(`clk_period);
157      #(`clk_period);
158
159      writeBMP;
160      $stop;
161      /*readBMP;
162      writeBMP;*/
163  end
164  endmodule
```

CHAPTER 3: RESULTS & CONCLUSION:

3.1 Results after image emboss:



Original image Embossed image

3.2 Used resources:

Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	894 / 33,216 (3 %)
Total combinational functions	866 / 33,216 (3 %)
Dedicated logic registers	394 / 33,216 (1 %)
Total registers	394
Total pins	52 / 475 (11 %)
Total virtual pins	0
Total memory bits	19,200 / 483,840 (4 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

3.3 Conclusion:

The image embossing project successfully demonstrated the implementation of digital embossing techniques using image processing algorithms. By converting 2D images into embossed visuals, we created a simulated 3D effect that enhances the texture and depth perception of the original images. This project encompassed various stages, including image preprocessing, gradient calculation, and the application of embossing filters.

3.4 Direction of development:

The image embossing project has laid a solid foundation in digital image processing, transforming 2D visuals into simulated 3D textures. As we look to the future, several promising directions for further development stand out.

Enhancing Realism and Detail is another priority. Extending techniques to support color images and capturing finer textures will produce high-fidelity embossed images, preserving color integrity and adding visual appeal. This requires advanced filtering and high-resolution processing.

END OF DOCUMENT