

Jpeg Lifting Scheme

Testbench Status

The testbench is currently using 5 modules `jp_process`, `rom_flg`, `m_flatten`, `ram`, and `ram_res`. Three instances of `m_flatten` are used to generate `flat_lf`, `flat_sa`, and `flat_rt` 144 bit signals. Three instances of `ram` are used to store `ram_lf`, `ram_sa`, and `ram_rt` each of 144 bit signals.

In Appendix A Verilog “`ram.v`” for 144 bits signals. is the Verilog file generated with “`ram.py`”. In Appendix B. Verilog “`jp_process.v`” using 9 bits and 16 lifting schemes in parallel is the Verilog file generated with “`array_jpeg.py`”. In Appendix C. Verilog “`rom_flg.v`” using 5 bits with 80 bits width is the Verilog file generated with “`rom.py`”. Appendix D.

Generates 144 bit Signal is the Verilog file generated with “`flatten.py`”. Appendix E Verilog `ram_res.v` generated with `ram_res.py`.

With samples from a (red, blue, green) sub band making the signals `lft_s_i`, `sa_s_i`, and `rht_s_i`. Which now can be indexed with row and col not yet defined This first index is the row and 2nd index is the col.

Even samples `flat_lf` row starts at 2

```
    "mrow mcol pass1 even lf
      3 3 3 2 3 1 3 0 2 3 2 2 2 1 2 0 1 3 1 2 1 1 1 0 0
3 0 2 0 1 0 0
(1 2 3) (3 4 5) (5 6 7) (7 8 9) (9 10 11) (11 12 13) (13 14 15) (15 16 17) (17 18 19) (19 20 21)
(21 22 23) (23 24 25) (25 26 28) (27 28 29) (29 30 31) (31 32 33)"
```

```
for mmrow in range(3,-1,-1):
```

```
    mrow.next = mmrow
```

```
    yield clk_fast.posedge
```

```
    for mmcol in range(3,-1,-1):
```

```
        mcol.next = mmcol
```

```
        yield clk_fast.posedge
```

```
        if (mrow == 3 and mcol == 3):
```

```
            x.next = r[row-1][col]
```

```
        elif (mrow == 3 and mcol == 2):
```

```
            x.next = r[row+1][col]
```

```
        elif (mrow == 3 and mcol == 1):
```

```
            x.next = r[row+3][col]
```

```
        elif (mrow == 3 and mcol == 0):
```

```
            x.next = r[row+5][col]
```

```
        elif (mrow == 2 and mcol == 3):
```

```
            x.next = r[row+7][col]
```

```
        elif (mrow == 2 and mcol == 2):
```

```
            x.next = r[row+9][col]
```

```
        elif (mrow == 2 and mcol == 1):
```

```
            x.next = r[row+11][col]
```

```
        elif (mrow == 2 and mcol == 0):
```

```
            x.next = r[row+13][col]
```

```
        elif (mrow == 1 and mcol == 3):
```

```
            x.next = r[row+15][col]
```

```
        elif (mrow == 1 and mcol == 2):
```

```
            x.next = r[row+17][col]
```

```
        elif (mrow == 1 and mcol == 1):
```

```
            x.next = r[row+19][col]
```

```
        elif (mrow == 1 and mcol == 0):
```

```

        x.next = r[row+21][col]
    elif (mrow == 0 and mcol == 3):
        x.next = r[row+23][col]
    elif (mrow == 0 and mcol == 2):
        x.next = r[row+25][col]
    elif (mrow == 0 and mcol == 1):
        x.next = r[row+27][col]
    elif (mrow == 0 and mcol == 0):
        x.next = r[row+29][col]
    yield clk_fast.posedge
    print (" %d %s %d %d %d") % (now(),bin(x,W0), x, mrow, mcol)
    yield clk_fast.posedge
    z.next = x[W0:]
    yield clk_fast.posedge
    matrix_lf[mrow][mcol].next = z
    yield clk_fast.posedge
    lft_s_i.next = flat_lf
    yield clk_fast.posedge
even flat_sa

```

```

#combinel_sig_s.next = 0
"""mrow mcol pass1 even sa
    3 3  3 2  3 1  3 0  2 3  2 2  2 1  2 0  1 3  1 2  1 1  1 0  0
3   0 2   0 1   0 0
    (1 2 3) (3 4 5) (5 6 7) (7 8 9) (9 10 11) (11 12 13) (13 14 15) (15 16 17) (17 18 19) (19 20 21)
(21 22 23) (23 24 25) (25 26 28) (27 28 29) (29 30 31) (31 32 33)"""
for mmrow in range(3,-1,-1):
    mrow.next = mmrow
    yield clk_fast.posedge
    for mmcol in range(3,-1,-1):
        mcol.next = mmcol
        yield clk_fast.posedge
        if (mrow == 3 and mcol == 3):
            x.next = r[row][col]
        elif (mrow == 3 and mcol == 2):
            x.next = r[row+2][col]
        elif (mrow == 3 and mcol == 1):
            x.next = r[row+4][col]
        elif (mrow == 3 and mcol == 0):
            x.next = r[row+6][col]
        elif (mrow == 2 and mcol == 3):
            x.next = r[row+8][col]
        elif (mrow == 2 and mcol == 2):
            x.next = r[row+10][col]
        elif (mrow == 2 and mcol == 1):
            x.next = r[row+12][col]
        elif (mrow == 2 and mcol == 0):
            x.next = r[row+14][col]
        elif (mrow == 1 and mcol == 3):

```

```

        x.next = r[row+16][col]
    elif (mrow == 1 and mcol == 2):
        x.next = r[row+18][col]
    elif (mrow == 1 and mcol == 1):
        x.next = r[row+20][col]
    elif (mrow == 1 and mcol == 0):
        x.next = r[row+22][col]
    elif (mrow == 0 and mcol == 3):
        x.next = r[row+24][col]
    elif (mrow == 0 and mcol == 2):
        x.next = r[row+26][col]
    elif (mrow == 0 and mcol == 1):
        x.next = r[row+28][col]
    elif (mrow == 0 and mcol == 0):
        x.next = r[row+30][col]
    yield clk_fast.posedge
    print (" %d %s %d %d %d") % (now(),bin(x,W0), x, mrow, mcol)
    z.next = x[W0:]
    yield clk_fast.posedge
    matrix_sa[mrow][mcol].next = z
    yield clk_fast.posedge
    #print (" %d %s") % (now(),bin(flat_sa,W0*LVL0))

```

```

sa_s_i.next = flat_sa
yield clk_fast.posedge
"""mrow mcol pass1 even rt
    3 3   3 2   3 1   3 0   2 3   2 2   2 1   2 0   1 3   1 2   1 1   1 0   0
3   0 2   0 1   0 0
    (1 2 3) (3 4 5) (5 6 7) (7 8 9) (9 10 11) (11 12 13) (13 14 15) (15 16 17) (17 18 19) (19 20 21)
(21 22 23) (23 24 25) (25 26 28) (27 28 29) (29 30 31) (31 32 33)"""
for mmrow in range(3,-1,-1):
    mrow.next = mmrow
    yield clk_fast.posedge
    for mmcol in range(3,-1,-1):
        mcol.next = mmcol
        yield clk_fast.posedge
        if (mrow == 3 and mcol == 3):
            x.next = r[row+1][col]
        elif (mrow == 3 and mcol == 2):
            x.next = r[row+3][col]
        elif (mrow == 3 and mcol == 1):
            x.next = r[row+5][col]
        elif (mrow == 3 and mcol == 0):
            x.next = r[row+7][col]
        elif (mrow == 2 and mcol == 3):
            x.next = r[row+9][col]
        elif (mrow == 2 and mcol == 2):
            x.next = r[row+11][col]
        elif (mrow == 2 and mcol == 1):

```

```

        x.next = r[row+13][col]
    elif (mrow == 2 and mcol == 0):
        x.next = r[row+15][col]
    elif (mrow == 1 and mcol == 3):
        x.next = r[row+17][col]
    elif (mrow == 1 and mcol == 2):
        x.next = r[row+19][col]
    elif (mrow == 1 and mcol == 1):
        x.next = r[row+21][col]
    elif (mrow == 1 and mcol == 0):
        x.next = r[row+23][col]
    elif (mrow == 0 and mcol == 3):
        x.next = r[row+25][col]
    elif (mrow == 0 and mcol == 2):
        x.next = r[row+27][col]
    elif (mrow == 0 and mcol == 1):
        x.next = r[row+29][col]
    elif (mrow == 0 and mcol == 0):
        x.next = r[row+31][col]
    yield clk_fast.posedge
    print (" %d %s %d %d %d") % (now(),bin(x,W0), x, mrow, mcol)
    z.next = x[W0:]
    yield clk_fast.posedge
    matrix_rt[mrow][mcol].next = z
    yield clk_fast.posedge
    #print (" %d %s") % (now(),bin(flat_rt,W0*LVL0))

```

```

    rht_s_i.next = (flat_rt)
    yield clk_fast.posedge

```

Odd samples same even above but row starts at 1

```

    """mrow mcol pass1 odd lf
        3 3  3 2  3 1  3 0  2 3  2 2  2 1  2 0  1 3  1 2  1 1  1 0  0
3    0 2  0 1  0 0
        (0 1 2 ) (2 3 4 ) (4 5 6) (6 7 8) (8 9 10) (10 11 12) (12 13 14) (14 15 16) (16 17 18) (18 19 20)
(20 21 22) (22 23 24) (24 25 26) (26 27 28) (28 29 30) (30 31 32)"""

```

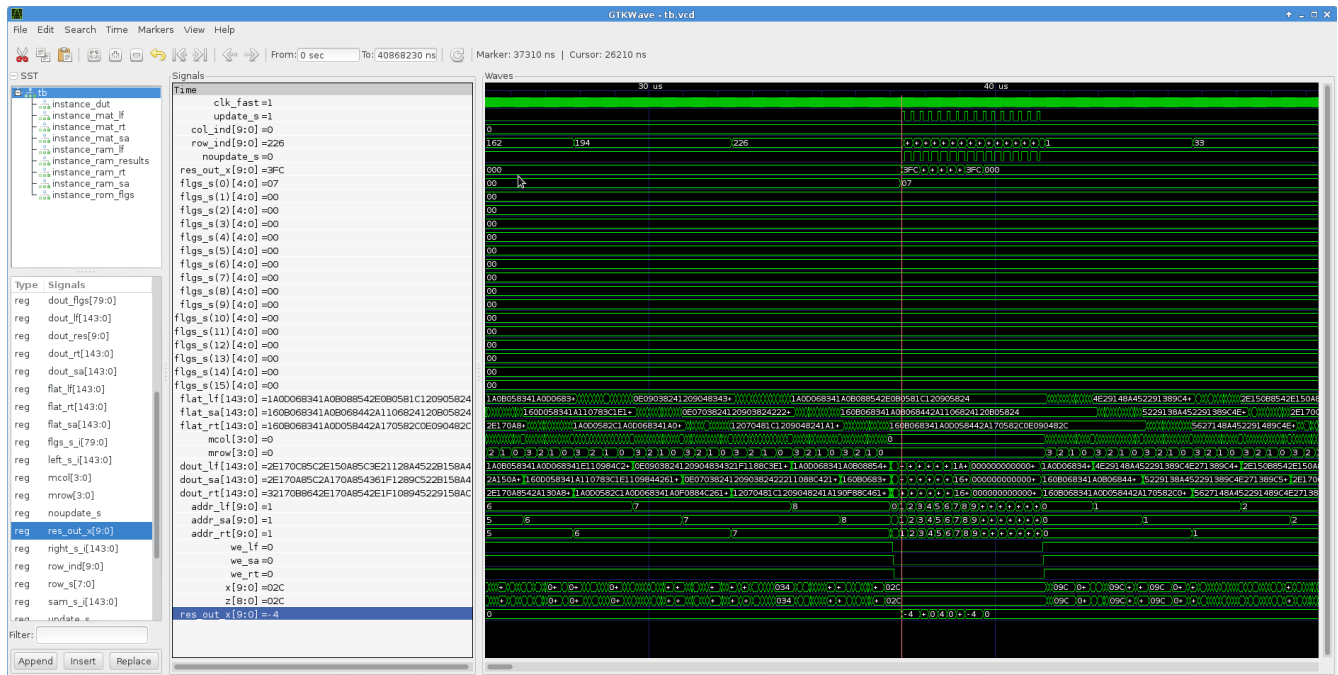
The outputs the module combine_sam are the inputs to signals left_s_i, sam_s_i, and right_s_i. These signals and the output of rom_flg are the inputs to the jp_process module. Currently bits are used for the flgs which determine the following:

```

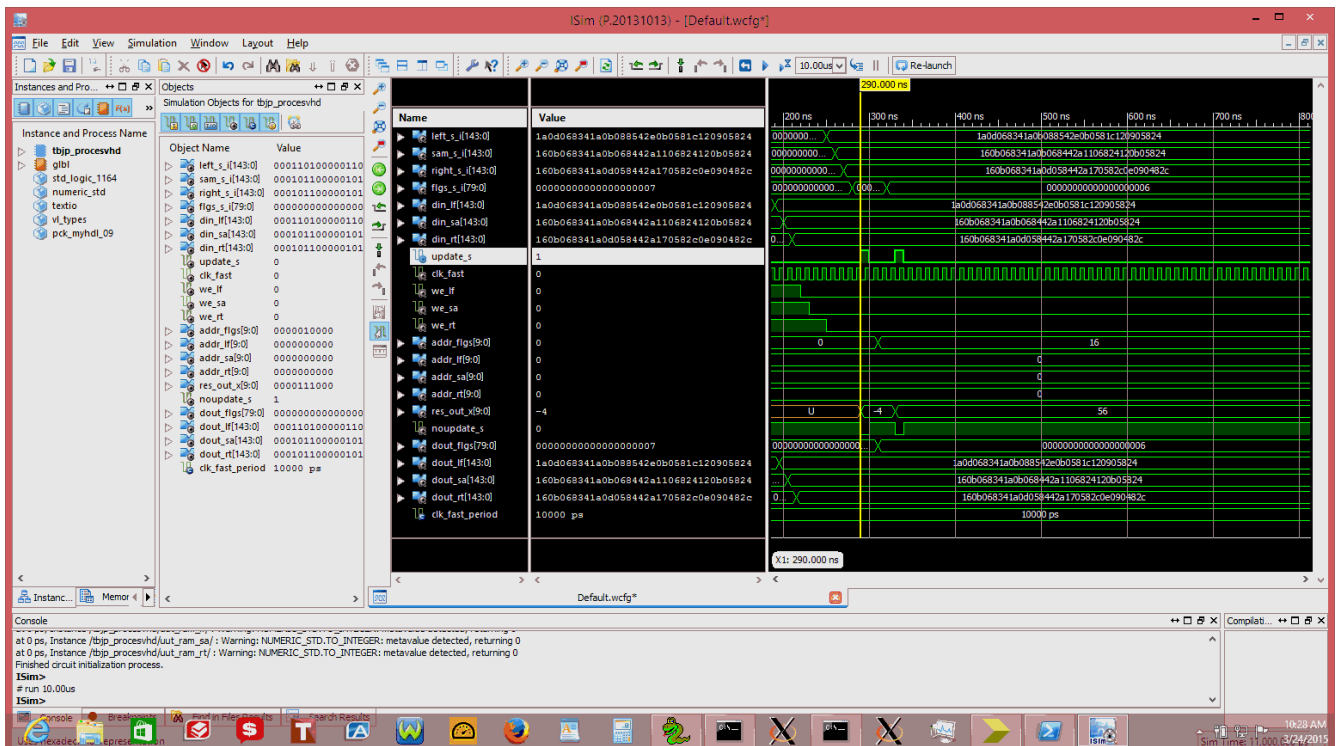
    update_s needs to be 1
    for the res_out_x to be valid
    nouupdate_s goes lo when a
    res_out_x valid
    fwd dwt even flgs_s eq 7
    inv dwt even flgs_s eq 5
    fwd dwt odd flgs_s eq 6
    inv dwt odd flgs_s eq 4

```

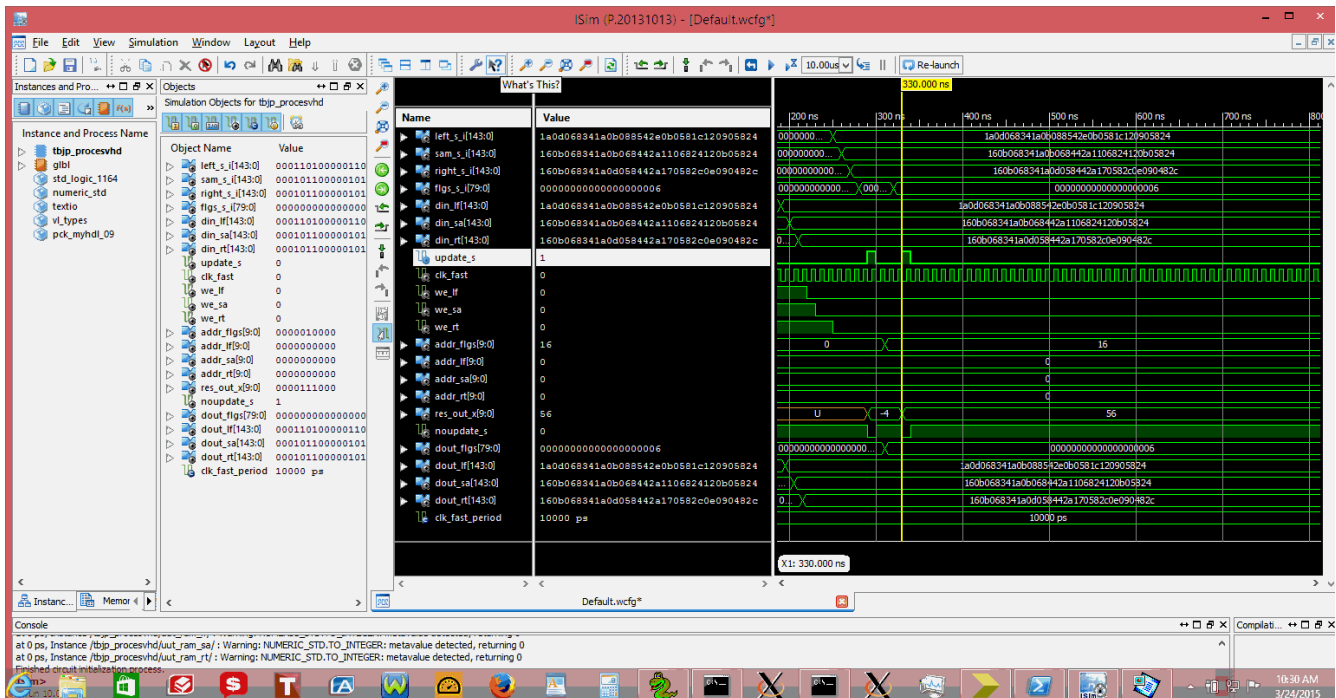
Using gtkwave to view the “gtkwave tb.vcd file”.



Using ISE to run a simulation of “tbjp_processvhd.vhd” for even sample.



Using ISE to run a simulation of “tbjp_processvhd” for odd sample.



The target image for the pass 1 is below.

This results was obtained using “t_wave_53.py” with “import waveletsim_53_16ops as dwt” instead of “import waveletsim_53 as dwt” and the lines 38 & 43 commented out.

To obtain just the pass 1 results lines 74-77 are commented out in “waveletsim_53_16ops.py”.

Note:

This can be fixed with systematical extension which currently being implemented in

“waveletsim_53_16ops.py”.

During the even pass the last

if (row != 226):

```
s[row+30][col] = (s[row+30][col] - ((int(s[row+30-1][col])>>1) +  
(int(s[row+31][col])>>1)))  
print row+30, col, int(s[row+30-1][col]), int(s[row+30][col]),  
int(s[row+31][col])
```

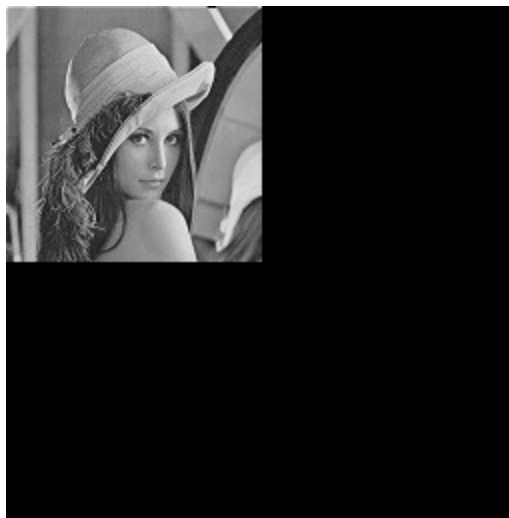
During the odd pass the last 2 operations need to be removed.

```
if (row != 225):  
s[row+30][col] = (s[row+30][col] + ((int(s[row+30-1][col]) +  
(int(s[row+31][col]) + 2))>>2))  
print row+30, col, int(s[row+30-1][col]), int(s[row+30][col]),  
int(s[row+31][col])  
s[row+32][col] = (s[row+32][col] + ((int(s[row+32-1][col]) +  
(int(s[row+33][col]) + 2))>>2))  
print row+32, col, int(s[row+32-1][col]), int(s[row+32][col]),  
int(s[row+33][col])
```

Using “t_wave_53_16ops.py” which uses “waveletsim_53_16ops.py” the target image was obtained with pass 1.



Using “t_wave_53_16ops.py” which uses “waveletsim_53_16ops.py” the target image was obtained with pass 1 and pass 2 is below.



With “python test_bench_array_jpeg_ram.py” the result “sim_pass1_2.png” is below when only pass1 is performed.



With “python test_bench_array_jpeg_ram.py” the result “sim_pass1_2.png” is below when only pass1 and pass 2 are performed.



The 512 X 512 Red sub band.



Appendix A Verilog “ram.v” for 144 bits signals.

// File: ram.v

// Generated by MyHDL 0.9dev

// Date: Tue Mar 24 10:04:43 2015

`timescale 1ns/10ps

```

module ram (
    dout,
    din,
    addr,
    we,
    clk_fast
);
// Ram model

output [143:0] dout;
wire [143:0] dout;
input [143:0] din;
input [9:0] addr;
input we;
input clk_fast;

reg [143:0] mem [0:512-1];

always @(posedge clk_fast) begin: RAM_WRITE
    if (we) begin
        mem[addr] <= din;
    end
end

```

```

assign dout = mem[addr];

```

```

endmodule

```

Appendix B Verilog “jp_process.v” using 9 bit input from 144 bit signal and 10 bit out and 16 lifting schemes in parallel.

```

// File: jp_process.v
// Generated by MyHDL 0.9dev
// Date: Wed Mar 18 10:19:47 2015

```

```

`timescale 1ns/10ps

```

```

module jp_process (
    res_out_x,
    left_s_i,
    sam_s_i,
    right_s_i,
    flgs_s_i,
    nouupdate_s,

```

```
    update_s  
);
```

```
output signed [9:0] res_out_x;  
reg signed [9:0] res_out_x;  
input [143:0] left_s_i;  
input [143:0] sam_s_i;  
input [143:0] right_s_i;  
input [79:0] flgs_s_i;  
output noupdate_s;  
reg noupdate_s;  
input update_s;
```

```
wire [8:0] right_s [0:16-1];  
wire [4:0] flgs_s [0:16-1];  
wire [8:0] sam_s [0:16-1];  
wire [8:0] left_s [0:16-1];
```

```
// update_s needs to be 1  
// for the res_out_x to be valid  
// noupdate_s goes lo when a  
// res_out_x valid  
// fwd dwt even flgs_s eq 7  
// inv dwt even flgs_s eq 5  
// fwd dwt odd flgs_s eq 6  
// inv dwt odd flgs_s eq 4  
always @(update_s, right_s[0], right_s[1], right_s[2], right_s[3], right_s[4], right_s[5], right_s[6],  
right_s[7], right_s[8], right_s[9], right_s[10], right_s[11], right_s[12], right_s[13], right_s[14],  
right_s[15], flgs_s[0], flgs_s[1], flgs_s[2], flgs_s[3], flgs_s[4], flgs_s[5], flgs_s[6], flgs_s[7], flgs_s[8],  
flgs_s[9], flgs_s[10], flgs_s[11], flgs_s[12], flgs_s[13], flgs_s[14], flgs_s[15], sam_s[0], sam_s[1],  
sam_s[2], sam_s[3], sam_s[4], sam_s[5], sam_s[6], sam_s[7], sam_s[8], sam_s[9], sam_s[10],  
sam_s[11], sam_s[12], sam_s[13], sam_s[14], sam_s[15], left_s[0], left_s[1], left_s[2], left_s[3],  
left_s[4], left_s[5], left_s[6], left_s[7], left_s[8], left_s[9], left_s[10], left_s[11], left_s[12], left_s[13],  
left_s[14], left_s[15]) begin: JP_PROCESS_JPEG_LOGIC  
    integer i;  
    if (update_s) begin  
        noupdate_s = 0;  
        for (i=0; i<16; i=i+1) begin  
            if ((flgs_s[i] == 7)) begin  
                res_out_x = ($signed(sam_s[i]) - ($signed($signed(left_s[i]) >>> 1) +  
$signed($signed(right_s[i]) >>> 1))));  
            end  
            else if ((flgs_s[i] == 5)) begin  
                res_out_x = ($signed(sam_s[i]) + ($signed($signed(left_s[i]) >>> 1) +  
$signed($signed(right_s[i]) >>> 1))));  
            end  
        end  
    end  
end
```

```

        end
        else if ((flgs_s[i] == 6)) begin
            res_out_x = ($signed(sam_s[i]) + $signed((((signed(left_s[i]) + signed(right_s[i])) + 2) >>>
2)));
        end
        else if ((flgs_s[i] == 4)) begin
            res_out_x = ($signed(sam_s[i]) - $signed((((signed(left_s[i]) + signed(right_s[i])) + 2) >>>
2)));
        end
    end
end
end
else begin
    nouupdate_s = 1;
end
end
endmodule

```

Appendix C Verilog “rom_flg.v” using 5 bits with 80 bits width.

```

// File: rom_flg.v
// Generated by MyHDL 0.9dev
// Date: Tue Mar 17 12:59:56 2015

```

```

`timescale 1ns/10ps

```

```

module rom_flg (
    dout_flg,
    addr_flg
);
// ROM model

```

```

output [79:0] dout_flg;
reg [79:0] dout_flg;
input [9:0] addr_flg;

```

```

always @(addr_flg) begin: ROM_FLGS_READ
    case (addr_flg)
        0: dout_flg = 7;
        1: dout_flg = 224;
        2: dout_flg = 7168;
        3: dout_flg = 229376;
        4: dout_flg = 7340032;
        5: dout_flg = 234881024;
    endcase
end

```

```
6: dout_flg = 34'h1c0000000;
7: dout_flg = 39'h3800000000;
8: dout_flg = 44'h70000000000;
9: dout_flg = 49'he00000000000;
10: dout_flg = 54'h1c00000000000;
11: dout_flg = 59'h38000000000000;
12: dout_flg = 64'h700000000000000;
13: dout_flg = 69'he00000000000000;
14: dout_flg = 74'h1c00000000000000;
15: dout_flg = 79'h38000000000000000;
16: dout_flg = 6;
17: dout_flg = 192;
18: dout_flg = 6144;
19: dout_flg = 196608;
20: dout_flg = 6291456;
21: dout_flg = 201326592;
22: dout_flg = 34'h180000000;
23: dout_flg = 39'h3000000000;
24: dout_flg = 44'h60000000000;
25: dout_flg = 49'hc00000000000;
26: dout_flg = 54'h18000000000000;
27: dout_flg = 59'h30000000000000;
28: dout_flg = 64'h600000000000000;
29: dout_flg = 69'hc00000000000000;
30: dout_flg = 74'h18000000000000000;
31: dout_flg = 79'h30000000000000000;
32: dout_flg = 5;
33: dout_flg = 160;
34: dout_flg = 5120;
35: dout_flg = 163840;
36: dout_flg = 5242880;
37: dout_flg = 167772160;
38: dout_flg = 34'h140000000;
39: dout_flg = 39'h2800000000;
40: dout_flg = 44'h50000000000;
41: dout_flg = 49'ha00000000000;
42: dout_flg = 54'h14000000000000;
43: dout_flg = 59'h28000000000000;
44: dout_flg = 64'h500000000000000;
45: dout_flg = 69'ha00000000000000;
46: dout_flg = 74'h14000000000000000;
47: dout_flg = 79'h28000000000000000;
48: dout_flg = 4;
49: dout_flg = 128;
50: dout_flg = 4096;
51: dout_flg = 131072;
52: dout_flg = 4194304;
53: dout_flg = 134217728;
54: dout_flg = 34'h100000000;
```

```

55: dout_flg = 39'h20000000000;
56: dout_flg = 44'h40000000000;
57: dout_flg = 49'h800000000000;
58: dout_flg = 53'h10000000000000;
59: dout_flg = 58'h200000000000000;
60: dout_flg = 64'h400000000000000;
61: dout_flg = 68'h800000000000000;
62: dout_flg = 73'h1000000000000000;
default: dout_flg = 78'h2000000000000000;
endcase
end

endmodule

```

Appendix D. Generates 144 bit Signal.

```

// File: m_flatten.v
// Generated by MyHDL 0.9dev
// Date: Wed Mar 18 10:54:07 2015

```

```

`timescale 1ns/10ps

module m_flatten (
    flat

);

output [143:0] flat;
wire [143:0] flat;

wire [143:0] _flat;
wire [8:0] mcol;

assign mcol = 0;

assign _flat[144-1:135] = None;
assign _flat[135-1:126] = None;
assign _flat[126-1:117] = None;
assign _flat[117-1:108] = None;
assign _flat[108-1:99] = None;
assign _flat[99-1:90] = None;
assign _flat[90-1:81] = None;
assign _flat[81-1:72] = None;
assign _flat[72-1:63] = None;
assign _flat[63-1:54] = None;
assign _flat[54-1:45] = None;
assign _flat[45-1:36] = None;

```

```
assign _flat[36-1:27] = None;  
assign _flat[27-1:18] = None;  
assign _flat[18-1:9] = None;  
assign _flat[9-1:0] = mcol[9-1:0];
```

```
assign flat = _flat;
```

```
endmodule
```

Appendix E Verilog ram_res.v generated with ram_res.py.

// File: ram_res.v

// Generated by MyHDL 0.9dev

// Date: Tue Mar 24 12:15:43 2015

```
`timescale 1ns/10ps
```

```
module ram_res (  
    dout_res,  
    din_res,  
    addr_res,  
    we_res,  
    clk_fast  
);
```

```
// Ram model
```

```
output [9:0] dout_res;  
wire [9:0] dout_res;  
input [9:0] din_res;  
input [9:0] addr_res;  
input we_res;  
input clk_fast;
```

```
reg [8:0] mem [0:512-1];
```

```
always @(posedge clk_fast) begin: RAM_RES_WRITE  
    if (we_res) begin  
        mem[addr_res] <= din_res;  
    end  
end
```

```
assign dout_res = mem[addr_res];
```

```
endmodule
```