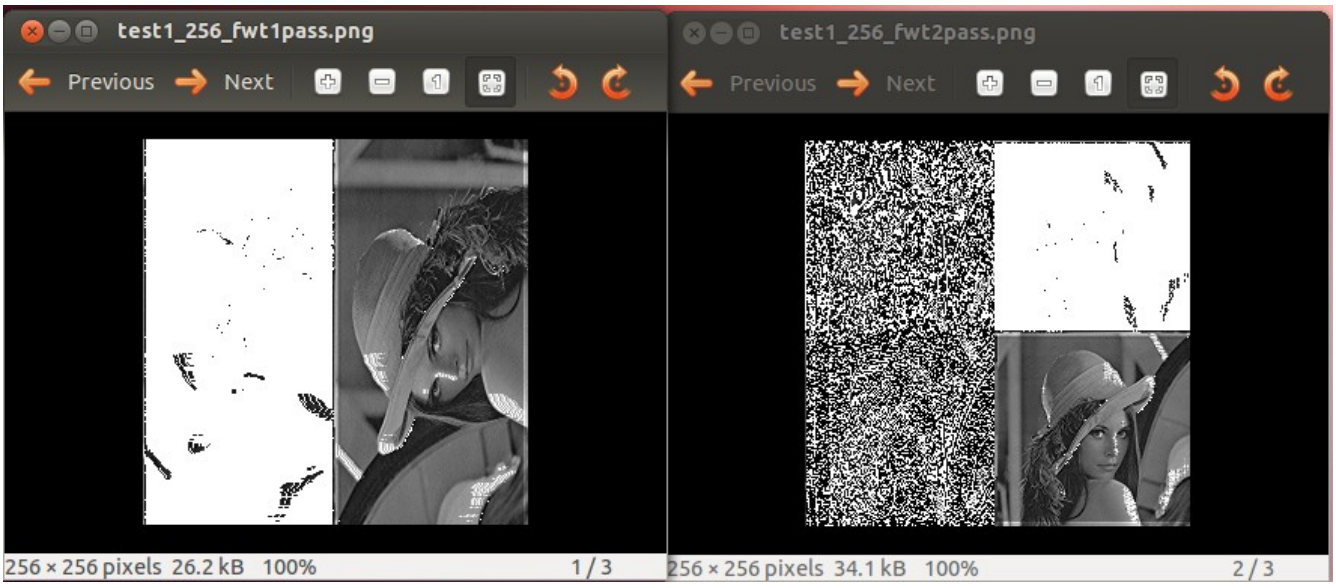
Simulation
Using 4 instances of DWT to perform JPEG-2000

Using multiple instance is to minimize the transfers between the host & FPGA and the FPGA & host.

The first level of the DWT is done with 2 passes of even & odd samples. The first pass is from top to bottom from left to right which takes the original and makes the half size of the original image.

The 2nd pass is from top to bottom from left to right which takes the half size and creates an image a fourth of the original in the lower right hand corner. ***Note: Currently several errors exits in the first even & odd pass. This can be observed in front edge of the hat and in the lower right of the image. In addition over the right eye an some locations on the feathers of the hat.***



On a 6-Core AMD running Ubuntu 12.04 the simulation takes

vidal@ws009:~/wkg/jpeg-2000-test/pc_fast_blinker_jpeg$ time python test_top_a.py --test

real 1m6.287s
user 1m6.036s
sys 0m0.228s

On a RPi2B with pypy the simulation takes

time /opt/pypy-4.0.1-linux-armhf-raspbian/bin/pypy test_top_a.py --test

real 4m22.736s
user 4m17.710s
sys 0m2.180s

pi@mysshserver ~/jpeg-2000-test/pc_fast_blinker_jpeg $ time python test_top_a.py --test

real 12m31.381s
user 12m19.160s
sys 0m2.330s
Using pypy provides a 1.43 times improvement over python.

Simulation time
23039990 ns 23.03 msec at 50MHz

120000000	2.4	9.60E-003
100000000	2	1.15E-002

Appendix A Verilog code for 4 instances of JPEG lifting step.

```
// File: dwt_top.v
// Generated by MyHDL 1.0dev
// Date: Fri Jan 29 15:25:12 2016


`timescale 1ns/10ps

module dwt_top (
    flgs0,
    upd0,
    lft0,
    sam0,
    rht0,
    done0,
    clock,
    z0,
    flgs1,
    upd1,
    lft1,
    sam1,
    rht1,
    done1,
    z1
```

```
);


input [2:0] flgs0;
input upd0;
input [8:0] lft0;
input [8:0] sam0;
input [8:0] rht0;
output done0;
reg done0;
input clock;
output [8:0] z0;
wire [8:0] z0;
input [2:0] flgs1;
input upd1;
input [8:0] lft1;
input [8:0] sam1;
input [8:0] rht1;
output done1;
reg done1;
output [8:0] z1;
wire [8:0] z1;

wire [8:0] lft2;
wire [8:0] lft3;
wire signed [9:0] res2;
wire signed [9:0] res3;
wire signed [9:0] res0;
wire signed [9:0] res1;
wire upd3;
wire upd2;
wire [2:0] flgs3;
wire [8:0] z2;
wire [8:0] z3;
wire [2:0] flgs2;
reg done3;
reg done2;
wire [8:0] rht4;
wire [8:0] rht2;
reg signed [9:0] lift1;
reg signed [9:0] lift2;
reg signed [9:0] lift3;
reg signed [9:0] lift0;
wire [8:0] sam3;
wire [8:0] sam2;


assign lft2 = 9'd0;
assign lft3 = 9'd0;
```

```
assign upd3 = 1'd0;
assign upd2 = 1'd0;
assign flgs3 = 3'd0;
assign flgs2 = 3'd0;
assign rht4 = 9'd0;
assign rht2 = 9'd0;
assign sam3 = 9'd0;
assign sam2 = 9'd0;



always @(posedge clock) begin: DWT_TOP_INSTANCE_2_RTL
   if ((upd2 == 1)) begin
      done2 <= 0;
      case (flgs2)
         'h7: begin
            lift2 <= ($signed(sam2) - ($signed($signed(lft2) >>> 1) + $signed($signed(rht2) >>> 1)));
         end
         'h5: begin
            lift2 <= ($signed(sam2) + ($signed($signed(lft2) >>> 1) + $signed($signed(rht2) >>> 1)));
         end
         'h6: begin
            lift2 <= ($signed(sam2) + $signed((($signed(lft2) + $signed(rht2)) + 2) >>> 2));
         end
         'h4: begin
            lift2 <= ($signed(sam2) - $signed((($signed(lft2) + $signed(rht2)) + 2) >>> 2));
         end
      endcase
   end
   else begin
      done2 <= 1;
   end
end


always @(posedge clock) begin: DWT_TOP_INSTANCE_3_RTL
   if ((upd3 == 1)) begin
      done3 <= 0;
      case (flgs3)
         'h7: begin
            lift3 <= ($signed(sam3) - ($signed($signed(lft3) >>> 1) + $signed($signed(rht4) >>> 1)));
         end
         'h5: begin
            lift3 <= ($signed(sam3) + ($signed($signed(lft3) >>> 1) + $signed($signed(rht4) >>> 1)));
         end
         'h6: begin
            lift3 <= ($signed(sam3) + $signed((($signed(lft3) + $signed(rht4)) + 2) >>> 2));
         end
         'h4: begin
```

```
            lift3 <= ($signed(sam3) - $signed((($signed(lft3) + $signed(rht4)) + 2) >>> 2));
         end
      endcase
   end
   else begin
      done3 <= 1;
   end
end


always @(posedge clock) begin: DWT_TOP_INSTANCE_0_RTL
   if ((upd0 == 1)) begin
      done0 <= 0;
      case (flgs0)
         'h7: begin
            lift0 <= ($signed(sam0) - ($signed($signed(lft0) >>> 1) + $signed($signed(rht0) >>> 1)));
         end
         'h5: begin
            lift0 <= ($signed(sam0) + ($signed($signed(lft0) >>> 1) + $signed($signed(rht0) >>> 1)));
         end
         'h6: begin
            lift0 <= ($signed(sam0) + $signed((($signed(lft0) + $signed(rht0)) + 2) >>> 2));
         end
         'h4: begin
            lift0 <= ($signed(sam0) - $signed((($signed(lft0) + $signed(rht0)) + 2) >>> 2));
         end
      endcase
   end
   else begin
      done0 <= 1;
   end
end



assign z0 = res0;



assign res0 = lift0[9-1:0];


always @(posedge clock) begin: DWT_TOP_INSTANCE_1_RTL
   if ((upd1 == 1)) begin
      done1 <= 0;
      case (flgs1)
         'h7: begin
            lift1 <= ($signed(sam1) - ($signed($signed(lft1) >>> 1) + $signed($signed(rht1) >>> 1)));
         end
```

```
        'h5: begin
           lift1 <= ($signed(sam1) + ($signed($signed(lft1) >>> 1) + $signed($signed(rht1) >>> 1)));
        end
        'h6: begin
           lift1 <= ($signed(sam1) + $signed((($signed(lft1) + $signed(rht1)) + 2) >>> 2));
        end
        'h4: begin
           lift1 <= ($signed(sam1) - $signed((($signed(lft1) + $signed(rht1)) + 2) >>> 2));
        end
      endcase
   end
   else begin
      done1 <= 1;
   end
end



assign res2 = lift2[9-1:0];



assign res3 = lift3[9-1:0];



assign z2 = res2;



assign z3 = res3;



assign z1 = res1;



assign res1 = lift1[9-1:0];

endmodule
```

Appendix B. Simulation of first even odd pass

```
        '''
        sending from host to FPGA
        m[row-5][col] m[row-4][col] m[row-3][col]
        m[row-6][col] m[row-3][col] m[row-2][col]
        flgs0 upd0 flgs1 upd1 flgs2 upd2 flgs3 upd3
```

returning to host from FPGA
m[row-4][col] m[row-5][col] m[row-2][col] m[row-3][col]
First pass even odd samples
1 2 3 0 1 2
1
'''
lft0.next = m[row-5][col]
yield clock.posedge
# 2
sam0.next = m[row-4][col]
yield clock.posedge
# 3
rht0.next = m[row-3][col]
yield clock.posedge
flgs0.next = 7
yield clock.posedge
upd0.next = 1
yield clock.posedge
upd0.next = 0
yield clock.posedge
#this needs to be the rht1
#1 2 3 0 1 2
2
m[row-4][col] = lift0[W0:]
# 2
rht1.next = z0
yield clock.posedge
# 0
lft1.next = m[row-6][col]
yield clock.posedge
# 1
sam1.next = lft0
yield clock.posedge
flgs1.next = 6
yield clock.posedge
upd1.next = 1
yield clock.posedge
upd1.next = 0
yield clock.posedge
#this needs to be the rht2
1
m[row-5][col] = lift1[W0:]

#3 4 5 2 3 4
# 5
rht2.next = z1
yield clock.posedge
# 3
lft2.next = rht0

```
#lft2.next = m[row-3][col]
yield clock.posedge
# 4
sam2.next = m[row-2][col]
yield clock.posedge

flgs2.next = 7
yield clock.posedge
upd2.next = 1
yield clock.posedge
upd2.next = 0
yield clock.posedge
#this needs to be the rht3
# 4
m[row-2][col] = lift2[W0:]
#3 4 5  2 3 4
# 4
rht3.next = z2
yield clock.posedge
# 2
lft3.next = z1
#lft3.next = m[row-4][col]
yield clock.posedge
# 3
sam3.next = lft2
yield clock.posedge
flgs3.next = 6
yield clock.posedge
upd3.next = 1
yield clock.posedge
upd3.next = 0
yield clock.posedge
#this needs to be the rht1
# 3
m[row-3][col] = lift3[W0:]
```