SAN FRANCISCO STATE UNIVERSITY

SCHOOL OF ENGINEERING

Final Project Report: Motion Estimator

ENGR 852-01

Fall 2020

By

Amir Modan, #918234621

amodan1@mail.sfsu.edu

Date report submitted: Dec. 20, 2020

Table of Contents

Problem Analysis: Today's world depends very heavily on video files especially since a majority of human communication is currently virtual. Videos are now being used everywhere, from the movie industry all the way to education. Especially with the rise of 4K (and soon 8K), video sizes are getting larger than ever, making it increasingly difficult to share and store on one's computer. This is what makes the practice of Motion Estimation is so important. Instead of simply storing each pixel for each frame, one can instead calculate and store the much more portable motion vector of subsequent frames, drastically decreasing file size.

We are tasked to implement this device using an Application Specific Integrated Circuit. By implementing the Motion Estimator using dedicated hardware, we have the potential to increase efficiency over a software implementation using algorithms. The 32/28nm CMOS Library will be used when designing the physical layout of the chip. As for the motion estimator itself, the design will be capable of reading a 16x16 Reference Frame coded in an 8-bit Grayscale. The subsequent Search Frame for which we will be determining the motion displacement vector will have dimensions 31x31 also coded in Grayscale. We have both 130MHz and 260MHz Clocks available to us, though we will have to meet the minimum search speed of 15 Frames/Second.

Hardware Design: Before writing a Verilog module to implement this Motion Estimator, we must first plan out the behavioral structure of our design. This will reduce the likelihood of creating bugs in the design. A Black-Box representation of our design can be seen below in Figure 1.

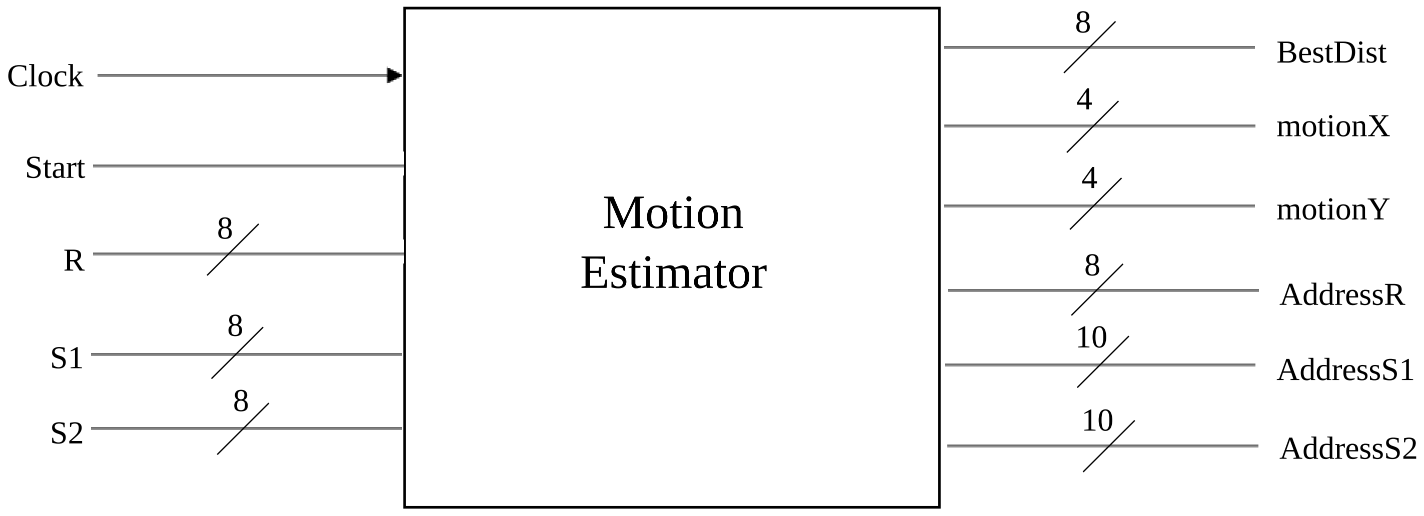


Figure 1 - Motion Estimator, Black Box Representation

As indicated above, all operations will take place on the rising edge of the clock signal as opposed to using a level-sensitive latch-based design. A start signal will signal the Motion Estimator to begin executing calculations when the start signal is high. Note, our memories R, S1, and S2 (representing Reference Frame and two Possible Search Frames) are external to the Motion Estimator and will not be included in our final design. Therefore, we will have to access the memory contents as input signals to the Motion Estimator, making it possible to switch out different memories to compute Motion Estimation on. Our design will be capable of calculating the amount of distortion seen on subsequent frames, allowing us to deduce how similar two frames really are. The estimated motion vectors for the X and Y axis will be in Two's Compliment form, meaning that motion can range anywhere from -8 to 7 pixels.

Fortunately, we have several reference materials available to us on how to compute the Motion Vectors. We will be using the most commonly used Block Matching Algorithm to implement the Motion Estimator. The way that this algorithm works is that a frame comparison must be made between Reference and Search frames for each possible position of the Reference Frame. If the frame comparison yields low distortion, the vector describing the movement of the original reference frame to that position on the search frame will be saved as the estimated motion until a comparison yielding an even lower distortion is found.

In order to calculate this distortion, we will first have to implement a Processing Element capable of determining an accumulated distortion value based on the difference between the R memory and either S1 or S2 depending on which memory we want to use. If the distortion is greater than the maximum value allowed by 8 bits (255), distortion will simply saturate at this maximum value. In order to meet the required speed of our motion estimator, we will utilize Pipelining by having 16 Processing Elements working in Parallel, though with the Reference Memory 'pipelining' through each one.

Then, we must make a Comparator Module that will receive the 16 distortion values from the Processing Elements (all accumulated into one long vector) and determine whether they are smaller than the currently saved value, initially set to the maximum value of 255. If distortion is smaller, we will save the frame displacement as our motion X and Y vectors.

Next, to generate all of the signals that will start calculating and comparing new distortion values, accessing memory addresses, etc., we will implement a controller that will mimic the behavior of a reference block moving through each possible position of a search block.

Finally, all of these modules will be brought together in one top module like the one shown in Figure 1.

Verification: In order to verify that our design actually works as intended, we will create several testbenches instantiating the Motion Estimator, then run each one through a waveform simulator. First, we will simulate and verify each of the submodules individually, so that it will be easier to debug the top module if necessary.

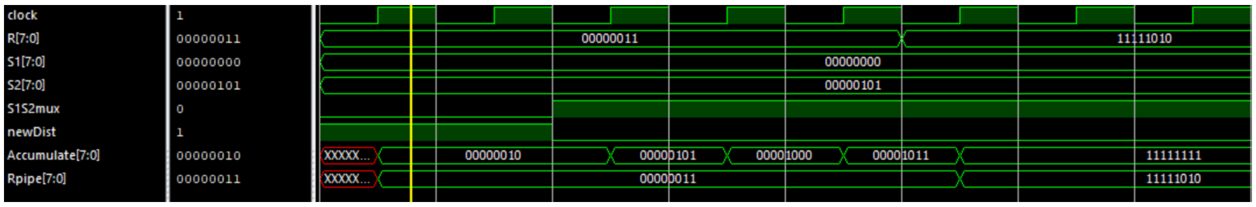


Figure 2 - Processing Element (PE) Waveform

Figure 2 shows what happens when we provide several stimuli to the Processing Element Module. One must set newDist to a high value in order to begin calculating a new distortion value. Distortion will be calculated by taking the difference between R and S1/S2 depending on the value of S1S2mux. S1S2mux is initially low, indicating we will be reading from S2. As seen by the Accumulate register, the accumulated distortion starts at two, appropriate since: $|R - S1| = |3 - 5| = 2$. Now newDist must be set to a low value, else accumulated distortion will continue to be set to the difference between R and S rather than adding to it. By setting S1S2mux high, we will now confirm that the lower value held by S1 is also functional. The difference will now be 3, indicating that 3 will now be added to the accumulated distortion. This behavior is correctly captured by the waveform as the value of *Accumulate* changes from $2 \to 5 \to 8 \to 11$. By changing R to a much higher value, we successfully validate the saturation feature of *Accumulate* as $11 + (250 - 0) = 266 \to 255$.

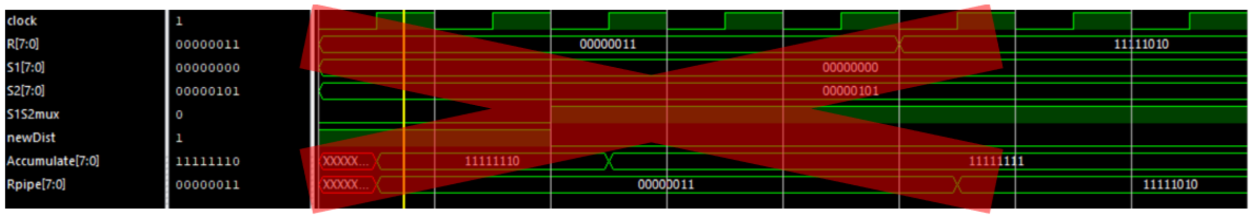
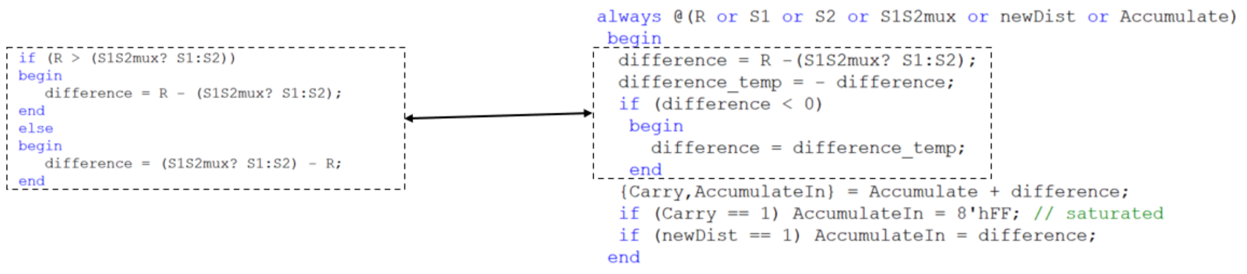


Figure 3 - Processing Element Bug

During simulation for the Processing Element, we had discovered a bug in which the difference between the R and S memories was not absolute. The distortion calculated when R is less than S would be the 2's compliment of the desired value, causing the distortion to be radically different. For instance, during the second clock pulse above: $R - S1 = 3 - 5 = -2 \to 254$. We can remedy this by using if-else logic to determine whether R is greater than or less than S, then execute the appropriate logic as shown below.



```
if (R > (S1S2mux? S1:S2))
begin
   difference = R - (S1S2mux? S1:S2);
end
else
begin
   difference = (S1S2mux? S1:S2) - R;
end
```

```
always @(R or S1 or S2 or S1S2mux or newDist or Accumulate)
begin
  difference = R -(S1S2mux? S1:S2);
  difference_temp = - difference;
  if (difference < 0)
   begin
     difference = difference_temp;
   end
  {Carry,AccumulateIn} = Accumulate + difference;
  if (Carry == 1) AccumulateIn = 8'hFF; // saturated
  if (newDist == 1) AccumulateIn = difference;
end
```

Figure 4 - Processing Element Bug Fix

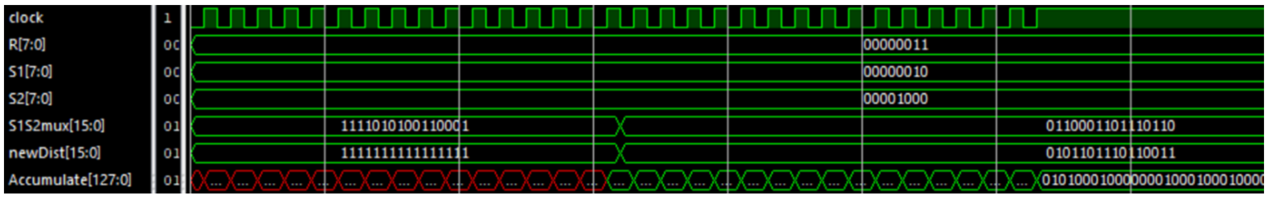


Figure 5 – Processing Element Toral (PETotal) Waveform

Figure 5 represents how all 16 Processing Elements will work together in the PETotal module. Note that it will take 16 clock cycles to fully obtain a real value for the Accumulated Distortion, as one clock cycle will be required for each Processing Element to pipe the R memory. The Accumulated Distortion will then be used as an input for the Comparator Module.

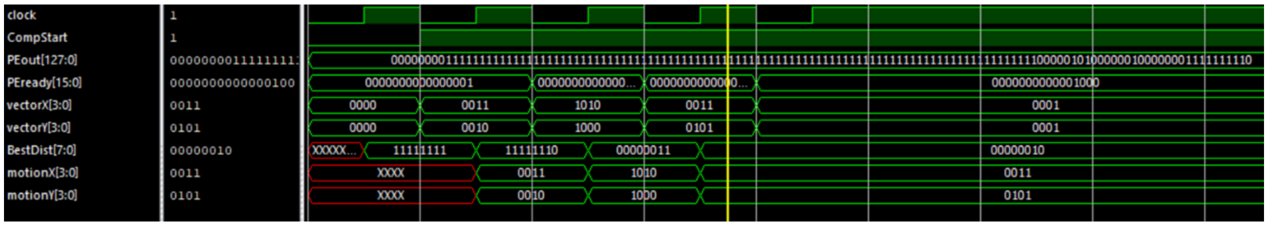
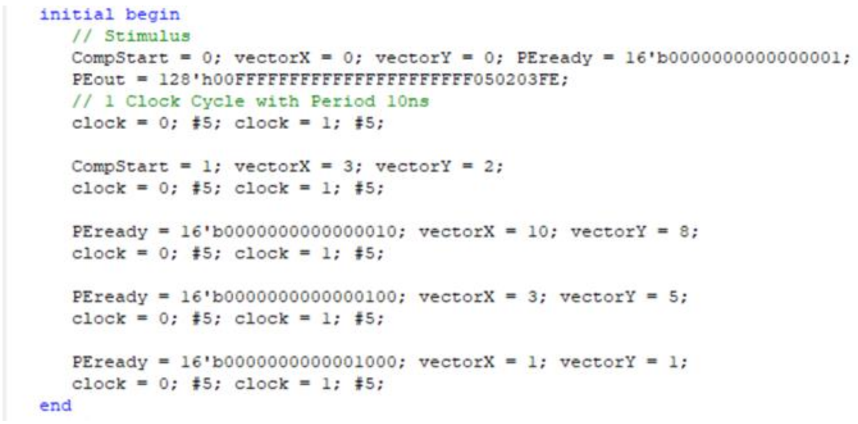


Figure 6 - Comparator Module Waveform

After signaling the Comparator to start using the CompStart Register, we can see difference frame placements being tested for the lowest distortion. We notice that the BestDist output starts at its maximum worse-case value and continues to decrease overtime as frame placements with lower distortions are found. After a number of clock cycles, we see that the outputs cease to change as the Best Distortion has been found, so BestDist and the motion vectors pertaining to this frame are saved.



```
initial begin
   // Stimulus
   CompStart = 0; vectorX = 0; vectorY = 0; PEready = 16'b0000000000000001;
   PEout = 128'h00FFFFFFFFFFFFFFFFFFFFFFF050203FE;
   // 1 Clock Cycle with Period 10ns
   clock = 0; #5; clock = 1; #5;

   CompStart = 1; vectorX = 3; vectorY = 2;
   clock = 0; #5; clock = 1; #5;

   PEready = 16'b0000000000000010; vectorX = 10; vectorY = 8;
   clock = 0; #5; clock = 1; #5;

   PEready = 16'b0000000000000100; vectorX = 3; vectorY = 5;
   clock = 0; #5; clock = 1; #5;

   PEready = 16'b0000000000001000; vectorX = 1; vectorY = 1;
   clock = 0; #5; clock = 1; #5;
end
```

Figure 7 - Comparator Stimulus

Looking more closely at the stimulus for the Comparator, we see that the smallest distortion vector is indeed 2, as the other tested vectors: 254, 3, and 5 are all greater. The motion vectors pertaining to this frame, (3, 5) are also saved. The only exception is 0, provided by PE16. However, this Processing Element was never signaled as ready and thus was never considered.

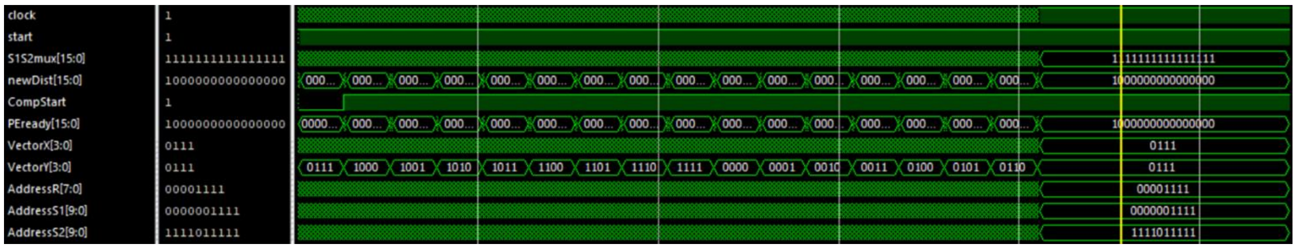
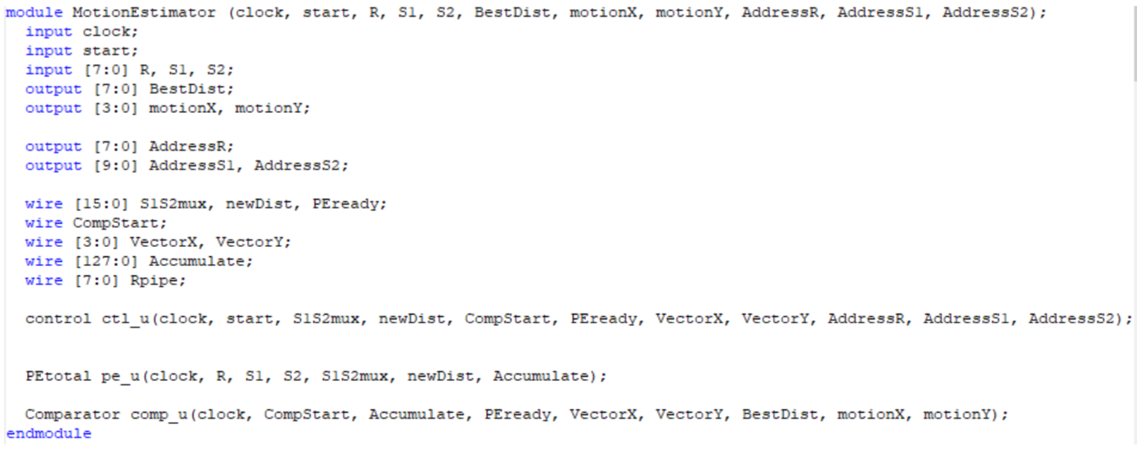


Figure 8 - Controller Waveform

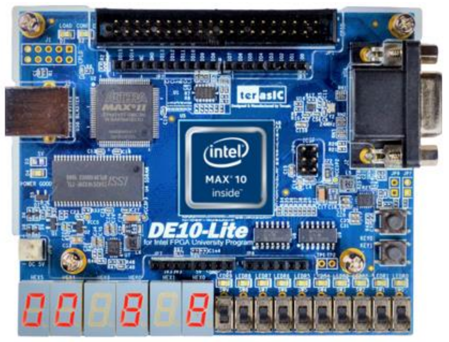
The controller generating all of the signals used in our other modules is seen in Figure 8. By zooming into any part of the waveform, one will see several of the output signals slowly incrementing on each clock pulse. This behavior is similar to how a reference frame would be moving left-to-right and up-to-down a search frame.

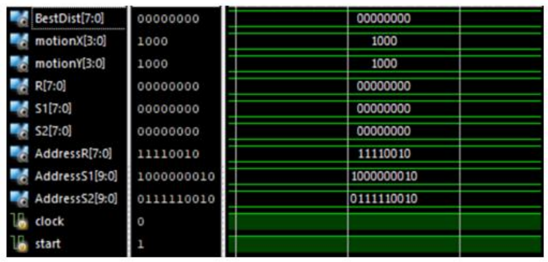


```
module MotionEstimator (clock, start, R, S1, S2, BestDist, motionX, motionY, AddressR, AddressS1, AddressS2);
  input clock;
  input start;
  input [7:0] R, S1, S2;
  output [7:0] BestDist;
  output [3:0] motionX, motionY;

  output [7:0] AddressR;
  output [9:0] AddressS1, AddressS2;

  wire [15:0] S1S2mux, newDist, PEready;
  wire CompStart;
  wire [3:0] VectorX, VectorY;
  wire [127:0] Accumulate;
  wire [7:0] Rpipe;

  control ctl_u(clock, start, S1S2mux, newDist, CompStart, PEready, VectorX, VectorY, AddressR, AddressS1, AddressS2);


  PEtotal pe_u(clock, R, S1, S2, S1S2mux, newDist, Accumulate);

  Comparator comp_u(clock, CompStart, Accumulate, PEready, VectorX, VectorY, BestDist, motionX, motionY);
endmodule
```

Figure 9 - Top Module

Finally, we will integrate all that we have verified previously into one top module. Remember that we intend to have our memories be external to our design, so they should only be seen in the I/O list similar to Figure 1. Because our goal when simulating previous sub-modules was to check for bugs at the RTL level, we used only waveform simulation. However for the top module, we will also want to see how the actual Motion Estimator will perform behaviorally using actual hardware. For this reason, we will emulate the Motion Estimator using an FPGA as it will give us a good idea how the Motion Estimator will perform on a dedicated chip. Because the FPGA is "Field-Programmable", downloading the RTL code to the FPGA will be relatively seamless, especially compared to the ASIC which is not reprogrammable and requires a longer design process. If we find any bugs in the RTL code, we can simply erase and reprogram the FPGA whereas the ASIC would have to be discarded and refabricated. Nevertheless, we did not use the FPGA to emulate sub-modules as the FPGA only has a limited number of I/O pins from which we can monitor. For the following emulations, the seven-segment displays represent in Hexadecimal Format: The Best Distortion, Motion Y, Motion X. The left-most switch will also be used as an active-high start signal.





BestDist[7:0]	00000000	00000000
motionX[3:0]	1000	1000
motionY[3:0]	1000	1000
R[7:0]	00000000	00000000
S1[7:0]	00000000	00000000
S2[7:0]	00000000	00000000
AddressR[7:0]	11110010	11110010
AddressS1[9:0]	1000000010	1000000010
AddressS2[9:0]	0111110010	0111110010
clock	0	
start	1	

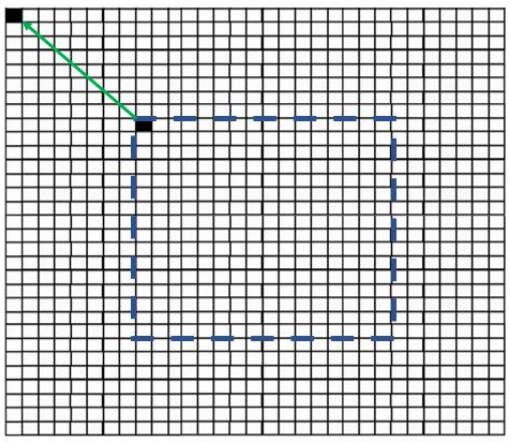
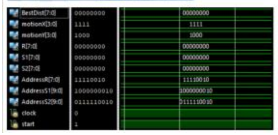
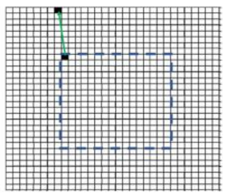


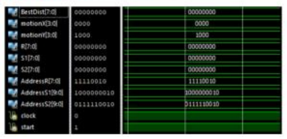
Figure 10 - Top Module Waveform Simulation and FPGA Emulation

In order to obtain the results in Figure 10, we had to set the first address in both R and S memories to the same colored pixel (11111111 in this case). This would be equivalent to moving a pixel from the top-left corner of the R frame to the top-left corner of the S frame. In total, there should be 256 addresses for the R memory and 961 addresses for the S memories. Because both pixels are the same color and no other pixels are present, there is no distortion present. This behavior is correctly estimated by both the waveform simulator and the FPGA. Do note that motion vectors are in 2's compliment form, indicating that a vector pointing to the left is negative and a vector pointing upwards is also negative.



BestDist[7:0]	00000000	00000000
motionX[3:0]	1111	1111
motionY[3:0]	1000	1000
R[7:0]	00000000	00000000
S1[7:0]	00000000	00000000
S2[7:0]	00000000	00000000
AddressR[7:0]	11110010	11110010
AddressS1[9:0]	1000000010	1000000010
AddressS2[9:0]	0111110010	0111110010
clock	0	
start	1	





BestDist[7:0]	00000000	00000000
motionX[3:0]	0000	0000
motionY[3:0]	1000	1000
R[7:0]	00000000	00000000
S1[7:0]	00000000	00000000
S2[7:0]	00000000	00000000
AddressR[7:0]	11110010	11110010
AddressS1[9:0]	1000000010	1000000010
AddressS2[9:0]	0111110010	0111110010
clock	0	
start	1	

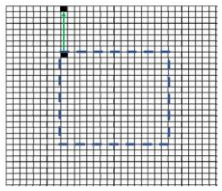
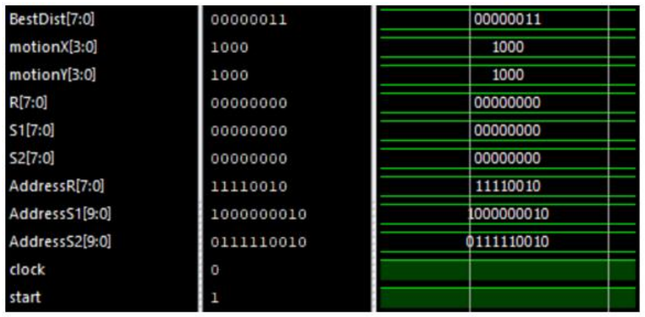


Figure 11 - Shifted Image

By shifting the pixel from Figure 10 to the right, we notice changes in the motionX vector, though no change is evident in motionY. This is analogous to moving the pixel first to Address 8, then to Address 9 in the search memory file (Assuming origin is Address 1). One can verify the correctness of both examples by simply counting the spaces the pixel has moved. I.E, the example on the left has a motion vector of [-1, -8] whereas the right has [0,8]. Additionally, distortion remains at 0 correctly suggesting that a frame shift does not imply distortion.



BestDist[7:0]	00000011	00000011
motionX[3:0]	1000	1000
motionY[3:0]	1000	1000
R[7:0]	00000000	00000000
S1[7:0]	00000000	00000000
S2[7:0]	00000000	00000000
AddressR[7:0]	11110010	11110010
AddressS1[9:0]	1000000010	1000000010
AddressS2[9:0]	0111110010	0111110010
clock	0	
start	1	

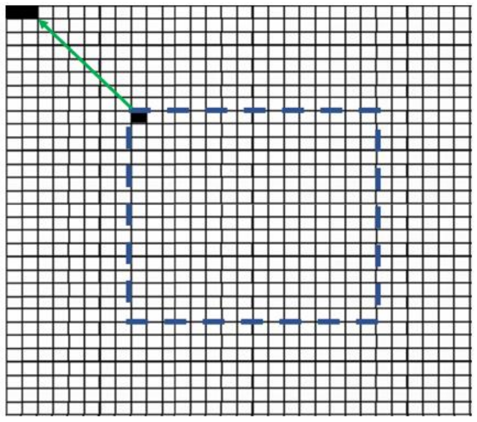


Figure 12 - Image Distortion

Finally, we test how adding an additional pixel affects the Motion Estimation from Figure 10. This is achieved by setting the first two addresses of the Search Memory to the pixel value (11111111). We see from motionX that the motion vector will be drawn to the pixel further from the reference frame. More importantly, we see the pixel closer to the reference frame acting as a blur to the pixel motion. This is captured as an image distortion value of 3, allowing us to determine how similar the two frames are. If we were to add more pixels to the search frame, we would see this distortion value increase further. Changing the darkness of the pixel will also have an effect on distortion when the two pixels are of different shades.

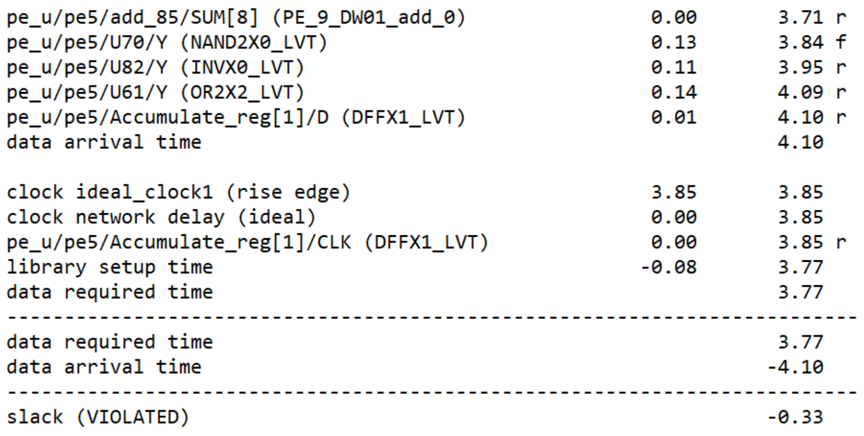
Synthesis: Having validated our design using both waveform simulation and emulation, we are ready to begin synthesizing our RTL design into a gate-level netlist using Design Compiler and Design Vision. From this step onwards, most of the process will be completely automated with scripts aside from a couple of specifications we will have to set. For instance, we will set our clock period to be 3.846 ns pertaining to a clock frequency of 260MHz. We were also asked to implement no clock delay, so we specify this as well.

```
elaborate   -architecture verilog -library WORK MotionEstimator


## Check if design is consistent
check_design  > reports/synth_check_design.rpt

## Create Constraints
create_clock clock -name ideal_clock1 -period 3.846
set_input_delay 0 [ remove_from_collection [all_inputs] clock ] -clock ideal_clock1
set_output_delay 0 [all_outputs ] -clock ideal_clock1

set_max_area 0
```

Figure 13 - Synthesis Constraints

From synthesis, we will receive two files that will be crucial to moving onto the physical design. The files we need to extract are the gate-level netlist and the .sdc timing constraints file.

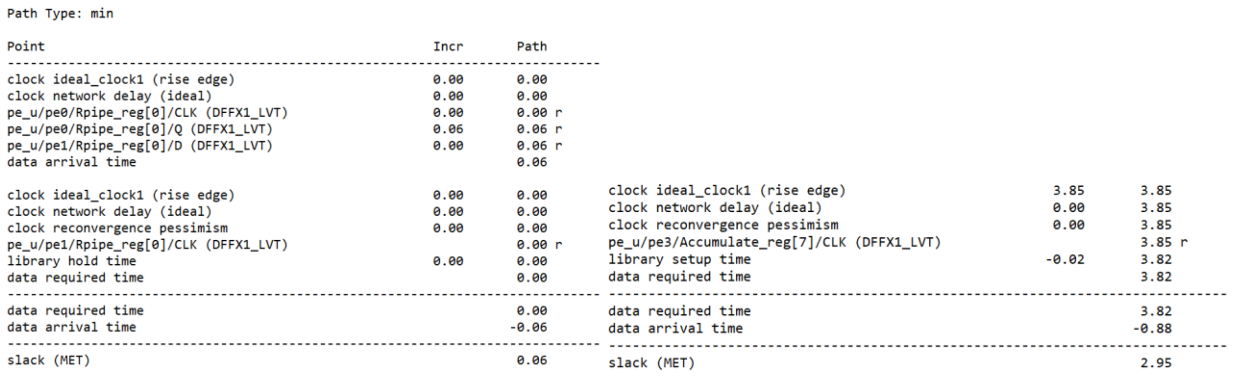
We can use Design Vision or Design Compiler to test the timing of our netlist. While our setup slack meets the required time to prevent a setup violation, our hold time is violated as shown in Figure 14.



```
pe_u/pe5/add_85/SUM[8] (PE_9_DW01_add_0)          0.00       3.71 r
pe_u/pe5/U70/Y (NAND2X0_LVT)                      0.13       3.84 f
pe_u/pe5/U82/Y (INVX0_LVT)                        0.11       3.95 r
pe_u/pe5/U61/Y (OR2X2_LVT)                        0.14       4.09 r
pe_u/pe5/Accumulate_reg[1]/D (DFFX1_LVT)          0.01       4.10 r
data arrival time                                            4.10

clock ideal_clock1 (rise edge)                    3.85       3.85
clock network delay (ideal)                       0.00       3.85
pe_u/pe5/Accumulate_reg[1]/CLK (DFFX1_LVT)        0.00       3.85 r
library setup time                               -0.08       3.77
data required time                                           3.77
-------------------------------------------------------------------------
data required time                                           3.77
data arrival time                                           -4.10
-------------------------------------------------------------------------
slack (VIOLATED)                                            -0.33
```

Figure 14 - Design Vision Hold Time Violation

However, using the Prime Time tool to perform static timing analysis on our Motion Estimator suggests that both hold and setup times are met.



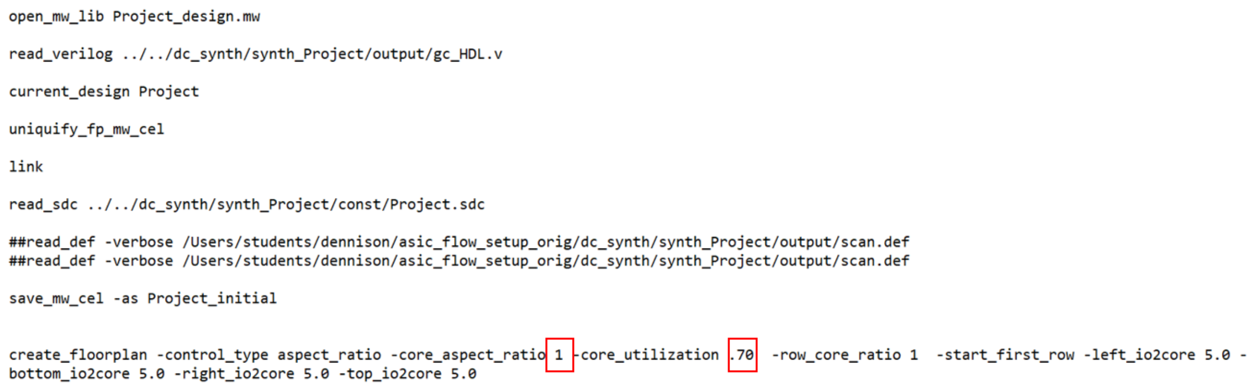
```
Path Type: min

Point                                             Incr       Path
---------------------------------------------------------------------------
clock ideal_clock1 (rise edge)                    0.00       0.00
clock network delay (ideal)                       0.00       0.00
pe_u/pe0/Rpipe_reg[0]/CLK (DFFX1_LVT)             0.00       0.00 r
pe_u/pe0/Rpipe_reg[0]/Q (DFFX1_LVT)               0.06       0.06 r
pe_u/pe1/Rpipe_reg[0]/D (DFFX1_LVT)               0.00       0.06 r
data arrival time                                            0.06

clock ideal_clock1 (rise edge)                    0.00       0.00
clock network delay (ideal)                       0.00       0.00
clock reconvergence pessimism                     0.00       0.00
pe_u/pe1/Rpipe_reg[0]/CLK (DFFX1_LVT)                        0.00 r
library hold time                                 0.00       0.00
data required time                                           0.00
---------------------------------------------------------------------------
data required time                                           0.00
data arrival time                                           -0.06
---------------------------------------------------------------------------
slack (MET)                                                  0.06
```

```
clock ideal_clock1 (rise edge)                    3.85       3.85
clock network delay (ideal)                       0.00       3.85
clock reconvergence pessimism                     0.00       3.85
pe_u/pe3/Accumulate_reg[7]/CLK (DFFX1_LVT)                   3.85 r
library setup time                               -0.02       3.82
data required time                                           3.82
---------------------------------------------------------------------------
data required time                                           3.82
data arrival time                                           -0.88
---------------------------------------------------------------------------
slack (MET)                                                  2.95
```

Figure 15 – Post Synthesis Primetime Hold (Left) and Setup (Right) Static Timing Analysis

Either way, we may proceed with the design flow as our set-up slack surpasses the required time. If there is a Hold Violation, this can be easily remedied by placing buffers to eliminate the chances of race-through.

Physical Design: Now we will begin designing the actual layout of our chip by using ICC Compiler to perform Floorplanning, Placement, Clock tree synthesis, and Routing, though these steps will be automated by a single script. However, we will specify some constraints for the physical dimensions of our chip as shown below.



```
open_mw_lib Project_design.mw

read_verilog ../../dc_synth/synth_Project/output/gc_HDL.v

current_design Project

uniquify_fp_mw_cel

link

read_sdc ../../dc_synth/synth_Project/const/Project.sdc

##read_def -verbose /Users/students/dennison/asic_flow_setup_orig/dc_synth/synth_Project/output/scan.def
##read_def -verbose /Users/students/dennison/asic_flow_setup_orig/dc_synth/synth_Project/output/scan.def

save_mw_cel -as Project_initial


create_floorplan -control_type aspect_ratio -core_aspect_ratio 1 -core_utilization .70  -row_core_ratio 1  -start_first_row -left_io2core 5.0 -
bottom_io2core 5.0 -right_io2core 5.0 -top_io2core 5.0
```

Figure 16 - Physical Constraints

Our chip will be square shaped as indicated by our aspect ratio of 1. More importantly, we will be specifying a utilization of 70%. If we specify a utilization that is too low, the ICC Compiler will not take full advantage of the entire area of the chip. However, using a utilization percentage that is too high may result in the ICC compiler not being able to place any buffers during clock tree synthesis to eliminate any possible hold violations. If we fail to meet timing constraints during post-layout timing analysis, we will likely change the utilization percentage used here rather than returning to the RTL code.

After performing the Physical Design, our final chip will appear as shown in Figure 17.

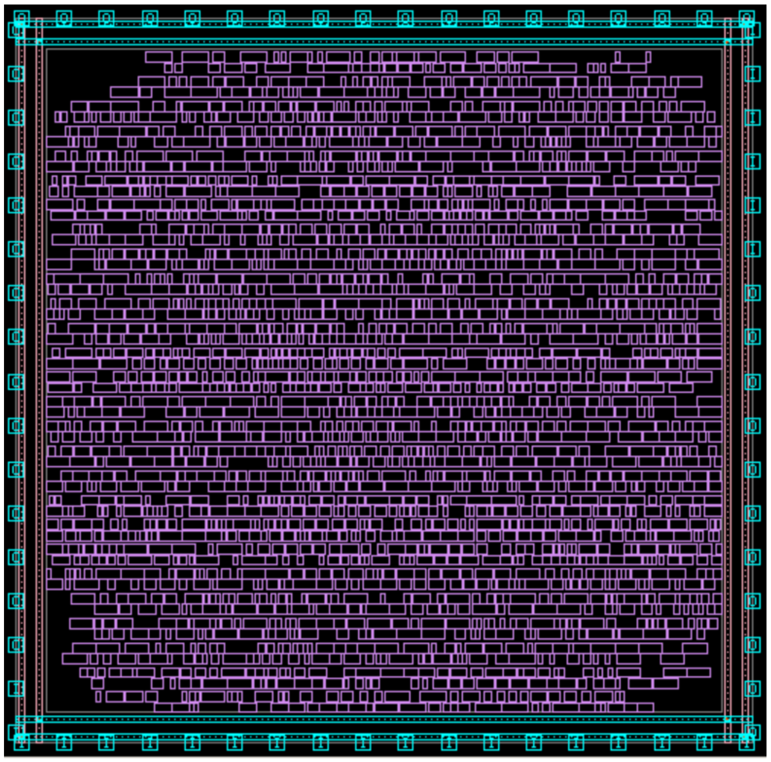
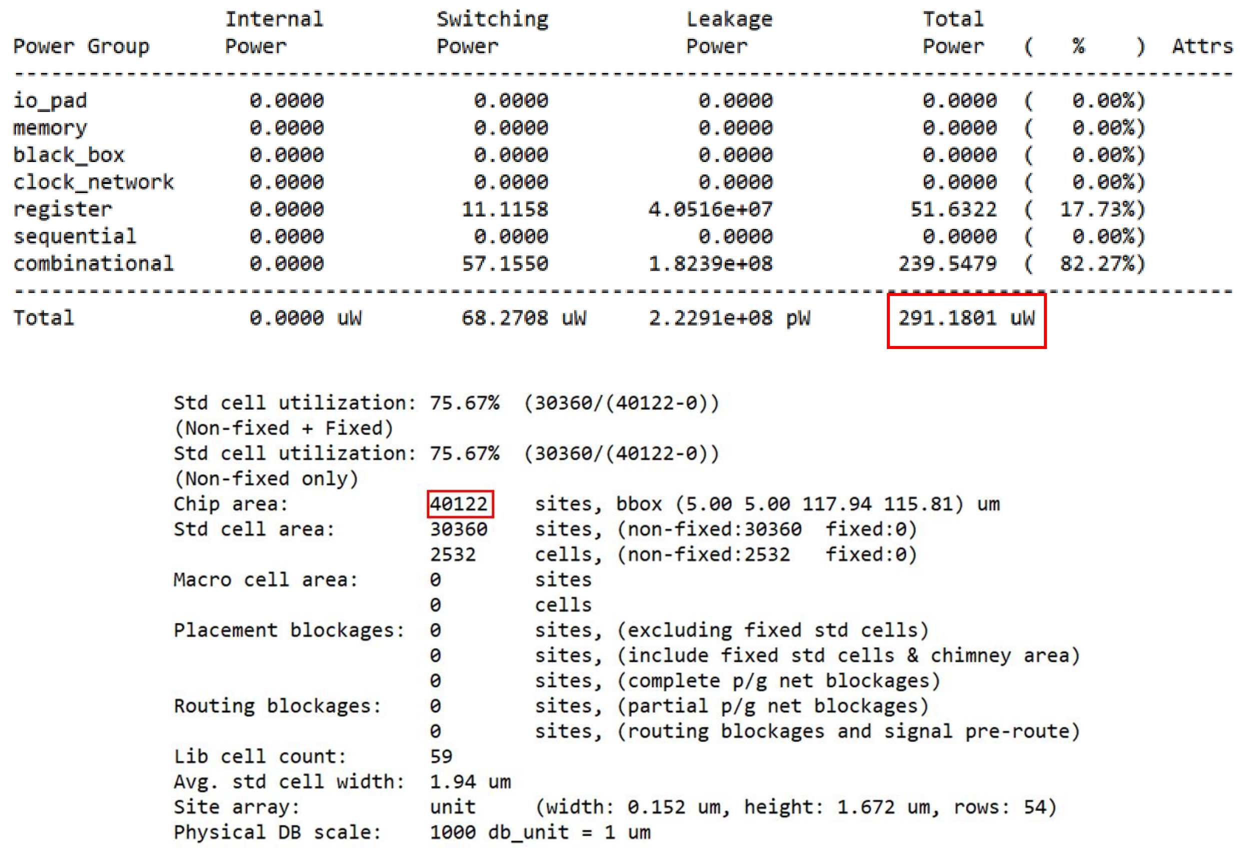


Figure 17 - Physical Chip

Power and Area reports are shown in Figure 18. Note that memory does not dissipate any power at all, confirming that the memory is indeed external to our design.

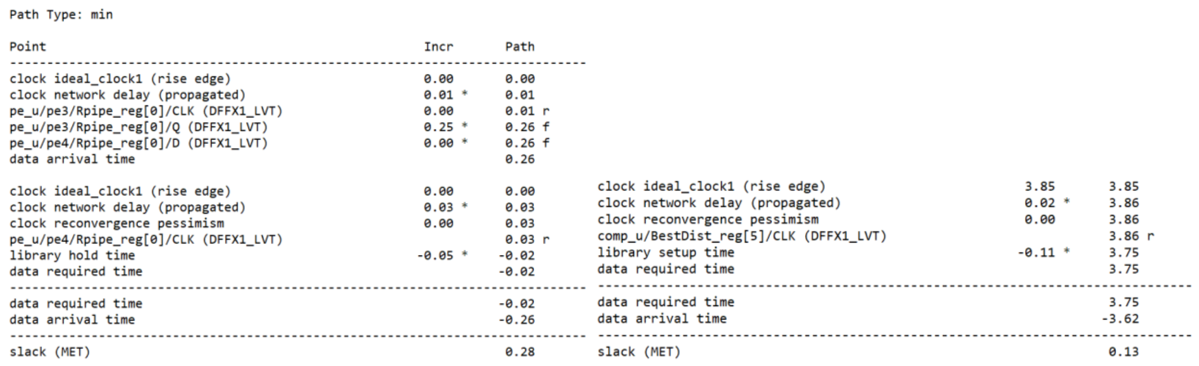


```
                  Internal         Switching          Leakage           Total
Power Group       Power            Power              Power             Power   (   %    )  Attrs
-------------------------------------------------------------------------------------------------
io_pad               0.0000           0.0000             0.0000            0.0000  (   0.00%)
memory               0.0000           0.0000             0.0000            0.0000  (   0.00%)
black_box            0.0000           0.0000             0.0000            0.0000  (   0.00%)
clock_network        0.0000           0.0000             0.0000            0.0000  (   0.00%)
register             0.0000          11.1158         4.0516e+07           51.6322  (  17.73%)
sequential           0.0000           0.0000             0.0000            0.0000  (   0.00%)
combinational        0.0000          57.1550         1.8239e+08          239.5479  (  82.27%)
-------------------------------------------------------------------------------------------------
Total                0.0000 uW       68.2708 uW      2.2291e+08 pW      291.1801 uW
```

```
Std cell utilization: 75.67%  (30360/(40122-0))
(Non-fixed + Fixed)
Std cell utilization: 75.67%  (30360/(40122-0))
(Non-fixed only)
Chip area:            40122    sites, bbox (5.00 5.00 117.94 115.81) um
Std cell area:        30360    sites, (non-fixed:30360  fixed:0)
                      2532     cells, (non-fixed:2532   fixed:0)
Macro cell area:      0        sites
                      0        cells
Placement blockages:  0        sites, (excluding fixed std cells)
                      0        sites, (include fixed std cells & chimney area)
                      0        sites, (complete p/g net blockages)
Routing blockages:    0        sites, (partial p/g net blockages)
                      0        sites, (routing blockages and signal pre-route)
Lib cell count:       59
Avg. std cell width:  1.94 um
Site array:           unit     (width: 0.152 um, height: 1.672 um, rows: 54)
Physical DB scale:    1000 db_unit = 1 um
```

Figure 18 - Power (Top) and Area (Bottom) Physical Design Reports

Sign-Off: Because buffers may have been added to long wires as well as to prevent hold violations, we must reverify the gate-level netlist using Prime Time. Additionally, how fast our chip will perform is also based off of the RC Parasitics that we have extracted during layout. Using a similar script as from pre-layout analysis, we obtain the following timing reports.

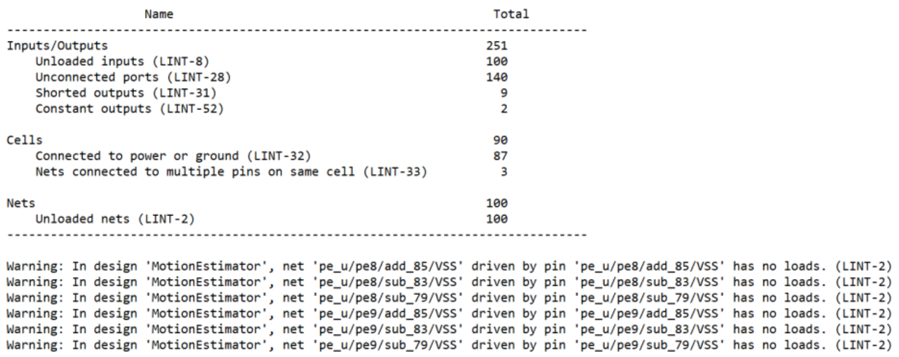


```
Path Type: min

Point                                              Incr       Path
-------------------------------------------------------------------
clock ideal_clock1 (rise edge)                     0.00       0.00
clock network delay (propagated)                   0.01 *     0.01
pe_u/pe3/Rpipe_reg[0]/CLK (DFFX1_LVT)              0.00       0.01 r
pe_u/pe3/Rpipe_reg[0]/Q (DFFX1_LVT)                0.25 *     0.26 f
pe_u/pe4/Rpipe_reg[0]/D (DFFX1_LVT)                0.00 *     0.26 f
data arrival time                                             0.26

clock ideal_clock1 (rise edge)                     0.00       0.00
clock network delay (propagated)                   0.03 *     0.03
clock reconvergence pessimism                      0.00       0.03
pe_u/pe4/Rpipe_reg[0]/CLK (DFFX1_LVT)                         0.03 r
library hold time                                 -0.05 *    -0.02
data required time                                           -0.02
-------------------------------------------------------------------
data required time                                           -0.02
data arrival time                                            -0.26
-------------------------------------------------------------------
slack (MET)                                                   0.28
```

```
clock ideal_clock1 (rise edge)                     3.85       3.85
clock network delay (propagated)                   0.02 *     3.86
clock reconvergence pessimism                      0.00       3.86
comp_u/BestDist_reg[5]/CLK (DFFX1_LVT)                        3.86 r
library setup time                                -0.11 *     3.75
data required time                                            3.75
-------------------------------------------------------------------
data required time                                            3.75
data arrival time                                            -3.62
-------------------------------------------------------------------
slack (MET)                                                   0.13
```

Figure 19 - Post Synthesis Primetime Hold (Left) and Setup (Right) Static Timing Analysis

Now we will perform a final Design Rule Check (DRC) of our chip to make sure it is following the specified rulesets. In Figure 20, we see several warnings, but no errors, indicating that our physical design is not violating any physical constraints such as space between wires, etc.



```
                  Name                                          Total
---------------------------------------------------------------------------------
Inputs/Outputs                                                   251
    Unloaded inputs (LINT-8)                                     100
    Unconnected ports (LINT-28)                                  140
    Shorted outputs (LINT-31)                                      9
    Constant outputs (LINT-52)                                     2

Cells                                                             90
    Connected to power or ground (LINT-32)                        87
    Nets connected to multiple pins on same cell (LINT-33)         3

Nets                                                             100
    Unloaded nets (LINT-2)                                       100
---------------------------------------------------------------------------------

Warning: In design 'MotionEstimator', net 'pe_u/pe8/add_85/VSS' driven by pin 'pe_u/pe8/add_85/VSS' has no loads. (LINT-2)
Warning: In design 'MotionEstimator', net 'pe_u/pe8/sub_83/VSS' driven by pin 'pe_u/pe8/sub_83/VSS' has no loads. (LINT-2)
Warning: In design 'MotionEstimator', net 'pe_u/pe8/sub_79/VSS' driven by pin 'pe_u/pe8/sub_79/VSS' has no loads. (LINT-2)
Warning: In design 'MotionEstimator', net 'pe_u/pe9/add_85/VSS' driven by pin 'pe_u/pe9/add_85/VSS' has no loads. (LINT-2)
Warning: In design 'MotionEstimator', net 'pe_u/pe9/sub_83/VSS' driven by pin 'pe_u/pe9/sub_83/VSS' has no loads. (LINT-2)
Warning: In design 'MotionEstimator', net 'pe_u/pe9/sub_79/VSS' driven by pin 'pe_u/pe9/sub_79/VSS' has no loads. (LINT-2)
```

Figure 20 - DRC Check

Having passed post-layout timing analysis and DRC Check, we have now completed sign-off and are ready for chip fabrication.

Discussion: While the verification stage of our design process was relatively seamless, we did make one error during Synthesis. Having failed hold timing analysis right after synthesis, we mistakenly assumed that we had to lower the clock frequency in order to meet timing constraints. While a setup violation would be quite hard to fix as such a violation occurs from slow and abundant logic in a data path, a hold violation occurs simply because logic is too fast, meaning that additional logic in the form of buffers can easily be inserted into the violating data paths in order to meet the hold time. This is in fact, one of the reasons why we perform clock tress synthesis. By slowing down the clock frequency, we may be alleviating the hold violation, however we are also missing our speed requirement of 15 frames/second. Such a decrease in frequency would require this loss in speed to be made up for in additional PE modules. Fortunately, we realized our mistake and went ahead with the design flow even though we had failed synthesis hold timing. As expected, Clock Tree Synthesis inserted enough buffers to eliminate the hold violation while still meeting setup requirements.

Conclusion: By following the ASIC Design Flow, we were able to successfully implement a Motion Estimator capable of drastically reducing video compression time using the Block-Matching Algorithm. The RTL Code we had design showed successful results using both Waveform Simulation and FPGA Emulation. By the end of this project we had designed a fully operational low power/area chip capable of performing the task we were assigned. More importantly, we have familiarized ourselves with all of the basic fundamentals behind Hardware design and its steps.