SIEMENS EDA

Edge Detect Lab #4 Optimizing the Memory Architecture

Lab Workbook

Catapult 10.6a

SIEMENS

Table of Contents

Lab4 – Optimizing the Memory Architecture 3

Before you Begin .... **Error! Bookmark not defined.**

Objectives 3

Verify the Synthesizable Model 3

Synthesis of the 3-process Design 4

Optimizing the Code to Use Single Port RAM for Shifting and Circular Buffer Implementations 7

Lab4 – Optimizing the Memory Architecture

Before you Begin

Do the following after logging into the AWS Virtual Machine (VM):

- Open a terminal by right-clicking on the desktop and select “Terminal”
- Enter “ls catapult_edge_detect”
    - If the folder is already there you have already copied the files over and can proceed to the rest of the lab
- Copy the lab files to the user home directory by entering the following at the terminal prompt:
    - cp /project/catapult_edge_detect_106.tar .
    - tar -xvf catapult_edge_detect_106.tar
- Source the setup script in the terminal
    - Type “source /project/setup.sh” and hit return

Objectives

Re-profiling the Edge Detect design after accelerating the Magnitude/Angle calculation into hardware showed that the performance bottleneck moved to the horizontal and vertical derivative calculations.

The objectives of this lab are:

- Review the horizontal and vertical derivative C++ code that has been optimized for HLS
- Synthesizes the horizontal and vertical derivatives along with the MagnitudeAngle computation into a design with three concurrent processes
- Verify the design using SCVerify automated verification
- Enhance the architecture for the vertical derivative to use only single port memories
- Enhance the architecture for the vertical derivative to use a circular buffer implementation for the line buffers to reduce power
- Perform power analysis in Catapult to measure the power consumption of the single port and circular buffer architectures

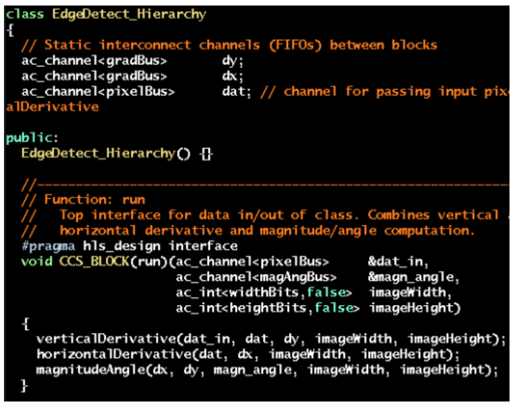
Verify the Synthesizable Model

1. Open a terminal and CD to catapult_edge_detect/system_designs/labs/lab4 directory
2. Type “make run” and click Enter. This will compile and execute the testbench which runs the original Edge Detect algorithm and the bit-accurate synthesizable model and calculates the error difference between the two
    a. Note that the percentage error is small
3. You can also visually compare the results
    a. In the terminal type “display ../image/people_gray.bmp&” and hit Enter, this is the original image

b. In the terminal type “display orig1.bmp&” and hit Enter, this is the magnitude of the algorithm
c. In the terminal type “display ba.bmp&” and hit Enter, this is the magnitude of the synthesizable model. Note they look pretty much the same
d. Close all the images

Synthesis of the 3-process Design

4. Launch Catapult by typing “catapult” in the terminal
5. Source the Catapult initial synthesis script by selecting File > Run Script and select the go_hls.tcl file and click OK.
   a. This will set the technology library to a Nangate sample library that ships with catapult, set the clock frequency to 333 MHz, and synthesize the design constrained to achieve a performance of ~4 pixels/clock cycle throughput
2. While the design is synthesizing (takes a couple of minutes) go back to the Linux terminal and open EdgeDetect_Hierarchy.h in an editor (emacs and VI are available)
   a. Scroll through the code and note that the synthesizable design not only contains the MagnitudeAngle calculation but also the derivatives
   b. Note that the EdgeDetect_Hierarchy class has a public member function that calls the vertical, horizontal, and magnitudeAngle functions. This function has a #pragma hls_design interface which tells catapult to synthesize function interface variables as hardware interfaces to the outside world
   c. Note that these member functions are interconnected using ac_channel interconnect which will allow data to stream between the processes

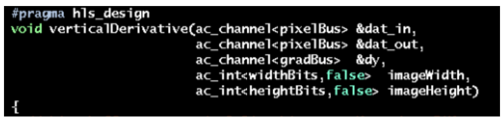


```
class EdgeDetect_Hierarchy
{
  // Static interconnect channels (FIFOs) between blocks
  ac_channel<gradBus>       dy;
  ac_channel<gradBus>       dx;
  ac_channel<pixelBus>      dat; // channel for passing input pix
alDerivative

public:
  EdgeDetect_Hierarchy() {}

  //---------------------------------------------------------------
  // Function: run
  //   Top interface for data in/out of class. Combines vertical
  //   horizontal derivative and magnitude/angle computation.
  #pragma hls_design interface
  void CCS_BLOCK(run)(ac_channel<pixelBus>     &dat_in,
                      ac_channel<magAngBus>    &magn_angle,
                      ac_int<widthBits,false>  imageWidth,
                      ac_int<heightBits,false> imageHeight)
  {
    verticalDerivative(dat_in, dat, dy, imageWidth, imageHeight);
    horizontalDerivative(dat, dx, imageWidth, imageHeight);
    magnitudeAngle(dx, dy, magn_angle, imageWidth, imageHeight);
  }
```

   d. Scroll down and look at the verticalDerivative member function
   e. Note that it is defined with a #pragma hls_design. This tells catapult to push the member function into a separate concurrent process. Note that this coding style requires ac_channel variables to be used to interconnect the member functions

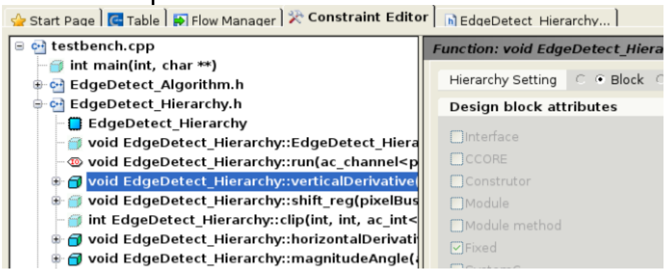


```
#pragma hls_design
void verticalDerivative(ac_channel<pixelBus> &dat_in,
                        ac_channel<pixelBus> &dat_out,
                        ac_channel<gradBus>  &dy,
                        ac_int<widthBits,false>  imageWidth,
                        ac_int<heightBits,false> imageHeight)
{
```

   f. Scroll through the rest of the code and note that the vertical and horizontal derivatives have be coded almost identical to the code covered in the presentation. The one major difference is that the presentation material worked on input data that was a single pixel wide. This design has 4 input pixels read in parallel. The vertical derivative code is mostly the same and just uses a struct to pass 4 pixels. However, the horizontal derivative is slightly different

    because 4 pixels read in parallel requires shifting by 4, and the horizontal sliding window implementation is slightly different than what was covered in the presentation
  g. Note that the magnitudeAngle function is the same as the last example in Lab2

3. Return to Catapult, the synthesis run should be finished by now
4. Look at the table view and note that the design takes ~520,000 cycles to process a 1920x1080 image, which is the same throughput that was achieved on the previous standalone MagnitudeAngle design
6. Click on the Hierarchy Icon in the Task Bar
   a. Note that the 3 functions, verticalDerivative, horizontalDerivative, and magnitudeAngle are all shown as design blocks. This means that Catapult will synthesize them as separate concurrent processes which will allow them to run in parallel
   b. Double-click on any of the 3 design blocks to cross-probe back to the C++ source. Note the #pragma hls_design before the functions. This told Catapult to synthesize the functions as concurrent processes



Start Page | Table | Flow Manager | Constraint Editor | EdgeDetect_Hierarchy...

```
testbench.cpp
  int main(int, char **)
EdgeDetect_Algorithm.h
EdgeDetect_Hierarchy.h
  EdgeDetect_Hierarchy
  void EdgeDetect_Hierarchy::EdgeDetect_Hiera
  void EdgeDetect_Hierarchy::run(ac_channel<p
  void EdgeDetect_Hierarchy::verticalDerivative(
  void EdgeDetect_Hierarchy::shift_reg(pixelBus
  int EdgeDetect_Hierarchy::clip(int, int, ac_int<
  void EdgeDetect_Hierarchy::horizontalDerivati
  void EdgeDetect_Hierarchy::magnitudeAngle(
```

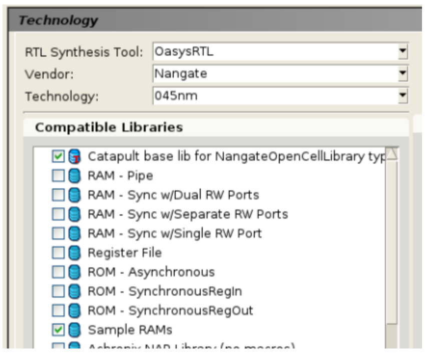
Function: void EdgeDetect_Hiera

Hierarchy Setting [✓ Block]

Design block attributes

- ☐ Interface
- ☐ CCORE
- ☐ Construtor
- ☐ Module
- ☐ Module method
- ☑ Fixed

7. Click on Libraries in the Task Bar and note that the Sample Ram library has been included. This will allow C++ arrays to be mapped to memories



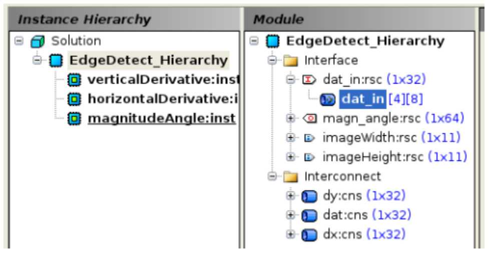
Technology

RTL Synthesis Tool:	OasysRTL
Vendor:	Nangate
Technology:	045nm

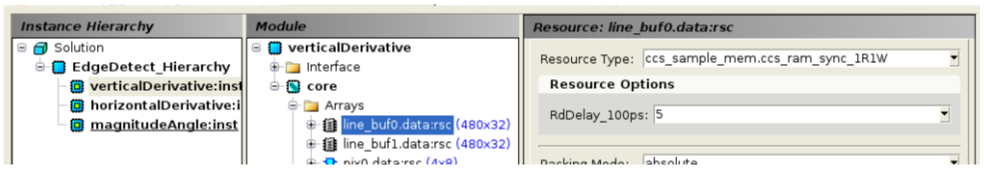
Compatible Libraries

- ☑ Catapult base lib for NangateOpenCellLibrary typ
- ☐ RAM - Pipe
- ☐ RAM - Sync w/Dual RW Ports
- ☐ RAM - Sync w/Separate RW Ports
- ☐ RAM - Sync w/Single RW Port
- ☐ Register File
- ☐ ROM - Asynchronous
- ☐ ROM - SynchronousRegIn
- ☐ ROM - SynchronousRegOut
- ☑ Sample RAMs

8. Click on Architecture in the Task Bar
   a. Note there are three design blocks shown in the instance hierarchy for the three member functions
   b. Expand the Interface and Interconnect Folders. The Interface folder shows the ports for the top-level design. These can be expanded "+" and cross-probed to the C++ source by double-clicking on the variable under the resource
   c. The Interconnect Folder shows the ac_channel interconnect that was used to "stitch" the three functions together. These channels can be constrained to be either wire or FIFO interconnect



d. Click on the verticalDerivative Icon in the Instance Hierarchy Pane and expand the Arrays folder. This shows the C++ arrays in the design and allows them to be constrained to registers or memories

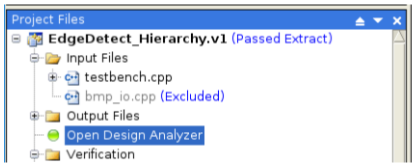


Resource Type: ccs_sample_mem.ccs_ram_sync_1R1W

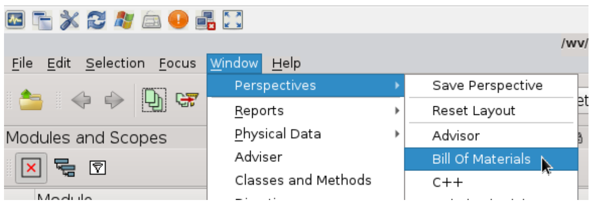
RdDelay_100ps: 5

e. Click on line_buf0.data:rsc, this is one of the line buffer arrays from the verticalDerivative function. The grey chip icon indicates that this array has been mapped to a memory. The Resource Type to the right indicate that it has been mapped to a memory with one read and one write port

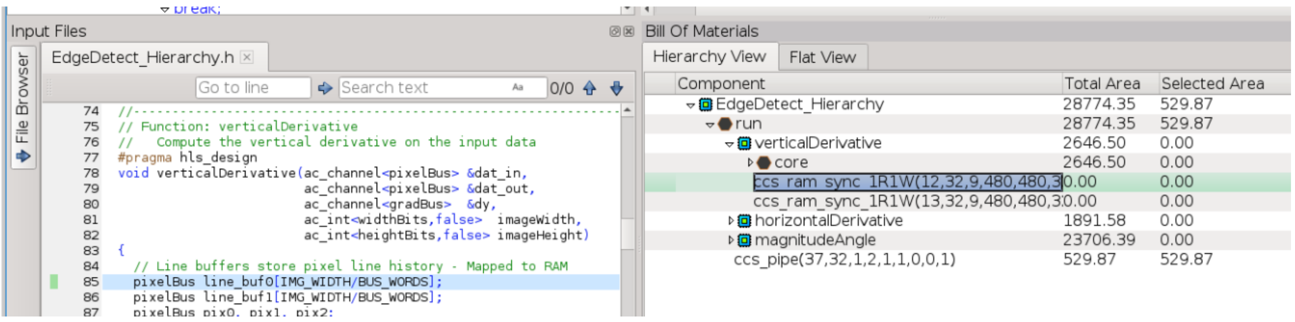
9. Double-click on Open Design Analyzer in the Project Files View



10. In Design Analyzer go to Window > Perspectives and select Bill of Materials. This will display an interactive BOM. Design Analyzer has a number of pre-built perspectives for viewing the design



11. Expand the BOM into the verticalGradient Block by clicking on the triangle next to the icons
    a. Note that it shows two memories
    b. Click on either of the memories and note that it highlights the linebuffer arrays in the C++ source



```
    break;
```

Input Files

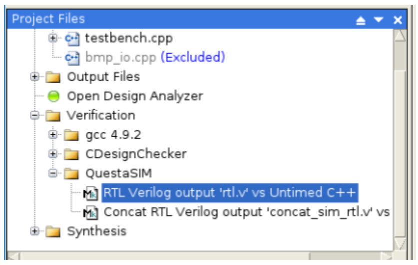
EdgeDetect_Hierarchy.h

```
//--------------------------------------------------------------------
// Function: verticalDerivative
//   Compute the vertical derivative on the input data
#pragma hls_design
void verticalDerivative(ac_channel<pixelBus> &dat_in,
                        ac_channel<pixelBus> &dat_out,
                        ac_channel<gradBus>  &dy,
                        ac_int<widthBits,false>  imageWidth,
                        ac_int<heightBits,false> imageHeight)
{
  // Line buffers store pixel line history - Mapped to RAM
  pixelBus line_buf0[IMG_WIDTH/BUS_WORDS];
  pixelBus line_buf1[IMG_WIDTH/BUS_WORDS];
  pixelBus pix0, pix1, pix2;
```

Bill Of Materials

Component	Total Area	Selected Area
EdgeDetect_Hierarchy	28774.35	529.87
run	28774.35	529.87
verticalDerivative	2646.50	0.00
core	2646.50	0.00
ccs_ram_sync_1R1W(12,32,9,480,480,3	0.00	0.00
ccs_ram_sync_1R1W(13,32,9,480,480,3	0.00	0.00
horizontalDerivative	1891.58	0.00
magnitudeAngle	23706.39	0.00
ccs_pipe(37,32,1,2,1,1,0,0,1)	529.87	529.87

12. Close Design Analyzer
13. Launch the SCVerify Verification Flow from the Verification > Questasim folder by double-clicking on “RTL Verilog output ‘rtl.v’ vs Untimed C++”



Project Files

```
testbench.cpp
bmp_io.cpp (Excluded)
Output Files
Open Design Analyzer
Verification
  gcc 4.9.2
  CDesignChecker
  QuestaSIM
    RTL Verilog output 'rtl.v' vs Untimed C++
    Concat RTL Verilog output 'concat_sim_rtl.v' vs
Synthesis
```

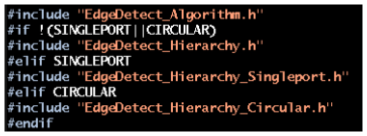
14. Type run -all in the Questasim Transcript
15. Questasim will simulate the design using a 128x128 image (this is done for runtime)
16. Look at the transcript and note that it returns a Simulation PASSED! Message and indicates that the design took 13,946 ns.
    a. 13,946 ns/3.333 clock period = 4184 cycles. A 128x128 image @ 4 pixels/clock = 4096 cycles, so the performance of the hardware is as expected
    b. The slightly slower performance of the hardware than the 4096 cycles is caused by only pipelining the COL loops of the 3 functions plus the memory architecture used introduces one dead cycle per line
    c. Look at the waveform view in Questasim and note that the input is read continuously (din_rsc_rdy is high) with occasional stalls at the end of each line (din_rsc_rdy is low a couple clocks)
    d. Both of these issues could be removed if desired by reconstraining the design and making some code changes
17. Close Catapult

Optimizing the Code to Use Single Port RAM for Shifting and Circular Buffer Implementations

This last section recodes the verticalGradient member function to use single port memories for the line buffers. This is done because using a dual port memory in an ASIC design is only done when absolutely necessary due to the area cost. In addition to recoding to use single port memories, the verticalGradient is re-coded to operate the line buffers in a “shifting” fashion as well as in a “circular” fashion. Then power is analyzed on both implementations

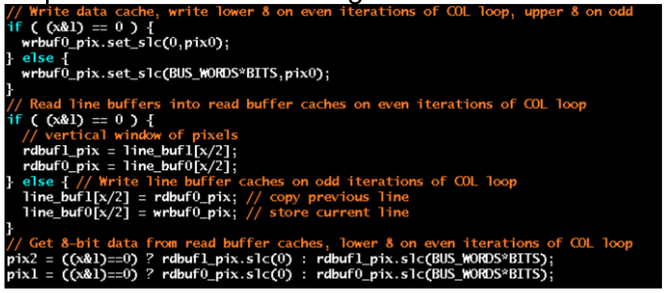
18. Launch Catapult by typing “catapult&” in the terminal

19. Select File > Run Script and select the go_hls_power.tcl file and click OK.
    a. This will set the technology library to a Nangate sample library that ships with catapult, set the clock frequency to 333 MHz, and synthesize the design constrained to achieve a performance of ~4 pixels/clock cycle throughput
    b. This script also adds in a Nangate single port memory library that includes a memory liberty file that will allow us to do power optimization. NOTE this memory library is for demo purposes only
    c. The script synthesizes both the “shifting” and “circular” buffer implementations by using a compiler define to include them in the testbench



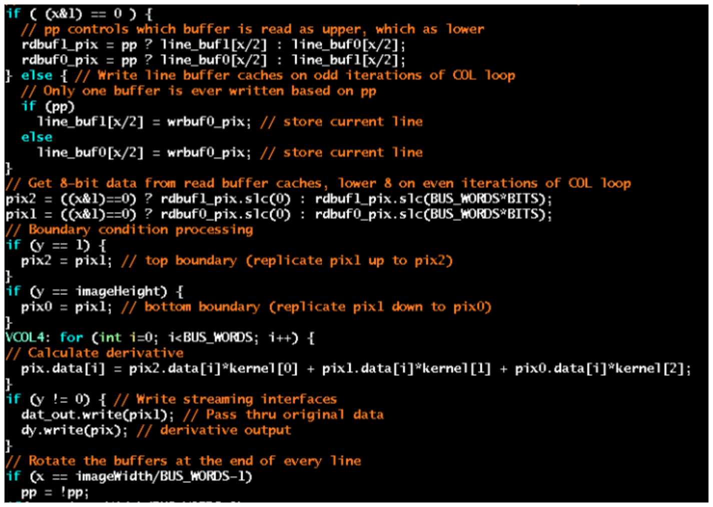
```
#include "EdgeDetect_Algorithm.h"
#if !(SINGLEPORT||CIRCULAR)
#include "EdgeDetect_Hierarchy.h"
#elif SINGLEPORT
#include "EdgeDetect_Hierarchy_Singleport.h"
#elif CIRCULAR
#include "EdgeDetect_Hierarchy_Circular.h"
#endif
```

5. While the design is synthesizing (takes a couple of minutes) go back to the Linux terminal
    a. Both designs can be compiled and executes by typing “make run_sp” or “make run_circ”. Running either implementation will show the same percentage error as the previous dual port memory version
    b. Open EdgeDetect_Hierarchy_Singleport.h in an editor (emacs and VI are available)
    c. Scroll down to the verticalGradient function and inspect the code. Note that it has implemented the architecture covered in the presentation that allows a single port memory to act like a dual port by caching the reads and writes and reading on even iterations of the loop variable “x” and writing on odd iterations of “x”



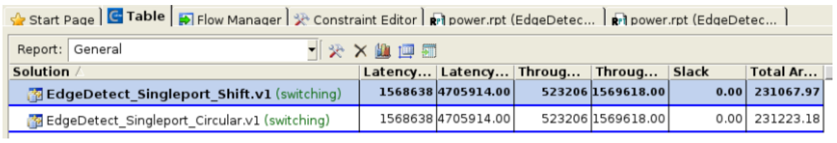
```
// Write data cache, write lower 8 on even iterations of COL loop, upper 8 on odd
if ( (x&1) == 0 ) {
  wrbuf0_pix.set_slc(0,pix0);
} else {
  wrbuf0_pix.set_slc(BUS_WORDS*BITS,pix0);
}
// Read line buffers into read buffer caches on even iterations of COL loop
if ( (x&1) == 0 ) {
  // vertical window of pixels
  rdbuf1_pix = line_buf1[x/2];
  rdbuf0_pix = line_buf0[x/2];
} else { // Write line buffer caches on odd iterations of COL loop
  line_buf1[x/2] = rdbuf0_pix; // copy previous line
  line_buf0[x/2] = wrbuf0_pix; // store current line
}
// Get 8-bit data from read buffer caches, lower 8 on even iterations of COL loop
pix2 = ((x&1)==0) ? rdbuf1_pix.slc(0) : rdbuf1_pix.slc(BUS_WORDS*BITS);
pix1 = ((x&1)==0) ? rdbuf0_pix.slc(0) : rdbuf0_pix.slc(BUS_WORDS*BITS);
```

    d. Open EdgeDetect_Hierarchy_Circular.h in an editor
    e. Scroll down to the verticalGradient function and inspect the code. Note that it has implemented the circular buffer architecture covered in the presentation. Only one buffer is ever written, and the buffers are rotated after each line based on the variable “pp”



```
if ( (x&1) == 0 ) {
  // pp controls which buffer is read as upper, which as lower
  rdbuf1_pix = pp ? line_buf1[x/2] : line_buf0[x/2];
  rdbuf0_pix = pp ? line_buf0[x/2] : line_buf1[x/2];
} else { // Write line buffer caches on odd iterations of COL loop
  // Only one buffer is ever written based on pp
  if (pp)
    line_buf1[x/2] = wrbuf0_pix; // store current line
  else
    line_buf0[x/2] = wrbuf0_pix; // store current line
}
// Get 8-bit data from read buffer caches, lower 8 on even iterations of COL loop
pix2 = ((x&1)==0) ? rdbuf1_pix.slc(0) : rdbuf1_pix.slc(BUS_WORDS*BITS);
pix1 = ((x&1)==0) ? rdbuf0_pix.slc(0) : rdbuf0_pix.slc(BUS_WORDS*BITS);
// Boundary condition processing
if (y == 1) {
  pix2 = pix1; // top boundary (replicate pix1 up to pix2)
}
if (y == imageHeight) {
  pix0 = pix1; // bottom boundary (replicate pix1 down to pix0)
}
VCOL4: for (int i=0; i<BUS_WORDS; i++) {
// Calculate derivative
  pix.data[i] = pix2.data[i]*kernel[0] + pix1.data[i]*kernel[1] + pix0.data[i]*kernel[2];
}
if (y != 0) { // Write streaming interfaces
  dat_out.write(pix1); // Pass thru original data
  dy.write(pix); // derivative output
}
// Rotate the buffers at the end of every line
if (x == imageWidth/BUS_WORDS-1)
  pp = !pp;
```

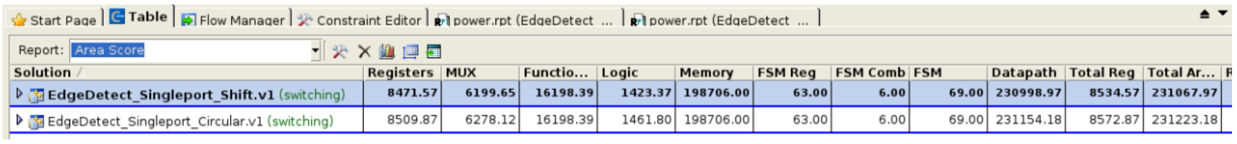
20. Go back to Catapult, it should have finished synthesizing both designs by now
21. Look at the table view and note there are two solutions, EdgeDetect_Singleport_Shift and EdgeDetect_Singleport_Circular
22. Note that the areas are very similar, with the *_Circular being slightly large due to the additional logic needed to rotate the buffers



Report: General

Solution	Latency...	Latency...	Throug...	Throug...	Slack	Total Ar...
EdgeDetect_Singleport_Shift.v1 (switching)	1568638	4705914.00	523206	1569618.00	0.00	231067.97
EdgeDetect_Singleport_Circular.v1 (switching)	1568638	4705914.00	523206	1569618.00	0.00	231223.18

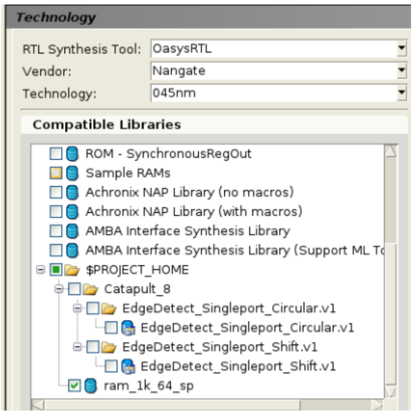
23. Also note that the area is much larger than the previous designs. This is because the single port memory library that was used includes area numbers. The Catapult sample memories used previously don’t include area
24. Click on the drop-down menu in the table view and change the Report to Area Score. Note that the design area is dominated by the single port memory



Report: Area Score

Solution	Registers	MUX	Functio...	Logic	Memory	FSM Reg	FSM Comb	FSM	Datapath	Total Reg	Total Ar...	R
EdgeDetect_Singleport_Shift.v1 (switching)	8471.57	6199.65	16198.39	1423.37	198706.00	63.00	6.00	69.00	230998.97	8534.57	231067.97	
EdgeDetect_Singleport_Circular.v1 (switching)	8509.87	6278.12	16198.39	1461.80	198706.00	63.00	6.00	69.00	231154.18	8572.87	231223.18	

25. Click on Libraries in the Task Bar
26. Scroll down in the Compatible Libraries Window and note that a custom memory library has been included for the single port ram



Technology

RTL Synthesis Tool: OasysRTL

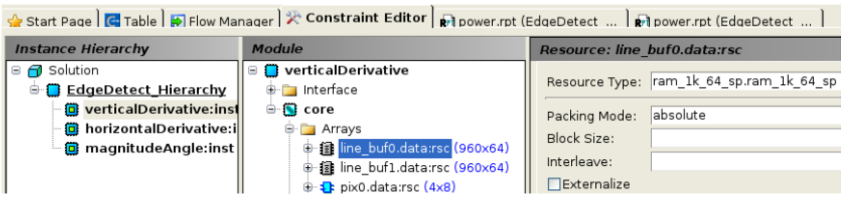
Vendor: Nangate

Technology: 045nm

Compatible Libraries

- ☐ ROM - SynchronousRegOut
- ☐ Sample RAMs
- ☐ Achronix NAP Library (no macros)
- ☐ Achronix NAP Library (with macros)
- ☐ AMBA Interface Synthesis Library
- ☐ AMBA Interface Synthesis Library (Support ML To
- $PROJECT_HOME
  - ☐ Catapult_8
    - ☐ EdgeDetect_Singleport_Circular.v1
      - ☐ EdgeDetect_Singleport_Circular.v1
    - ☐ EdgeDetect_Singleport_Shift.v1
      - ☐ EdgeDetect_Singleport_Shift.v1
  - ✓ ram_1k_64_sp

27. Click on Architecture in the Task Bar
28. Select the verticalGradient block and expand the Interface folder
29. Note that the linebuffer memories have been mapped to the custom single port memory



Instance Hierarchy

- Solution
  - EdgeDetect_Hierarchy
    - verticalDerivative:inst
    - horizontalDerivative:i
    - magnitudeAngle:inst

Module

- verticalDerivative
  - Interface
  - core
    - Arrays
      - line_buf0.data:rsc (960x64)
      - line_buf1.data:rsc (960x64)
      - pix0.data:rsc (4x8)

Resource: line_buf0.data:rsc

Resource Type: ram_1k_64_sp.ram_1k_64_sp

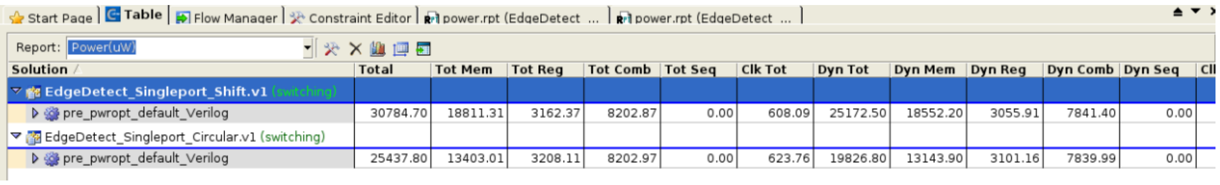
Packing Mode: absolute

Block Size:

Interleave:

☐ Externalize

30. Look at the Task Bar and note that there are two additional Icons for power analysis and optimization. The go_hls_power.tcl script ran the power flow for both solutions
31. Go to the table view and change the report to Power
32. Expand the solutions in the table view to show the power numbers (Note you will need to collapse the inner part of the power report to see the screen shot below)



Report: Power(uW)

Solution	Total	Tot Mem	Tot Reg	Tot Comb	Tot Seq	Clk Tot	Dyn Tot	Dyn Mem	Dyn Reg	Dyn Comb	Dyn Seq	Cl
EdgeDetect_Singleport_Shift.v1 (switching)												
pre_pwropt_default_Verilog	30784.70	18811.31	3162.37	8202.87	0.00	608.09	25172.50	18552.20	3055.91	7841.40	0.00	
EdgeDetect_Singleport_Circular.v1 (switching)												
pre_pwropt_default_Verilog	25437.80	13403.01	3208.11	8202.97	0.00	623.76	19826.80	13143.90	3101.16	7839.99	0.00	

33. Note the difference in power consumption between the two solutions
    a. The register and combinational power numbers are very similar
    b. The memory power consumption is significantly less for the circular buffer implementation
    c. The circular buffer implementation total power is about 16% less than the "shift" solution
34. Close Catapult

Done Lab4