SIEMENS EDA

Edge Detect Lab#2 Synthesis and Optimization of the Magnitude/Angle Accelerator

Lab Workbook

Catapult 10.6a

SIEMENS

Unpublished work. © 2021 Siemens

This material contains trade secrets or otherwise confidential information owned by Siemens Industry Software Inc. or its affiliates (collectively, "SISW"), or its licensors. Access to and use of this information is strictly limited as set forth in Customer's applicable agreement with SISW. This material may not be copied, distributed, or otherwise disclosed outside of Customer's facilities without the express written permission of SISW, and may not be used in any way not expressly authorized by SISW.

This document is for information and instruction purposes. SISW reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult SISW to determine whether any changes have been made. SISW disclaims all warranties with respect to this document including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement of intellectual property.

The terms and conditions governing the sale and licensing of SISW products are set forth in written agreements between SISW and its customers. SISW's **End User License Agreement** may be viewed at: www.plm.automation.siemens.com/global/en/legal/online-terms/index.html.

No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of SISW whatsoever.

TRADEMARKS: The trademarks, logos, and service marks ("Marks") used herein are the property of Siemens or other parties. No one is permitted to use these Marks without the prior written consent of Siemens or the owner of the Marks, as applicable. The use herein of third party Marks is not an attempt to indicate Siemens as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A list of Siemens' trademarks may be viewed at: www.plm.automation.siemens.com/global/en/legal/trademarks.html. The registered trademark Linux® is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis.

Support Center: support.sw.siemens.com
Send Feedback on Documentation: support.sw.siemens.com/doc_feedback_form

Table of Contents

Lab2 – Synthesis and Optimization of the Magnitude/Angle Accelerator 3

Before you Begin 3

Objectives 3

Verify the Synthesizable Model 3

Initial Synthesis 4

Analyzing Performance Bottlenecks 6

Optimizing the Design 6

Taking Advantage of Bus Bandwidth to Increase Performance 8

Lab2 – Synthesis and Optimization of the Magnitude/Angle Accelerator

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

The objectives of this lab are:

- Perform an initial synthesis of the design in Catapult and determine the performance
- Use the Catapult design analysis capabilities to pinpoint performance bottlenecks
- Optimize the design for the initial performance goals
    - Read 1 dx,dy sample/clock throughput or process 1920x1080 image in ~2,073,600 cycles
- Re-run synthesis with modified C++ architecture that reads 4 dx,dy samples/clock. This architecture matches the MagnitudeAngle block IO bandwidth to the RiskV memory system bandwidth. The RiscV bus in this example is 64-bits wide.
- Optimize the 4x architecture to achieve 4 dx,dy samples/clock throughput or process 1920x1080 image in ~518,400 cycles
- Verify the Catapult generated RTL functionality and performance by running the Catapult SCVerify automated verification flow

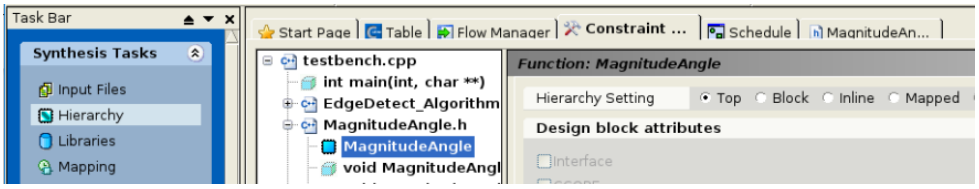
Verify the Synthesizable Model

1. Open a terminal and cd to ~/catapult_edge_detect/system_designs/labs/lab2 directory
2. Type “make run” and click Enter. This will compile and execute the testbench which runs the original 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

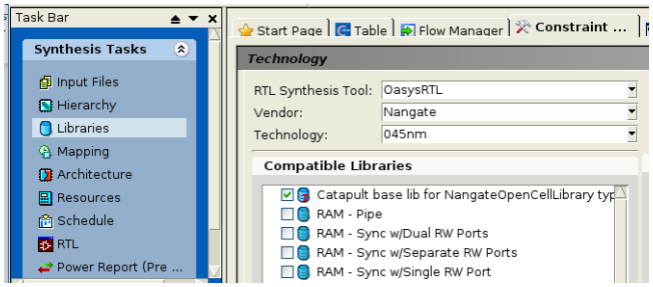
c. In the terminal type “display ba.bmp&” and hit Enter, this is the magnitude of the synthesizable model. Note they look pretty much the same
d. Close all the images

Initial Synthesis

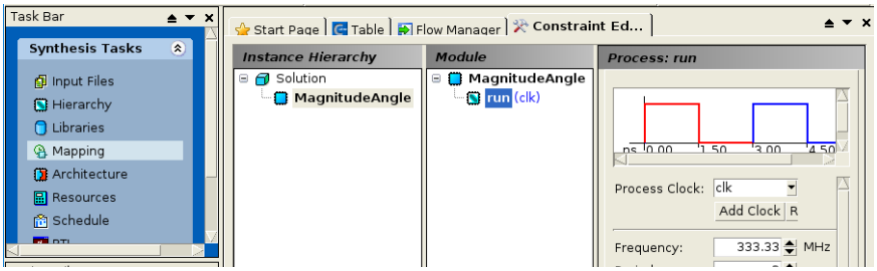
4. Launch Catapult by typing “catapult” in the terminal
5. Source the Catapult initial synthesis script by selecting File > Run Script and select the go_hls.tcl file and click OK.
    a. This will set the technology library to a Nangate sample library that ships with catapult, set the clock frequency to 333 MHz, and synthesize the design unconstrained
6. Click on the Hierarchy Icon in the Task Bar. This is where design blocks can be constrained or mapped in a bottom-up design flow
    a. Note that the Magnitude angle block is shown as the “top” design
    b. Double-click on the MagnitudeAngle chip icon and note that it cross-probes back to the class definition in the C++ source
    c. Note that there is a #pragma hls_design top before the class definition that tells Catapult this is the top-level design



7. Click on Libraries in the Task Bar. This is where the synthesis tool, technology library, and memory libraries are selected

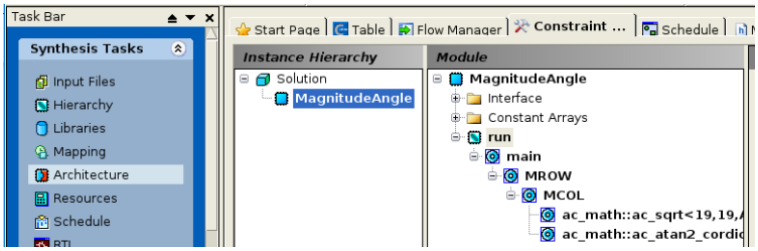


8. Click on Mapping in the Task Bar. This is where the clock frequency is specified
    a. Click on the “run (clk)” icon and note that the clock frequency is set to 333 MHz

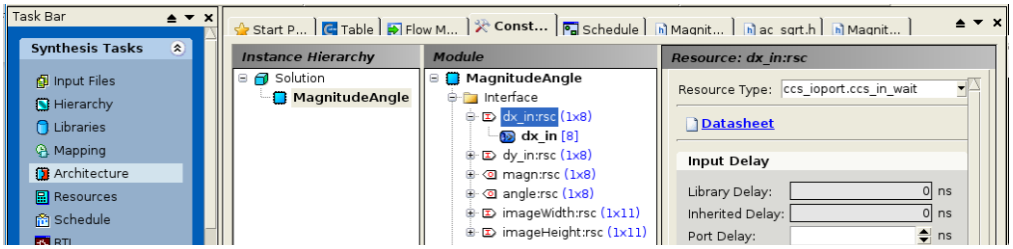


9. Click on Architecture in the Task Bar. This is where design optimization constraints are set. It also provides architectural details about the design.

a. Note that the Architecture View shows the top-level design and the loops in the design
b. Double-click on the “MROW” loop and note that it cross-probes back to the loop in the C++ source with the MROW label. Labeling loops is useful for understanding where things came from in the C++ source. Though it’s not required.



c. Go back to the Architecture View (click on the “Constraint Editor tab) and expand the Interface folder

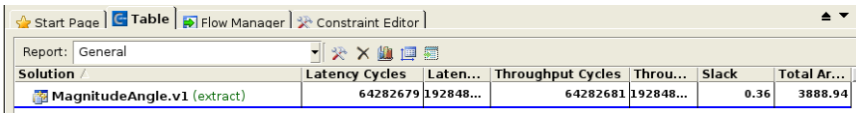


d. Note that it shows hardware interfaces that are similar to the variable names on our C++ design interface. It also shows the interface bit-width in parenthesis
e. Click on dx_in_rsc and note that the Resource Type is mapped to a ccs_in_wait. This is a handshaking interface with a data/ready/valid protocol
f. Click on the “+” for dx_in_rsc to expand it. This shows the C++ variable “dx_in” that is mapped to the interface dx_in_rsc
g. Double-click on the “dx_in” variable and note that it cross-probes back to the ac_channel interface variable in the C++ for dx_in

```
#pragma hls_design interface
void run(ac_channel<ac_fixed<8,8,true>>                    dx_in,
```

h. Go back to the Architecture View and click through the loops, ROW, COL, and loops for the sqrt and atan2. Note that when a loop is selected it shows the loop iterations. Also note that the loops are unconstrained (No loop unrolling or pipelining has been set)

10. Go to the Table View by clicking on the Table Tab in the Catapult GUI
    a. Note that the latency and throughput are both greater than 64 million cycles, which is significantly larger than the design goal of ~518,400 cycles



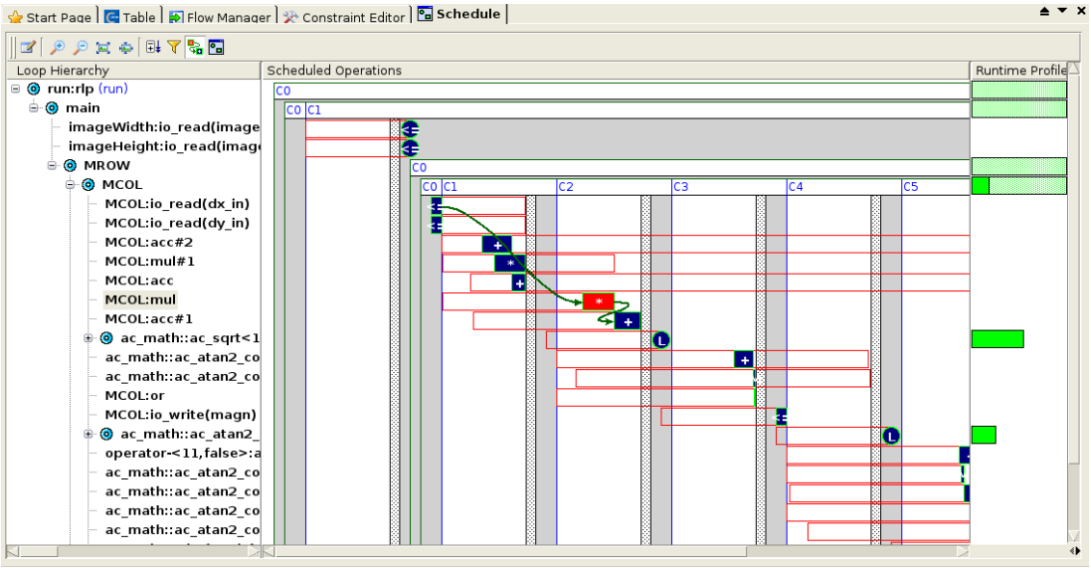
Solution	Latency Cycles	Laten...	Throughput Cycles	Throu...	Slack	Total Ar...
MagnitudeAngle.v1 (extract)	64282679	192848...	64282681	192848...	0.36	3888.94

11. The question that needs to be answered is “Where are the performance bottlenecks in this design, and how do I identify them and optimize the design”?

Analyzing Performance Bottlenecks

The Catapult Gantt Chart provides a view of the scheduled design that shows how much time is being spent within the various loops of the C++ design. This guides the user to know which loops need to be optimized

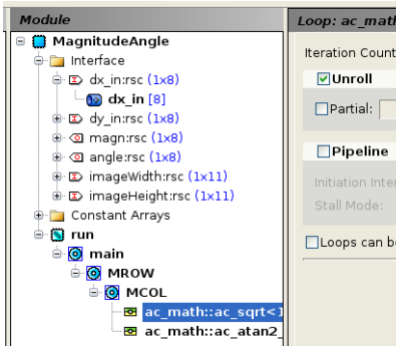
12. Click on Schedule in the Task Bar. This will open the Catapult Gantt Chart.
13. Expand the main, ROW, and COL loops by clicking on the “+” next to the loop label



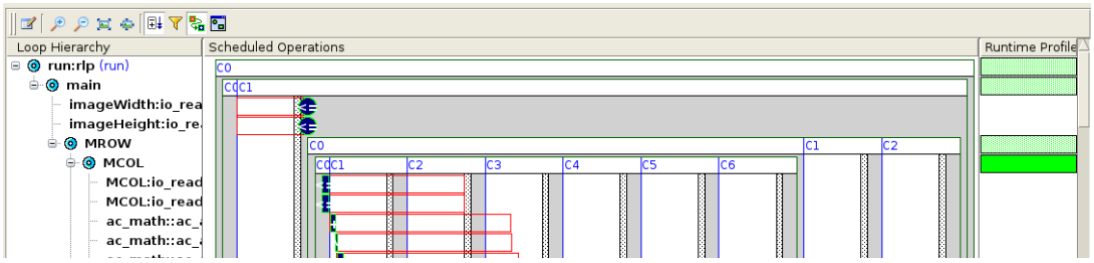
14. Double-click on the multiplier shown above and note how it cross-probes back to a multiply operation in the C++ source
15. Note that the green arrow shows that the multiplier is feeding the adder shown above. Double-click on the adder and note it cross-probes to an adder in the C++ source that adds the results from the multipliers
16. Look at the green bars on the right side of the Gantt Chart. These show the runtime profile of the algorithm as it would execute in hardware. The dark green bars show how much time is spent executing the hardware inside of a loop. The bigger the dark green bar is, the more time is spent executing that loop. What they show is that most of the time is being spent inside of the loops for the sqrt and the atan2 function calls. These loops should be where we begin optimizing the design

Optimizing the Design

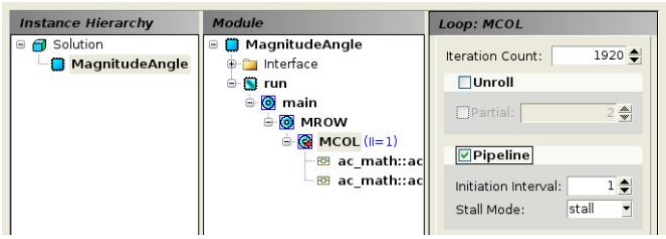
17. Click on Architecture in the Task Bar
18. Unroll both the ac_math::sqrt and ac_math::atan2 loops by selecting each of them and checking the unroll box



19. Click on RTL in the Task Bar
20. Look at the Table View
    a. Note that the design now takes about 12 million cycles to process the image
    b. Note that the design has also increased in area
21. Click on Schedule in the Task Bar and expand the Gantt Chart main, MROW, and MCOL loops
22. Zoom out full on the Gantt Chart. There are Zoom Tools at the top left of the Gantt Chart



23. Look at the Runtime Profile (Big green bar) for the MCOL loop and note that it indicates about 100% of the time is being spent executing the MCOL loop
24. Note that there are 6 C-steps, C1, C2,…C6, in the loop body
    a. These 6 c-steps are essentially the data path of the design
    b. This means that each loop body takes 6 clock cycles to execute
    c. The MCOL loop has 1920 iterations and MROW is 1080. So 1080x1920x6 = 12441600 cycles. The MCOL loop needs to go faster!
    d. The MCOL loop is currently unconstrained which means that each iteration of the MCOL loop takes 6 clock cycles to complete before the next iteration can start. This loop should be constrained using loop pipelining constraints. Loop pipelining will allow the next loop iteration to start before the current iteration has finished. So instead of starting every 6 clock cycles, each loop iteration can be made to start every clock cycle, or every 2 clock cycles, and so on.
25. Click on Architecture in the Task Bar
    e. Select the MCOL loop and enable Loop Pipelining by checking the Pipeline box. This sets the default Initiation Interval (II) to 1, which means that a new loop iteration will start every clock cycle



26. Click on RTL in the Task Bar
27. Look at the Table View and note that the performance goal of ~2 million clock cycles to process an image has been achieved. Also note that area has increased somewhat

Start Page | Table | Flow Manager | Constraint Editor | MagnitudeAngle.h (...

Report: General

Solution	Latency Cycles	Laten...	Throughput Cycles	Throu...	Slack	Total Ar...
MagnitudeAngle.v1 (extract)	64282679	192848...	64282681	192848...	0.36	3888.94
MagnitudeAngle.v2 (extract)	12443758	373312...	12443761	373312...	0.01	5987.18
MagnitudeAngle.v3 (extract)	**2077919**	**623375...**	**2077921**	**623376...**	**0.00**	**7222.25**

Taking Advantage of Bus Bandwidth to Increase Performance

The MagnitudeAngle algorithm that was optimized and synthesized reads one 8-bit value each of the derivatives “dx” and “dy” from the design interface in parallel, with “dx” and “dy” are mapped to separate handshake interfaces. However, this accelerator is going to be integrated into a RiscV platform where the processor and memory subsystem have a 64-bit data bus. Therefore, the C++ memory architecture can be rewritten to match the processor data bus bandwidth. This will also allow the MagnitudeAngle accelerator to be further optimized to increase performance

28. This new model can be reverified against the algorithm by repeating steps 1 to 3 but instead enter “make run_4x” in the terminal
29. In Catapult go to File > Run Script, select the go_hls_4x.tcl file, and click OK. This will create a new project and synthesize the modified architecture to match the previous performance goals of ~2 million clock cycles to process a single 1920x1080 image
30. Go to the Table View and verify that the previous performance goal is met
31. Go to File > Open, and select the MagnitudeAngle_4x.h file, and click Open
    a. Note that the design interface has been modified to combine the dx and dy inputs, and the magn and angle outputs
    b. Also note that typedefs and constants have been used to make the design easily reconfigurable

```
#pragma hls_design interface
void CCS_BLOCK(run)(ac_channel<gradBus>      &dxdy_in,
                    ac_channel<magAngBus>    &magn_angle,
                    ac_int<widthBits,false>  imageWidth,
                    ac_int<heightBits,false> imageHeight) {

  gradBus dxdy;
  gradType dx, dy;
  sqType dx_sq;
  sqType dy_sq;
  sumType sum; // fixed point integer for sqrt
  angType at;
  magAngBus ma;
  sqRtType sq_rt; // square-root return type

  HROW: for (int y = 0; y < IMG_HEIGHT; y++) {
    HCOL: for (int x = 0; x < IMG_WIDTH/BUS_WORDS; x++) {
```

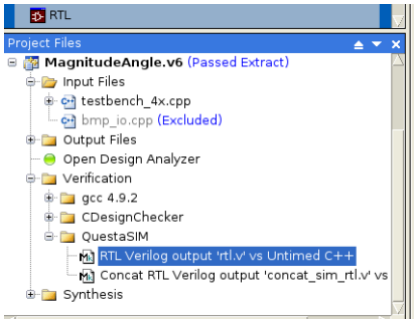
32. Go to File > Open, select types.h, and click Open
    a. Note that constants have been defined for the image height, width and number of words
    b. Typedefs have been created for the various data types and bit-widths used in the design
    c. A enum is used along with the ac-datatypes helper function “nbits” to compute the minimum number of bits needed for the loop control variables
    d. Structs, gradBus and magAngBus, have been created to pack the input (dx and dy) and output (magn and angle) data so that it matches the bus bandwidth.

```
7   const int IMG_WIDTH=1920;
8   const int IMG_HEIGHT=1080;
9   const int BUS_WORDS = 4;
10
11  // Define some bit-accurate types to use in this model
12  typedef ac_fixed<8,8,true,AC_RND,AC_SAT>  gradType;
13  typedef ac_fixed<18,18,false>             sqType;
14  typedef ac_fixed<19,19,false>             sumType;
15  typedef ac_fixed<8,8,false,AC_RND,AC_SAT> magType;
16  typedef ac_fixed<8,3,true>                angType;
17  typedef ac_fixed<16,9,false>              sqRtType;
18
19  enum{
20   widthBits = ac::nbits<IMG_WIDTH>::val,
21   heightBits = ac::nbits<IMG_HEIGHT>::val
22  };
23  struct gradBus {
24    gradType data0[BUS_WORDS];
25    gradType data1[BUS_WORDS];
26  };
27
28  struct magAngBus {
29    magType data0[BUS_WORDS];
30    angType data1[BUS_WORDS];
31  };
32
```

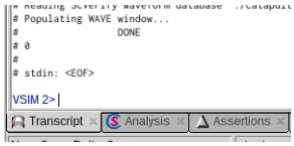
33. Click on Architecture in the Task Bar
    a. Expand the Interface Folder and note that the dxdy and magnAng interfaces are each 64-bits wide
    b. Look at the loops and note that there is a new loop MCOL4 that has been added to the design
    c. Double-click on MCOL4 to cross-probe to the C++ source
    d. Note that the dxdy_in interface is read n the MCOL loop, and then the MCOL4 process the read data over BUS_WORDS=4 iterations

```
39    MROW: for (int y = 0; y < IMG_HEIGHT; y++) {
40      MCOL: for (int x = 0; x < IMG_WIDTH/BUS_WORDS; x++) {
41        dxdy = dxdy_in.read();
42        MCOL4: for (int i=0; i<BUS_WORDS; i++) {
43          dx = dxdy.data0[i];
44          dy = dxdy.data1[i];
45
```

34. Go to the Verification > Questasim Folder in the Project Files View and double-click on “RTL Verilog output ‘rtl.v’ vs Untimed C++’. This will launch Questasim



35. Select the Questasim transcript tab and type “run -all” and press Enter

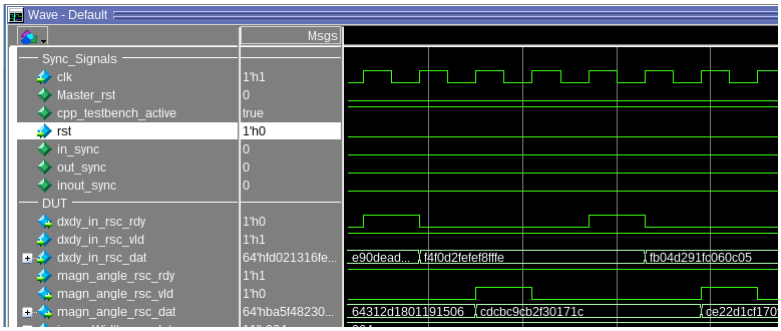


36. The simulation will run to completion. Note that this simulation has reduced the image size to 128x128 so it finishes quickly. It takes about 5 to 10 minutes to simulate a 1920x1080 image
37. When the simulation has finished you will see the transcript issue a “Simulation Passed” message indicating that it compared the outputs of the C++ and the RTL and they’re the same

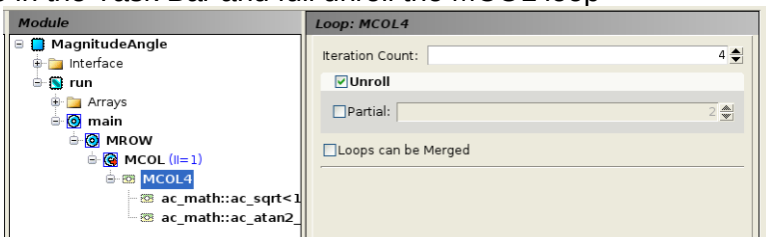
```
# Checking results
# 'magn_angle_data0'
#    capture count        = 2500
#    comparison count     = 2500
#    ignore count         = 0
#    error count          = 0
#    stuck in dut fifo    = 0
#    stuck in golden fifo = 0
# 'magn_angle_data1'
#    capture count        = 2500
#    comparison count     = 2500
#    ignore count         = 0
#    error count          = 0
#    stuck in dut fifo    = 0
#    stuck in golden fifo = 0
#
# Info: scverify_top/user_tb: Simulation PASSED @ 31205 ns
# ** Note: (vsim-6574) SystemC simulation stopped by user.
```

38. Click on the “Waveform” Tab in Questa. This shows the simulation waveforms for the IO of the hardware
    a. Zoom out full by clicking on the “Zoom Full” button on the tool bar

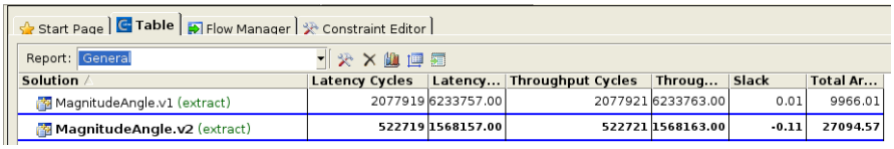
    b. Click on the “Zoom in(+)” button till you can see the waveforms
    c. Note that the dxdy_in input is read every 4 clock cycles (dxdy_in_rsc_rdy is high every 4 cycles) and the magnAng output is written every 4 clock cycles (magnAng_rsc_vld is high every 4 clock cycles)



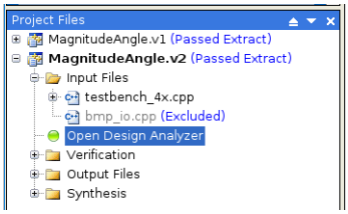
39. Close Questasim
40. Click on Architecture in the Task Bar and full unroll the MCOL loop



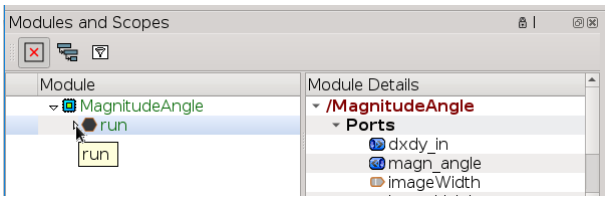
41. Click on RTL in the Task Bar
42. Go to the Table View and note that the second performance goal has been achieved
    a. The design takes ~520,000 cycles to process a 1920x1080 image
    b. Note that the area has increase substantially. This is a result of adding more parallelism by unrolling the MCOL4 loop



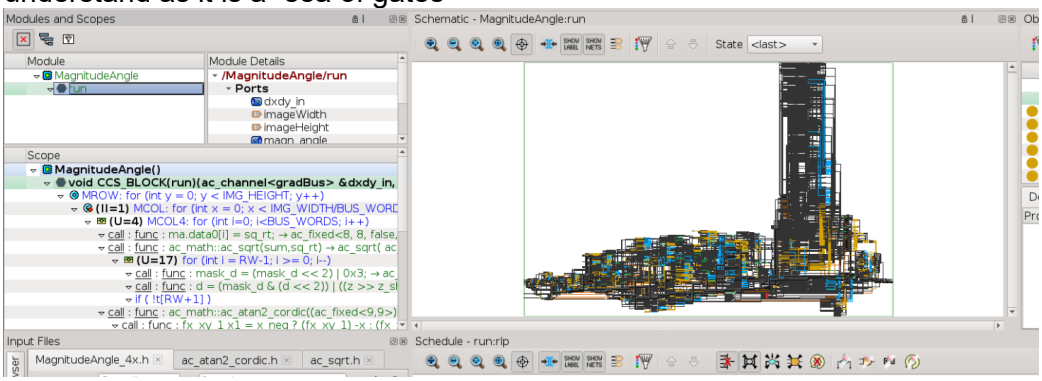
43. As a final step lets look at the hardware that got built from the synthesis process
44. Double-click on “Open Design Analyzer” Icon in the Project Files View. This will launch the Catapult Design Analyzer which shows the relationships between the C++, design constraints, and the generated hardware



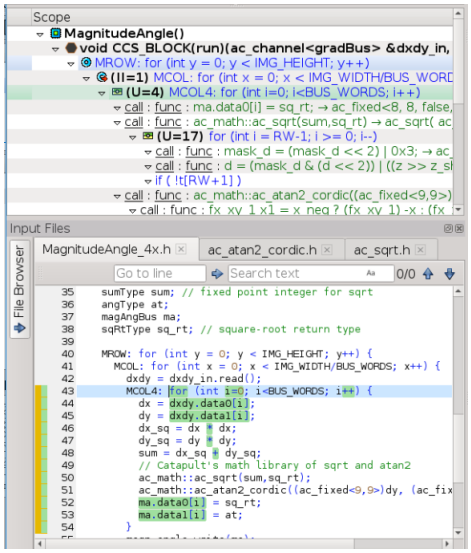
45. Expand the entire design by clicking on the “run” icon in the Modules and Scopes View



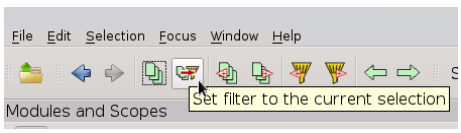
46. This shows us the entire design in the RTL Schematic and Schedule View, which is hard to understand as it is a “sea of gates”



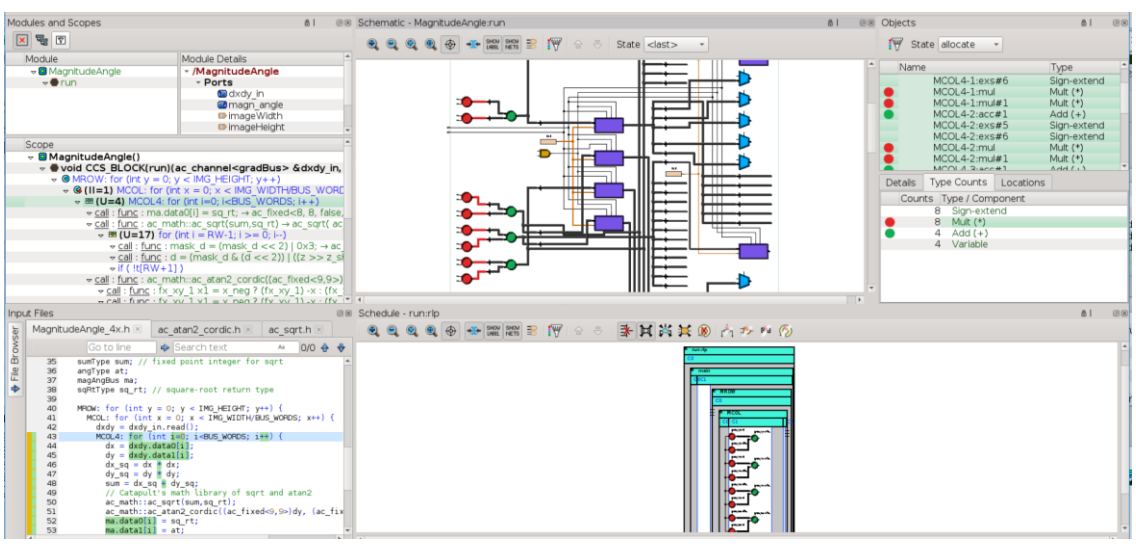
47. One of the most powerful features of Design Analyzer is that it allows the design to be filtered to specific areas
48. Click on the MCOL4 loop in the Scopes View. Note that it also highlights the C++ source that is inside that loop



49. Click on the Filter Button to filter the design to just the MCOL4 loop



50. Note that the RTL Schematic and other views are now displaying just the resources that are inside of the MCOL4 loop



51. Note that the Type Counts Tab shows 8 multipliers generated from the MCOL4 loop. Why?
52. Click through the list of objects in the Object View and note how the other views highlight that object
53. Close Design Analyzer
54. Re-run the SCVerify verification flow to verify the performance

Done Lab2