# High-Level Synthesis Flow with Intel® FPGAs

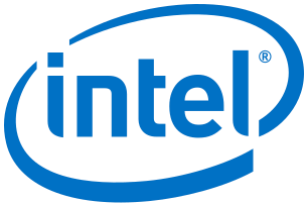
# Exercise

**Software Requirements**

64-bit Linux Software Development Environment with g++
Intel® HLS Compiler version 17.1
Intel Quartus® Prime Pro software version 17.1 with Arria® 10 family

*In this exercise, you will practice going through the HLS flow, we will first perform a g++ compile, then an i++ emulation compile, then co-simulation compile, and finally we will integrate the generated component with a Quartus project.*

*If you have difficulty with any of the tasks in this exercise, please consult the instructor. There is also a solution folder included with the exercise files that you may reference.*



## Step 1. Perform g++ Compile

____ 1. Go to the IntroHLS directory in the terminal

*a.* "cd <install_directory>/IntroHLS"

____ 2. Examine mult.cpp in your favorite text editor

*a.* "gedit mult.cpp"

*In this file, my HLS component is called mymult and it simply does a multiply of 2 integer arguments. The testbench main will loop through 10 random set of values and verify that the HLS component results matches the multiply results from the testbench.*

*b.* Close gedit window when done by clicking the x in the top left corner

____ 3. Examine the Makefile

*a.* "gedit Makefile"

*In this file you see the 4 targets that's been setup. The gpp.exe target performs a g++ compile. The emu.exe target performs i++ with -march=x86-64, the fpga.exe is the cosimulation compile. And the fpga_ghdl.exe is the cosimulation compile with full signal logging.*

*b.* Close gedit window when done by clicking the x in the top left corner

____ 4. Perform the g++ compile.

*a.* In the terminal type "make gpp.exe"

*Ensure the compile is error free.*

____ 5. Execute gpp.exe

*a.* Type "./gpp.exe"

*You should see the ten sets of correct results*

```
[student]:~/fpga_trn/IntroFPGA_v17.1/IntroHLS $ make gpp.exe
g++ -I/home/student/inteldevstack/intelFPGA_pro//19.1/hls/bin/../include -std=c++11 mult.cpp -o
 gpp.exe
[student]:~/fpga_trn/IntroFPGA_v17.1/IntroHLS $ ./gpp.exe
3*6=18
7*5=35
3*5=15
6*2=12
9*1=9
2*7=14
0*9=0
3*6=18
0*6=0
2*6=12
```

## Step 3. Perform i++ x86 Emulation Compile

____ 1. Type "make emu.exe"

*This will perform the same compile as the g++ compile but using the makefile-compatible i++ Intel® HLS Compiler. In emulation, the component is still compiled and executed just like any other c++ function.*

____ 2. Execute emu.exe by typing "./emu.exe"

```
[student]:~/fpga_trn/IntroFPGA_v17.1/IntroHLS $ make emu.exe
i++  -I/usr/include/x86_64-linux-gnu/c++/5/bits -march=x86-64 mult.cpp -o emu.exe
[student]:~/fpga_trn/IntroFPGA_v17.1/IntroHLS $ ./emu.exe
3*6=18
7*5=35
3*5=15
6*2=12
9*1=9
2*7=14
0*9=0
3*6=18
0*6=0
2*6=12
```

*You should see that the emulation compile behaves exactly the same as the gpp compile*

## Step 4. Perform Co-simulation Compilation

____ 1. Type "make fpga.exe"

*In the cosimulation flow, the main() testbench is still executed as software, but the component is executed in ModelSim simulator. When using -march=Arria10, the HLS compiler is generating the HDL for the mymult component and because of that it will take a few minutes.*

____ 2. Execute fpga.exe by typing "./fpga.exe" in the terminal window

```
[student]:~/fpga_trn/IntroFPGA_v17.1/IntroHLS $ make fpga.exe
i++  -march=10AX115N2F40E2LG mult.cpp -o fpga.exe
[student]:~/fpga_trn/IntroFPGA_v17.1/IntroHLS $ ./fpga.exe
3*6=18
7*5=35
3*5=15
6*2=12
9*1=9
2*7=14
0*9=0
3*6=18
0*6=0
2*6=12
```

*You should see that the cosimulation compile should match the result from the previous runs. Now you see how easy it is to perform a functional simulation of your HLS component.*

## Step 5. Perform co-simulation with logging of HDL signals

____ 1. Type "make fpga_ghdl.exe"

*With i++ -ghdl, the ModelSim testbench generated will log all HDL signals in a wlf file.*

____ 2. Execute fpga_ghdl.exe by typing "./fpga_ghdl.exe"

*You should again see results that match the previous compiles*

____ 3. Open the vsim.wlf file in ModelSim

*a. Type "vsim fpga_ghdl.prj/verification/vsim.wlf"*

____ 4. Add signal to the Waveform viewer

*a.* Find mymult_inst in the vsim-Default window and click on it

*The objects window should now display signals from mymult*

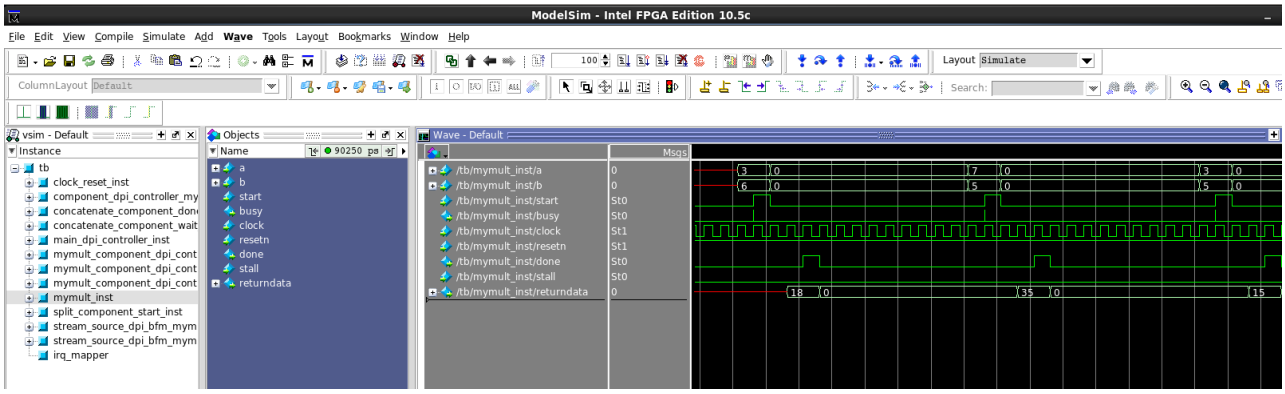
*b.* In the objects windows press Ctrl+a to select all objects

*c.* Right click and select Add Wave

*d.* In the Wave viewer, right click and choose Zoom Full

*e.* For a, b, and return data, right click and choose Radix→Decimal

*f.* Zoom in and out until you see a clear picture of the waveform and results




____ 5. As you can see, the results should match the command line.

*Notice how the start signal matches with a and b inputs while the return data matches with the done signal.*

*Notice also that the component executing inside this testbench is not pipelined. In the HLS class we would teach you how to generate an enqueued pipelined testbench.*

____ 6. Close the ModelSim simulator

## Step 7. Perform Quartus Compilation to Generate QoR Results

____ 1. Perform i++ with the --quartus-compile option

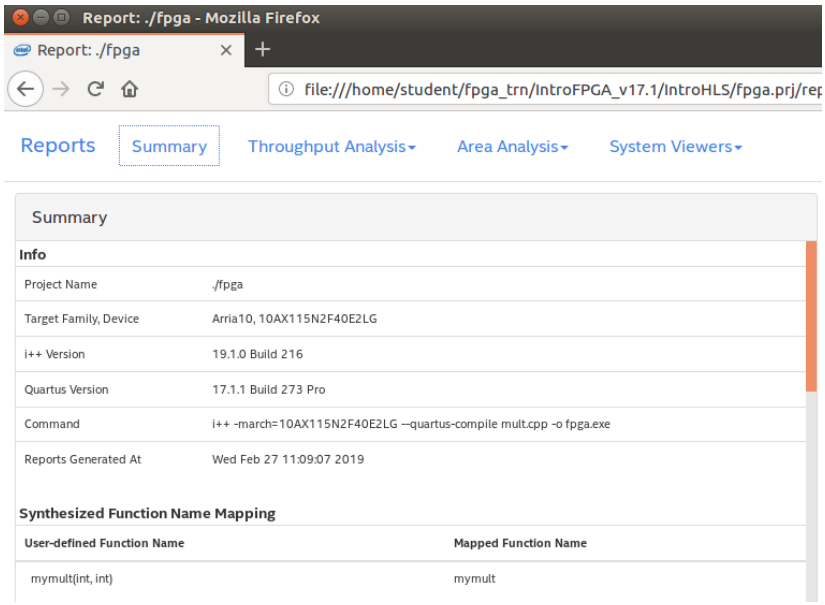
*a.* i++ -march=10AX115N2F40E2LG --quartus-compile mult.cpp -o fpga.exe

*This will perform a Quartus compilation on all of the components in the cpp file. The final QoR results such as fmax and resource utilization will be added to the HTML report. A Quartus compilation will take several minutes.*

____ 2. Open the HTML report

a. firefox fpga.prj/reports/report.html

*This is the main HTML report containing static feedback on the component.*




____ 3. Examine the Summary page

*This is where you'll see the Quartus generated Fmax and Resource utilization information along with estimated resources for each component*

____ 4. Go to loop analysis by using the drop down menu at the top of the page

*This tab contains information on how loops in the components are optimized*

____ 5. Go to Area analysis of source

*Here if you expand the System and the component you'll see the estimated resource consumption of each line of code*

____ 6. Experiment with the Component Viewer and the Component Memory Viewer

____ 7. Go to verification statistics

*Here you'll see the performance details of the 10 invocations we ran from the testbench*

____ 8. Close the HTML Report

# Exercise Manual

*for*

# High-Level Synthesis Advanced Optimization Techniques

**Software Requirements**

64-bit Linux Software Development Environment with g++
Intel® HLS Compiler version 17.1
Intel Quartus® Prime Pro software version 17.1 with Arria® 10 family

# Exercise 1

# Optimizing loop pipelining performance by relaxing data dependencies

*In this exercise, you will practice using a few common techniques to improve the loop pipelining performance. In this lab, our component will simply sum the elements in an array*

*If you have difficulty with any of the tasks in the exercises, please consult the instructor. There is also a solution folder included with the exercise files that you may reference.*

## Step 1. Setup Virtual Machine Lab Environment

____ 1. In Windows explorer, navigate to the flash drive

____ 2. Go into the FPGA_Train_CentOS6 directory

____ 3. Double click on FPGA_Train_CentOS6.vbox

*This should start Oracle VM VirtualBox and boot the machine.*

*If the VM did not boot automatically, try removing but NOT deleting the existing machines inside VirtualBox and add the .vbox file on the flash drive and start the VM.*

____ 4. Log into the VM as User:Student Password: QPrime.1

____ 5. Open a terminal

*a.* Applications→System Tools→Terminal

____ 6. Navigate to the course directory

____ 7. Type "cd fpga_trn/HLS_ad"

____ 8. Perform "ls"

____ 9. If there's an "hls_ad_17_1" directory, please remove it

*a.* Type "rm -rf hls_ad_17_1"

____ 10. Unzip hls_ad_17_1_v2.tar.gz

*a.* Type "tar -xvf hls_ad_17_1_v2.tar.gz"

*This will unzip the course files into the appropriate folder. If the tar.gz files is not there with the **exact** same name, please consult the instructor.*

____ 11. Go into the course directory

*a.* Type "cd hls_ad_17_1"

____ 12. Source the course environment

*a.* Examine init.sh using your favorite text editor (For example gedit)

*i. gedit init.sh*

*This script set up the directories for ModelSim, Quartus, and HLS tools*

*b.* Execute the script

i. Type "source init.sh" in the terminal

## Step 2. Optimize the Loop

____ 1. Change directory into the **summation** directory

**cd summation**

____ 2. Open summation.cpp

*In this program we simply loop through all the double elements of an array and perform an summation*

____ 3. Compile the component in emulation mode

**i++ summation.cpp**

____ 4. Execute the newly created executable

**./a.out**

Ensure the HLS component result are the same as the CPU result

____ 5. Compile the component in cosimulation mode

**i++ -march=Arria10 summation.cpp**

____ 6. Execute the newly created executable and ensure the results are the same

**./a.out**

____ 7. Open the HTML report

**firefox a.prj/reports/report.html**

____ 8. Note the resource consumption in the summary report

ALUTS_____ FFs_____ RAMs_____DSPs_____

____ 9. Note the latency of the component

Avg Latency _____

____ 10. Examine the loop report and note the II of the for loop

II:_____

*You should see that the loop inside the component has a high II due to data dependency caused by the double-precision floating-point add*

____ 11. Use the shift register technique to relax the distance between dependencies for the loop

*If you feel confident about your understanding of the technique, go ahead and attempt this yourself. If you need additional help, follow the substeps below, and it'll guide you step-by-step.*

a. Delete the line that sets the result inside the loop.

b. Before loop, define a double array of II (II should = 14) elements. Name the array sum_copies[14]

c. Initialize all member of the sum_copies array to 0 using a loop;

d. Inside the original inner loop, create the double variable "**cur"** and set it to the sum of the top of the sum_copies array and a_in[i]

*In our accumulation, we're only using the top copy of sum_copies*

e. After the previous step, shift all values of sum_copies up. Create a loop that starts at the top and decrements. And set each element to be the value of the previous element in the array.

f. After the previous step, set sum_copies[0]=cur;

g. After the main loop. Loop through all elements of sum_copies and accumulate it to "**result**".

____ 12. Recompile using the emulation flow and verify the component still works.

*Because we're changing the order of floating point operation. You may see a slight difference in the HLS result and the CPU result.*

____ 13. Compile the design in the cosimulation mode

**i++ -march=Arria10 summation.cpp**

____ 14. Test the executable

**./a.out**

____ 15. Open the HTML report

**firefox a.prj/reports/report.html**

____ 16. Go to the Loop Analysis Report

*Is the II of the main loop at 1 now? If it's not, you may need to go back and fix your code.*

*You should also see that the other loops are Auto-enrolled*

*If you do not see this result, you may need to use #pragma unroll to unroll the shift register loop.*

| Component: summation (summation.cpp:10) | | | | Task function |
|---|---|---|---|---|
| summation.B1.start (Component invocation) | No | n/a | n/a | Out-of-order inner loop |
| Fully unrolled loop (summation.cpp:13) | n/a | n/a | n/a | Auto-unrolled |
| Fully unrolled loop (summation.cpp:26) | n/a | n/a | n/a | Auto-unrolled |
| summation.B2 (summation.cpp:16) | Yes | 1 | n/a | |
| Fully unrolled loop (summation.cpp:20) | n/a | n/a | n/a | Auto-unrolled |

____ 17. Go to the Verification statistics report

What's the avg Latency now? _____

____ 18. Go to the summary report and note the resource consumption

ALUTs_____ FFs_____ RAMs_____ DSPs_____

*As you can see there is a significant increase in resources to achieve the maximum throughput.*

____ 19. As you probably have noticed, the compiler automatically unrolled 3 loops with constant loop counts but also threw a compiler warning.

To remove the warning, manually apply the **#pragma unroll** to those 3 loops

____ 20. Compile for cosimulation, execute the component and make sure the result stayed the same in the HTML report.

**Exercise Summary**

- Practiced executing component in emulation and cosimulation mode
- Examined the HTML reports
- Fixed data dependency issues from the optimization report

## END OF EXERCISE 1

# Exercise 2

# Local Memory Optimizations

*In this exercise, you will practice using a few common techniques to improve local memory architecture in a component. We will look at how to bank explicitly on specified bits, how to guide the compiler to build a stall-free memory architecture, and how to merge memory.*

## Step 1. Memory Banking and Coalescing

____ 1. If you haven't already done so, open a terminal inside the VM, and source init.sh in the $HOME/fpga_trn/HLS_ad/hls_ad_17_1 directory

____ 2. Change directory into the **memory** directory

____ 3. Open bankbits.cpp

*This is a simple memory example, we have a static local array a. Based on the values of the input arguments, we will perform 4 unrolled writes and 4 reads.*

____ 4. Compile bankbits for the FPGA

**i++ --fpga-only -march=arria10 bankbits.cpp**

____ 5. Open the HTML report

**firefox a.prj/reports/report.html**

____ 6. Note the resource consumption on the Summary page

ALUTs_____ FFs_____ RAMs_____

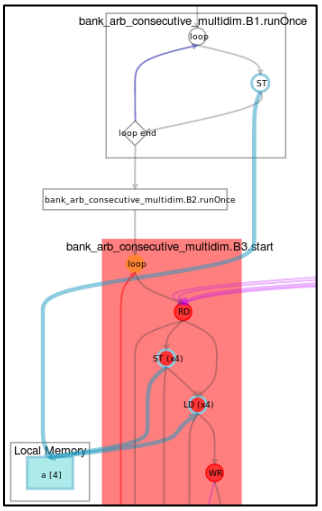
____ 7. Go to Loop analysis

What's the II of the component invocation? _____

*Here you see that the compiler attempted to pipeline the component but got a large II due to memory dependency on a*

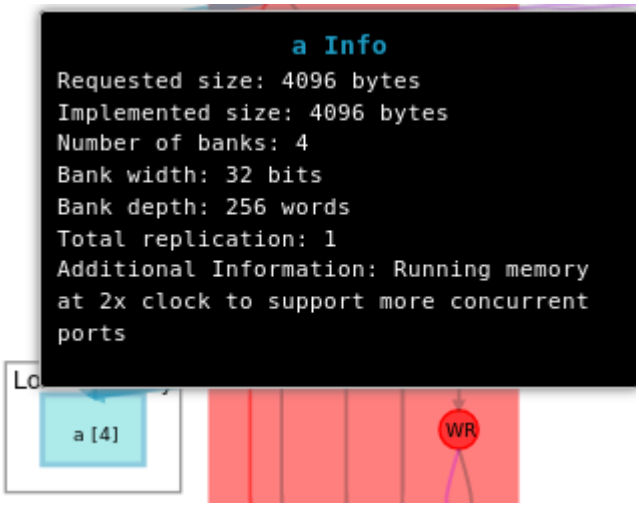
____ 8. Go to the Component Viewer

____ 9. Here you immediately see that there are 9 access to the memory, 8 from the unrolled reads and writes and 1 from the static variable initialization.



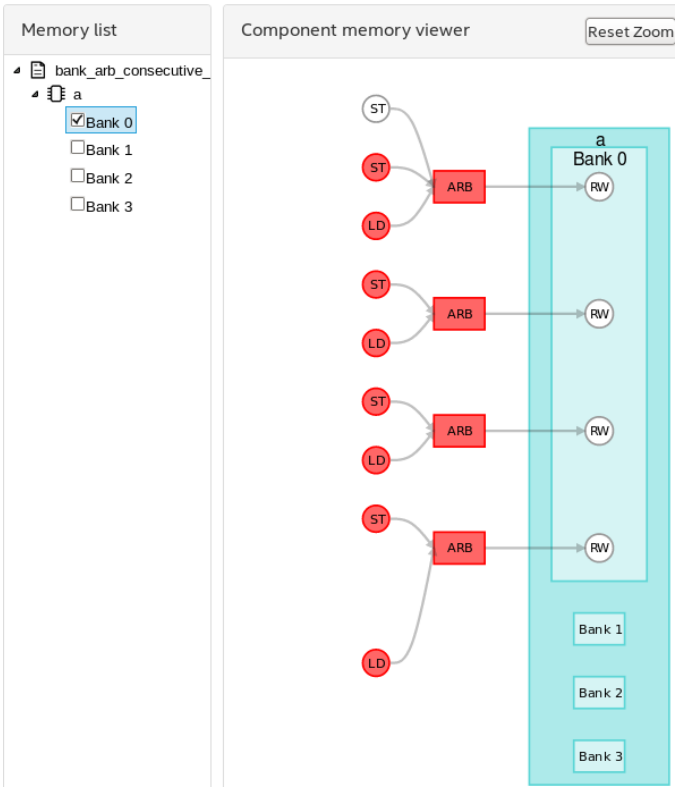
____ 10. Hover over the a[4] Local Memory




*You can see that the compiler ran this memory at 2x clock giving it 4 ports. It attempted to bank the memory on the lower bits but it was not able to coalesce or replicate.*

*Because there are 8 simultaneous access aggregating on 4 ports, these accesses are stallable and needs to be arbitrated*

____ 11. Look in the Component Memory Viewer and enable just one bank




*Here again you see the 9 access aggregating on the 4 ports with the 8 simultaneous accesses being stallable.*

____ 12. Open bankbits.cpp if it's not already open

____ 13. Add the attribute to the memory **a** stating the static local memory only needs to be initialized at FPGA power up and not component reset.

____ 14. Figure out the correct banking based on access pattern and use the hls_bankbits attribute to set the bank bits.

*Hint: The only address bits where the 8 access differ is based on the loop variable i. The lowest dimention of the a array [128] uses 7 bits.*

____ 15. Compile bankbits.cpp for FPGA

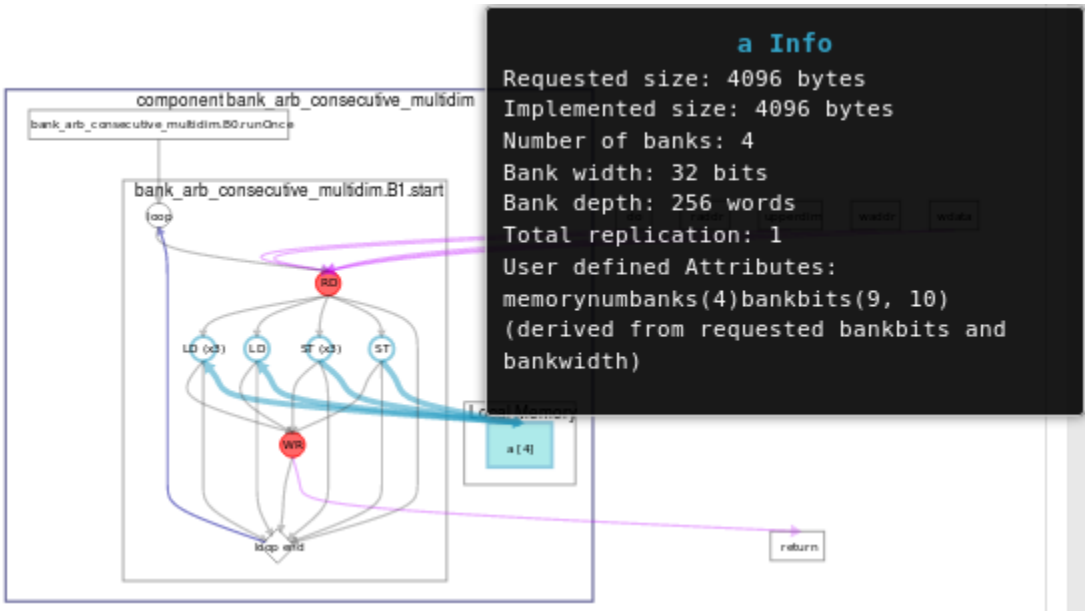
**i++ --fpga-only -march=arria10 bankbits.cpp**

____ 16. Open the HTML report

____ 17. Now much resource doesthe component take now?

ALUTs_____ FFs_____RAMs_____

____ 18. Go to the component viewer and hover over the local memory




*As you can see from this report, we still have 4 banks but because we are not banking on the optimal bits, the local memory is no longer stallable.*

*We also reduced the number of access to 8 instead of 9.*

____ 19. Go to the loop report.

*You should now see the component II is ~1*

*By improving the bankbits, we also improved the performance while reducing resource usage.*

____ 20. Now lets see if we can also fix the original problem by simply changing the order of the index.

____ 21. Open bankbits.cpp if it's not already open.

____ 22. Remove the bankbits attribute

____ 23. Change the order of the two lower indexes, **a** is now declared as

**static int a[2][128][4]**

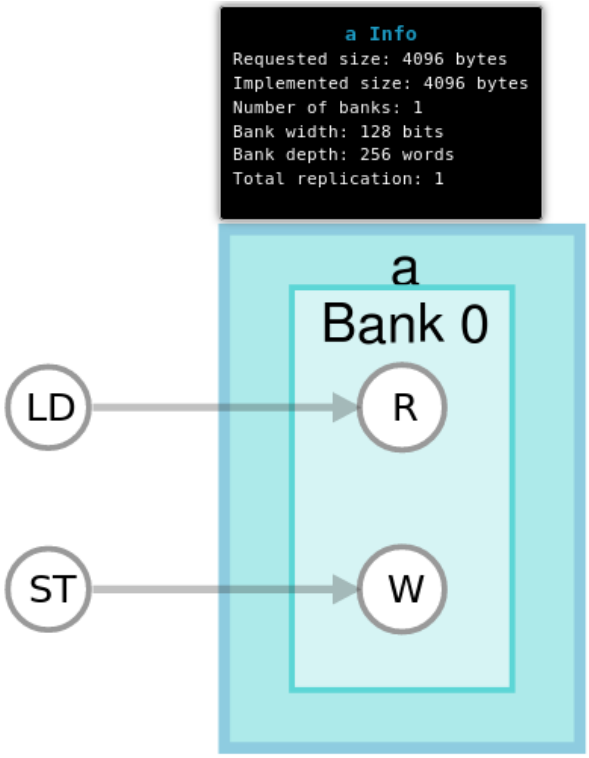
____ 24. Also, swap the two lower index locations in the reading and writing statements, anywhere **a** is accessed.

____ 25. Compile bankbits.cpp for FPGA

**i++ --fpga-only -march=arria10 bankbits.cpp**

____ 26. Open the HTML report

____ 27. Go to the Component Memory Viewer, and hover over memory **a**




*You can see here that we are no longer separating the memory into banks but instead coalescing 4 access in to a wider 128 bit access.*

____ 28. Go to the Loop report.

*You should see that the component invocation II is still 1*

____ 29. Note the resource utilization

ALUTS: _____ FFs_____ RAMs_____

*You should see with automatic coalescing the interconnect is simpler, resulting in less resources used while maintaining the same high performance.*

____ 30. Open bankbits.cpp again

____ 31. Keeping the array dimensions for **a**, apply hls_numbanks(4)

*Lets examine the affect of banking on the lower bits now.*

____ 32. Open the HTML report

____ 33. What's the resource utilization now?

ALUTs_____ FFs_____ RAMs_____

____ 34. Examine the loop report as well as the memory report.

*Is banking now better or worse than automatic coalescing done by the compiler?*

## Step 2. Memory Merging

____ 1. Open depthwisemerge.cpp

*In this design, we have two local memory arrays but they access to them are always mutually exclusive since the access switch on the use_a argument.*

____ 2. Compile the component for FPGA

**i++ --fpga-only -march=arria10 depthwisemerge.cpp**

____ 3. Open the HTML report

**firefox a.prj/reports/report.html**

____ 4. In the HTML report note the resource consumption of the component

ALUs_____ FFs_____ RAMs_____

____ 5. Examine the memory and/or the component for memory implementation.

*Since we have relatively shallow memories and that the accesses are mutually exclusive we can try to merge the memory to save resources.*

*Should this merge be depthwise or widthwise?*

____ 6. In depthwisemerge.cpp, use the hls_merge attribute on both a and b to merge them.

____ 7. Compile the component for FPGA

**i++ --fpga-only -march=arria10 depthwisemerge.cpp**

____ 8. Open the HTML report

**firefox a.prj/reports/report.html**

____ 9. In the HTML report note the resource consumption of the component

ALUs_____ FFs_____ RAMs_____

____ 10. Examine the memory and/or the component for memory implementation in the HTML report

*You should see a reduction in RAM usage.*

____ 11. Open **widthwisemerge.cpp**

*In this design, we have two local memory arrays but they access to them are always together*

____ 12. Compile the component for FPGA

**i++ --fpga-only -march=arria10 widthwisemerge.cpp**

____ 13. Open the HTML report

**firefox a.prj/reports/report.html**

____ 14. In the HTML report note the resource consumption of the component

ALUs_____ FFs_____ RAMs_____

____ 15. Examine the memory and/or the component for memory implementation.

*Since we have narrow memories and that the accesses are always together we can merge them to save resources.*

*Should this merge be depthwise or widthwise?*

____ 16. In widthwisemerge.cpp, use the hls_merge attribute on both a and b to merge them.

____ 17. Compile the component for FPGA

**i++ --fpga-only -march=arria10 widthwisemerge.cpp**

____ 18. Open the HTML report

**firefox a.prj/reports/report.html**

____ 19. In the HTML report note the resource consumption of the component

ALUs_____ FFs_____ RAMs_____

____ 20. Examine the memory and/or the component for memory implementation in the HTML report

*You should see a reduction in RAM usage. And that the width of the access is wider.*

**Exercise Summary**

- Practiced using the bank attributes as well as using desirable access patterns in order for the HLS compiler to generate the optimal memory architecture.
- Practiced merging memories both depth-wise and width-wise

## END OF EXERCISE 2