OpenCL FPGA Lab 0 – Vector Add

This handout will take you through the steps required for completing Lab 0. After this lab, you will become familiar with the complete Intel FPGA OpenCL emulation, compilation, and simulation based on Boston University CAAD Lab's tool flow.

Getting Started:

- Before beginning, please get a copy of source code from Github: https://github.com/FPGA-Bot-Yang/HK_FPGA_OpenCL_Labs

  "git clone https://github.com/FPGA-Bot-Yang/HK_FPGA_OpenCL_Labs.git"

  "cd HK_FPGA_OpenCL_Labs"

- Setup OpenCL environment

  "source init_OpenCL.sh"

- Verify environment setup by typing:

  "which aocl"

  You should expect the path of the OpenCL executable as shown below:

```
cyang@americano00.eee.hku.hk:~/HK_FPGA_OpenCL_Labs$ which aocl
/opt/Altera/Quartus/17.1/hld/bin/aocl
```

- Enter Lab0 directory:

  "cd Labs/lab0"

Under this directory, "device" folder contains the OpenCL kernel code (.cl), "host" folder contains the OpenCL host code. "support_files" folder contains library files that will be used later in this lab for simulating the OpenCL generated kernel, do not touch or move this folder. "Makefile" is used for compiling the host code. Those 2 bash scripts (.sh file) contains all the compiling commands for the entire process, which is provide for your convenience.

OpenCL Host Code:

Open the host code (host/src/main.cpp) and perform the following tasks:

1, First, notice that the correct Altera OpenCL headers are included:

```
#include "CL/opencl.h"
#include "AOCLUtils/aocl_utils.h"
```

2. Find the lines in the code that sets up the compute context, finds the target platform, and loads the kernel; these are done in the same manner as what you have seen before.

3. In the code, search for clCreateBuffer() which allocates memories for the device. You will see that the second argument is a flag (i.e. CL_MEM_READ_ONLY) that provides information to the OpenCL compiler to optimize storage of the device arrays.

4. Next, search for clKernelArg – this builds the list of arguments for the kernel. It may appear long and tedious, but only needs to be completed once per kernel.

5. Search for clEnqueueNDRangeKernel – this is the function that is used to run your kernel in parallel on the accelerator. The global argument specifies the image size and is the same as specifying the required work group size as we did in the kernel.

6. Finally, search for clEnqueueReadBuffer – here we are reading the results from the device to verify the output. Notice that d_out_image was one of the memories we had allocated with clCreateBuffer, that you saw in Step 3.

OpenCL Kernel Code:

Open the kernel code (device/mmm.cl). The kernel code performs a simple vector_add task.

1, In each kernel, it adds the related element from A and B and write back to C.

2, The get_global_id(0) function indentifies the positon of the current element being processed. This is important as the OpenCL runtime will take this kernel and parallelize it across the specified work group. Each work item in the work group will have a unique ID.

Run OpenCL Emulation:

The hardware compilation is a lengthy process, depending on the complexity of the kernel code and platform, the full compilation process may take 1~10 hours to finish. Thus, the developers should first emulate the code for functional correctness on our x86-64 machines. (**Note that the emulation is only performed on the software level.** Each function used in the kernel code has a related C model that mimic the hardware function. The emulation process will execute these C functions in a sequential order. **Thus even the emulation has passed, does not guarantee a 100% correctness when you executing on board.**)

The emulation generally has 2 steps: 1) compile host code, 2) compile kernel code for emulation. These 2 steps are covered in the provided script (run_emulation.sh). Run the emulation by typing the following command:

"source run_emulation.sh"

Your output should look like:

```
cyang@americano00.eee.hku.hk:~/HK_FPGA_OpenCL_Labs/Labs/lab0$ source run_emulation.sh

****************** Building Host Code ******************
****************** Compiling Kernels for Emulation ************************
aoc: Environment checks are completed successfully.
aoc: If necessary for the compile, your BAK files will be cached here: /var/tmp/aocl/
You are now compiling the full flow!!
aoc: Selected target board a10gx
aoc: Running OpenCL parser....
aoc: OpenCL parser completed successfully.
aoc: Compiling for Emulation ....
aoc: Emulator Compilation completed successfully.
Emulator flow is successful.
To execute emulated kernel, invoke host with
        env CL_CONTEXT_EMULATOR_DEVICE_INTELFPGA=1 <host_program>
 For multi device emulations replace the 1 with the number of devices you wish to emulate
****************** Run Emulation ************************
Initializing OpenCL
Platform: Intel(R) FPGA SDK for OpenCL(TM)
Using 1 device(s)
  EmulatorDevice : Emulated Device
Using AOCX: mmm.aocx
Launching for device 0 (10000 elements)

Time: 82.357 ms
Kernel time (device 0): 82.098 ms

Verification: PASS
****************** Genearating Compilation Report ************************
aoc: Environment checks are completed successfully.
aoc: If necessary for the compile, your BAK files will be cached here: /var/tmp/aocl/
aoc: Selected default target board a10gx
aoc: Running OpenCL parser....
aoc: OpenCL parser completed successfully.
aoc: Optimizing and doing static analysis of code...
aoc: Linking with IP library ...
Checking if memory usage is larger than 100%
aoc: First stage compilation completed successfully.
aoc: To compile this project, run "aoc bin/mmm.aoco"
```

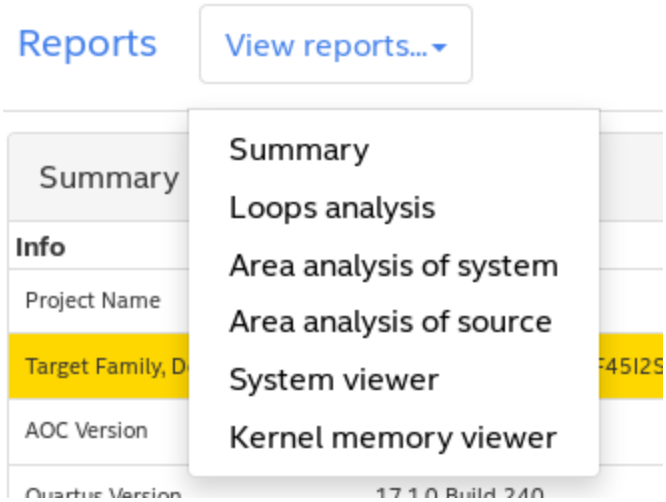
Note that in the emulation report, the execution time is only a simulated one, which should not be used for performance evaluation. The actual runtime can either be collected by running on board after a full compilation, or by simulating the generated kernel, which will be covered in the later section of this lab.

Read OpenCL Report:

After the emulation, a pre-compilation report is generated to give developer the estimation of resource usage and kernel memory access pattern, plus a brief report for optimization. The report file is located in “reports/report.html”. Open the report file:

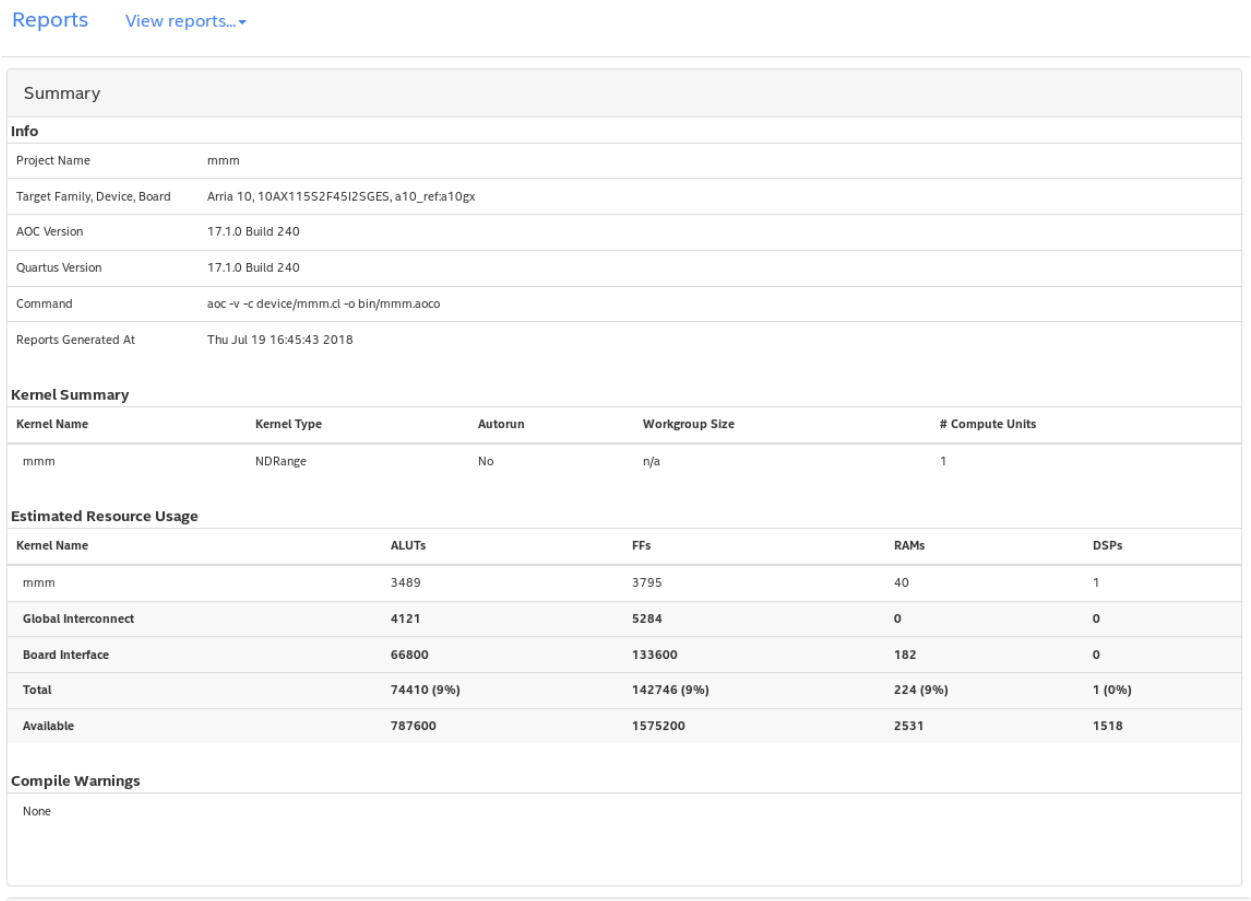
“firefox reports/report.html”

- Select reports: using the shown tab to select different reports.



- Summary:

  This page provides the resource usage.



Reports View reports...

Summary

Info

Project Name	mmm
Target Family, Device, Board	Arria 10, 10AX115S2F45I2SGES, a10_ref:a10gx
AOC Version	17.1.0 Build 240
Quartus Version	17.1.0 Build 240
Command	aoc -v -c device/mmm.cl -o bin/mmm.aoco
Reports Generated At	Thu Jul 19 16:45:43 2018

Kernel Summary

Kernel Name	Kernel Type	Autorun	Workgroup Size	# Compute Units
mmm	NDRange	No	n/a	1

Estimated Resource Usage

Kernel Name	ALUTs	FFs	RAMs	DSPs
mmm	3489	3795	40	1
Global Interconnect	**4121**	**5284**	**0**	**0**
Board Interface	**66800**	**133600**	**182**	**0**
Total	**74410 (9%)**	**142746 (9%)**	**224 (9%)**	**1 (0%)**
Available	**787600**	**1575200**	**2531**	**1518**

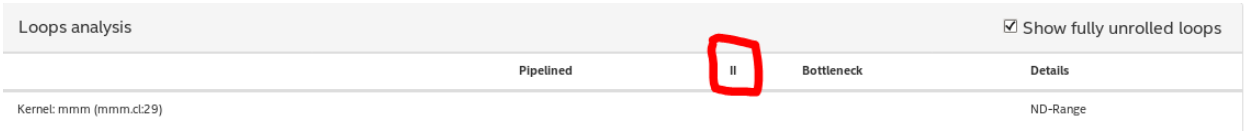
Compile Warnings

None

- Loop analysis:

This page provides the loop unrolling status, pipeline status of the generated kernel. You will see more details in the next lab. Among those, II represents Initial Interval, which denotes how many data can be

fed into the pipeline each cycle. Ideally, this number should be 1, which means there is no bubble in the pipeline.



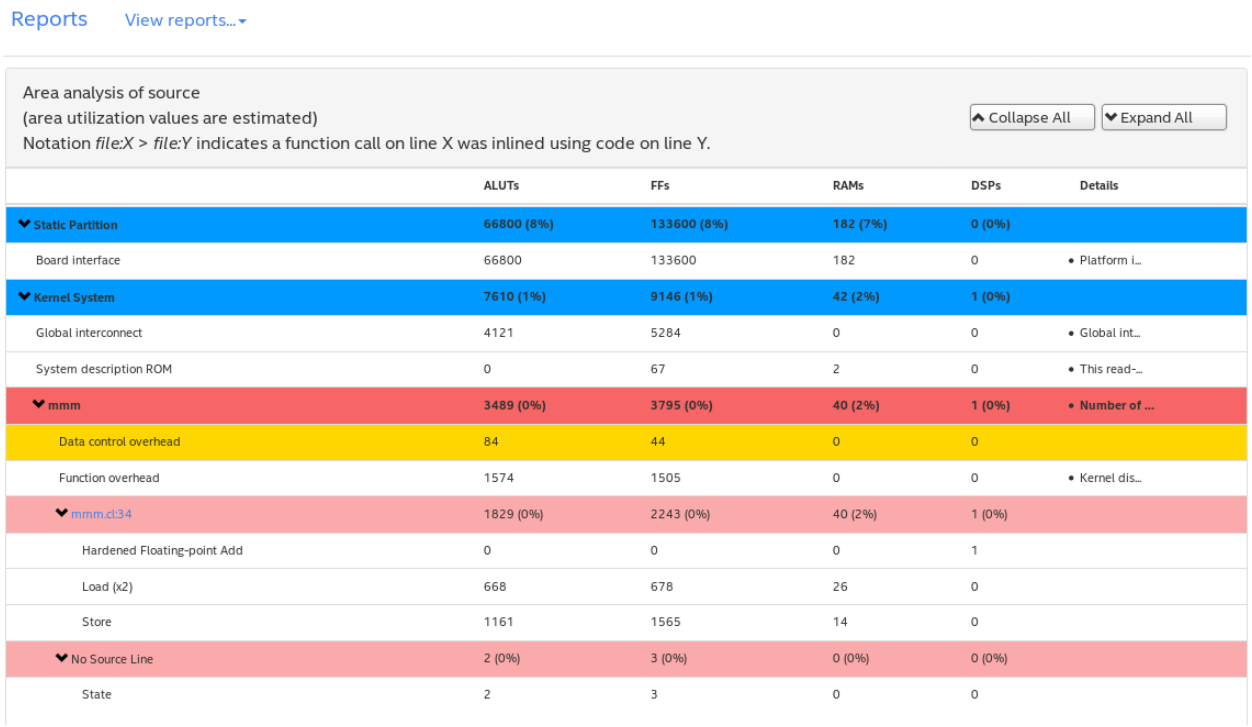
Loops analysis

☑ Show fully unrolled loops

	Pipelined	II	Bottleneck	Details
Kernel: mmm (mmm.cl:29)				ND-Range

- Area analysis of system & Area analysis of source:

These 2 pages gives you detailed resource usage of each module in the generated kernel. Note, this resource usage is just a estimation. After the full compilation, the resource usage may have slight changes.



Reports View reports...

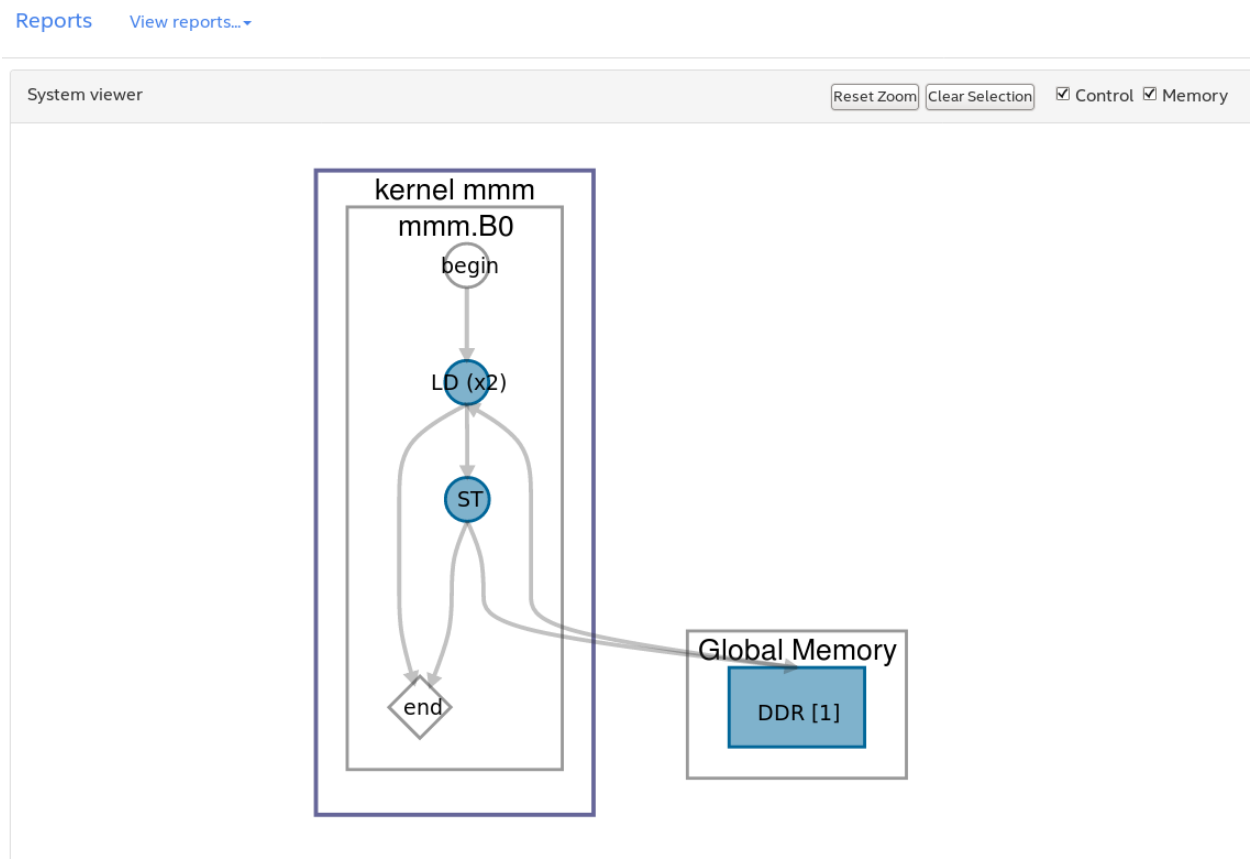
Area analysis of source
(area utilization values are estimated)
Notation *file:X* > *file:Y* indicates a function call on line X was inlined using code on line Y.

Collapse All | Expand All

	ALUTs	FFs	RAMs	DSPs	Details
Static Partition	**66800 (8%)**	**133600 (8%)**	**182 (7%)**	**0 (0%)**	
Board interface	66800	133600	182	0	• Platform i...
Kernel System	**7610 (1%)**	**9146 (1%)**	**42 (2%)**	**1 (0%)**	
Global interconnect	4121	5284	0	0	• Global int...
System description ROM	0	67	2	0	• This read-...
mmm	**3489 (0%)**	**3795 (0%)**	**40 (2%)**	**1 (0%)**	• **Number of ...**
Data control overhead	84	44	0	0	
Function overhead	1574	1505	0	0	• Kernel dis...
mmm.cl:34	1829 (0%)	2243 (0%)	40 (2%)	1 (0%)	
Hardened Floating-point Add	0	0	0	1	
Load (x2)	668	678	26	0	
Store	1161	1565	14	0	
No Source Line	2 (0%)	3 (0%)	0 (0%)	0 (0%)	
State	2	3	0	0	

- System Viewer:

This page provides the memory access pattern of the kernel.



Simulate the Kernel:

The full compilation of OpenCL may take hours to finish, which hugely slows down the developing process. On the other hand, only performs emulation is not enough to quickly collect the performance number, especially if one wants to apply a variety of optimization on the kernel code. The Boston University CAAD Lab provides a simulation workflow. By modifying the OpenCL Board Support Package, we enabled the automated simulation of the generated kernel HDL code, which provides low-level simulation and performance measurements. The entire flow takes less than 10 minutes to finish in general.

In order to use the simulation flow, the developers only need to use the normal OpenCL compilation command, instead of generating the aocx file, the new flow will lead to the simulation process using ModelSim. A simulation script (run_simulation.sh) is provided for your convenience.

"source run_simulation.sh"

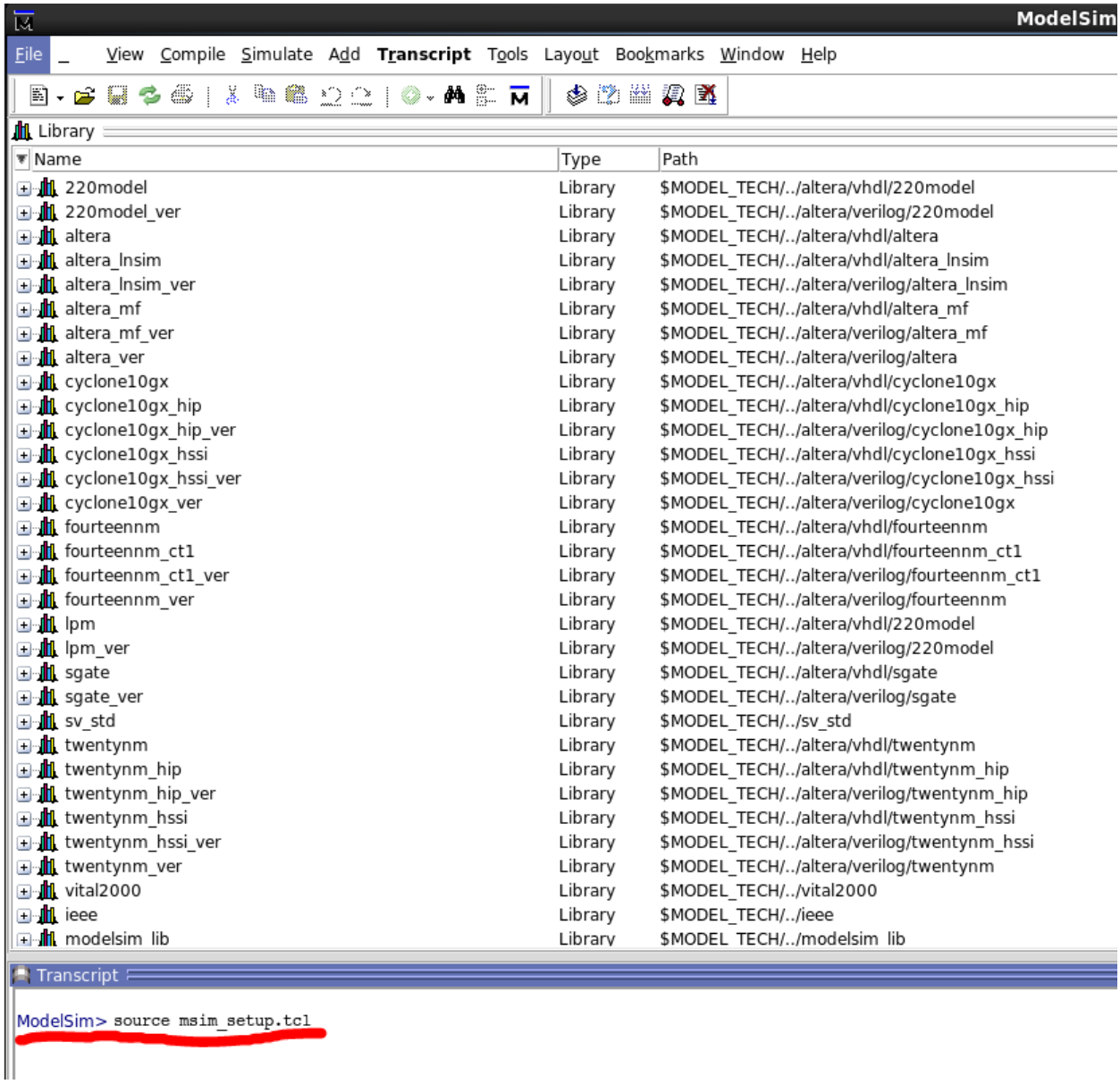
The output should look like:

```
cyang@americano00.eee.hku.hk:~/HK_FPGA_OpenCL_Labs/Labs/lab0$ source run_simulation.sh
****************** Compiling Kernels for Simulation ***********************
aoc: Environment checks are completed successfully.
aoc: If necessary for the compile, your BAK files will be cached here: /var/tmp/aocl/
You are now compiling the full flow!!
aoc: Selected default target board a10gx
aoc: Running OpenCL parser....
aoc: OpenCL parser completed successfully.
aoc: Optimizing and doing static analysis of code...
aoc: Linking with IP library ...
Checking if memory usage is larger than 100%
aoc: First stage compilation completed successfully.
Compiling for FPGA. This process may take a long time, please be patient.
aoc: Hardware generation completed successfully.
aoc: Warning: Cannot find a FPGA programming (.sof) file
****************** Run Emulation ***********************
```

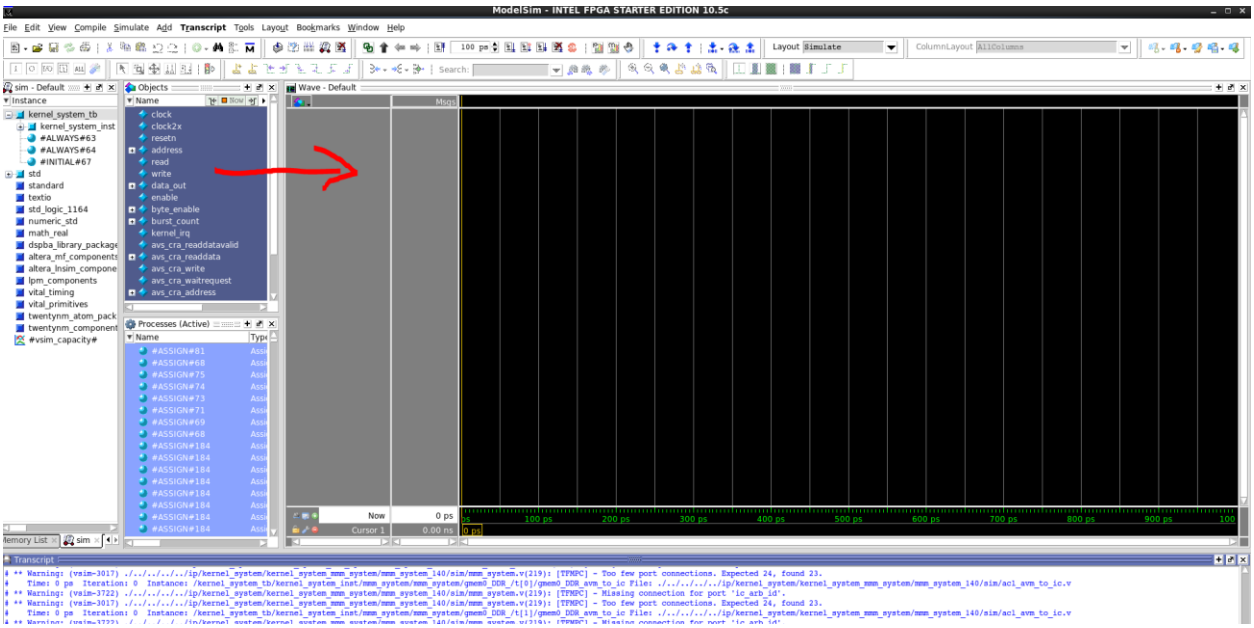
After that, a ModelSim window will be launched. In the transcript window, type:

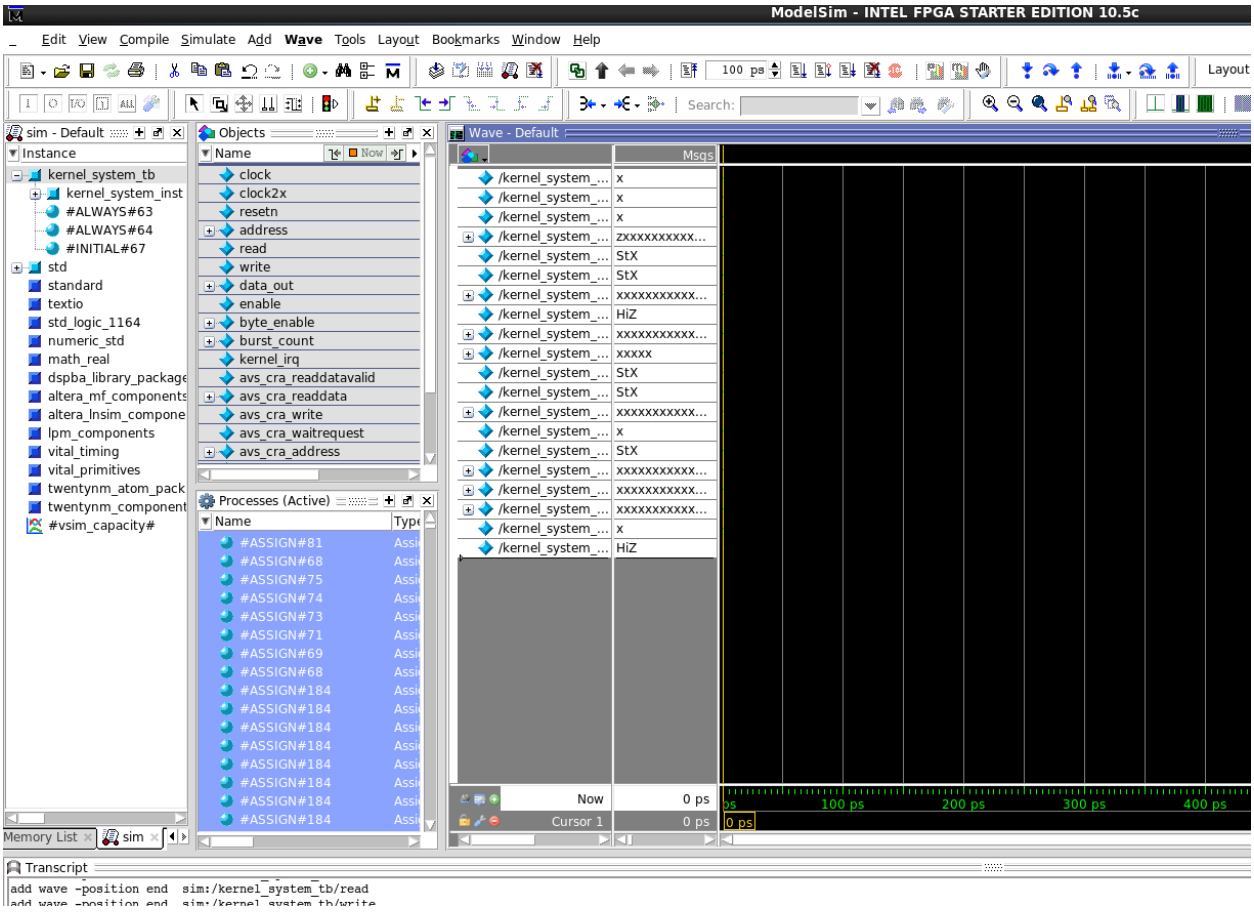
"source msim_setup.tcl"

"ld" (this may take 3-5 minutes)

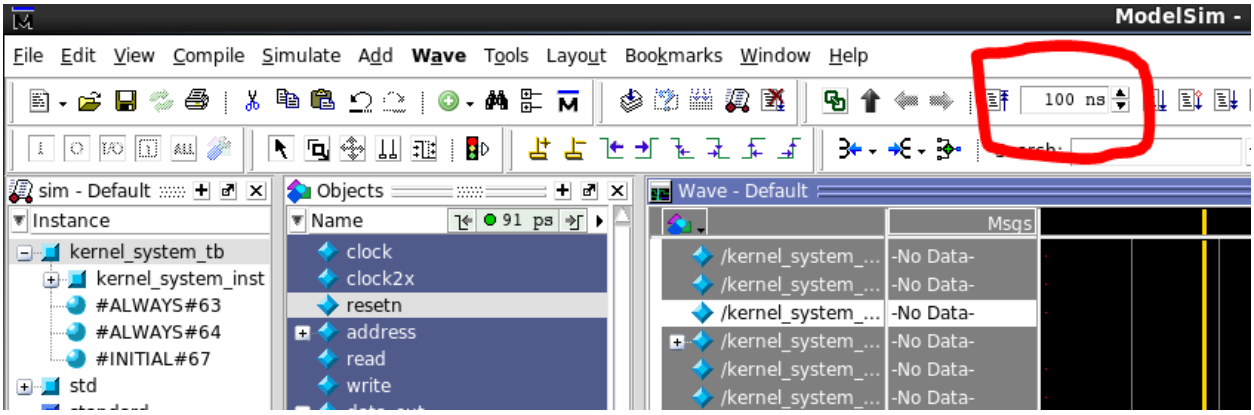


After the “ld” is finished, you will see the simulation window: drag all the signals from the object window to the Wave window:

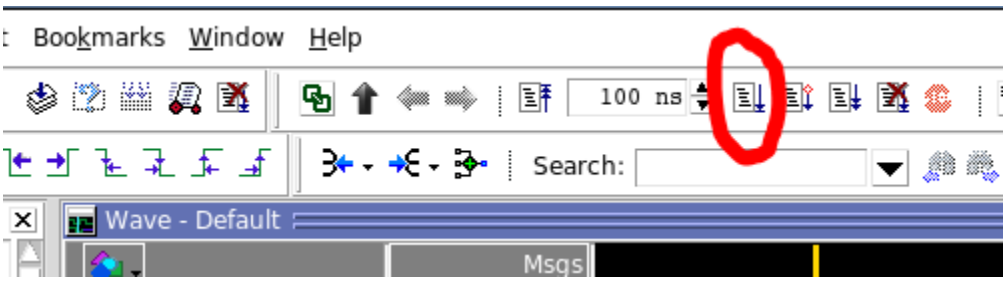




Change the simulation time step to 100ns:

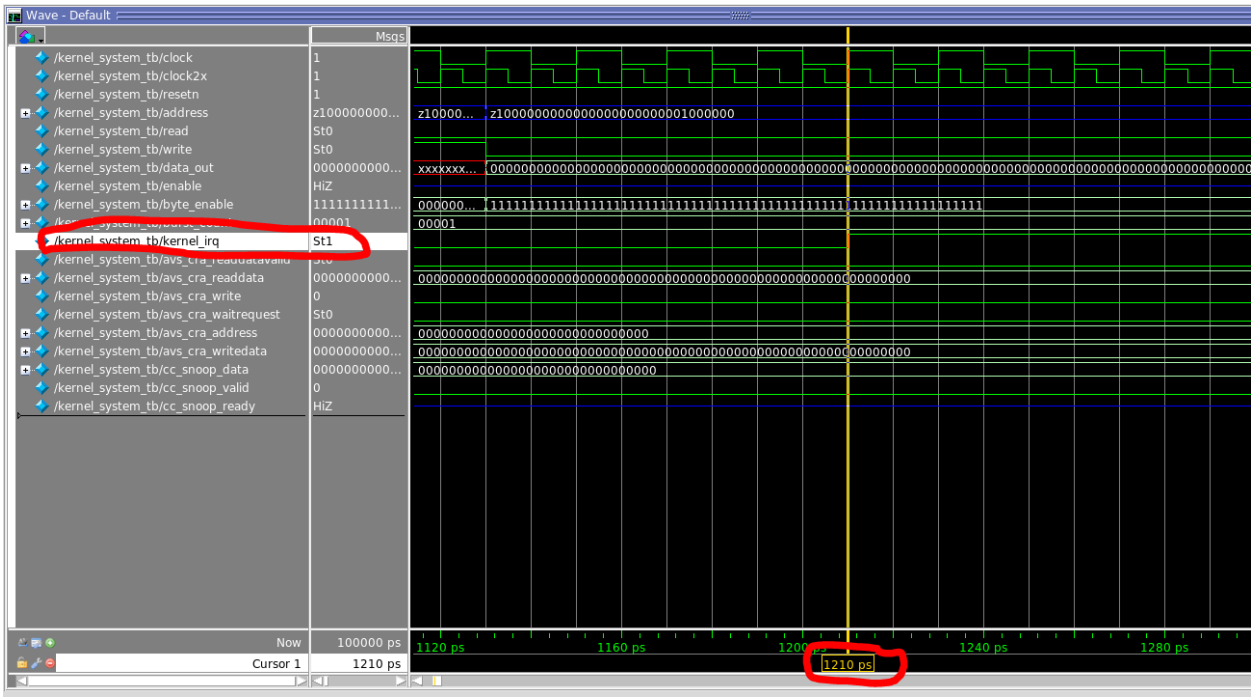


Now launch the simulation by pressing the button to the right of the timestep box:



After the simulation is done, zoom out the simulation waveform:

The key signal we are looking for is the kernel_irq signal, when this one turns to high, denotes the kernel has finished the job. By locating the timestamp of the point this one turns to high, we now know the “latency” of this kernel:



In this testbench, we set the clock cycle as 20ps. So when the kernel finished at 1210ps, it actually runs for 60 cycles which is the latency of the kernel.

Summary:

In this lab, we go over the process of performing OpenCL emulation and simulation, and how to interpreting the OpenCL report and simulation result. In the next lab, we will apply a couple of simple techniques to improve the performance of OpenCL kernel.