# University of Pennsylvania Department of Electrical and System Engineering System-on-a-Chip Architecture

ESE532, Fall 2020	I/O and FPGA Milestone	Wednesday, November 11
-------------------	------------------------	------------------------

**Due:** Friday, Nov. 20, 5:00PM

- 1. Move some part of your design onto the FPGA for acceleration. Writeup should identify what you moved onto the FPGA, how you validated it, and how you tuned it. Identify the current throughput achieved.
- 2. Use the supplied measurement routines (Refer to Tutorial 1) to report the input throughput to your encoder.
- 3. Use the supplied measurement routines to report the maximum real-time throughput the current design can sustain (Refer to Tutorial 2 and see how timers are used. Additionally, notice how you can use getProfilingInfo on cl::Event to get the kernel execution time).
- 4. Turn in a tar file with your FPGA accelerated code to the designated assignment component in canvas (one per group).
- 5. Turn in a tar or zip file with binaries to support execution of your code to the designated assignment component in canvas (one per group).
  - (a) encoder.xclbin for FPGA kernel
  - (b) encoder for OpenCL host code executable
  - (c) decoder executable configured to work with your encoded file and that can be run on the Ultra96. (Most likely, this is just a compilation of the Decoder.cpp we supplied; however, if you chose a different maximum block size, you may need to change CODE\_LENGTH; so give us back one with that change made.) Make sure to compile it with the aarch64-linux-gnu-g++ compiler and test it on the Ultra96. While you could run the decoder on your host machine (which could be Linux/Mac OS/Windows), we will run your decoder on the Ultra96.
    - Your compression program (OpenCL host code) should take one argument:
      - the file name where the program should store the compressed data.

Your program should assume that encoder.xclbin is in the same directory as the host executable.

• Your compression program should start up ready to receive inputs.

We don't expect significant FPGA acceleration on this milestone, but we do want you to start exploring acceleration options.

# Tutorials

## 1. Measuring Ethernet Throughput in the Encoder

In P2, you measured the raw ethernet throughput using iperf3 and got about 895 Mbits/s. Note that, by default iperf3 sent TCP packets to the receiver in the Ultra96, whereas we are using UDP in the project, which is a faster protocol. We will now show you how to measure the input throughput in your encoder.

(a) We have updated the server code we gave you in P2 with some instrumentation. Do the following to get the latest changes:

> cd ese532\_code/ git pull origin master

- (b) Compile the encoder code, copy the binary to the Ultra96 and run it.
- (c) Compile the client code and run the client with the supplied vmlinuz.tar file as follows:

```
./client -f vmlinuz.tar -i 10.10.7.1
```

(d) You should see the following output in the Ultra96 terminal:

```
root@ultra96v2-2020-1:~# ./encoder.elf
setting up sever...
server setup complete!
write file with 69079040
------ Key Throughputs ------
Input Throughput to Encoder: 1079.33 Mb/s. (Latency: 0.512015s).
root@ultra96v2-2020-1:~# diff vmlinuz.tar output_cpu.bin
```

You should see the following output in the host terminal:

filename is vmlinuz.tar ip is set to 10.10.7.1 payload\_size is 8192 bytes\_read 69079040

(e) You can see that we are indeed getting about 1 Gb/s input throughput. You can look into encoder.cpp and see that we are using a timer to measure the total latency taken by the call: server.get\_packet(input[writer]) (ignoring the first call which waits for the first packet to arrive). Later in the code, we calculated the throughput as follows:

```
float ethernet_latency = ethernet_timer.latency() / 1000.0;
float input_throughput = (bytes_written * 8 / 1000000.0) / ethernet_latency;
```

- (f) Note that it is very important that you verify the output using diff. You can lose packets if your encoder cannot keep up with the input throughput, in which case you should use the -s option in the client to transfer at a lower speed.
- (g) Note that we have increased the PAYLOAD\_SIZE to 8192 bytes in the updated code.

### 2. FPGA Acceleration Tutorial: Bloom Filter

Using bloom filter as the application, this tutorial shows you:

- a significant speedup  $(8\times)$  when computations are offloaded on the FPGA efficiently.
- how to write an HLS kernel for a CPU implementation (using ap\_uint, hls::stream, and pragmas).
- how to achieve communication-compute overlap using sub-buffers.
- (a) Clone the ese532\_code repository using the following command:

git clone https://github.com/icgrp/ese532\_code.git

If you already have it cloned, pull in the latest changes using:

cd ese532\_code/ git pull origin master

The code you will use for this section is in the vitis\_tutorials/bloom directory. The directory structure looks like this:

```
bloom/
cpu/
fpga/
MurmurHash2.c
common.h
compute_score_fpga_kernel.cpp
compute_score_host.cpp
hls_stream_utils.h
main.cpp
sizes.h
xcl2.cpp
xcl2.hpp
```

The cpu folder has a standalone CPU implementation of the bloom filter, which you can compile using the Vitis GUI flow from P2 and run it. The main.cpp code in the fpga folder has the OpenCL host code. The top level HLS function is in compute\_score\_fpga\_kernel.cpp. We will now show how to use the Vitis GUI flow to compile OpenCL and HLS code (if you would like to use a Makefile, modify and use the one from Homework 6).

(b) Use the instructions from Milestone 2 and start vitis. Create or use an existing workspace, create a new application project and use the provided platform.

		Eclips	e Launcher			<u> </u>
elect a direct	tory as wo	kspace				
/itis IDE uses th	- ne workspac	e directory to store	its preferences a	nd developr	nent artifa	cts.
		,				
	1. 1.1.1.1		522 1 (			
Norkspace: /	media/lilbiri	b/research/code/es	e532_code/projec	:t 🔻	Browse	
Use this as t	ho dofault a	od do pot ock opoir				
Use this as t		nu uu not ask agan	1			
Restore othe	г Workspace	2				
Recent Works	spaces					
<u>02-bloom</u>						
projects						
concurrent ke	rnel executi	on				
linux files						
app						
				Canc	el	Launch
<b>atform</b> hoose a platform for	your project. You	New Ap	plication Project ion from XSA through the	'Create a new p	latform from ha	o ( ardware
atform hoose a platform for (SA)' tab.	your project. You	New Ap u can also create an applica The create a new platfor	plication Project ion from XSA through the m from hardware (XSA)	'Create a new p	latform from ha	ardware
atform hoose a platform for (SA)' tab. Select a platform Find:	your project. You h from repositor	New Ap I can also create an applica R Create a new platfo	plication Project ion from XSA through the m from hardware (XSA)	'Create a new p	latform from ha	ardware Manage
atform hoose a platform for (SA)' tab. Select a platform Find: Name	your project. You In from repositor	New Ap I can also create an applica Create a new platfo Board Flow	plication Project ion from XSA through the m from hardware (XSA) Vendor PR	'Create a new p	latform from ha	ardware Manage
atform hoose a platform for (SA) tab. Select a platform Find: Name Brese532, hwo, pt	your project. You n from repositor () fm (custom)	New Ap I can also create an applica I can also a new platfor I create a new platfor Board Flow ULTRASOV, Embedded A	plication Project ion from XSA through the m from hardware (XSA) Vendor Pe vendor / Pe vendor / M	'Create a new p th edia/lilbirb/res	latform from ha	ardware Manage
atform hoose a platform for Saltab. Select a platform Finds Name Eleces32_hw6_pf Ceneral Info Name: eset Part: xzzz Part: xzzz	your project. You from repositor; © 'm (custom) 532 hw6 pfm 532 hw6 pfm 532 hw6 pfm 532 hw6 pfm	New Ap a can also create an applica board Flow ULTRASOV Embedded A Acceleration Ress Clock Frequenci PL 0 PL 0	plication Project ion from XSA through the m from hardware (XSA) Vendor Pr vendor Pr avmet.com /m surces Frequency (MH2) 120.00000 130.0000000 130.0000000 130.0000000000 130.000000000 1	Create a new p	+ Add + Add carch/git/avnet/ is n name _cortexas3 CPI os	Details ContexeS3
atform hoose a platform for SA) tab. Select a platform Find: Name Stesse532_hw6_pf Platform info Ceneral Info Name: eset Part: xc20 Family: zc20 Family: zc20	your project. You from repositor from [custom] 532 hw6 pfm 1329 shva484-14 uplus n	New Ap can also create an applica P C Create a new platfor Board Flow ULTRASOY, Embedded A Acceleration Ress Clock Frequenci Clock PL 0 PL 2 PL 2 PL 4	plication Project ion from XSA through the m from hardware (XSA) Vendor Pa amet.com /m ccel amet.com /m from sets f5 Fequency (MH2) 75.00000 150.00000 200.00000 200.00000 200.00000	*Create a new p th edia/filbirb/rese Domain Deta Domains Domains Domains Domains	+ Add + Add carcly(glt/avnet) is a name _cortexa53 CPI OS	Manage Manage petalinux/projec
atform hoose a platform for GA) tab. Select a platform Find: Mame Selects a platform Selects a platform Platform Info Ceneral Info Name: eset Part: xzz Family: zync Description: eset532_hw6_pfr	your project. You from repositor from (custom) 532 hw6 pfm 1380-5bv4884-14 upplus n	New Ap a can also create an applica b create a new platfor Create a new platfor Board Flow ULTRAGOV, Embedded A Acceleration Ress Clock Frequenci Clock	plication Project ion from XSA through the m from hardware (XSA) Vendor Pa vendor Pa exect avendor Pa avents.com /nr scal aven	'Create a new p th edia/IIIbirb/res Domain Deta Domains Domains Ilnux on psu	Latform from h + Add arct/glt/avnety lis name _cortexes3 CPI _Cortexes3 CPI	Manage petalinux/project
atform hoose a platform for Sol tab. Select a platform Find: Name Platform Info Ceneral Info Name: eset Part: xzzz Part: xzzz P	your project. You from repositor. im (custom)	New Ap a can also create an applica board Flow UUTRAGOV Embedded A Acceleration Ress Clock Frequenci PL 0 PL 0 PL 3 PL 4	plication Project ion from XSA through the wrom hardware (XSA) vendor Pro conditional and the second	'Create a new p th edia/filbirb/rese Domain Deta Domains Domains Inux on psu	Latform from ha	Industry  Indus
atform hoose a platform for SAY tab. Select a platform Find: Name Platform info Ceneral info Name: ese! Part: xzzy Family. zyno Description: ese532_hw6_pfr	your project. You from repositor () from [custom] 532 hwó p/m 1320 hwó p/m 1320 hwó p/m 1320 hwó p/m 1320 hwó p/m 1320 hwó p/m	Acceleration Ress Clock Frequent Clock Frequ	plication Project ion from XSA through the m from hardware (XSA) Vendor Provide excel avnet.com /m avnet.com /m solution Solution	"Create a new p th edia/illoirs/res Domain Deta Domains Domains	Latform from ha	Manage petalinuv/projec

(c) Choose Empty Application from the SW acceleration templates as follows:

New Applic	ation Project 🛛 🖉 😣
Templates Select a template to create your project.	•••
Available Templates: Find:  SW acceleration templates  Tempty Application Vector Addition SW development templates Empty Application(++) Linux Empty Application Linux Hello World Vitis IDE Examples Vitis IDE Libraries	Empty Application Creates a new Empty application
0	<back next=""> Cancel Finish</back>

(d) Right-click on the **src** folder and click on import sources. Import the source files for this project as follows:

Import So	urces	
File system Import resources from the local file system.		
From directory: /media/lilbirb/research/code/ese5	32_code/vitis_tutorials/bl 👻	Browse
V № fpga		
Filter Types Select All Deselect All		
Into folder: bloom/src		Browse
Options Overwrite existing resources without warning Create top-level folder Advanced >>		
(?)	Cancel	Finish

(e) Click on bloom.prj from the Explorer and change the Active build configuration to Hardware as shown below:

vitis_workspace - bloom/bloo	m.prj - Vitis IDE	
₩ Φ <b>*</b> Φ *		Quick Access
👗 bloom_system 🛛 🛠 bloom 🛱		🗢 🗉 🔠 🗄 Outline
X Application Project Settings	Active build configuration: Hardware	An outline
General	Options	
Project name: <u>bloom</u>	Target: Hardware	
Platform: ese532 hw6 pfm	Host debug:	
Runtime: OpenCL	Kernel debug:	
Number of devices: 1 – +	Kernel debug mode: Waveform	
	Report level: Default	
	Hardware optimization: Default optimization (-00)	$\sim$
Hardware Functions	E E 🗲 🗡	4
Name Compute Units Port Data Width	Max Memory Ports	
	Vitis_workspace - bloom/bloo Solutions	Vitis_workspace - bloom/bloom.prij - Vitis IDE

(f) Add an xclbin container by clicking on the button circled in red below. Rename the container to runOnfpga\_hw by clicking on the name binary\_container\_1. This is the xclbin name that the host code uses.



(g) Now click on the button circled in blue below. Choose the runOnfpga function as the hardware function.



(h) From the Assistant view, double click on Hardware as follows:



(i) Click on Hardware→runOnfpga\_hw→runOnfpga. This brings the screen where you can specify the number of compute units, compiler options for the kernel, assign different ports to inputs etc. Keep the defaults for now:

					Ha	ardware Function	Settings		 		a 😣
type filter t 🛞 🗎 🖿	🕖 runOnfpga										⇔ • ⇔ • •
<ul> <li>➡ ➡ bloom_system</li> <li>➡ ➡ bloom</li> <li>▶ &lt; &lt; Emulation-SW</li> </ul>	Name: Compute units:	runOnfpga									
Kenulation-HW	Max memory ports:										
✓ Nardware ✓ Intervention of the second	Port data width:	Auto 👻									
📕 runOnfpga	Extra source files:										
	V++ compiler options:										
	Compute Unit Settings										
	Name	Memory	SLR	ChipScope Debug	Protocol Checker	Data Transfer	Execute Profiling	Stall Profiling			
	👻 🐔 runOnfpga	Auto	Auto			None					
	runOnfpga_1	Auto	Auto			None					
	output_flag	s Auto				None					
	input_words	s Auto				None					
	- bloom_filter	r Auto				None					
	Cotal_size										
	Refresh										
	V++ Compiler Comman \${XILINX_VITIS}/ target hw compile -I"/src/" config common- -o"runOnfpga_hw	d Line bin/v++ config.cfg ga hw-runOnfpg .build/runOnfp	<u>a-compi</u> ga.xo"	le.cfg "/src/compute_sc	ore_fpga_kernel.cp	p."					
										Revert	Apply
										Cancel	Apply and Close

(j) Click on bloom.prj. Check out the Hardware optimization option where you can change the optimization level for the hardware function. Additionally, recall from P2 that you can change the optimization level of the host code from the C/C++ build settings. Now click on the build button on the menu bar to start compilation:

		vitis_	workspace - bloom/bloo	m.prj - Vitis IDE					- " 😣
Eile Edit Search Xilinx Project Window Help									
🖆 • 🖩 🐚 🗧 • • • • • • 🖉 • 🖾 🖉 🖬	$\Leftrightarrow \Leftrightarrow \bullet \bullet \bullet$						Q	uick Access 🛛 🖓 Design	🎋 Debug
🖳 Explorer 🛙 📄 🔁 🔻 🗖 🗖	🛎 bloom_system 🛛 😹 bl	oloom 🛿 🖻 main.cpp	compute_score_fp	ga_kernel.cpp				문 Outline 회	- 0
shoom_system [ese532_code master] [ese532_hw6_pfm]	X Application Proje	ect Settings			Active build co	onfiguration: Hardware	• @	An outline is not available.	
►	General			Options					
► 📴 > Emulation-HW	Project name: blo	moom		Target:	Hardware				
► @Ş > Emulation-SW ► @\$ > Hardware	Platform: ese	e532_hw6_pfm		Host debug:					
	Runtime: Op	DenCL		Kernel debug:			_		
▶ ∰ common.h	Number of devices: 1			Kernel debug mode:	Waveform				
<ul> <li>is compute_score_rpga_kernet.cpp</li> <li>is compute score host.cpp</li> </ul>				Report level:	Default		$\sim$		
▶ Is hls_stream_utils.h				Hardware optimication	h: Default optimization (-0	<u>=(0)</u>	$\sim$		
G main.cpp	Hardware Functions					😑 🖻 🖻 🗡 🗙	A		
▶ is sizes.h	Name	Compute Units	Port Data Width		Max Memory Ports				
Kalandari Kal	🕶 🖻 runOnfpga_hw								
K xcl2.hpp	🕖 runOnfpga	1	Auto		•				
4 bloom_system.sprj									
♥ Bloom_system [System]     ♥ Bloom [Enbedded OpenCL]     * < EmulationsW [Software Emulation]     * < Emulation-HW [Hardware Emulation]     * < Hardware [Hardware]									
	📮 Console 🛿 🚺 Problem	ns 🗐 Vitis Log  i Gui	dance 🛛 🤴 🗘 🛸			Emulation Console ន		<b>k</b> 🖻	~
	Build Console [bloom, Hard	dware]							

(k) Once the compilation completes, open the Hardware folder from the Explorer. The binaries are in the package/sd\_card folder.



(l) Copy the binaries and the xrt.ini to the Ultra96 as follows and then reboot the Ultra96.

$\langle \rangle$			D	research code	ese532_code	vitis_works	pace bloom	Hardware	package	sd_card 🔫	Q			i -		- 0	۲
③ Recent	Ô	0100 0010 1001	DLDD DDLD LODL	0100 0010 1001	>_		0100 0010 1001	D100 D010 1001									
★ Starred	bloom	boot.scr	BOOT.BIN	image.ub	init.sh	platform_	runOnfpga_	system.dtb									
습 Home						Uesc.txt	INV.ACIDITI										
Desktop																	
Docur 🔳			lilb	irb@stingy: /media,	'lilbirb/researc	h/code/ese532_	code/vitis_works	pace/bloom/Ha	rdware/pacl	age/sd_card			Q = -		≥.		
Down rdware/packag     cost010 10 7	ge/sd_card\$ sc	p BOOT.BIN b	poot.scr imag	e.ub root@10.10	.7.1:/mnt/sc	-mmcblk0p1/											
∬ Music BOOT.BIN	.1 3 password.		100%	6627KB 38.0MB	/s 00:00												
Picturimage.ub lilbirb@stin			100% 100%	8117KB 33.0MB	/s 00:00 kspace/bloor		ckage/sd card	S scp bloom	runOnfoda	hw.xclbin ro	ot010.10.7	.1:~/					
⊟ Videoroot@10.10.7 bloom	.1's password:		100%	1045KB 4.5MB	/s 00:00					=							
Trash runOnfpga_hw	<pre>.xclbin gy:/media/lilb</pre>		100% /code/ese532	5468KB 43.2MB _code/vitis_wor	/s 00:00 kspace/bloor			\$									
🖹 resea																	
+ Other																	

(m) Run the code using the following commands in the Ultra96:

ifconfig eth0 10.10.7.1 netmask 255.0.0.0
export XILINX\_XRT=/usr
./bloom 40000 64

You should see the following output in the terminal:

```
root@ultra96v2-2020-1:~# ./bloom 40000 64
Initializing data
Creating documents - total size : 559.858 MBytes (139964416 words)
Creating profile weights
[ 1018.547572] [drm] Pid 769 opened device
[ 1018.551450] [drm] Pid 769 closed device
[ 1018.558627] [drm] Pid 769 opened device
Loading runOnfpga_hw.xclbin
[ 1018.617733] [drm] zocl_xclbin_read_axlf The XCLBIN already loaded
[ 1018.617765] [drm] zocl_xclbin_read_axlf 3c650f2f-9cc2-408a-8c92-0ec3
[ 1018.633496] [drm] bitstream 3c650f2f-9cc2-408a-8c92-0ec3bc335ce3 loc
[ 1018.641197] [drm] Reconfiguration not supported
[ 1018.652995] [drm] bitstream 3c650f2f-9cc2-408a-8c92-0ec3bc335ce3 unl
Processing 559.858 MBytes of data
Splitting data in 64 sub-buffers of 8.748 MBytes for FPGA processing
_____
[ooqueu]: Completed buffer migrate
[ooqueu]: Completed buffer migrate
[ooqueu]: Completed buffer migrate
[ooqueu]: Completed buffer migrate
```

[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	${\tt Completed}$	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	${\tt Completed}$	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	${\tt Completed}$	buffer	migrate
[ooqueu]:	${\tt Completed}$	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
waiting			
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
waiting			
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	Completed	buffer	migrate
[ooqueu]:	${\tt Completed}$	buffer	migrate
waiting			

•

Executed Software-Only version	I	11779.4325 ms		
Executed FPGA accelerated version		1414.5707 ms	( FPGA	247.878 ms

Verification: PASS

- (n) The verbose outputs in the terminal is caused by the call set\_callback(flagDone, "oooqueue"); in the host code. This is a helper function in xcl2.hpp that prints out the state of an OpenCL event. You can use it to debug OpenCL calls. In addition, you can also use the OCL\_CHECK macro from xcl2.hpp to see if an OpenCL call succeeded.
- (o) Copy the generated run summary and csv files to your host computer and open vitis analyzer. You can see overlap of kernel execution with data transfer and OpenCL API calls.

kernel.xclbin (System Run)	×									
Summary × Application Tir	neline ×									802
Q 🔛 @ Q 🔀 📲 I	±   ±   +									۰
						20	,775.505 ms			^
Name	Value	 20,765.000000 ms	2	0,770.000000 m	IS	20, 775	.000000 ms	20	,780.000000 ms	
∨ Host		 								
V Open CL API Calls										
General 0	clSetKernelArg									
General 1	INACTIVE									
Queue: 55C93B8780	INACTIVE		81 88			1 1	<u>                                     </u>			
V Data Transfer										
Read								_		
Row 0	">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>					-				
Write		_				L I.		_		
Row 0	"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX		-00	× "UX			"UX "UX.	*0x	····	
Row I						2		~ .		
Kernel Enqueues	3			2	3	2 3	4	8 4	5	6 5
Row U	runontpga	L		rununipga		aux Octobra		runun rpga	aux Oofe an	
Row 1	runonipga					runompga	eupOn f		runorrpga	_
Row 2	runonipga						- Griori	pya cun0	ofora	
Row 4									run0	nfoga
Bow 5										run
Bow 6										
Bow 7										
Row 8		 								
Row 9		 								
Row 10										
Row 11		 								
Row 12	_	 								
Row 13										
Row 14		 								
Row 15	—	 								
Row 16		 								
Row 17										
Row 18		 								
Row 19										
Row 20										
Row 21		 								
Row 22		 								
Row 23		 								
Row 24	_									
	$\langle \rangle \rangle \langle \rangle$					_				>

(p) If you scroll forward in the timeline, you can see overlap between computation in the cpu and the fpga as shown below. From the output in 2m, these calls



correspond to the waiting... print outs. You can check in the host code, how we wait for a cl::Event to finish based on a condition, and when the event notifies that it's finished, we start executing the cpu code, so that it overlaps with the fpga execution:

```
needed += size;
if (needed > available) {
   clWaitForEvents(1, (const cl_event *) &flagWait[iter]);
   std::cout << "waiting..." << std::endl;
   available += subbuf_doc_info[iter].size / sizeof(uint);
   iter++;
}
```

(q) Now run with a different ITER value and look at the updated trace:

```
./bloom 40000 128
```



You can see that since the kernel execution time gets smaller as you increase the iteration number, the next kernel execution starts almost immediately.

(r) Running a sweep on the number of iterations, we see that ITER=32 is the most performant for this design:

```
./bloom 40000 8
Executed FPGA accelerated version | 1413.3252 ms (FPGA 305.796 ms)
Executed Software-Only version | 11780.3703 ms
_____
Verification: PASS
./bloom 40000 16
Executed FPGA accelerated version | 1396.5072 ms (FPGA 298.034 ms)
Executed Software-Only version | 11770.1858 ms
 _____
                          _____
Verification: PASS
./bloom 40000 32
Executed FPGA accelerated version | 1391.2062 ms (FPGA 284.336 ms)
Executed Software-Only version | 11768.1306 ms
_____
Verification: PASS
./bloom 40000 64
Executed FPGA accelerated version | 1414.5707 ms (FPGA 247.878 ms)
```

- (s) This concludes a top-down walk-through of this tutorial. To learn more about this design, read the following in-order:
  - i. Overview of the Original Application
  - ii. Architect a Device-Accelerated Application
  - iii. Implementing the Kernel
  - iv. Data Movement Between the Host and Kernel

Note that the tutorial is written for data center cards. Some of the parameter choices, such as port data width, DDR memory etc. should be reconsidered for the Ultra96 to get optimal performance (refer to this paper: Unexpected Diversity: Quantitative Memory Analysis for Zynq UltraScale+ Systems).

#### 3. Using Multiple Compute Units

The code you will use for this section is in the vitis\_tutorials/mult\_compute\_units directory. The directory structure looks like this:

```
mult_compute_units/
host.cpp
vadd.cpp
xcl2.cpp
xcl2.hpp
```

The host.cpp code has the OpenCL host code. The top level HLS function is in vadd.cpp.

- (a) Create an application project as described in Tutorial 2, compile and run the project.
- (b) The system diagram in vitis analyzer looks like:



# which shows that there is one vadd kernel. The application timeline looks like:



From the application trace, we can see that although the host scheduled all kernel executions concurrently, the second, third and fourth execution requests are delayed as there is only one compute unit on the FPGA.

(c) Increase the number of compute units to 4 and assign separate ports by going to the window mentioned in 2i. Compile and run the updated configuration. The vitis analyzer system diagram would look like:

nmary     ×     Platform Diagram     ×     System Diagram     ×				
		НРО	vadd_1 in1 out_r size	Profile Calls: N/A Utilization: N/A Total: N/A Average: N/A
	ZYNQ	НР1	vadd_2 in1 in2 out_r size	Profile Calls: N/A Utilization: N/A Total: N/A Average: N/A
		HP2	vadd_3 in1 in2 out_r -size vadd	Profile Calls: N/A Utilization: N/A Total: N/A Average: N/A
		HP3	vadd_4 in1 in2 out_r -size	Profile Calls: N/A Utilization: N/A Total: N/A Average: N/A

kernel.xclbin (System Run) ×											
Summary × Platform Diagram	n × System D	iagram × 🖌	Application Timeline	×							
Q 🗑 🔍 Q 🔀 📲 🕅	<b>H</b>   ±   ±   +	F   Te   +F   H									
										818.086 ms	
Name	Value		200.000000 ms	300.000000 ms	400.000000 ms	500.000000 ms	600.000000 ms	700.000000 ms	1800	000000 ms	1900.0
✓ Host ✓ Open CL API Calls											
General	INACTIVE			c10	CreateProgramWith		clFinis	h			
Queue: 557A2307A0 V Data Transfer	INACTIVE										
~ Read											
Row 0	"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX										
Row 1	*XXXXXXXXXXXX										
V Write											
Row 0	"X0000000000"					-					
Row 1	"X0000000000"										
✓ Kernel Enqueues							4	3	2		
Row 0							vadd				
Row 1							vadd				
Row 2							vadd				
Row 3							vadd				
V Device "edge-0"											
> Binary Container "kernel"											

The application timeline looks like:

You can now see that the application takes advantage of the four compute units, and that the kernel executions overlaps and executes in parallel.

(d) Look into the host code and learn how the multiple compute units are utilized:

```
for (int i = 0; i < num_cu; i++) {
    int narg = 0;
    // Setting kernel arguments
    OCL_CHECK(err, err = krnls[i].setArg(narg++, buffer_in1[i]));
    OCL_CHECK(err, err = krnls[i].setArg(narg++, buffer_output[i]));
    OCL_CHECK(err, err = krnls[i].setArg(narg++, chunk_size));
    // Copy input data to device global memory
    OCL_CHECK(err, err = q.enqueueMigrateMemObjects( { buffer_in1[i],
        buffer_in2[i] }, 0 /* 0 means from host*/));
    // Launch the kernel
    OCL_CHECK(err, err = q.enqueueTask(krnls[i]));
}</pre>
```

You can see from the code that by creating an array of kernels and enqueueing them in a loop, you can utilize the multiple compute units.

#### 4. Streaming Kernel to Kernel Memory Mapped

The code you will use for this section is in the vitis\_tutorials/streaming\_k2k\_mm directory. The directory structure looks like this:

```
streaming_k2k_mm/
host.cpp
krnl_stream_vadd.cpp
krnl_stream_vadd_vmult.ini
krnl_stream_vmult.cpp
xcl2.cpp
xcl2.hpp
```

The host.cpp code has the OpenCL host code. There are two disjoint HLS kernels: krnl\_stream\_vadd.cpp and krnl\_stream\_vmult.cpp. krnl\_stream\_vadd\_vmult.ini specifies how the two kernels are connected with each other. Read about the tutorial from here and then continue.

(a) Create an application project as described in Tutorial 2. Add the two kernels as hardware functions, add the V++ linker option:



--config ../src/krnl\_stream\_vadd\_vmult.ini

(b) Compile and run the project. The system diagram in vitis analyzer looks like:



which shows that the two kernels are reading from the DRAM and are also connected via a stream connection. The application timeline looks like:

0	∓   ≜   st														
→ bina	ary_container_1.xclbin ()	binary_container_1.xclbin (Sys	stem Run) ×												
•	Summary System Diagram	m Summary x Application Timeline x System Diagram x Platform Diagram x							8 0 8						
•	Platform Diagram	Q 11 Q Q X + H	N 1 to 1 to 1 H	F   Fe   aF   B	4										¢
	Run Guidance												4,209.263 ms		^
	Profile Summary	Name	Value	2,500.0	00000 ms		3,000.000000 m		3, 500. 0	00000 ns		4,000.000000 ns		4,500.0	00000
• -	Application Timeline	∨ Host													
		V Open CL API Calls													
		General	clFinish	clCre	clCrclCr	•				clFinish					
		Queue: 55C77CA770	INACTIVE			1				-					
		V Data Transfer													
		~ Read													
		Row 0	"20000000000"												
		~ Write													
		Row 0	"20000000000"			"0X4810"	0								
		Row 1	"20000000000"			*0×401	D								
		✓ Kernel Enqueues	2							2					
		Row 0	krnl_stream_vadd							krnl_stre	an_vadd				
		Row 1	krnl_stream_vmult							krnl_stre	am_vmult				
		V Device "edge-0"													
		> Binary Containey_container_1"													

From the application trace, we can see that the two kernels are running concurrently.

## 5. Using Faster Clocks

(a) In Homework 6, we saw the our platform provides multiple clocks:

**ESE532** 

======================================	mation						
======================================	armatton ======= ese532_hw6_pfm /medla/lilbirb/research/git/avnet/petalinux/projects/ese532_hw6_pfm/export/ese532_hw6_pfm/ese532_hw6_pfm.xpfm						
======================================	Shell) Informat	=== ion					
Vendor: Board: Name: Version: Generated Version: Software Emulation: FPGA Family: FPGA Device: Board Vendor: Board Vandor: Board Name: Board Part: Maximum Number of Co	ompute Units:	=== avnet.com ULTRA96V2 ULTRA96V2 2020.1 1 0 2020.1 1 0 2ynquplus xczu3eg avnet.com avnet.com:ultra96v2:1.1 xczu3eg-sbva484-1-i 60					
Clock Information Default Clock Inde Clock Index: Frequency: Clock Index: Frequency: Clock Index: Frequency: Clock Index: Frequency: Clock Index: Frequency: Clock Index: Frequency: Clock Index: Frequency: Clock Index: Frequency:	<pre>ex: 0 0 150.000000 1 300.000000 2 75.000000 3 100.000000 4 200.000000 5 400.000000 6 600.000000</pre>						
Resource Availabilit  Total  LUTS: 57915 FFS: 126868 BRAMs: 212 DSPs: 360	:= :y :=						

(b) We can assign faster clocks to our kernels in Tutorial 4. You can specify them in a configuration file and pass it in the V++ Linker Options. Looking at the krnl\_stream\_vadd\_vmult.ini, you can see that we have assigned Clock Index 1 (300 Mhz) to the kernels:

```
[connectivity]
stream_connect=krnl_stream_vadd_1.out:krnl_stream_vmult_1.in2:64
```

```
[clock]
id=1:krnl_stream_vadd_1
id=1:krnl_stream_vmult_1
```

where the format of the specification is id=<clock index>:<compute unit name>. You should start with a slower clock in your project so that you can meet timing easily. After you have made HLS and host code optimizations, you can try increasing the clock frequency until your design fails to meet timing.

(c) You can check if the clocks were correctly assigned by opening the vivado project as instructed in Homework 6:



You can see from the vivado block diagram that clock index 1 is assigned. Moreover, you can also see that an AXI Stream FIFO is connecting the two kernels.