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

ESE532, Fall 2020

Design and Function Milestone

Wednesday, November 4

Due: Friday, Nov. 13, 5:00PM

Group: Develop functional code. Identify design space options. Writeup (single turn-in for group).

1. Identify major design space axes that could be explored for your implementation.
   - For this milestone, aim for breadth (quantity of options)
   - Each axis description can be 2–3 sentences. Identify challenge being addressed, basic solution opportunity, and continuum. A single point in the design space is not a continuum; except in rare cases, this should capture a range of potential parameter values.
   - Include a simple equation to illustrate ideal benefit (e.g., running N tasks in parallel reduces runtime by a factor of N; $T(N) = T(1)/N$) and the associated resource costs.
   - Cover all operations that must be accelerated including communication among operators. (i.e., CDC, SHA, Deduplication, LZW, and communication/integration)
   - Aim for at least 6 axes per operation. Identify a few associated with how operations interact with each other.
   - Some of this should build on the parallelism opportunities you identified on the previous milestone.

   Example from FFT design discussed in class.

Axis:	P, number of butterfly units.
Challenge:	Improving the throughput of the FFT
Opportunity:	Implement multiple hardware butterfly datapath units.
Continuum:	This can range from 1 to a fully spatial design with $P = \frac{N}{2}\log(N)$ butterfly units.
Equation for Benefit:	$Throughtput(P) = P \times SingleButterflyThroughput$
Equation for Resources:	$Resources(P) = P \times SingleButterflyResources$

2. Refine your placeholder implementation into a functional implementation for the project task that can run on a Zynq ARM Cortex A53 processor and produce a valid compressed output stream that works with the supplied decompressor. Integrate with the provided ethernet input flow. Compress from ethernet input to SDCard output.
   - The primary goal for this assignment is functionality. As such, you should focus on a simple design that captures the necessary behavior.
   - As a result, this design need not be efficiently synthesizable to hardware.
   - However, you will eventually be optimizing this design and likely exploring HLS mappings to hardware. So, given a choice, you might want to use design constructs and idioms that you know will be more amenable to HLS hardware mappings.
   - Alternately, you should be prepared to rewrite your code later for efficient hardware mappings.
3. Turn in a tar or zip file with your functional code to the designated assignment component in canvas.
4. Turn in a tar or zip file with binaries to support execution of your code to the designated assignment component in canvas.
   (a) The tar (or zip) files should include:
      - `encoder` – binary for your encoder to run on the the Ultra96

   (b) Your encoder should take one argument:
      - the file name where the program should store the compressed data.
5. Measure the raw ethernet performance from your host machine to your Ultra96 (See Item 2d in "Setup and Walkthrough" at end of this milestone document).
6. Document your design.
   (a) Code sources (e.g., URLs) for any open-source code you used as a starting point or as a primary reference

   (b) Current compression ratio and breakdown of contribution from deduplication and from LZW compression.

   (c) Overall throughput (Gb/s) of your current implementation.

   (d) Description of all validation performed on your current functional implementation.

   (e) Report the raw ethernet speed measurements (Problem 5) *for all 3 partner's machines.*

   (f) Description of who did what. How did your team collaborate on the design, implementation, and validation?

7. Identify any challenges your group had in collaboration and design integration this week and how you plan to address them for future weeks.
   - This is not a question about technical status – that should be addressed above.
   - This is for teamwork, coordination, collaboration, communication, and workflow issues.
   - These may be things you've overcome by submission but didn't go as smoothly as they should have.
   - In the unlikely case that everything went perfectly, identify the things you did that made it work well. For your future plans, look forward to next week to see if the same techniques are applicable or if there are new challenges that might require different or additional techniques for things to continue to go well.

Setup and Walkthrough

1. **Get New Ultra96 Platform**

   We have updated the `hw6_platform_v2.tar.gz` from Homework 6 and have included the following in the new platform:
   - Increased Contiguous Memory Allocation (CMA) size to 1024 MB
   - Updated XRT runtime to version `2020.1_PU1`
   - Added DPDK library which you may use in the project to send packets to your encoder
   - Added `iperf3` which you will use to measure ethernet speeds
   - Enabled SHA3 unit and userspace API
   - Added `gdb`

   So please re-download the new platform from here:
   - Ultra96 Platform
   - Ultra96 Platform (Asia)

   You will be using this platform in Vitis to compile your code as usual. Additionally, you will need a new `sd_card.img` that utilizes this new platform. You can either download that from the following links or make your own by building the hello world example from Homework 6 using the new platform:
   - sd_card.img
   - sd_card.img (Asia)

You will need to write this new image to your SD-card and put it in your Ultra96. Usually Vitis would produce this `sd_card.img` for you, but it doesn't when you just compile CPU code (i.e. without any xclbin). We acknowledge that the above downloads are big and are frustrating to download every time, but bear with us since this is our second iteration using the linux platform in this course.

2. **Measure Ethernet Speed**

   (a) Make sure that you have the following pre-requisites figured out (you should already have this setup from homework 6):

      i. Communication over ethernet. If you are using Windows, follow this document. If you are on Mac, use the following instructions:

         A. Download and install the AX88179 driver in the Mac (which is for the ethernet usb):

         B. And then in Mac, you can do `screen /dev/tty.usbserial-1234_oj11 115200` to open the serial console for the Ultra96. Assign the ip address to ultra96 like you would do normally (see Homework 6 instructions). You can exit the serial console by doing `CTRL-A CTRL-\` and pressing `y`.

         C. Once the ethernet driver is installed on your Mac, you can assign it an ip address using `sudo ifconfig en4 10.10.7.2 netmask 255.0.0.0` where `en4` is the interface name you will find from ifconfig.

      ii. Install `iperf3` on your computer: `https://iperf.fr/iperf-download.php`

      iii. Find out what kind of USB ports you have in your computer: USB-2.0 or USB-3.0.

   (b) Open the Ultra96 terminal and issue the command: `/usr/bin/iperf3 -s`

   (c) Open a terminal in your computer and issue the command: `/usr/bin/iperf3 -c 10.10.7.1` (assuming `10.10.7.1` is the IP address you assigned to the Ultra96).

   (d) You should see outputs similar to the following if you connected to a USB-3.0 port in your computer:

```
Connecting to host 10.10.7.1, port 5201
[  5] local 10.10.7.2 port 38484 connected to 10.10.7.1 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec   110 MBytes   920 Mbits/sec    0    266 KBytes
[  5]   1.00-2.00   sec   109 MBytes   917 Mbits/sec    0    266 KBytes
[  5]   2.00-3.00   sec   109 MBytes   916 Mbits/sec    0    276 KBytes
[  5]   3.00-4.00   sec   109 MBytes   915 Mbits/sec    0    276 KBytes
[  5]   4.00-5.00   sec   106 MBytes   893 Mbits/sec    0    287 KBytes
[  5]   5.00-6.00   sec   104 MBytes   874 Mbits/sec    0    287 KBytes
[  5]   6.00-7.00   sec   105 MBytes   878 Mbits/sec    0    287 KBytes
[  5]   7.00-8.00   sec   105 MBytes   877 Mbits/sec    0    287 KBytes
[  5]   8.00-9.00   sec   105 MBytes   880 Mbits/sec    0    287 KBytes
[  5]   9.00-10.00  sec   105 MBytes   878 Mbits/sec    0    287 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
```

```
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-10.00  sec  1.04 GBytes   895 Mbits/sec    0             sender
[  5]   0.00-10.00  sec  1.04 GBytes   893 Mbits/sec                  receiver
iperf Done.
```

If you connected to a USB-2.0 port, you should see the following:

```
      Connecting to host 10.10.7.1, port 5201
[  5] local 10.10.7.2 port 37752 connected to 10.10.7.1 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec  41.1 MBytes   345 Mbits/sec    0    120 KBytes
[  5]   1.00-2.00   sec  40.5 MBytes   340 Mbits/sec    0    120 KBytes
[  5]   2.00-3.00   sec  40.5 MBytes   340 Mbits/sec    0    120 KBytes
[  5]   3.00-4.00   sec  40.5 MBytes   340 Mbits/sec    0    120 KBytes
[  5]   4.00-5.00   sec  40.5 MBytes   340 Mbits/sec    0    120 KBytes
[  5]   5.00-6.00   sec  40.8 MBytes   342 Mbits/sec    0    120 KBytes
[  5]   6.00-7.00   sec  40.5 MBytes   340 Mbits/sec    0    120 KBytes
[  5]   7.00-8.00   sec  40.5 MBytes   340 Mbits/sec    0    120 KBytes
[  5]   8.00-9.00   sec  40.8 MBytes   342 Mbits/sec    0    120 KBytes
[  5]   9.00-10.00  sec  40.6 MBytes   340 Mbits/sec    0    120 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-10.00  sec   406 MBytes   341 Mbits/sec    0             sender
[  5]   0.00-10.00  sec   406 MBytes   340 Mbits/sec                  receiver
iperf Done.
```

This tells you that the upper bound on the throughput achieved by a placeholder receiver is around 895 Mb/s (341 Mb/s if using USB-2.0 port) limited by the Ugreen ethernet-to-USB interfaces. This will be your actual target instead of the 1 Gb/s stated in the project handout. That is, we're not asking you to exceed the speed supported by the ethernet-transceiver, but we are asking you to match it. Measure on the setup for all project members. Plan to use the machine with the fastest raw results for your highest-performance measurements.

3. **Obtaining Starter Code and Integrating Ethernet Input**

   We will now describe how data will be sent through ethernet using the client and server, the packet layout, and other relevant information for getting started to receiving real-time data.

   (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 `project` directory. The directory structure looks like this:

```
project/
  Client/
    client.cpp
  Decoder/
    Decoder.cpp
  Server/
    encoder.cpp
    encoder.h
    event_timer.cpp
    event_timer.h
    server.cpp
    server.h
  compile_on_biglab.sh
  LittlePrince.txt
  Makefile
```

(b) Make sure the `PLATFORM_REPO_PATHS` is setup to the new platform you downloaded.

(c) Open a terminal and issue the following command. Change the command to reflect the directory you installed Vitis in. The command sets up the paths used by the Makefile.

```
source /opt/Xilinx/Vitis/2020.1/settings64.sh
```

If you are compiling on BigLab:

- git clone your repo on BigLab.
- wget the Ultra96 platform in BigLab and extract it in a folder.
- open the `compile_on_biglab.sh` file and change the `PLATFORM_REPO_PATHS` to reflect the folder you put the platform in.
- Run `compile_on_biglab.sh` compile the code. Note that by default `compile_on_biglab.sh` calls `make all`. Change it if you want to run a different make command. Also note that you need to run with a shell script on BigLab.

(d) Use `make all` to compile all the targets `client`, `encoder`, and `decoder`.

(e) Use `make clean` to clean all the generated files.

(f) Our basic model will be communication between two systems—your computer and the Ultra96—over ethernet. Your computer will send packets at a fixed rate. The Ultra96 will receive the data and compress it. Figure 2.5 from homework 6 shows you the setup and cabling. Since the first system is sending data at a fixed rate, it is necessary for the receiver to compress the data at that rate or data will be lost. We provide the code for the sender (`Client/client.cpp`). Your project is connected to the receiver (`Server/encoder.cpp`). And then you can use the decoder (`Decoder/Decoder.cpp`) to verify that you can recover the original, unencoded file from the compressed file.

(g) Let's run the given code with the system we have setup. After compiling the code, copy over `encoder` binary, and the `LittlePrince.txt` as follows (adjust the commands if you are not using Linux):

```
scp encoder LittlePrince.txt root@10.10.7.1:~/
```

And then open the Ultra96 terminal and run the encoder with `./encoder`. The program waits for a packet to arrive. Open a terminal in your computer and issue the following command:

```
./client -i 10.10.7.1 -f LittlePrince.txt
```

You should see the following output in your Ultra96 terminal:

```
root@ultra96v2-2020-1:~# ./encoder
setting up sever...
server setup complete!
write file with 14247
--------------- Key execution times ---------------
Reading packets and processing :    0.228 ms
```

You should see the following output in your host terminal:

```
ip is set to 10.10.7.1
filename is LittlePrince.txt
bytes_read 14247
```

You can verify the output by doing the following in the Ultra96:

```
diff output_cpu.bin LittlePrince.txt
```

Note that our example is just writing the packets to a file. Your project will process these packets with the encoder pipeline and write a compressed output.

(h) We'll now describe what's happening in the client and the server:

i. **Packet Layout**: We will be sending data via UDP datagrams. Linux supports the UDP protocol and receiving packets from the client can be done easily using Linux IP. The code provided will direct you on how to setup your compression pipeline to listen as well as handle incoming packets. The maximum size of a packet will be 16K Bytes. The header of the packet will be 2 bytes consisting of a done bit denoting that all of the data has been transmitted as well as the length of the data contained inside the packet.

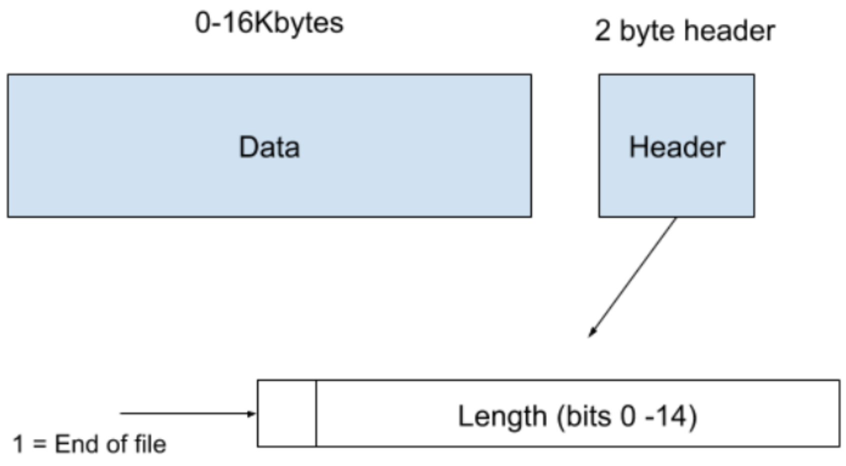


Figure 1: Packet Layout

ii. You will be specifically interested in the files `Server/server.cpp` and `Server/server.h`. These are the files provided and can be directly copied and pasted into your project to set up your server pipeline to receive data. The rest of the repository can serve as an example of how one could implement the design (`Server/encoder.cpp`). At a high level you need to call the function `setup_server()` once. Following this you can makes calls to the function `getpacket()` to receive your next datagram. Please note that your buffer passed into this function needs to be able to handle the maximum payload size plus the two byte header. Please also note that the `recvfrom()` Linux call is blocking. This means that your process will be blocked until a datagram has arrived. Depending on your design this may be ok. If you do not want to block you may look into the `select()` system call as an alternative. This will allow you to check if data has arrived in your socket and take actions accordingly.

    You are of course free to write your own application. For more information on how to receive the data you can refer to the man page. https://linux.die.net/man/2/recvfrom.

    Alternatively, you can create your own client and server using the DPDK library. Examples of how to write client and server code using DPDK can be found in the following links:

    - https://doc.dpdk.org/guides/index.html
    - https://zenhox.github.io/2018/01/25/dpdk-pktSR/

Design Considerations

1. **Configuring Sender**: You will likely need to slow down the client's data transfer to begin with. If your system cannot keep up with the pace of the sender, packets will be dropped.

   To configure the sender, when you start the client from the Linux shell, it takes arguments as shown:

   ```
   ./client -s 5 -f file -i ip_address_of_ultra96 -c chunksize
   ```

   Usage example is:

   ```
   ./client -s 5 -f LittlePrince.txt -i 10.10.7.1 -c 2048
   ```

   `-s` option specifies the sleep time or delay between packets (in microseconds)
   `-i` option specifies ip address to send to
   `-f` option specifies what file to send
   `-c` option specifies the chunk size

   For Problem 6c (and for later times where you characterize your throughput), you should adjust the `-s` argument until your design fails. Report your maximum throughput as the throughput associated with the smallest value of `-s` on which your design successfully receives and correctly compresses the input. Measure the actual throughput by measuring the time it takes for the client to send the file. You can use `/usr/bin/time` to measure the time.

2. **Debugging Sending and Receiving of Packets**: If you encounter problems with sending and receiving packets between the client and the server, you can emulate the socket programming. You can see an example of that here. Specifically in `server.cpp` and `server.h`, you can see that instead of using sockets, you can read a file and send it in pieces to the encoder pipeline to emulate socket programming.

3. **Options for SHA implementation**: You have four options for implementing SHA in your encoder pipeline:

   - Writing serialized SHA-256 running on the ARM processor.
   - Using the dedicated SHA3-384 unit in the Zynq Ultrascale.
   - Using SHA-256 NEON intrinsics.
   - Implementing SHA-256 on FPGA using HLS (not until P3).

   The specific option you choose will have implications on the maximum throughput you can achieve for your encoder pipeline. Moreover, you will need to adjust your design accordingly if you end up using the SHA3 unit, since it has a different digest size.

- You can find throughput comparisons of SHA-256 on NEON and the dedicated SHA3 unit here: `https://www.xilinx.com/support/documentation/white_papers/wp512-accel-crypto.pdf`.
- You can find an example of how to use the SHA3 unit here: `https://xilinx-wiki.atlassian.net/wiki/pages/viewpage.action?pageId=18841654&pageVersion=1`. Note we have already enabled the SHA3 unit and user-space API in the new platform. You will just need to write the driver.

4. **Multiple Cores**: Recall from Homework 3 that you can utilize multiple cores using `std::threads`. There are four cores (ARM Cortex-A53) on the Ultra96 and you should consider how the cores are different from the Cortex A-72 you utilized in Homework 3. Moreover, recall that each core on the Ultra96 has one 64-bit NEON SIMD unit with 128-bit registers that you can utilize simultaneously with `std::threads`.

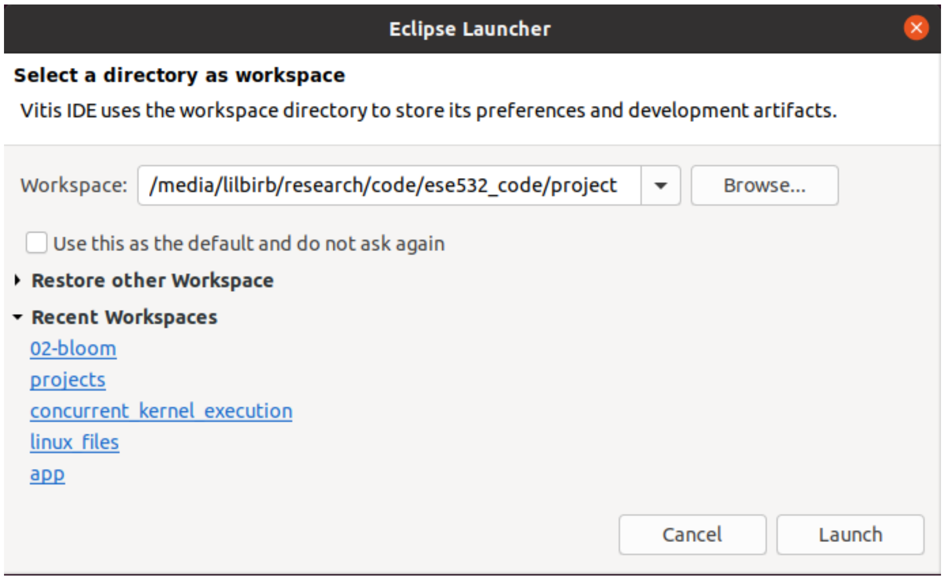
Vitis GUI Workflow

We have given you a `Makefile` so far (and still recommend that you use makefiles). However, if you would like a GUI based compilation workflow, following is a tutorial on how you can compile code for the Ultra96 using the Vitis GUI.

1. Open a terminal and issue the following command. Change the command to reflect the directory you installed Vitis in. If you have installed Vitis on Windows, you can open Vitis from the Programs.

```
source /opt/Xilinx/Vitis/2020.1/settings64.sh
```

2. Launch Vitis by typing `vitis` in the terminal. You will see the following screen:



Eclipse Launcher

Select a directory as workspace

Vitis IDE uses the workspace directory to store its preferences and development artifacts.

Workspace: /media/lilbirb/research/code/ese532_code/project

Use this as the default and do not ask again

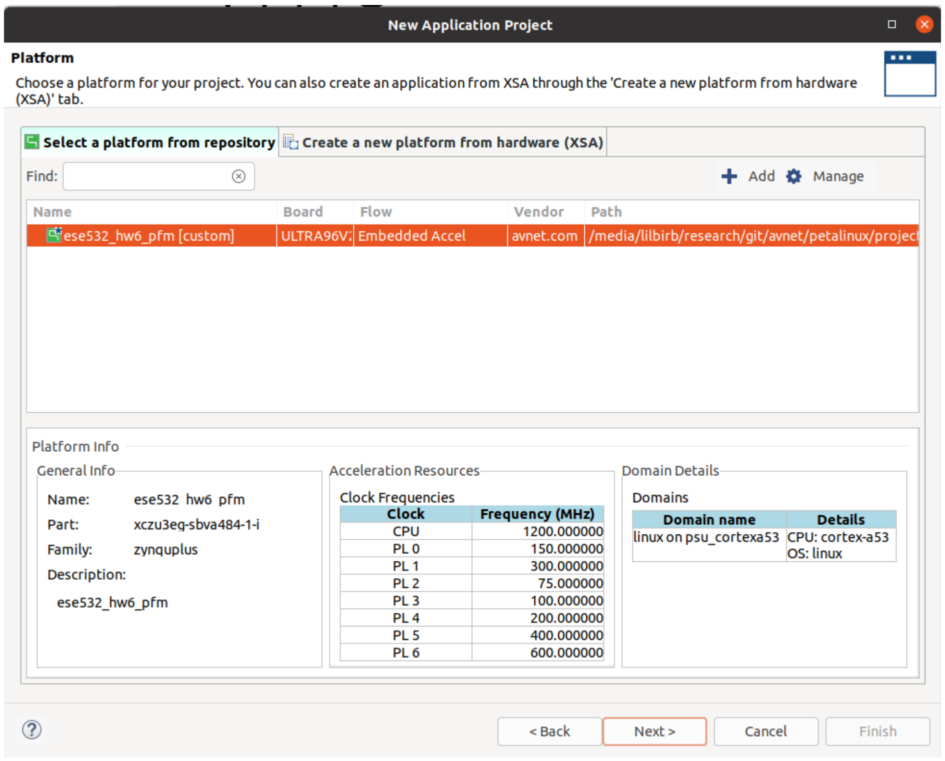
Restore other Workspace

Recent Workspaces

02-bloom
projects
concurrent_kernel_execution
linux_files
app

3. Enter a directory in the `Workspace` field where you would like to store your projects and then click `launch`.

4. From the next screen, click on `Create Application Project` and press `Next`. You will then see the screen on selecting a platform:


New Application Project
Select a platform from repository
Create a new platform from hardware (XSA)
Find:
Add
Manage
< Back
Next >
Cancel
Finish

Platform

Choose a platform for your project. You can also create an application from XSA through the 'Create a new platform from hardware (XSA)' tab.

Name	Board	Flow	Vendor	Path
ese532_hw6_pfm [custom]	ULTRA96V:	Embedded Accel	avnet.com	/media/lilbirb/research/git/avnet/petalinux/projec

Platform Info

General Info

Name:	ese532_hw6_pfm
Part:	xczu3eg-sbva484-1-i
Family:	zynquplus
Description:	
ese532_hw6_pfm	

Acceleration Resources

Clock Frequencies

Clock	Frequency (MHz)
CPU	1200.000000
PL 0	150.000000
PL 1	300.000000
PL 2	75.000000
PL 3	100.000000
PL 4	200.000000
PL 5	400.000000
PL 6	600.000000

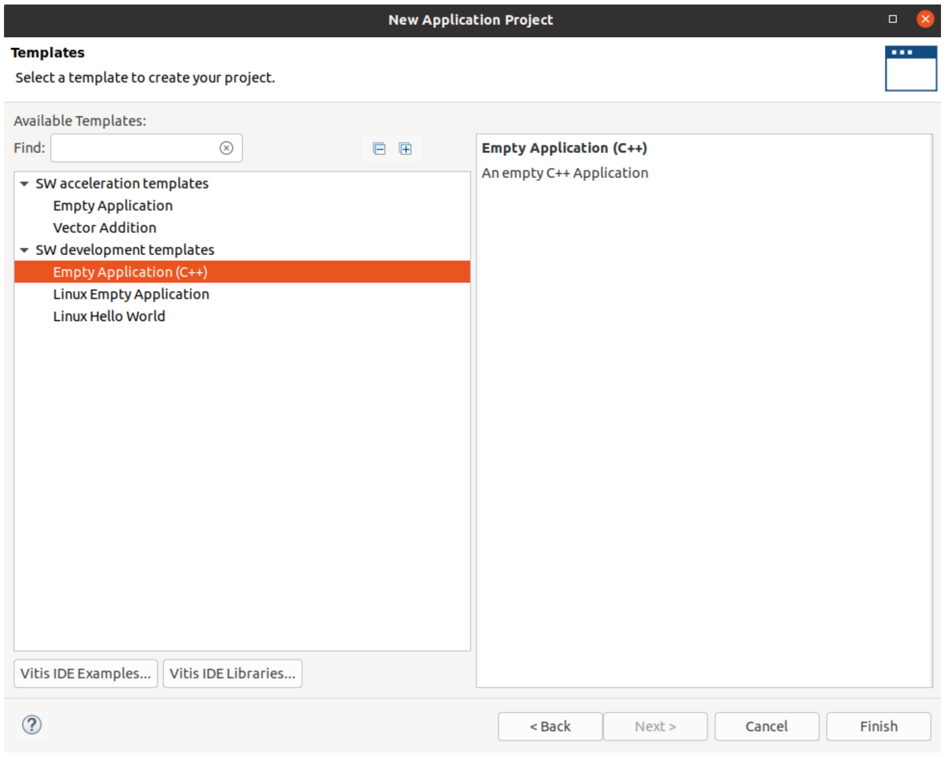
Domain Details

Domains

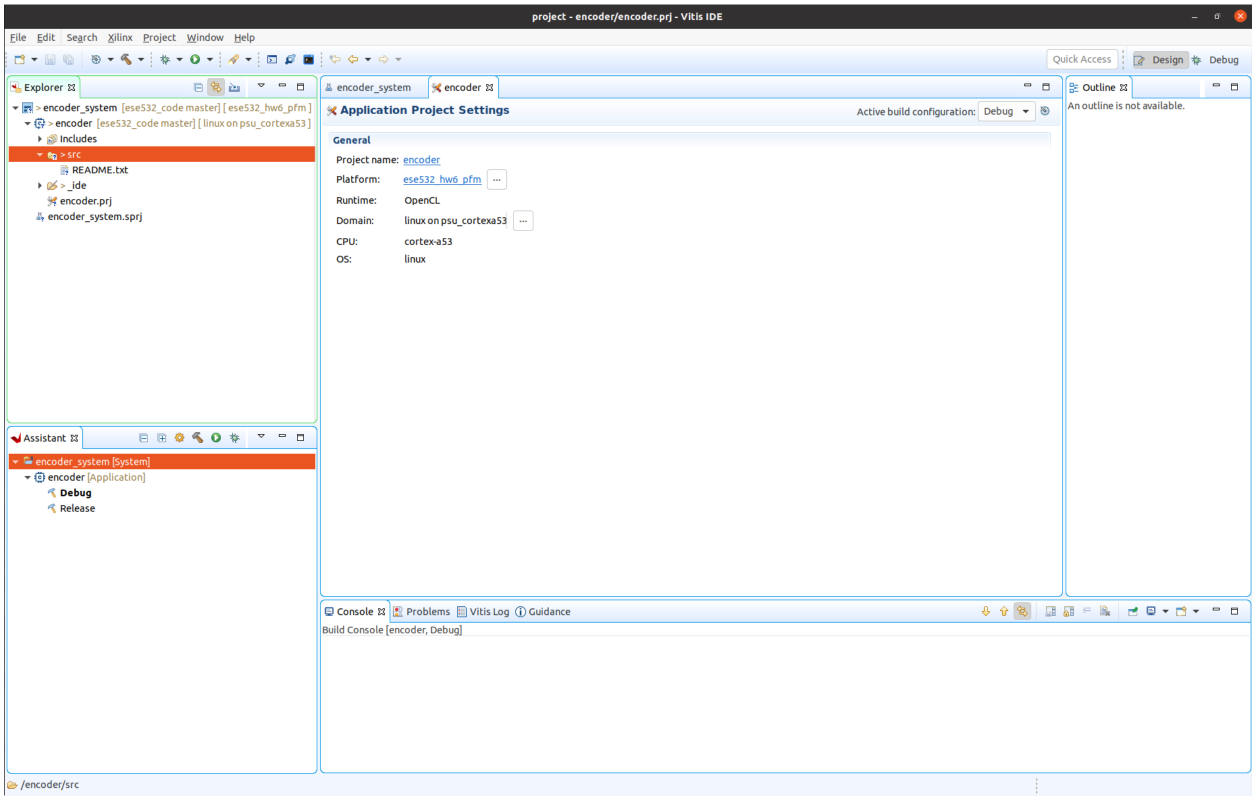
Domain name	Details
linux on psu_cortexa53	CPU: cortex-a53 OS: linux

5. If you don't see your `ese532_hw6_platform`, browse to the platform directory (where the `ese532_hw6_platform.xpfm` is) by clicking on `Add`. Click `Next`.

6. Give an application name and click `Next` and then `Next`. You will then see the following screen:



7. When you will start writing FPGA code, you should select the `Empty Application` template from `SW acceleration templates` option. Since in P2 we are only writing CPU code, you should select the `Empty Application` template from the `SW development templates` option. Click on `Finish`. You should now see the following screen:



Application Project Settings

Active build configuration: Debug

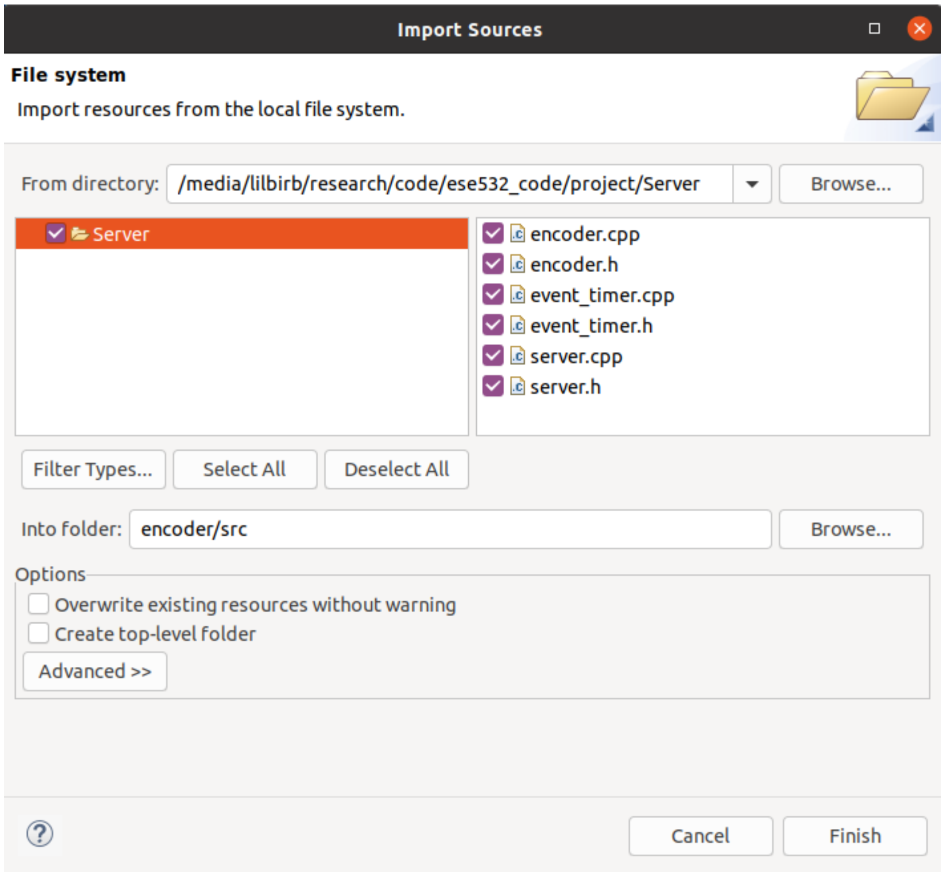
General

Project name: encoder
Platform: ese532_hw6_pfm
Runtime: OpenCL
Domain: linux on psu_cortexa53
CPU: cortex-a53
OS: linux

Build Console [encoder, Debug]

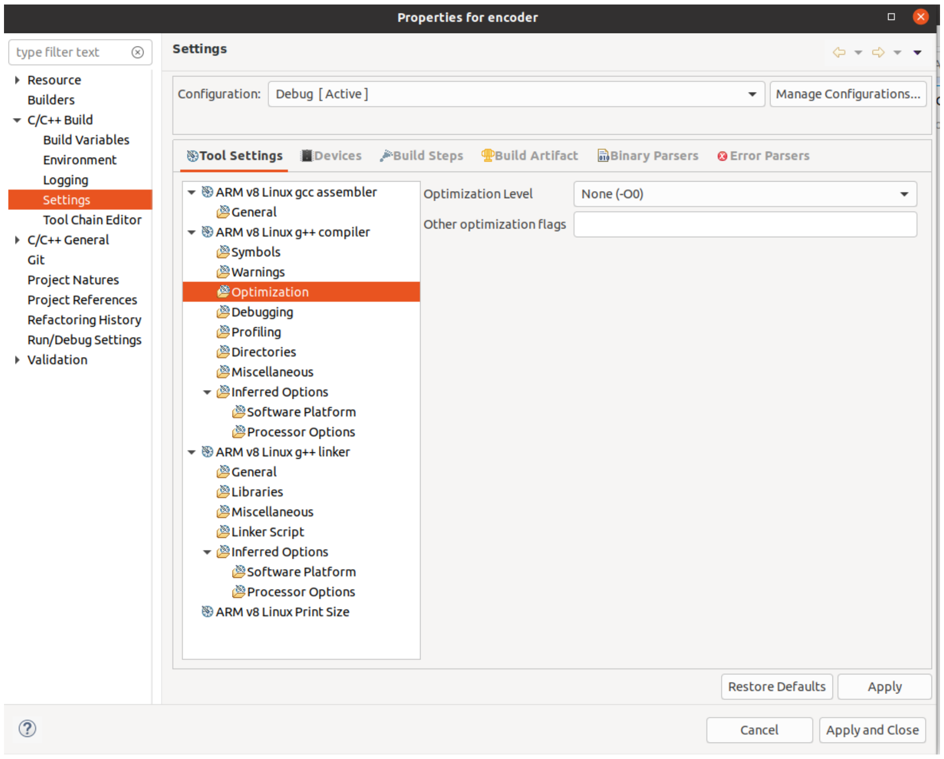
8. You can learn more about the Vitis IDE from here.

9. In the `Explorer` tab, right-click on `src` folder and click on `Import Sources`. You will see a screen as follows:



10. Browse to the directory where you will import sources from and click `OK`. Check all the sources you want to import and click `Finish`.

11. Right click on the encoder project from the `Explorer` tab and select `C/C++ Build Settings`. You will see a screen as follows:

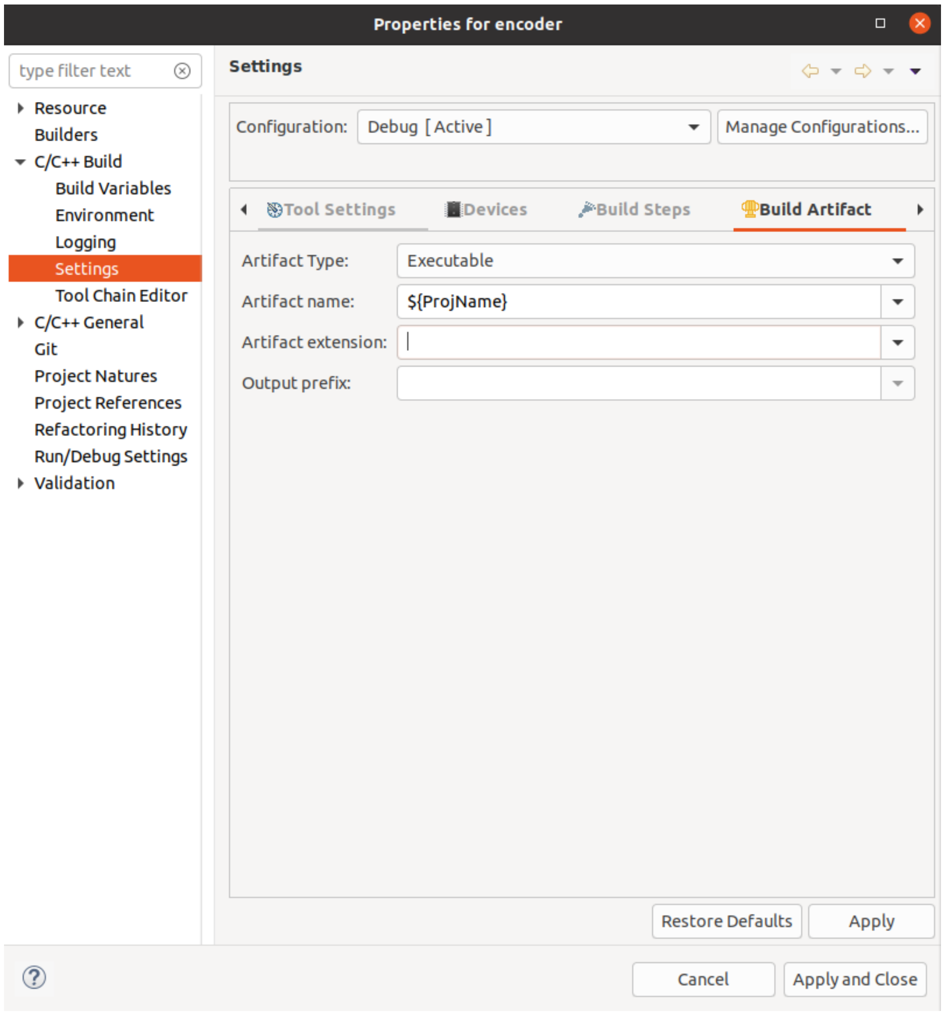


Configuration: Debug [Active]

Optimization Level: None (-O0)

Other optimization flags:

12. Specify optimization level to be `O3`. In the miscellaneous tab, you can specify other compiler options, such as `-march=armv8-a+simd -mtune=cortex-a53` to enable NEON SIMD unit for instance. Click on Apply. Now go to the `Build Artifact` tab as follows:



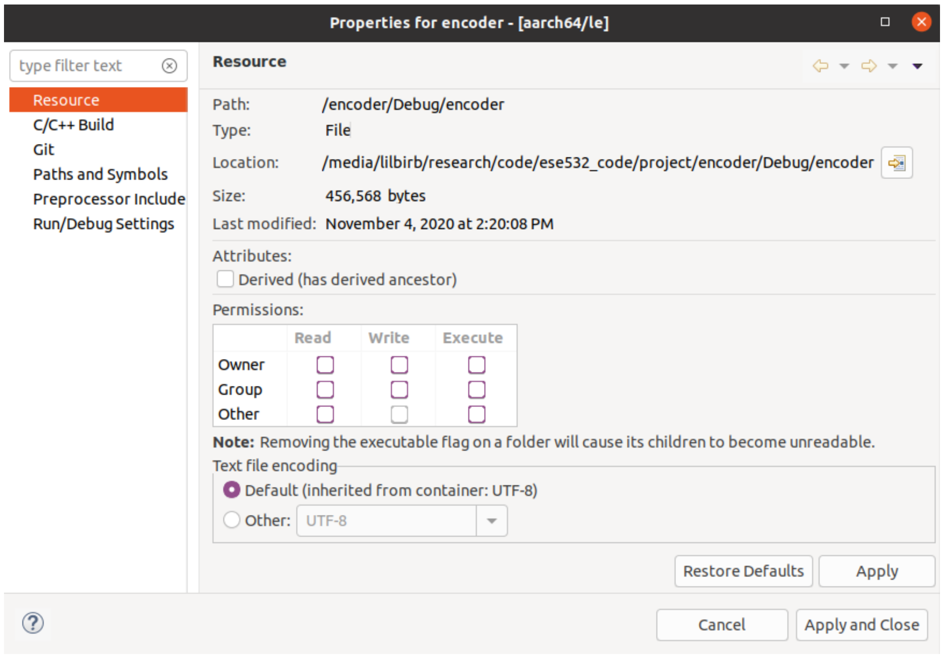
Configuration: Debug [Active]

Artifact Type: Executable
Artifact name: ${ProjName}
Artifact extension:
Output prefix:

13. Make the artifact extension blank (this is just so that the binary we are producing in the Makefile is the same as this GUI workflow. `.elf` is the extension for the executable). Click on `Apply and Close`.

14. Now click on the Build button from the menu to build the project. You should see in the `Console` that `Build Finished`.

15. In the `Explorer` tab, you should now see a `Debug` folder. This folder has the `encoder` that you should copy over to the Ultra96. Find the directory of the folder by right-clicking on the `encoder` and clicking as shown below. Open that folder in the terminal and copy the binary as you would do usually.


Properties for encoder - [aarch64/le]

Resource

Path: /encoder/Debug/encoder

Type: File

Location: /media/lilbirb/research/code/ese532_code/project/encoder/Debug/encoder

Size: 456,568 bytes

Last modified: November 4, 2020 at 2:20:08 PM

Attributes:

☐ Derived (has derived ancestor)

Permissions:

	Read	Write	Execute
Owner	☐	☐	☐
Group	☐	☐	☐
Other	☐	☐	☐

Note: Removing the executable flag on a folder will cause its children to become unreadable.

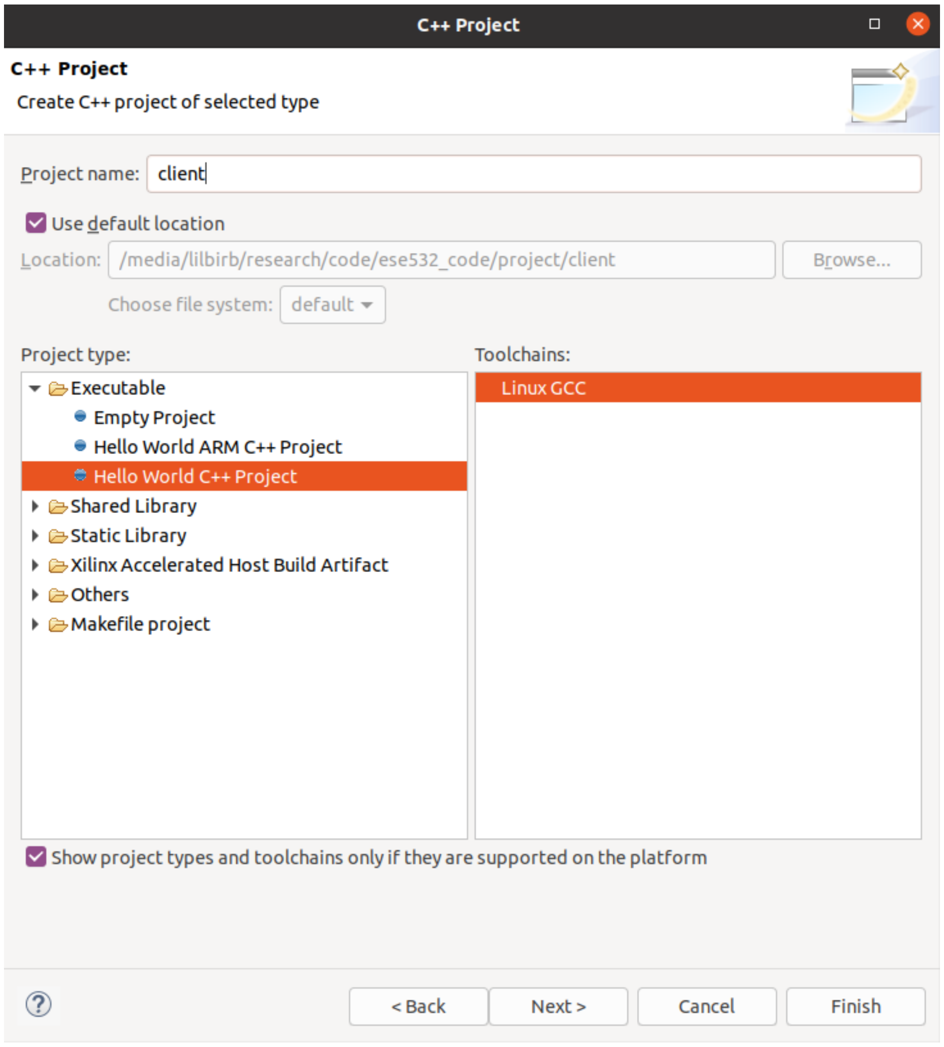
Text file encoding

● Default (inherited from container: UTF-8)

○ Other: UTF-8

16. This concludes how to compile Ultra96 code using the Vitis GUI.

17. You can also compile the client code using the Vitis GUI. From the menu, click on `File`→`New`→`Other`→`C++ Project`→`Hello World C++ Project` as shown below. Give a name to the project and click on Finish as follows. Click on Open Perspective and continue.



C++ Project

Create C++ project of selected type

Project name: client

☑ Use default location

Location: /media/lilbirb/research/code/ese532_code/project/client

Choose file system: default

Project type:
- Executable
  - Empty Project
  - Hello World ARM C++ Project
  - Hello World C++ Project
- Shared Library
- Static Library
- Xilinx Accelerated Host Build Artifact
- Others
- Makefile project

Toolchains:
- Linux GCC

☑ Show project types and toolchains only if they are supported on the platform

18. Delete the Hello World code and import your client code. Change the optimization level to `O3` from the `C/C++ Build Setting`. Build and you should see the binary in the Debug folder as usual.

19. You can repeat the above steps to compile the Decoder as well. It's up to you which target you compile the Decoder against—aarch64 or x86—depending on whether you want to test the decoding on the Ultra96 or your host machine.

Questions

If anything is unclear please post on piazza or come to office hours, and we will be glad to assist.