Building ADI Kernel

A goal of our project is to build the ADI kernel from ADI directly so we can have all of the necessary drivers for our project built-in, instead of having to integrate the new drivers into our existing software stack. To build this kernel we followed this guide on ADI's website here: https://wiki.analog.com/resources/tools-software/linux-build/generic/zynqmp

What Didn't Work

The Script Method

We started off by trying the section titled "The script method" at the top of this page, but this wouldn't generate the necessary artifacts to have a usable build. When we used it, the only generated artefact was an Image file but none of other required files for booting on the target device.

On the Development Host

What we ended up following was the "On the development host" steps. This seemed to work for a while, and we got all the way to the step to copy the two generated files to an SD card for booting. u-boot.elf is a necessary file from this step for building a bootable image. However, we would get no output when trying to boot using the generated files from this step.

Building using Petalinux

We tried "Building using Petalinux" the most, as building using Petalinux tools is standard for AMD SoCs, like the one on the BytePipe. We had tried this earlier in the semester, using the older version of the AMD Xilinx suite of tools (2021.1). When trying to use this version when building with Petalinux, we would get an older kernel version that was undesired. When building with Petalinux, we used the existing device tree (.dts) and HDL (.xsa) files when building using Petalinux, and that's how we got that result. What we tried later was using newer versions of the tools and picking a generic device tree and HDL combination that would give us a newer kernel version with all the necessary drivers, but these builds would result with build artefacts that couldn't successfully boot. Within the meta-adi-xilinx github repository (**link below**), we tried two different generic HDL/device tree combinations for zcu102 evaluation boards:

HDL	Device Tree
adrv9001_zcu102	zynqmp-zcu102-rev10-adrv9002
adrv9002_zcu102	zynqmp-zcu102-rev10-adrv9002-rx2-tx2

github.com

https://github.com/analogdevicesinc/meta-adi/tree/main/meta-adi-xilinx

A necessary step before building HDL is to have a valid Vivado Enterprise Edition license. This took us a little bit to find out, but this was our process for finding out how this kind of license works and how to get one for your system.

Getting a Valid Vivado License

To build HDL a Vivado Enterprise Edition license is necessary. To get this license, open Vivado and go to *Help* > *Manage License*. Once the license manager is open, go to the "Obtain License" page, select the fourth option ("Get My Full or Purchased Certificate-Based License") and select the "Connect Now" button. From there you have to sign into your AMD account and from there you'll be able to manage your licenses. At the very least you have to get the Vivado ML Enterprise Edition license. Something good to know about this license is that it needs to be renewed every 30 days. Once you've claimed your license you need to check your email for the confirmation email that's sent directly after requesting the license. Once you get to the page where you're viewing your license, you have to click a little blue down arrow icon that downloads a xilinx.lic file to your system. This file then needs to be loaded within the Vivado license manager program, and once its loaded you can build the HDL for this part.

What Does Work

Building the Project with Petalinux

> **github.com**
> https://github.com/analogdevicesinc/meta-adi/tree/main/meta-adi-xilinx

Using the guide within this GitHub repository, scroll down and start at the beginning at the "Building with Petalinux" header. As stated under "This layer supports:" text, Petalinux-v2023.2 will be necessary to build the project. If you don't already have Petalinux version 2023.2 installed, see "Setting Up the Build Environment" > "Xilinx Tool Suite" to install the necessary tools to proceed.

With Petalinux installed, follow the commands down below. These can also be found under the "To build a petalinux project using Analog Devices yocto lauer, run:" text within the ADI guide.

```
source  <path-to-installed-PetaLinux>/settings.sh
petalinux-create -t project --template <PLATFORM> --name <PROJECT_NAME>, where
<PLATFORM> is:
    * zynqMP (for UltraScale + MPSoC)
    * zynq (for Zynq)
    * microblaze (for MicroBlaze)
git clone https://github.com/analogdevicesinc/meta-adi.git
cd <path-to-project># Check the Building HDL link below
petalinux-config --get-hw-description=<path to hdf file># Select the devicetree that
fits the project being built
echo "KERNEL_DTB=\"${dts_to_use}\"" >> project-spec/meta-user/conf/petalinuxbsp.conf
cd build
petalinux-build
```

Building With Petalinux Step-by-Step:

source <path-to-installed-PetaLinux>/settings.sh
When sourcing the settings.sh file, you will likely find it within your tools/ directory within the root directory of your Linux file system. If the source command does not work with your settings.sh file, try the following command and try sourcing again:

```
chmod +rwx <path-to-settings.sh>/settings.sh
```

This will allow the settings file to be readable, writable, and executable. When initially installing Petalinux, it may not have these permissions by default so manual modifications like this may be necessary.

petalinux-create -t project --template <PLATFORM> --name <PROJECT_NAME>
This step creates the base of the Petalinux project. For the platform field, use zynqMP as the SoC falls under the zynqMP platform. For the <PROJECT_NAME> field, you can use whatever name you want for the project, like adi-petalinux-build for example.

git clone https://github.com/analogdevicesinc/meta-adi.git
This step is fairly straightforward, here we're cloning ADI's medi-adi GitHub repository.

cd <path-to-project>
Traverse into the project directory you just made. This would be the same name you used for the <PROJECT_NAME> field above when creating the Petalinux project.

petalinux-config --get-hw-description=<path to xsa file>
For this step we're going to be configuring the Petalinux build. For the <path to hdf file> field, we'll be using the hdl generated from following the steps in the "RFLAN Documentation" document. The resulting artifact should be in the "bytepipe_sdk" directory. From there go to "workspace" > "rflan". Within the folder, copy the rflan_xczu3eg-sbva484-1-e.xsa file and paste the path in that <path to hdf file> field.
After running this command, the configuration menu interface will appear. From there navigate to "Yocto Settings" > "User Layers". Here we'll add a single user layer, and this can be added by copying the "meta-adi-xilinx" folder from the meta-adi repo cloned previously and pasting it in the user layer field. This should paste the path to the folder. Once that's done, you can exit the configuration menu. After exiting, the configuration should automatically complete.
IF THIS DOESN'T WORK:
Sometimes configuration will fail for whatever reason. In the event it fails, try closing your terminal and starting from the beginning again by sourcing the Petalinux tools, creating another Petalinux project, cd into the project, and run the petalinux-config command again. You don't need to clone the meta-adi repo again.

echo "KERNEL_DTB=\"${dts_to_use}\"" >> project-spec/meta-user/conf/petalinuxbsp.conf
For this step we're setting the variable KERNEL_DTB to the name our device tree (.dts) file. This command never quite worked for me, so I had did the following workaround.

First, I copied the .dts file resulting from the RFLAN build (system.dts) and pasted it into the root directory of this Petalinux project.
Next, I went into the petalinuxbsp.conf file directly (can be found within project-spec/meta-user/conf/) and wrote the following on line 5:

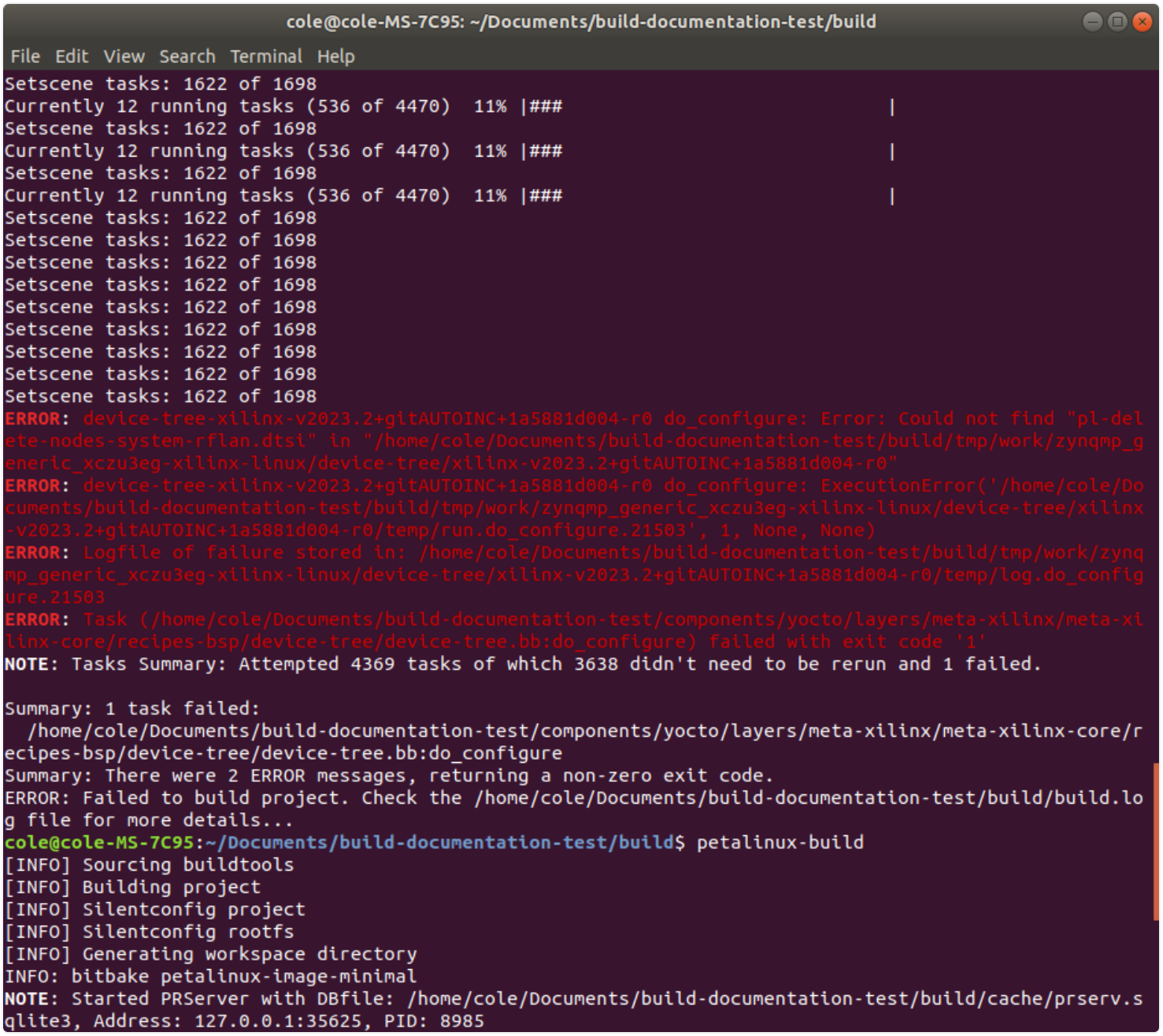
```
KERNEL_DTB="system"
```

And that does the same thing as the command listed for this step.

cd build

petalinux-build

For this step we'll traverse into the build directory with `cd build` and we'll begin the build with the `petalinux-build` command.
The build will run for a while, but **eventually will return an error and exit before completion**.
The error will likely look like this:



This is because the build process is meant to target generic boards, but our board is not like-for-like with the generic options. To work around this, you have to go into the following directory: <petalinux-build-directory>/build/tmp/work/zynqmo_generic_xczu3eg-xilinx-linux/device-tree/xilinx-v2023.2+gitAUTOINC+1a5881d004-r0
Within this directory there will be multiple remove pl-delete-nodes-*.dtsi files. What I did was **duplicate the pl-delete-nodes-zynqmp-zcu102-rev10-adrv9002.dtsi** file within the same directory, and **rename it to pl-delete-nodes-<dts file name>.dtsi.**
After that's done run the `petalinux-build` command again to continue the build process.
Shortly after starting the build process again you'll be met with an error that looks like this:

```
cole@cole-MS-7C95: ~/Documents/build-documentation-test/build
File  Edit  View  Search  Terminal  Help
ecipes-bsp/device-tree/device-tree.bb:do_configure
Summary: There were 2 ERROR messages, returning a non-zero exit code.
ERROR: Failed to build project. Check the /home/cole/Documents/build-documentation-test/build/build.log file for more details...
cole@cole-MS-7C95:~/Documents/build-documentation-test/build$ petalinux-build
[INFO] Sourcing buildtools
[INFO] Building project
[INFO] Silentconfig project
[INFO] Silentconfig rootfs
[INFO] Generating workspace directory
INFO: bitbake petalinux-image-minimal
NOTE: Started PRServer with DBfile: /home/cole/Documents/build-documentation-test/build/cache/prserv.sqlite3, Address: 127.0.0.1:35625, PID: 8985
Loading cache: 100% |#########################################| Time: 0:00:01
Loaded 6323 entries from dependency cache.
Parsing recipes: 100% |#########################################| Time: 0:00:02
Parsing of 4404 .bb files complete (4384 cached, 20 parsed). 6343 targets, 333 skipped, 1 masked, 0 errors.
NOTE: Resolving any missing task queue dependencies
Initialising tasks: 100% |#####################################| Time: 0:00:02
Checking sstate mirror object availability: 100% |#############| Time: 0:00:07
Sstate summary: Wanted 525 Local 48 Mirrors 442 Missed 35 Current 1173 (93% match, 97% complete)
NOTE: Executing Tasks
ERROR: device-tree-xilinx-v2023.2+gitAUTOINC+1a5881d004-r0 do_configure: Error: Could not find selected device tree:"system-rflan.dts" in:"/home/cole/Documents/build-documentation-test/build/tmp/work-shared/zynqmp-generic-xczu3eg/kernel-source/arch/arm64/boot/dts/xilinx"!!
ERROR: device-tree-xilinx-v2023.2+gitAUTOINC+1a5881d004-r0 do_configure: ExecutionError('/home/cole/Documents/build-documentation-test/build/tmp/work/zynqmp_generic_xczu3eg-xilinx-linux/device-tree/xilinx-v2023.2+gitAUTOINC+1a5881d004-r0/temp/run.do_configure.10591', 1, None, None)
ERROR: Logfile of failure stored in: /home/cole/Documents/build-documentation-test/build/tmp/work/zynqmp_generic_xczu3eg-xilinx-linux/device-tree/xilinx-v2023.2+gitAUTOINC+1a5881d004-r0/temp/log.do_configure.10591
ERROR: Task (/home/cole/Documents/build-documentation-test/components/yocto/layers/meta-xilinx/meta-xilinx-core/recipes-bsp/device-tree/device-tree.bb:do_configure) failed with exit code '1'
NOTE: Tasks Summary: Attempted 4378 tasks of which 4368 didn't need to be rerun and 1 failed.

Summary: 1 task failed:
  /home/cole/Documents/build-documentation-test/components/yocto/layers/meta-xilinx/meta-xilinx-core/recipes-bsp/device-tree/device-tree.bb:do_configure
Summary: There were 2 ERROR messages, returning a non-zero exit code.
ERROR: Failed to build project. Check the /home/cole/Documents/build-documentation-test/build/build.log file for more details...
cole@cole-MS-7C95:~/Documents/build-documentation-test/build$
```

Here you'll go back to the root directory of your Petalinux project and copy the rflan device tree file (I named it system-rflan.dts in this case). Then, paste it into the <petalinux-build-directory>/build/tmp/work-shared/zynqmp-generic-xczu3eg/kernel-

source/arch/arm64/boot/dts/xilinx directory. Once you've pasted the .dts file there, run the `petalinux-build` command again and your build should complete successfully!

Booting Your Image

Once the build is complete, there is one more command that needs to be run to generate the necessary boot files. To do this, run `petalinux-package --boot --fsbl --fpga --u-boot` .

Now, to boot your fully built image, you will need an SD card formatted as FAT32 (this is typically how SD cards are formatted by default). The files needed to boot can then be found within your Petalinux build directory throught the following path:

<petalinux-build-directory>/images/linux/

Within this directory you will need to copy and paste four files onto your SD card:

- boot.scr
- boot.bin
- Image
- rootfs.cpio.gz.u-boot

With these four files on the SD card, you can boot ADI's kernel image (should be a 6.1.0-xilinx kernel) on any HDK unit and interact with the Linux shell via a serial terminal program. As a team we used Tera Term within a Windows installation, but programs like Minicom within Linux should also work.