Notes on Building UUB Firmware

D. Nitz

April 14, 2020

This document contains notes (mostly for myself) for bulding the UUB firmware, device tree, and first stage boot loader. These notes are a work in progress and have not yet been verified to work for the device tree and first stage boot loader, and furthermore, are specific to my working environment.

Generating the Bitstream and Exporting the Hardware

1. After cloning a fresh copy:
   (a) cd uub-firmware/wp2
   (b) Unzip the archive using: unzip uub_v3/uub_v3.zip
2. Set up Vivado environment using "setup_vivado" command. (Private command on my cluster)
3. After starting Vivado fix IP path to point to uub-firmware/ip_repo only rather than the in-project path that is set up when unzipping the archive.
4. Open the block diagram and compile the firmware using the tcl script "make_it_afs.tcl". This will need to be modified if the location of the git clone is changed. The tcl script will run through all of the compile steps up to and including writing the bitstream file and exporting the hardware.
5. Exit from Vivado

Building the Device Tree

This will need to be done whenever AXI devices are added or removed from the design, or when any device address is changed.

1. Using the setup_vivado environment, start up the Xilinx SDK using: xsdk.
2. Choose workspace ~wp2/uub_v3/uub_v3.sdk. After xsdk loads, you should see the projects listed in Fig. 1.
3. Regenerate BSP sources for "device_tree_bsp_0". This will make new versions of pl.dtsi, skeleton.dtsi, system.dts, zynq-7000.dtsi incorporating the latest .hdf file. We will not use these files directly, but instead merge them in to Auger customized versions for the UUB.

Figure 1: xSDK projects required for this work.

4. cd to the directory ~wp2/device_tree. Run the command "./merged_diff" to check for changes that need to be merged into ~auger_customized/pl.dtsi or ~auger_customized/zynq-7000.dtsi. Unfortunately, the names of some of the components have been customized, and the order in the file may not be exactly the same, so this will require careful hand checking. It is good to review the ~sdk_generated/pl.dtsi and zynq-7000.dtsi to verify changed made in Vivado are correctly noted.

5. After merging in any required changes go to the ~auger_customized directory and run the command "make" to compile the source device tree files into the system.dtb file.

Building the first stage boot loader

The first stage boot loader code has been customized by Patrick Allison. Thus we can't just use the standard xSDK setup to build it. Rather we use the project uub-fsbl that he created. But we need to update the ps7_parameters.xml file. We do this automatically by sym-linking the ps7_parameters.xml file in the uub-fsbl project to the one generated by Vivado in the uub_v3.srcs tree. Rebuild the uub-fsbl project. This will create a new uub-fsbl.elf file.

Building the new uboot.bin

Here we combine the fsbl.elf made in the previous step with u-boot.elf. At this time the source code for u-boot.elf is not available, which is a problem for the future. Nevertheless, as long as the the u-boot.elf provided by Roberto is up to date, we can make a new uboot.bin. The entries to fill out for this process are shown in Fig. 2.

Package the files and send for review

Package the files together using ~uub-firmware/package_for_roberto command to put everything together to send to Roberto for checking.

Loading the new uboot.bin onto a UUB

The UUB QSPI memory organization is shown in Fig. 3.

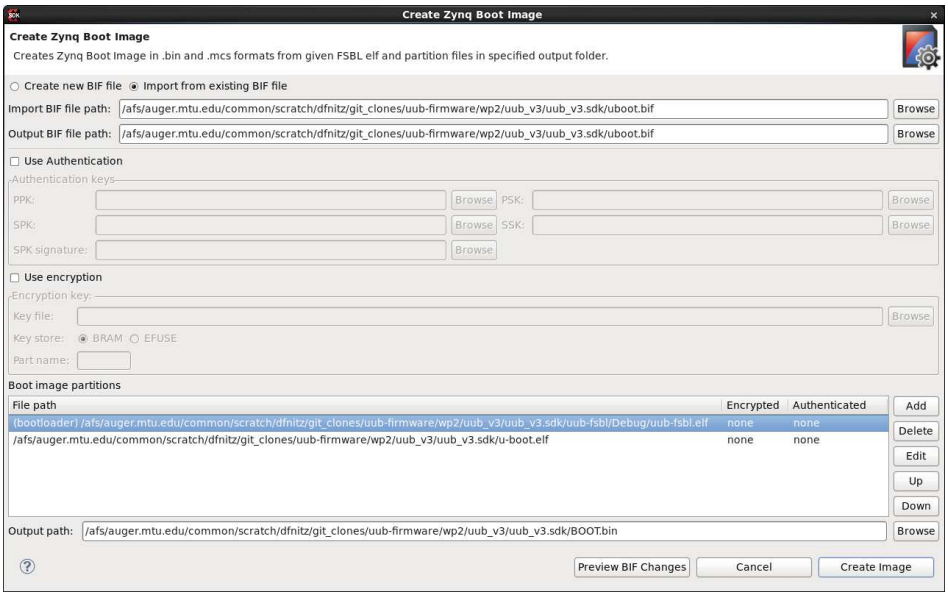

Figure 2: xSDK screen to create uboot.bin

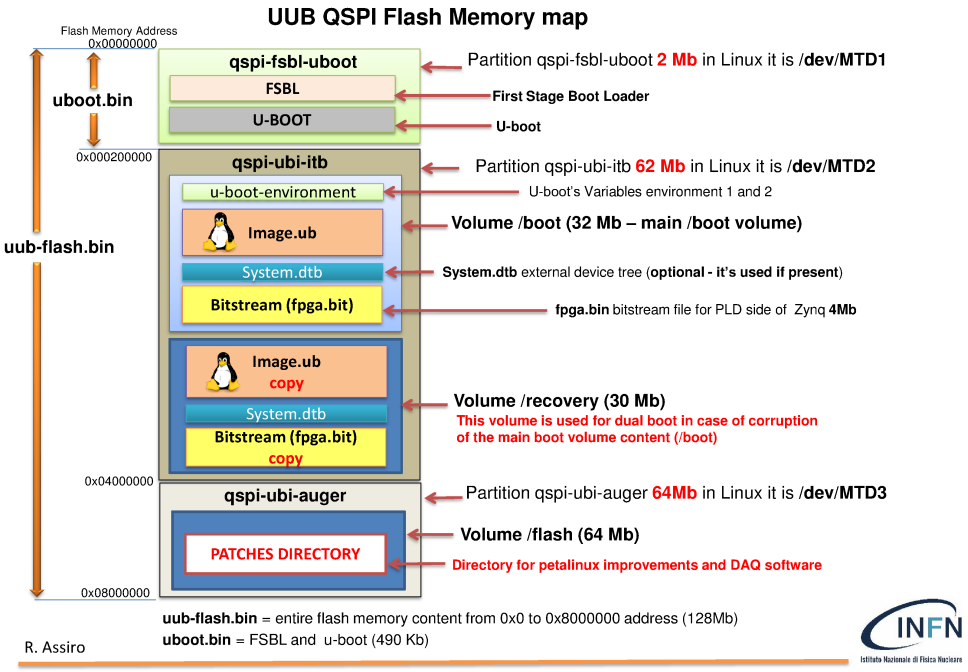

Figure 3: UUB QSPI Flash Memory Map

The new uboot.bin can be loaded from a running Petalinux system. After transferring the new uboot.bin to the system, run the following sequence of commands:

1. flash_erase /dev/mtd1 0 0 to clean the MTD1 partition
2. flashcp uboot.bin /dev/mtd1 to copy the new uboot.bin to the flash memory

If the UUB does not restart, one must use the JTAG programmer to restore a working system. The uub-integration GitHub repository has instructions for doing so.