ECE 4730: Embedded Systems II

Project 4: Linux Compilation

Objectives

* Compile Linux for the FPGA-Arm processor combo.
* Create a FSBL (First Stage Boot Loader).
* Create a U-boot (Universal Boot Loader).
* Create a RAMDISK (a temporary file system mounted for Kernel boot).
* Edit a 'device tree blob' to include the Multiply IP.
* Use the former to boot Linux on the Zybo board from an SD card.

Deliverables

* Booted Kernel	8
* Commented scripts	7
* Answers to the questions	10
Total:	25

Procedure

Part 1- Hardware

1. Open a new project in Vivado as explained in lab 3. Make sure to select the ZYBO board in the parts window.

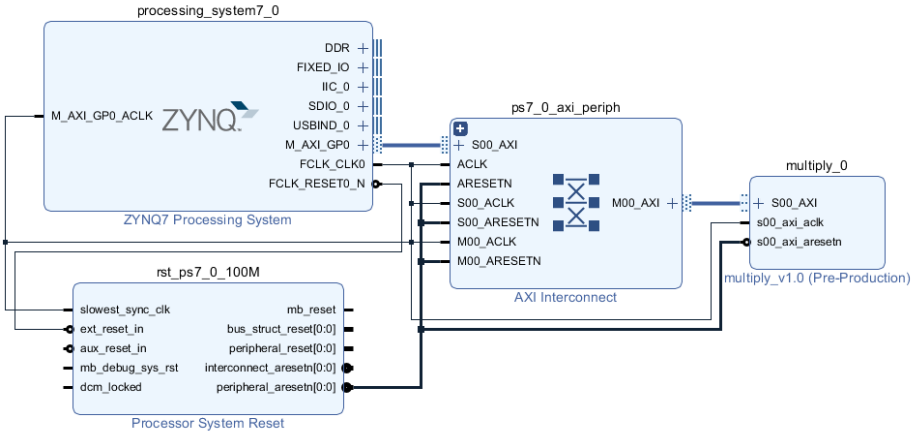
2. Create a Block Design, add the ‘ZYNQ7 Processing System’ IP, and import “ZYBO_zynq_def.xml” from the project directory by double-clicking on the PS and selecting ‘Import XPS Settings’.

3. Run block automation. Re-customize the IP and go to ‘Peripheral I/O pins’ to observe the connected peripherals. Make sure that SD0 and UART1 are enabled. Leave the rest unchanged.

4. Copy the ‘ip_repo’ directory containing ‘multiply’ from Lab 3 and paste it into the lab 4 directory.

5. Click on ‘Settings’ under ‘Project Manager’. Open ‘IP’ and click on ‘Repository’ in the project setting window. Add the ‘ip_repo’ to the IP repositories; then click ‘Apply’.

6. Add your ‘multiply’ IP to the diagram. Then run connection automation and regenerate the layout. The block design should resemble the following:



7. Create an HDL wrapper for your system as done previously.

1: Bitstream Generation

8. Generate the bitstream.

Part 2- boot.bin generation

In order to generate boot.bin, we need the following:

- first stage bootloader (FSBL, hardware-specific)
- bitstream (hardware, already completed)
- Universal Bootloader (U-boot, hardware-agnostic)

2: FSBL Build

1. Open Vivado and export the bitstream and hardware to the SDK as done before. Click File -> New -> Application Project to create a First Stage Bootloader (FSBL).

2. Name the project FSBL and select ‘Next’. In the Templates window select ‘Zynq FSBL’ and finish. *Make sure the ‘Hardware Platform’ selected is the hardware we created in step 1.*

3. Click on Project -> Build All. This builds the FSBL. While this is running you may continue to the next steps.

3: U-BOOT Compilation

Because the u-boot is universal, we do not need the bitstream or hardware specifications for the first part of this step.

> ***Warning: U-boot compilation needs to be done on a partition that has been formatted with the NTFS filesystem because it requires symbolic link generation. Make sure your USB or hard disk is formatted correctly.***

4. Type man tar into the server terminal to read how to untar files.

5. Untar u-boot version 2019.1 from the folder tars-an. *Hint: I would suggest renaming the folder to u-boot and moving it to ~. Read basic Linux commands here.*

6. Navigate to the extracted u-boot directory (now referred to as **<u-boot>**) using the Linux commands “cd” and “ls”.

Two changes must be made to the u-boot source code.

7. Add the following line to the end of the file **<u-boot>**/configs/zynq_zybo_defconfig: CONFIG_OF_EMBED=y

8. Remove the file **<u-boot>**/tools/version.h from the source code. The file is unnecessary and is not written in proper C syntax.

You are now ready to compile U-boot.

9. Navigate to the u-boot directory in the terminal and run u-boot_compile.sh. Comment the file appropriately.
   The output is found in the u-boot directory and named ‘u-boot’.

10. Copy the file to your personal computer and place it in an easily-retrievable location.

11. Add the ‘.elf’ (Executable and Linkable File) extension so Xilinx will recognize the file format to generate boot.bin.

boot.bin Generation

12. Click ‘Xilinx -> Create Boot Image’. In the popup image, select ‘Browse’ to set the path for output.bif. give the path to your lab 4 directory.

13. Generate boot.bin by placing the three files in the following order, with the following partition types:

FSBL	bootloader
Bitstream.bit	datafile
U-boot.elf	datafile

    The following steps will give a few more details:

14. Click add in the boot partitions section. Select ‘Browse’ and find FSBL under XilinxSDK/FSBL/-Debug/. Make sure the partition type is set as bootloader. Select ‘OK’.
15. Add the bistream by searching “*.bit” within the project directory. Set the partition type to datafile (lover of data).
16. Add the U-boot file. Set the partition type to datafile as well.

Part 3- The Post-Boot OS (Linux, DTB, and RAMDISK)

In order to generate the operating system, we need the following:

- Linux (RTOS, hardware-agnostic)
- DTB (Information about all device hardware)
- RAMDISK (Temporary Filesystem, hardware-agnostic)

Different configurations for Linux can be found in the <Linux>/arch/arm/configs/ directory. We will be using the configurations found in xilinx_zynq_defconfig.

4: Linux Build

Warning: Linux compilation needs to be done on a partition that has been formatted with the NTFS filesystem because it requires symbolic link generation. Make sure your USB or hard disk is formatted correctly.

1. Untar the linux folder from tars-an. *Hint: I would suggest renaming the folder to linux and moving it to ~.*
2. The linux_compile.sh script needs to be edited to add **<u-boot>**/tools. Change the path to match the path for that directory.
3. Run the linux_compile.sh script from inside the **<linux>** directory (it is found in the lab 4 directory downloaded with this lab). As usual, please add comments to explain the steps.

You should find uImage in **<Linux>**/arch/arm/boot. Congratulations! You’ve compiled Linux!

5: Device Tree Blob (DTB) Build

4. To compile the .dtb file, we need to edit the file located at <Linux>/arch/arm/boot/dts/zynq-7000.dtsi. Append the text in dtb_append.txt to the end of the file, creating a new line after watchdog0.
5. The dts needs to be converted to a dtb (the compatible format for arm). Run the dtb_build.sh script from the **<linux>** directory.
6. Rename the output (**<Linux>**/arch/arm/boot/dts/zynq-zybo.dtb) to devicetree.dtb and place it in an easily-retrievable location. You will need it.

Congratulations! You’ve built the Device Tree Blob!!

6: Ramdisk Configuration

7. Edit ramdisk.h file to insure the exported path points to the utilities in **<u-boot>**/tools .

8. Run ramdisk.h from any directory, as long as it is the same directory as the ramdisk file.

9. You now have all of the necessary files. Wahooo!

Part 4- Booting and Connecting to the board

1. Change JP5 to SD card mode and power on the ZYBO. Connect the USB cable to the ZYBO and open PuTTY.

2. Use Windows device manager to determine the board's associated COM port.

3. In PuTTY, make sure 'Session' is selected, click on 'Serial', type the name of the COM port (all caps, no spaces) in its field and the following baud rates into the field labeled 'Speed':

4. These Linux and Boot compiles are a bit buggy. The baud rates for the device are the following:

Pre-Boot:	76800	(I believe this is functional for everything)
Boot:	unknown…	
Post-Boot:	9600	

5. The SD card needs the following files placed on it:

Boot.Bin	wherever you set your output.bif
uImage	**<Linux>**/arch/arm/boot
uramdisk.image.gz	wherever you ran ramdisk.sh
devicetree.dtb	**<Linux>**/arch/arm/boot/dts, or where you placed it

6. Pull uImage, uramdisk.tar.gz, and devicetree.dtb from the server if you haven't already.

7. Disconnect the SD card from the PC and plug it into the ZYBO board. Press the reset button to start booting Linux. You should see Linux booting up on the ZYBO board via the PuTTY console.

8. Congratulations!! You've successfully compiled and booted Linux on your very own embedded system!!

Part 5- Your Turn

1. What is the PATH environment variable?

The lab 4 microprocessor system has 512 MB of SDRAM, but our system still includes a small amount of local memory.

2. What is the function of the local memory?

3. Does this 'local memory' exist on a standard motherboard? Where?

After booting, navigate through the various directories.

4. Which are writable? (the “man” command {manual} for ‘ls’ may be helpful).

Create a few files by typing ‘touch <filename>’. If the file is created, then the directory must be writable.

5. What happens to the files when the ZYBO is rebooted? Why?
6. Create a new IP called “speaker” to the match following specifications:

    - Register 0 receives the frequency in Hz.
    - That frequency is output as a square wave to one of the PMOD ports on the Zybo board.
    - The PMOD port is connected to a speaker that will play the note.
    - It will be in your advantage to use the PMOD port.
    - Did I mention the PMOD port is a good idea (though not as practical, it’s still a good idea to use the PMOD port)

    Then rebuild the necessary files to modify your boot process and OS. Consider the following as you create the IP and other files associated with it:

    - You will need to add an output port for your microphone and instantiate it 3x:
        - Under “user ports” in both the wrapper and the instantiated module.
        - At the end of the Verilog wrapper in the IP packager (should be around line 52) (this one is less intuitive).
    - You need to add clk_100MHz for a clock. (lab 2 should be helpful here)
    - Consult the following site for PMOD port pins and their associated .xdc file pins:

    https://reference.digilentinc.com/reference/programmable-logic/zybo-z7/reference-manual

    - Other boot files may have changed due to hardware changes. Which?
    - Post-boot files may need to be changed due to hardware changes. Which?
7. Which steps did you need to re-do, which could you skip? Why?