Kris Burney
Kyle Fischer
Khoi Cao
Group B-2
3/2/17

MP-3

Embedded Linux

Makefile

In the setup.sh file, an environment variable called CROSS_COMPILE is created. This declares a cross compiler toolchain prefix to be used later.

Bootargs

The console arg sets up UART and sets the baud rate to 115200. The root arg sets up the printk function. This allows us to get useful debug information before the kernel is completely setup and before we get access to other print functions.

Xilinx.dts

#gpio-cells = <2>; -- commented out

compatible = "xlnx,axi-gpio-1.01.b", "xlnx,xps-gpio-1.00.a"; -- specifies what Xilinx versions are compatible

gpio-controller ; -- It does something with the gpio controller

reg = < 0x41220000 0x10000 >; -- sets the base address of this reg to 0x41220000 and the reg has a length of 0x10000 bytes.

xlnx,all-inputs = <0x0>; -- Not sure what this does. It could be setting default values for the inputs.

xlnx,all-inputs-2 = <0x0>; -- Not sure what this does. It could be setting default values for the inputs.

xlnx,dout-default = <0x0>; -- Sets the default output.

xlnx,dout-default-2 = <0x0>; -- Sets the default output

xlnx,gpio-width = <0x8>; -- Sets the width of the first gpio.

xlnx,gpio2-width = <0x20>; -- Sets the width of the second gpio.

xlnx,instance = "LEDs_8Bits"; -- Names the instance of the device.

xlnx,interrupt-present = <0x0>; -- Sets up an interrupt. Possibly a status reg.

Kris Burney
Kyle Fischer
Khoi Cao
Group B-2
3/2/17

xlnx,is-dual = <0x0>; -- Filling a status reg. Setting it to false.

xlnx,tri-default = <0xffffffff>; -- Sets a default value.

xlnx,tri-default-2 = <0xffffffff>; -- Sets a default value.

My First Boot

Memory: ECC disabled

DRAM: 512 MiB

WARNING: Caches not enabled

MMC: zynq_sdhci: 0

SF: Detected S25FL256S_64K with page size 64 KiB, total 32 MiB

***** Warning - bad CRC, using default environment**

These messages tell us about the system that the linux is running on.

In: serial

Out: serial

Err: serial

These specify standard in and standard out. This is also where error messages are sent.

usbcore: registered new interface driver usbhid

usbhid: USB HID core driver

usbcore: registered new interface driver usbfs

usbcore: registered new interface driver hub

usbcore: registered new device driver usb

New drivers were found for the USB and they were loaded.

Kris Burney
Kyle Fischer
Khoi Cao
Group B-2
3/2/17

Linux version 3.9.0-xilinx (kfisch13@linux-4.ece.iastate.edu) (gcc version 4.7.2 (Sourcery CodeBench Lite 2012.09-104)) #1 SMP PREEMPT Thu Mar 2 18:02:29 CST 2017 -- Linux kernel and the boot time, gcc compiler version

CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=18c5387d -- CPU version

CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache -- CPU configuration

Machine: Xilinx Zynq Platform, model: Xilinx Zynq -- vendor name

cma: CMA: reserved 16 MiB at 1e400000 -- reserved memory for CMA at boot time

Memory policy: ECC disabled, Data cache writealloc -- disable EEC mode and allow the memory allocation using pointer

RAMDISK: gzip image found at block 0 -- boot image on RAM memory

++ Mounting filesystem

FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be corrupt. Please run fsck.

We tried to mount our FAT file system but it wasn't mounted properly. It says to run some cleanup.

++ Setting up mdev

++ Starting telnet daemon

++ Starting http daemon

++ Starting ftp daemon

++ Starting ssh daemon

These are setting up ways to connect to our linux system.

Kris Burney
Kyle Fischer
Khoi Cao
Group B-2
3/2/17

Zynq clock init

This one seemed important. Sets up our clock.

Console: colour dummy device 80x30

This says that our device doesn't have color output. It is just a dummy.

Copying Linux from SD to RAM...

reading uImage

reading devicetree.dtb

reading uramdisk.image.gz

These are reading some of the files we had on our SD card.

Starting kernel ...

The kernel was started …

Plugged in the Turret for the First Time

usb 1-1: new low-speed USB device number 2 using xusbps-ehci	found a new device
usb 1-1: New USB device found, idVendor=2123, idProduct=1010	self declared device properties
usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=0	self declared device properties
usb 1-1: Product: USB Missile Launcher	recognized that the device is a missile launcher
usb 1-1: Manufacturer: Syntek	Found the manufacturer info
hid-generic 0003:2123:1010.0001: device has no listeners, quitting	Nothing was using this device so linux stopped putting in an effort

Kris Burney
Kyle Fischer
Khoi Cao
Group B-2
3/2/17

Changes to launcher_driver.c

1. Removed the references to “skel”
2. Usually ended up saying “eton” in place of “skeleton”
3. Replaced the IDs in line 33 to their appropriate defines in launcher_commands.h

How launcher_fire.c Works

This file has a main function and a helper function. The helper function has some debug information about the launch and it has the actual driver call in it. The main function gets a reference to the driver, calls the helper function to start and stop firing, and closes the reference to the driver.

Target Acquisition

To start off, we didn’t get this part to work. We spent over 10 hours trying to get the /dev/mem to read our framebuffer but we couldn’t get it. We tried the other two frame buffers, we tried overwriting our image in software to see any changes, we had issues with linux not launching our kernel at all, and we had issues with it thinking we had a corrupt filesystem. The rest of this is the algorithm we would have started with. We expected it to be slow since it involves a lot of image processing but we would have optimized it once we got it working.

Some notes that help our explanation:

- The target is a circle; the very top pixel will be in the correct x-axis coordinate. Similarly for the y-axis on the left side of the circle.
- We would have a helper function to tell if the current pixel we are comparing is reasonably close in color to the target’s color or to the white between the rings of the target. We would figure out what color the camera reads our target as separately from running our acquisition program.

We would first read the image left to right, top to bottom, just like a book. We will call this reading it by rows. When we found a pixel that was close to the color of our target, we would then start comparing the pixels below it on the screen. If the color of the pixels alternated in a target color, white, target color pattern (it would alternate enough to check through all 3 rings of the target) we would say that we found the target. This gets us the x coordinate that we need to move. We would then do the same thing to find the y coordinate via reading columns.

Kris Burney
Kyle Fischer
Khoi Cao
Group B-2
3/2/17

Once we had the x and y coordinates that we needed to move, we would move the camera relative to the values we found. It wouldn't be a 1 to 1 matching but we would figure this out using guess and check. After we moved we would fire and go back to searching for a target.

We thought that by checking the alternating pattern of the target we would be able to detect the target further away. If we checked a specific number of pixels we could speed up the detection at 3 feet but we wouldn't be able to find it further away.