Mauro Lopez

LAB 4 Report

ECEN-449

Section 502

October 3, 2018

Introduction:

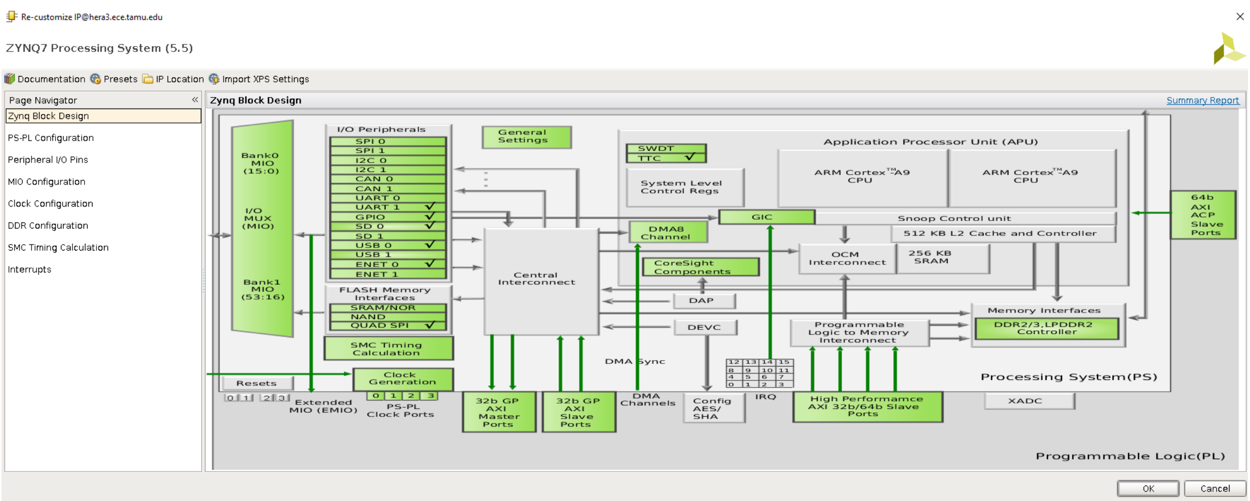
In this lab we will work on implementing the Linux operating system into our ZYBO board via SD card. We will be creating our microprocessor to be compatible with Linux. Using our FSBL and u-boot we are going to create an Zynq Boot Image. In this lab we will be creating each component to make our board load Linux from our SD card.

Procedure:

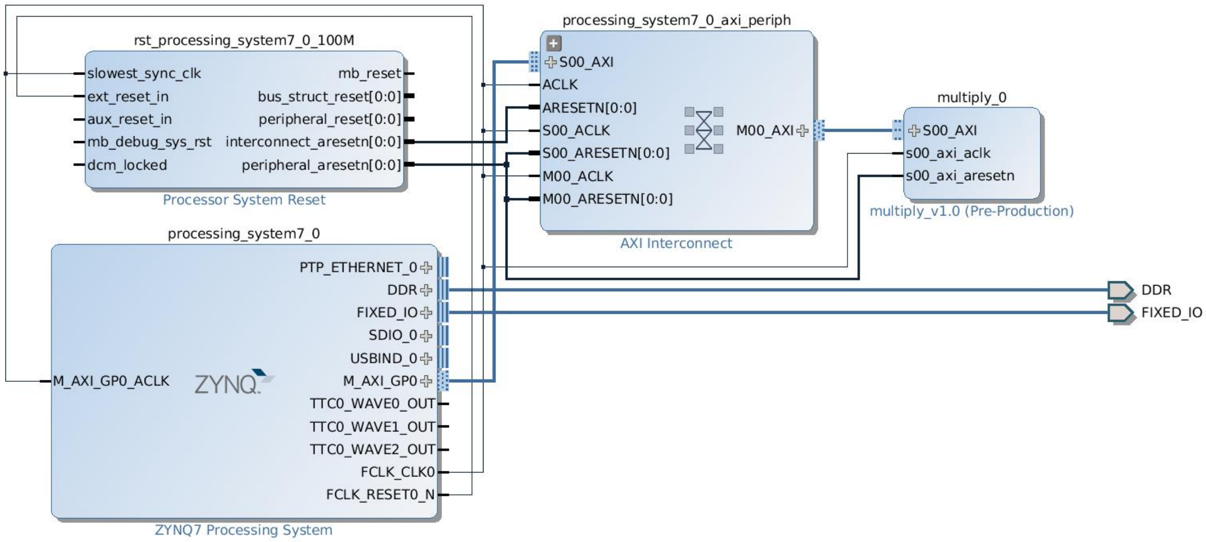
This lab will be like our previous lab, in which we implemented our own custom multiplication peripheral. We will be adding a SD card drive, a timer and a DDR3 controller.

We start by creating a new project using Vivado and selecting ZYBO boards. Then we add “ZYNQ7 Processing System “, and we import the ‘ZYBO_zynq_def.xml’ from the 449NeededFiles directory.

We then run the block automation. Now we need to look our pins and only turn on SD 0 , UART 1 and TTC 0.



To add our multiplication IP, we need to copy it from our lab 3. We simply copy the repo folder and copy it into our lab 4 directory. We then add it to our IP library. Finally, we then add the block to our block diagram, and run connection automation. Our diagram should look like this:



We then create our HDL wrapper and generate our bitstream.

u-boot:

We first need to get the file and un tar it into our lab4 directory.

To compile our u-boot file, we need to use our tools from Vivado to compile it using the following command.

- Make CROSS_COMPILE=arm-xilinx-linux-gnueabi- zyng_zybo_config

This will configure our boot loader to our ZYBO board.

With this, we can now configure our u-boot by typing the following command.

- Make CROSS_COMPILE=arm-xilinx-linux-gueabi-

Once the compiler has finish we will end up with an ELF (Executable and Linkable File). In order for Xilinx SDK to recognize our file we need to add ".elf" at the end of our file.

Now we move on, and create our boot.bin file. We do this by exporting our bitstream, and launching SDK. We need to create our First Sage Bootloader(FSBL). We create a new project and use the Zynq FSBL template, and build all.

Now we work on creating our Image. We change the destination, to our lab 4 directory. We need to add our FSBL.elf as the bootloader, and system.bit and u-boot.elf as data files. We then create the image, and it'll appear

Linux Kernel:

We need to un tar the file "linux-3.14.tar.gz" into our lab 4 directory. Once we have the file into our directory, we need to configure our Linux Kernel for our ZYBO board. To compile this we use the following commands:

- Make ARCH=arm CROSS_COMPOLE=arm-xilinx-linuxgnueabi- Xilinx_zynq_defconfig

- make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi-

once this is complete we should see the zImage in our directory. The image is however in a zip, to unzip this file we need to use the tools from our u-boot. We use the following command for that

first we add the u-boot tool to our path.

- PATH=$PATH:<directory_to_u_boot>/tools

With this, we can now go to our zimage and type the following commands in our terminal

- Make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- UIMAGE_LOADADDR=0x8000 uImage.

We also need a our .dtb file.

We need to edit our IP for multiply. Open the multiply in the .dts file and add the following

```
Multiply {
    Compatible = "ecen449,multiply";
    Reg=<0x43C00000 0X10000>;
};
```

This will store the address of our multiplier.

Finally we need to convert our .dts to .dtb.

We use the following command:

- ./scrips/dtc/dtc -I dts -O -dtb -o ./devicetree.dtb arch/arm/boot/dts/zynq-zybo.dts

Then we copy the ramdisk file for the ECEN449 directory. To crate our ramdisk8m file we type the following command.

- ./u-boot/tools/mkimage -A arm -T ramdisk -c gzip -d ./ramdisk8m.image.gz uramdisk.image.gz

Once we completed this, we have all of our components to boot Linux into our board.

Boot Linux on ZYBO

To have a visual on the board, we have to use PICOCOM. We change the ZYBO board, to SD card. We launch PICOCOM by using the following commands:

- Picocom -b 115200 -r -l /dev/ttyUSB1

Once we have this, we upload our BOOT.bin, uImage, uramdisk.image.gz and devicetree.dtb into our SD card. Once it completes, we place the SD card into the board and reset the board.

If everything is successful, we should see Linux booting up. We get the following screen

```
Starting kernel ...

Booting Linux on physical CPU 0x0
Linux version 3.18.0-xilinx (mauro31@lin16-424cvlb.ece.tamu.edu) (gcc version 4.9.1 (Sourcery CodeBench Lite 2014.11-30) ) #1 SMP PREEMPT Fri Sep 28 18:03:12 CDT 2018
CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=18c5387d
CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
Machine model: Xilinx Zynq
cma: Reserved 16 MiB at 0x1e400000
Memory policy: Data cache writealloc
PERCPU: Embedded 10 pages/cpu @5fbd3000 s8768 r8192 d24000 u40960
Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 130048
Kernel command line: console=ttyPS0,115200 root=/dev/ram rw earlyprintk
PID hash table entries: 2048 (order: 1, 8192 bytes)
Dentry cache hash table entries: 65536 (order: 6, 262144 bytes)
Inode-cache hash table entries: 32768 (order: 5, 131072 bytes)
Memory: 492632K/524288K available (4650K kernel code, 258K rwdata, 1616K rodata, 212K init, 219K bss, 31656K reserved, 0K highmem)
Virtual kernel memory layout:
    vector  : 0xffff0000 - 0xffff1000   (   4 kB)
    fixmap  : 0xffc00000 - 0xffe00000   (2048 kB)
    vmalloc : 0x60800000 - 0xff000000   (2536 MB)
    lowmem  : 0x40000000 - 0x60000000   ( 512 MB)
    pkmap   : 0x3fe00000 - 0x40000000   (   2 MB)
    modules : 0x3f000000 - 0x3fe00000   (  14 MB)
      .text : 0x40008000 - 0x40626b1c   (6267 kB)
      .init : 0x40627000 - 0x4065c000   ( 212 kB)
      .data : 0x4065c000 - 0x4069cb60   ( 259 kB)
       .bss : 0x4069cb60 - 0x406d3a78   ( 220 kB)
Preemptible hierarchical RCU implementation.
        Dump stacks of tasks blocking RCU-preempt GP.
        RCU restricting CPUs from NR_CPUS=4 to nr_cpu_ids=2.
RCU: Adjusting geometry for rcu_fanout_leaf=16, nr_cpu_ids=2
NR_IRQS:16 nr_irqs:16 16
L2C-310 erratum 769419 enabled
L2C-310 enabling early BRESP for Cortex-A9
L2C-310 full line of zeros enabled for Cortex-A9
L2C-310 ID prefetch enabled, offset 1 lines
L2C-310 dynamic clock gating enabled, standby mode enabled
L2C-310 cache controller enabled, 8 ways, 512 kB
L2C-310: CACHE_ID 0x410000c8, AUX_CTRL 0x76360001
ps7-slcr mapped to 60804000
zynq_clock_init: clkc starts at 60804100
Zynq clock init
sched_clock: 64 bits at 325MHz, resolution 3ns, wraps every 3383112499200ns
ps7-ttc #0 at 60806000, irq=43
Console: colour dummy device 80x30
Calibrating delay loop... 1292.69 BogoMIPS (lpj=6463488)
pid_max: default: 32768 minimum: 301
Mount-cache hash table entries: 1024 (order: 0, 4096 bytes)
Mountpoint-cache hash table entries: 1024 (order: 0, 4096 bytes)
CPU: Testing write buffer coherency: ok
```

Results:

When we have Linux into our board, we should see the image above. If you type “ls”, to list all the files, we get the following:

```
can: netlink gateway (rev 20130117) max_hops=1
zynq_pm_ioremap: no compatible node found for 'xlnx,zynq-ddrc-a05'
zynq_pm_late_init: Unable to map DDRC IO memory.
Registering SWP/SWPB emulation handler
drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
ALSA device list:
  No soundcards found.
RAMDISK: gzip image found at block 0
mmc0: new high speed SDHC card at address aaaa
mmcblk0: mmc0:aaaa SS08G 7.40 GiB
 mmcblk0: p1
EXT2-fs (ram0): warning: mounting unchecked fs, running e2fsck is recommended
VFS: Mounted root (ext2 filesystem) on device 1:0.
devtmpfs: mounted
Freeing unused kernel memory: 212K (40627000 - 4065c000)
Starting rcS...
++ Mounting filesystem
++ Setting up mdev
++ Starting telnet daemon
++ Starting http daemon
++ Starting ftp daemon
++ Starting dropbear (ssh) daemon
random: dropbear urandom read with 1 bits of entropy available
rcS Complete
zynq> ls
bin         lib         lost+found  proc        sys         var
dev         licenses    mnt         root        tmp
etc         linuxrc     opt         sbin        usr
zynq> cd bin
zynq> ls
addgroup       dnsdomainname  iplink         mt             setarch
adduser        dumpkmap       iproute        mv             sh
ash            echo           iprule         netstat        sleep
base64         ed             iptunnel       nice           stat
busybox        egrep          kill           pidof          stty
cat            false          linux32        ping           su
catv           fdflush        linux64        ping6          sync
chattr         fgrep          ln             pipe_progress  tar
chgrp          fsync          login          powertop       touch
chmod          getopt         ls             printenv       true
chown          grep           lsattr         ps             umount
cp             gunzip         lzop           pwd            uname
cpio           gzip           makemime       reformime      uncompress
cttyhack       hostname       mkdir          rev            usleep
date           hush           mknod          rm             vi
dd             ionice         mktemp         rmdir          watch
delgroup       iostat         more           rpm            zcat
deluser        ip             mount          run-parts
df             ipaddr         mountpoint     scriptreplay
dmesg          ipcalc         mpstat         sed
zynq> 
```

Here we can see the initial files, that are in the board already.

Since this is running Linux we can use the same commands to navigate trough the files.

Overall, the lab was a success and I was able to boot Linux from the FPGA board.

Conclusion:

In this lab I followed the steps necessary to design our microprocessor to run Linux. I started off by creating a Vivado project, copying the repo folder from our multiply from lab 3. Adding the necessary peripherals to the Zynq processor. Creating the HDL wrapper and making the bit stream. Launch SDK and create a zImage using the bit stream and our u-boot. Finally I created the uImage using our zImage. Edited our .dts file, and convert it to our .dtb file. lastly I copied the ramdisk file from the ECEN449 directory, and compile it for our board. I then use the sd card to store all of my components, and uploaded it into the zboard and use the PICOCOM command to visualize what was happening.

Questions:

Compared to lab 3, the lab 4 microprocessor system shown in Figure 1 has 512 MB of SDRAM. However, our system still includes a small amount of local memory. What is the function of the local memory? Does this ‘local memory” exist on a standard motherboard? If so, where?

The local memory stores the bootloader that are being executed when launching the operating system. This bootloader once its running, it fetches the components from the disk.

After your Linux system boots, navigate through the various directories. Determine which of these directories are writable. (Note that the man page for ’ls’ may be helpful). Test the permissions by typing ‘touch ’ in each of the directories. If the file, , is created, that directory is writable. Suppose you are able to create a file in one of these directories. What happens to this file when you restart the ZYBO board? Why?

```
zynq> ls -l
total 24
drwxr-xr-x    2 12319    300           2048 Jan  1 00:13 bin
drwxr-xr-x    6 root     0             2500 Jan  1 00:14 dev
drwxr-xr-x    4 12319    300           1024 Jan  1 00:00 etc
drwxr-xr-x    3 12319    300           2048 Jul 12  2012 lib
drwxr-xr-x   11 12319    300           1024 Jan  9  2012 licenses
lrwxrwxrwx    1 12319    300             11 Jan  9  2012 linuxrc -> bin/busybox
drwx------    2 root     0            12288 Jan  9  2012 lost+found
drwxr-xr-x    2 12319    300           1024 Aug 21  2010 mnt
drwxr-xr-x    2 12319    300           1024 Aug 21  2010 opt
dr-xr-xr-x   53 root     0                0 Jan  1 00:00 proc
drwxr-xr-x    2 12319    300           1024 Jul 12  2012 root
drwxr-xr-x    2 12319    300           1024 Jan  9  2012 sbin
dr-xr-xr-x   12 root     0                0 Jan  1 00:00 sys
drwxrwxrwt    2 root     0               40 Jan  1 00:00 tmp
drwxr-xr-x    5 12319    300           1024 Mar 30  2012 usr
drwxr-xr-x    4 12319    300           1024 Oct 25  2010 var
zynq>
```

mauro31@lin16-424cvlb:~

File Edit View Search Terminal Help

```
dr-xr-xr-x   12 root     0                0 Jan  1 00:00 sys
drwxrwxrwt    2 root     0               40 Jan  1 00:00 tmp
drwxr-xr-x    5 12319    300           1024 Mar 30  2012 usr
drwxr-xr-x    4 12319    300           1024 Oct 25  2010 var
zynq> cd proc
zynq> touch textfile.txt
touch: textfile.txt: No such file or directory
zynq> cd sys
zynq> touch textfile.txt
touch: textfile.txt: No such file or directory
zynq> cd etc
-/bin/ash: cd: can't cd to etc
zynq> cd lib
-/bin/ash: cd: can't cd to lib
zynq> cd /
zynq> cd sys
zynq> touch textfile.txt
touch: textfile.txt: Permission denied
zynq> ls
block     class     devices   fs        module
bus       dev       firmware  kernel    power
zynq> pwd
/sys
zynq> cd /
zynq> ls
bin         lib         lost+found  proc        sys         var
dev         licenses    mnt         root        tmp
etc         linuxrc     opt         sbin        usr
zynq> cd bin
zynq> ls
addgroup       dnsdomainname  ipcalc         mpstat         sed
adduser        dumpkmap       iplink         mt             setarch
ash            echo           iproute        mv             sh
base64         ed             iprule         netstat        sleep
busybox        egrep          iptunnel       nice           stat
cat            false          kill           pidof          stty
catv           fdflush        linux32        ping           su
chattr         fgrep          linux64        ping6          sync
chgrp          fsync          ln             pipe_progress  tar
chmod          getopt         login          powertop       touch
chown          grep           ls             printenv       true
cp             gunzip         lsattr         ps             umount
cpio           gzip           lzop           pwd            uname
cttyhack       helloworl.txt  makemime       reformime      uncompress
date           hostname       mkdir          rev            usleep
dd             hush           mknod          rm             vi
delgroup       ionice         mktemp         rmdir          watch
deluser        iostat         more           rpm            zcat
df             ip             mount          run-parts
dmesg          ipaddr         mountpoint     scriptreplay
zynq>
```

Applications Places Terminal Tue 12:06

mauro31@lin16-424cvlb:~

File Edit View Search Terminal Help

```
zynq> cd /
zynq> ls
bin         lib         lost+found  proc        sys         var
dev         licenses    mnt         root        tmp
etc         linuxrc     opt         sbin        usr
zynq> touch testingfile.txt
zynq> ls
bin              linuxrc          root             usr
dev              lost+found       sbin             var
etc              mnt              sys
lib              opt              testingfile.txt
licenses         proc             tmp
zynq> mkdir testdir
zynq> ls
bin              linuxrc          root             tmp
dev              lost+found       sbin             usr
etc              mnt              sys              var
lib              opt              testdir
licenses         proc             testingfile.txt
zynq> touch helloword.cpp
zynq> ls
bin              licenses         proc             testingfile.txt
dev              linuxrc          root             tmp
etc              lost+found       sbin             usr
helloword.cpp    mnt              sys              var
lib              opt              testdir
zynq>
```

Balance for mauro31 | lab4.pdf - Mozilla Firefox | mauro31@lin16-424cvlb:~ | Check Linux file permissions with ls ... | 1 / 4

If we use the ls -l command into our board Linux, we can see the permissions of each directory. If we do, have permission a “w” will be placed in front of the directory file. If we reset the board, we go back to zero and all our files get deleted, since it boots up from the beginning.

If you were to add another peripheral to your system after compiling the kernel, which of the above steps would you have to repeat? Why?

To add another peripheral, I would go back to our block diagram and add this peripheral. Once that has been done, I run the connection automation. I must recreate a new bit stream, which means I have to re-create the u-image. Lastly I re-generate a new .dtb tree. Other than that it should all be the same.