EMBEDDED LINUX KERNEL DEVELOPMENT REPORT

1 INTRODUCTION

The goal of this Lab is to run a Linux distribution on a ZedBoard development board and to setup application and module development on the CortexA9/Linux system.

2 PLATFORM CONFIGURATION

We first need to configure the board in order to run the Linux kernel, following this diagram.

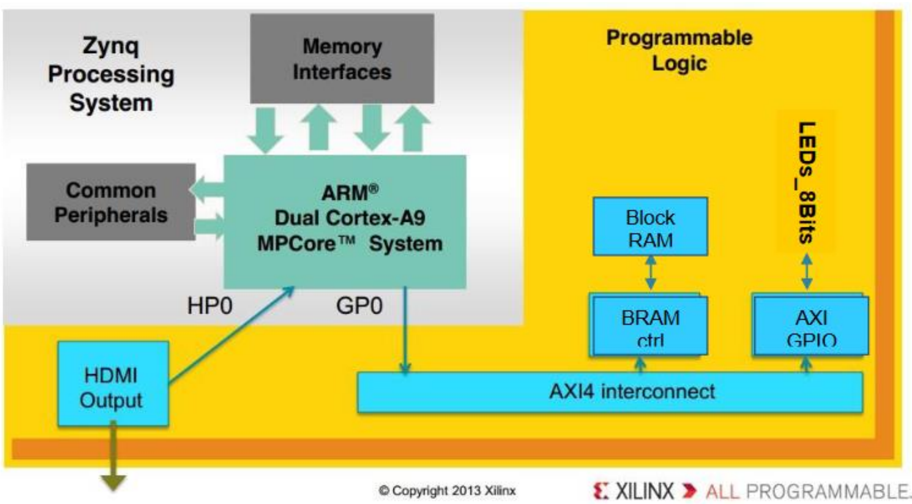


Figure 1- platform configuration diagram

Using Vivado we can implement this diagram.

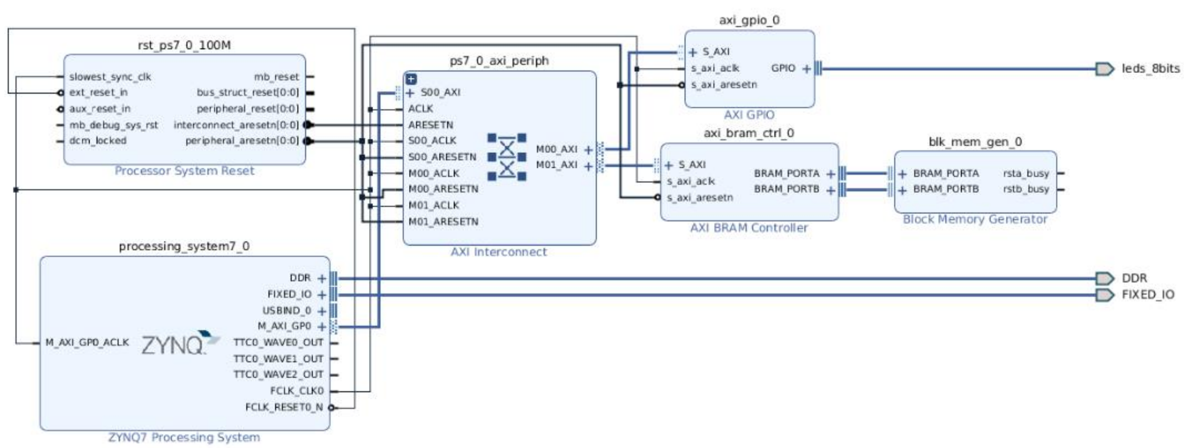


Figure 2- platform configuration diagram on Vivado

Still in Vivado, we can run the RTL analysis and the synthesis to get the final layout we will export to the board.

We can see the RAM slots as well as the used LUT highlighted on this view.

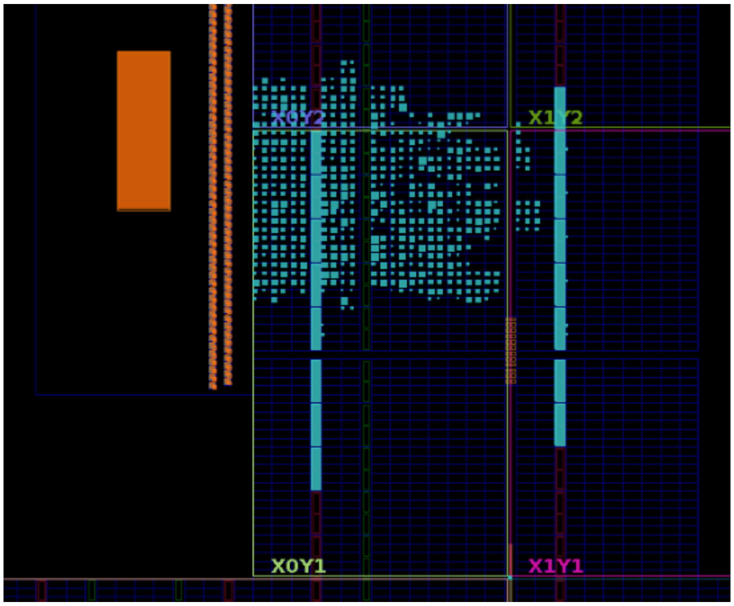


Figure 3- Zynq platform occupation

As a last step we need to check the design integrity. Checking if the GPIO0 address is the right one should suffice to assert the validity of the design.

```
axi_gpio_0: gpio@41200000 {
        #gpio-cells = <2>;
        compatible = "xlnx,xps-gpio-1.00.a";
        gpio-controller ;
        reg = <0x41200000 0x10000>;
        xlnx,all-inputs = <0x0>;
        xlnx,all-inputs-2 = <0x0>;
        xlnx,all-outputs = <0x1>;
        xlnx,all-outputs-2 = <0x0>;
        xlnx,dout-default = <0x00000000>;
        xlnx,dout-default-2 = <0x00000000>;
        xlnx,gpio-width = <0x8>;
        xlnx,gpio2-width = <0x20>;
        xlnx,interrupt-present = <0x0>;
        xlnx,is-dual = <0x0>;
        xlnx,tri-default = <0xFFFFFFFF>;
        xlnx,tri-default-2 = <0xFFFFFFFF>;
};
```

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 0x40000000 [1G])					
axi_gpio_0	S_AXI	Reg	0x4120_0000	64K	0x4120_FFFF
axi_bram_ctrl_0	S_AXI	Mem0	0x4000_0000	64K	0x4000_FFFF

Figure 4- GPIO0 address check

Comparing the GPIO0 address in the generated .dtsi file and the address written in Vivado, we indeed get the same value.

We can now export a HelloWorld C code to the board to check if all design is running correctly (using *minicom* to receive the *printf*).

```
Hello World
```

Figure 5- hello word C code running on the board

3 LINUX KERNEL

3.1 KERNEL CONFIGURATION

Before compiling the kernel, we need to make sure that the compilation will target an ARM architecture, as it is the type of processor that is embedded on our FPGA. To do that, we can either run the command "**export ARCH=arm"** to update the environment variable or set the variable when calling the compilation command with "**ARCH=arm**". We can check the target platform in the configuration menu.

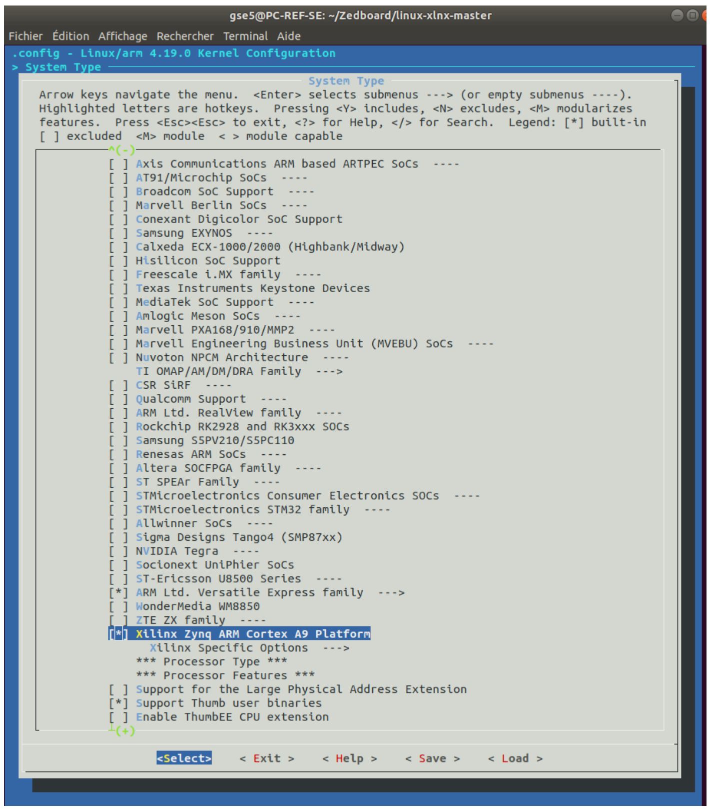


Figure 6- kernel configuration interface

The target processor is an ARM Cortex A9 type, which is what we want.

3.2 CROSS COMPILE PROGRAM

When compiling a program or kernel module for the Cortex processor from our main computer (that has an intel processor), we need to use a specific toolchain of cross-compilation. For that we use the tool **arm-linux-gnueabihf**. We include the tool in the toolchain with the environment variable CROSS_COMPILE.

export CROSS_COMPILE=arm-linux-gnueabihf-

We can manually compile a program by writing "**arm-linux-gnueabihf-gcc** [source files]" or **"${CROSS_COMPILE}gcc** [sourcefiles]

```
gse5@PC-REF-SE:~/Zedboard$ ${CROSS_COMPILE}gcc -o source/hello_world source/hello_world.c
```

```
gse5@PC-REF-SE:~/Zedboard$ arm-linux-gnueabihf-gcc -o source/hello_world source/hello_world.c -static
```

Figure 7- cross compile hello_world code

We copy the executable file on the *rootfs* partition of the SD card.

```
gse5@PC-REF-SE:~/Zedboard$ sudo cp source/hello_world /media/gse5/rootfs/
[sudo] Mot de passe de gse5 :
gse5@PC-REF-SE:~/Zedboard$
```

Figure 8- copy program to board file system

Now can test the cross compiled program on our ZedBoard.

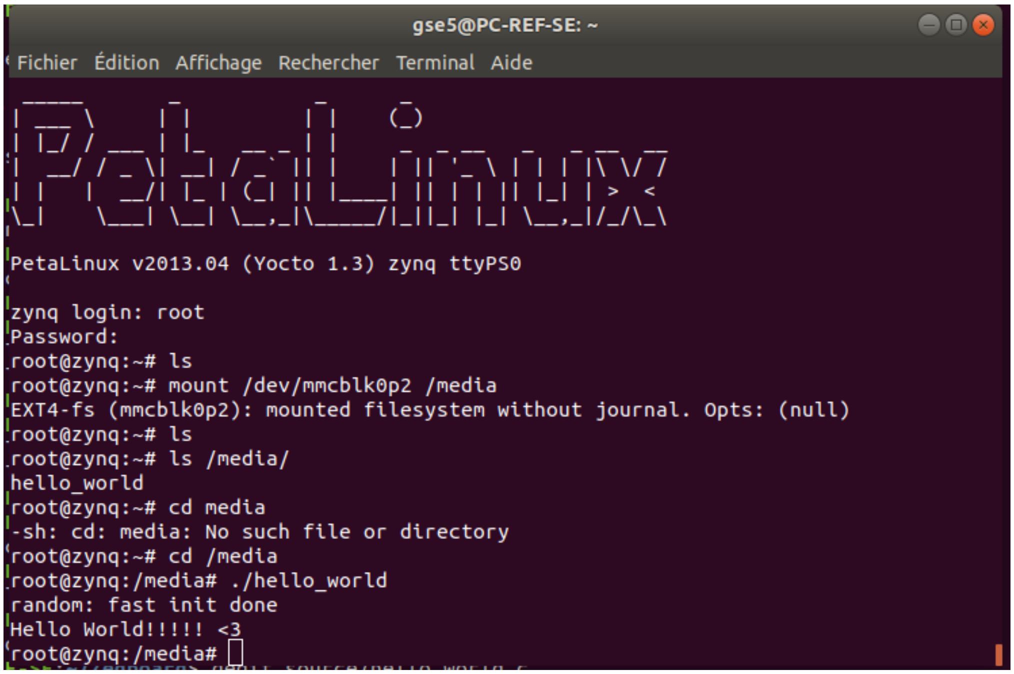
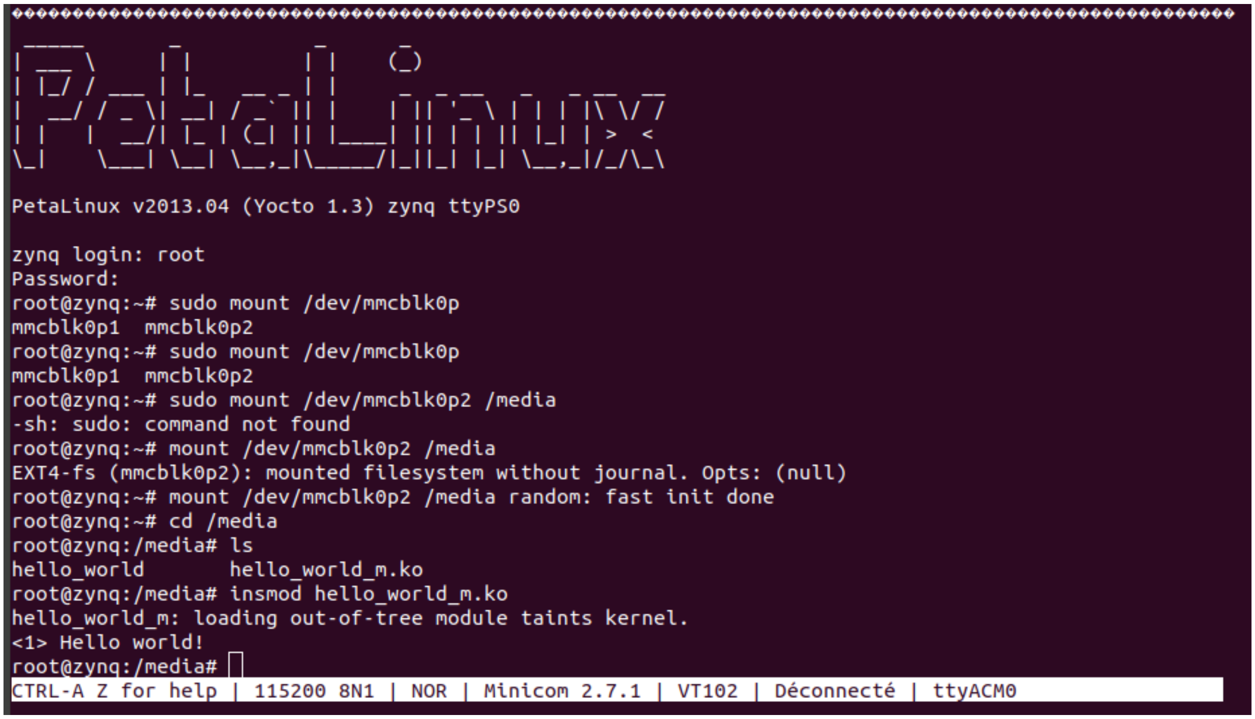


Figure 9- run hello_world on platform

3.3 MODULE DEVELOPMENT

Similarly, we can compile a kernel module that can be run on a Cortex processor. We compile the "Hello_World" program found in the slides of chapter 1 of the course. We compile it with a Makefile based on the template found in the same slides. Then we copy the *hello_world.ko* file on the *rootfs* partition of the SD card.

To install the kernel module from the .ko file, we use the "**insmod**" command. The result ouputs "hello world" to the standard output.



```
PetaLinux v2013.04 (Yocto 1.3) zynq ttyPS0

zynq login: root
Password:
root@zynq:~# sudo mount /dev/mmcblk0p
mmcblk0p1  mmcblk0p2
root@zynq:~# sudo mount /dev/mmcblk0p
mmcblk0p1  mmcblk0p2
root@zynq:~# sudo mount /dev/mmcblk0p2 /media
-sh: sudo: command not found
root@zynq:~# mount /dev/mmcblk0p2 /media
EXT4-fs (mmcblk0p2): mounted filesystem without journal. Opts: (null)
root@zynq:~# mount /dev/mmcblk0p2 /media random: fast init done
root@zynq:~# cd /media
root@zynq:/media# ls
hello_world       hello_world_m.ko
root@zynq:/media# insmod hello_world_m.ko
hello_world_m: loading out-of-tree module taints kernel.
<1> Hello world!
root@zynq:/media#
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Déconnecté | ttyACM0
```

Figure 10- running hello_world module on platform

4 CONCLUSION

We can now compile a Linux kernel on the wanted platform, and develop some modules to customize it. The next step should be to develop a working Linux driver to run on our custom platform.