Tutorial / Lab n° 4
Building an Embedded System

The purpose of this TD is to synthesize the creation of an embedded system including all phases necessary to obtain a system running on different architecture from your workstation and create a functional system.

1 Which embedded system?

To build a cross-compiling toolchain, we must determine the type of material for which we hope to implement this tool. We will work with qemu to emulate an ARM system. You can see all the platforms that qemu is able to emulate using the command:

```
qemu-system-arm -M \?
```

We will work with default qemu emulated platform namely:

```
integratorcp926 ARM Integrator/CP (ARM926EJ-S)
```

So we configure all the tools for the characteristics of this machine.

2 Building a cross-compiling toolchain

Our aim now is to build a cross compiler environment to obtain a functional system. We will use BuildRoot. You will need about 2GB of free space on the partition on which you install BuildRoot.

2.1 Getting BuildRoot

Start by retrieving the disk image containing the installation package BuildRoot and necessary (to avoid too long downloads):

```
http://kistren.polytech.unice.fr/cours/iam03/td6/hdb-buildroot.qcow2.bz2
```

This disk image integrates the following tools:

```
http://kistren.polytech.unice.fr/cours/iam03/td6/buildroot-20090114-clean.tar.bz2
http://kistren.polytech.unice.fr/cours/iam03/td6/packages.tar
```

2.2 BuildRoot Configuration

Configure the BuildRoot sources (`make menuconfig`) with the following options. You will learn by yourself of a particular option using the help menu.

```
Target Architecture: arm
Target Architecture variant: arm926t
Target ABI: EABI
Target options : ARM Ltd Device Support / Integrator 926
Build options :
  Toolchain and header file location: /work/toolchain
  Number of jobs to run simultaneously: 2
  strip: sstrip
Toolchain
  Kernel headers: Latest Linux 2.6.26.x kernel headers
  Binutils version: binutils 2.19
  Install sstrip for the target system: yes
  Include target utils in cross toolchain: no
Package selection for the target: unselect all
Target filesystem options: unselect all
Kernel type: none
```

Remember that a cross toolchain is not relocatable on your file system. So choose an appropriate manner where your cross toolchain will be installed (in our case `/work/toolchain` for example). If for some reason or another, you need to relocate your cross toolchain, it will create a link from the old location (the one specified at compile time) to the new location.
You'll notice that BuildRoot also allows compiling a kernel and creating the image of a filesystem for the target platform. We will not use these features in this lab to preserve time and in particular to avoid having the time of compiling the kernel that is additional to the already lengthy process. And we produced ourselves a filesystem with BusyBox so, we won't do it one more time. But know that these processes can be automated through BuildRoot.

2.3 Compilation of BuildRoot and options of cross compiling toolchain

Now we will to the compilation process with a simple `make` command (do not use `make-j X` because if it will lead to an error, you probably will not see it in the console logs).

Depending on the speed of your machine (and your Internet connection as necessary packages are downloaded when required), this operation may take between 45 and 90 minutes. When compiling, you can watch what is created in the directory `/work/toolchain`.

After successful compilation, you can delete the source directory, so you will get a lot of space on your disk.

3 Using the cross-compiling toolchain

3.1 Using a kernel to boot the ARM target

You may download the following file containing a Linux kernel compiled for ARM target.

```
http://kistren.polytech.unice.fr/cours/iam03/td6/zImage.arm
```

You can then run qemu and boot to an ARM architecture.

```
qemu-system-arm –kernel zImage.arm
```

After starting the kernel, you get an error telling you that there is no root file system. We will build one in the next section of this lab.

3.2 Compiling BusyBox for ARM

The purpose for the rest of this lab is to make the lab N° 3 with the additional constraint of producing a file system for ARM architecture using the channel cross-compilation that we have put in place. Remember to position the shell environment variables in which you run the cross-compilation for ARM (modify `ARCH` and `CROSS_COMPILE` variables).

3.3 Creating a filesystem for the target

Take the first issue of lab N° 3 to create a filesystem for the ARM target. You will compare the position occupied by your file system depending on whether you created it with either of two commands:

```
dd if=/dev/zero of=root1.img bs=1024 count=2048
```

and

```
dd if=/dev/zero of=root2.img bs=1M count=2
```

Once your file system filled with the result of BusyBox compiling and once compressed (using gzip), compare the size of both files. What does this tell you?

You can test that your system is functional with `qemu` using a command line type:

```
qemu-system-arm –kernel zImage.arm –initrd root.img.gz –append "root=/dev/ram0"
```

4 End of lab

4.1 Add a program on this target

You will see to add to your program "Hello World" running on your target. How big is this program? How can you make it more compact and make it take the least possible storage space?

4.2 Filesystem comparison between x86 and ARM

During lab N° 3, you've produced a filsystem necessary for your x86 system to work. You should compare the needed resources (particularly the memory size needed and the disk space) to work a system on both x86 and ARM architecture.

4.3 Lab evaluation

At the end of 4 hours, before leaving the room, you should call your teacher to demonstrate that you have made this lab and you got a complete system for ARM and possibly answer some questions.