Embedded-Linux-ZedBoard-Yocto

Step-by-step instructions to creatre embedded linux applications on ZedBoard SoC using Yocto Project

Yocto Project provides tools and resources for building custom Linux-based systems for embedded devices. It aims to streamline the development process and ensure consistency across a wide range of embedded platforms. Yocto provides a layering structure that makes it easier to scale and enables better organization and management of project configurations, dependencies, and customizations. Layers can be added or removed to tailor the build to specific project needs.

In this repository, you will find step-by-step instructions on how to configure and bring up linux and develop applications on ZedBoard SoC using Yocto project. Please refer to this page for development with Buildroot.

Getting Started with Yocto Project

Follow the steps in the original website:

- Starting from: Yocto Project Quick Build
- Until this section: Customizing Your Build for Specific Hardware

After following these steps, you make sure that you:

- have at least 90 Gbytes of free disk space and 8 Gbytes of RAM in your Build Host
- Build your host packages:
  - `sudo apt install gawk wget git diffstat unzip texinfo gcc build-essential chrpath socat cpio python3 python3-pip python3-pexpect xz-utils debianutils iputils-ping python3-git python3-jinja2 python3-subunit zstd liblz4-tool file locales libacl1`
  - `sudo locale-gen en_US.UTF-8`
- Clone the Poky repository:
  - `git clone git://git.yoctoproject.org/poky`
- Checkout to the desired branch of poky repository (for the ZedBoard, note that AMD 2023.2 release is based on Langdale (4.1.4)):
  - `cd poky`
  - `git checkout <<poky_branch>>`
- Build your Image:
  - Initialize the Build Environment: `oe-init-build-env`
  - Examine Your Local Configuration File (local.conf in poky/build/conf directory). The default machine is set to qemux86 target, which is suitable for emulation.
  - Start the Build: `bitbake core-image-sato`
  - Simulate Your Image Using QEMU: `runqemu qemux86-64`

Adding Meta Layers for Xilinx Devices

To customize the target hardware for the ZedBoard, we need to add the meta-xilinx layers. Go to poky directory and add the following layers: meta-openembedded, meta-xilinx, meta-xilinx-tools.

You can follow the Build instructions on the meta-xilinx layer.

Pay attention that **AMD 2023.2** release is based on **Langdale (4.1.4)** branch, so checkout the poky branch to Langdale (4.1.4) and clone the Langdale branch for all the meta-xilinx layers:

```
$git clone -b Langdale https://git.openembedded.org/meta-openembedded.git
$git clone -b Langdale https://github.com/Xilinx/meta-xilinx.git
$git clone -b Langdale https://github.com/Xilinx/meta-xilinx-tools.git
```

Adding Dependency layers:

- `$cd poky`
- `$source oe-init-build-env`
- configure `bblayers.conf` (poky/build/conf/bblayers.conf) by adding dependency layers as shown below using `bitbake-layers` command:

```
$bitbake-layers add-layer ./<path-to-layer>/meta-openembedded/meta-oe
$bitbake-layers add-layer ./<path-to-layer>/meta-openembedded/meta-python
$bitbake-layers add-layer ./<path-to-layer>/meta-openembedded/meta-filesystems
$bitbake-layers add-layer ./<path-to-layer>/meta-openembedded/meta-networking
$bitbake-layers add-layer ./<path-to-layer>/meta-xilinx/meta-microblaze
$bitbake-layers add-layer ./<path-to-layer>/meta-xilinx/meta-xilinx-core
$bitbake-layers add-layer ./<path-to-layer>/meta-xilinx/meta-xilinx-standalone
$bitbake-layers add-layer ./<path-to-layer>/meta-xilinx/meta-xilinx-bsp
$bitbake-layers add-layer ./<path-to-layer>/meta-xilinx/meta-xilinx-vendor
$bitbake-layers add-layer ./<path-to-layer>/meta-xilinx/meta-xilinx-contrib
$bitbake-layers add-layer ./<path-to-layer>/meta-xilinx-tools
```

Machine Configuration for ZedBoard

Set the machine configuration in local.conf to zedboard-zynq7:

- `MACHINE ?= "zedboard-zynq7"`

If you receive a warning in the build process related to xilinx license, you can add this line to the local.conf file: - `LICENSE_FLAGS_ACCEPTED += "xilinx"`

After following the build instructions on meta-xilinx layer, run the build command for core-image-minimal: - `$bitbake core-image-minimal`

This will generate the linux image and root file system required for the booting on the target. ## Prepare the SD card Now you can prepare your SD card by

storing the required files and images.

This page explains the instructions for booting Xilinx devices with SD card.

Note that the generated files are located in this directory:

`${DEPLOY_DIR_IMAGE} = poky/build/tmp/deploy/images/zedboard-zynq7/`

You need to:

1- Create two partitions in the SD card: - FAT32, 800 MB (`/mnt/boot/`) - EXT4, Rest of the SD card (`/mnt/rootfs/`)

You can use the `fdisk` and `mkfs` commands to create the two partitions and format them with the corresponding file systems. Considering sdb as the storage drive: - `$sudo fdisk /dev/sdb` –> follow the displayed information to create the two partitions - `$sudo mkfs -t vfat -F 32 /dev/sdb1` –> to create the fat32 partition - `$sudo mkfs -t ext4 /dev/sdb2` –> to create the ext4 partition - `$sudo mount /dev/sdb1 /mnt/boot/` - `$sudo mount /dev/sdb2 /mnt/rootfs/`

Note that /mnt could be different (/media/...).

2- Copy the required files based on the above-mentioned page.

The following files must be copied to the FAT32 partition:

```
$cp ${DEPLOY_DIR_IMAGE}/boot.bin /mnt/boot/boot.bin
$cp ${DEPLOY_DIR_IMAGE}/boot.scr /mnt/boot/boot.scr
$cp ${DEPLOY_DIR_IMAGE}/Image /mnt/boot/Image
$cp ${DEPLOY_DIR_IMAGE}/system.dtb /mnt/boot/system.dtb
```

The following command is to extract the root file system in the EXT4 partition:

```
$sudo tar -xf ${DEPLOY_DIR_IMAGE}/core-image-minimal-${MACHINE}-${DATETIME}.rootfs.tar.gz -C
$sync
```

After completing the previous steps, insert SD card and boot the ZedBoard from SD. using a serial commnication interface like picocom, you can see the linux boot messages in generated by Yocto in the terminal.

Creating a custom layer for password addition

In this section we create a custom layer in Yocto to add a password to the root user. Use this command to create **meta-custom** layer in the directory that contains the poky repository:

```
$ cd poky/../
$ bitbake-layers create-layer meta-custom
$ ls meta-custom/
COPYING.MIT  README  conf  recipes-example
```

Inside meta-custom layer, create a directory (recipe-core) that contains the recipe for our custom layer:

```
$ mkdir -p recipes-core/images
```

Next, copy the recipe for the core-image-minimal to this directory. We want to start from core-image-minimal.bb recipe and modify its contents in the meta-custom layer:

```
$ cp <Yocto_path>/poky/meta/recipes-core/images/core-image-minimal.bb
<Yocto_path>/meta-custom/recipes-core/images/custom-image.bb
```

Now you can modify the content of the custom-image.bb recipe. Check the custom-image.bb file in this repository.

Notice that you should use a hashed password which you can generate using this command:

```
openssl passwd -5
```

Also notice in this part of the recipe:

```
EXTRA_USERS_PARAMS = "usermod -p <hashed password> root;"
```

that you need to insert a \ before the $ signs in the generated hashed password.

At the end go to the poky directory and run:

```
bitbake custom-image
```

It will create the images and files for linux booting with the password.

Creating a Hello World Application

In Yocto, we need to create a recipe for any embedded application. While in Buildroot we can directly apply a cross-compiler tool on our C code to obtain the executable file for our application, in Yocto we must provide a recipe containing all the information about our cross-compiler, the storage location of our C code, the dependencies required to build the application and other meta-data. These recipes (with a .bb extention) are then processed by the bitbake command to find and compile the code for us and store the executable in the root file system location.

We must create a recipe even for a simple Hello World application. here is the process to create the recipe for the Hello-World application:

1- Create **recipes-apps** directory inside **meta-custom** layer. It contains all our applications including Hello-World:

```
$ cd meta-custom
$ mkdir -p meta-custom/recipes-apps/hello/
```

2- Copy the provided **hello-world_0.1.bb** recipe file in this repository to the recipes-apps/hello/ directory

3- Create `files/src/` directory inside `recipes-apps/hello` directory and copy the provided **hello-world.c** application code inside this src directory

4- Note that after creating your application, you must add it to the final image. To do this, you must add this line inside `meta-custom/recipes-core/images/custom-image.bb`:

```
IMAGE_INSTALL += "hello-world"
```

5- Finally, you will run the bitbake command again 'bitbake custom-image"

Blinking LED

In this section, we want to toggle an LED on our ZedBoard for 10 times every 1 second.

The LED that we use is connected to the PS MIO7. In ZedBoard, there are 54 MIO pins and 64 EMIO pins (extended to PL), so in total there are 118 pins.

Similar to the Hello-World application, we need to create a recipe. In our **meta-custom** layer, in the **recipes-apps** directory create the **led** directory that contains our recipe:

```
cd meta-custom/recipes-apps/
mkdir led
```

Then add the **led_0.1.bb** recipe from this reposiroty to the **led/** directory Note that we need to add the libgpiod for our application, so libgpiod is added as a Dependency and as the LDFLAGS option:

```
# Adding the gpiod.h library
DEPENDS = "libgpiod"

#pass arguments to linker
LDFLAGS := "-lgpiod"
TARGET_CC_ARCH += "${LDFLAGS}"
```

After that you need to add the C code application. Copy the **led.c** file from this reposiroty to the **led/files/src/** directory. Note that the exact gpiochip number might be different and it is better that you check the exact port after booting the linux on ZedBoard for the first time.

```
const char *chipname = "/dev/gpiochip<number>"; // the exact
number can vary
```

Then, we must install the libgpiod library and install (call) the led application recipe in the custom layer recipe. Note that in `meta-custom/recipes-core/images/custom-image.bb`, these lines are added:

```
IMAGE_INSTALL += " libgpiod libgpiod-dev libgpiod-tools"
IMAGE_INSTALL += "led"
```

Bitstream Addition

In this section, we want to add a bitstream file that is generated by Vivado. The application is a switch that is connected to an LED through a wire in the Programmable Logic (PL) in Zynq. Refer to this page for a detailed explanation (it shows the same design in Buildroot).

After completing the design in Vivado and generating the bitstream file, you only need to add the **.xsa** file to a local directory and provide the path that contains this file in **local.conf** file in poky/build/conf/ directory:

```
HDF_BASE = "file://"
HDF_PATH = "<poky_directory>/poky/meta-xilinx-tools/recipes-bsp/hdf/design_1_wrapper.xsa"
```

NAND function with PL and PS

In this section, we want to create a NAND gate in our Zynq SoC that uses both PL and PS. In the PL we create the AND gate and in the PS we create the NOT logic. Refer to this page for the details of the design (in Buildroot).

Update the **.xsa** file from the previous part (5-bitstream-addition) with the new generated .xsa file from the new Vivado design.

Then create the recipe and the C code for the NAND gate application. Follow the same steps in 4-LED-Blink considering the provided recipe and C code for the NAND application.

Note that unlike the Buildroot that we need to modify the device tree manually, in Yocto, all the information about the device tree is already included in the .xsa file, and the recipes in the meta-xilinx layers will generate the device tree from the .xsa file for us.

Ethernet Configuration

To enable Ethernet in our ZedBoard, we must change some default configurations in Yocto meta-xilinx-tool layer.

Specifically, if you check the **zc702.dtsi** file (poky/build/tmp/work/zedboard_zynq7-poky-linux-gnueabi/device-tree/xilinx-v2023 .2+gitAUTOINC+1a5881d004-r0/device-tree-build/device-tree/zc702.dtsi), you will see the following configuration for the ethernet:

```
&gem0 {
        phy-handle = <&ethernet_phy>;
```

```
        pinctrl-names = "default";
        pinctrl-0 = <&pinctrl_gem0_default>;
        phy-reset-gpio = <&gpio0 11 0>;
        phy-reset-active-low;

        ethernet_phy: ethernet-phy@7 {
                reg = <7>;
                device_type = "ethernet-phy";
        };
};
```

In this configuration, the PHY address is set to 7 (reg parameter), while in the ZedBoard it must be set to 0.

Also in **pcw.dtsi** file (poky/build/tmp/work/zedboard_zynq7-poky-linux-gnueabi/device-tree/xilinx-v2023 .2+gitAUTOINC+1a5881d004-r0/device-tree-build/device-tree/pcw.dtsi), the **Write Protection** option must be disabled; otherwise we cannot write to the rootfs files, so the ssh connection from the host to the target hardware will not be possible.

Since this dtsi files are generated by bitbake when reading the recipes, we cannot manually modify the dtsi files (unlike the Buildroot). Therefore, we must modify the recipe which is responsible for generating these dtsi files. This recipe is located in the **meta-xilinx-tools** layer and its name is **device-tree.bbappend** (poky/meta-xilinx-tools/recipes-bsp/device-tree/device-tree.bbappend).

Add these lines to the device-tree.bbappend recipe to modify the above-mentioned configurations:

```
do_configure:append() {
       sed -i '77,78 s/7/0/' ${B}/device-tree/zc702.dtsi
       sed -i '41 i\\tdisable-wp;' ${B}/device-tree/pcw.dtsi

}
```

The first line replaces the PHY address of ethernet in lines 77 and 78 in the zc702.dtsi file from 7 to 0. The second line adds “disable-wp;” line after the line 41 to the pcw.dtsi file.

Finally, to be able to use ssh connection, you must istall the following IMAGE in custom-image.bb:

```
IMAGE_INSTALL += openssh
```