RDFA DLA Report

FPGA Team

June 6, 2023

Contents

1 Board Setup

1.1 Downloading ubuntu image for zcu104

the official xilinx image for zcu104 was downloaded from their website and was flashed onto a 32GB SD card using Balena Etcher.

The board was then booted using the prepared SD card and then we proceeded with the intial board setup instructions.

1.2 Initial board setup

Using putty we got a terminal from the serial port connection to the board. The setup config for the same are as follows:

- baud rate : 115200
- port : ttyUSB1

1.2.1 Network setup

Internet connection was configured and verified using net-tools package.

1.2.2 xlnx-config

The snap package for xlnx-config was installed on th eboard using the following command:

```
#!/bin/bash
sudo snap install xlnx-config --classic --channel=1.x
```

2 Build Petalinux with DLA Hardware Design

2.1 Download zcu104 BSP

The boards support packages corresponding to the board in use were downloaded from the official website.

2.2 Create Petalinux Project

The petalinux project was created using the following command:

```
petalinux-create -t project -s
<path to the xilinx-zcu104-v2021.2-final.bsp>
```

2.3 Reconfigure Project

The project thus created uses the default bord support packages for the petalinux configuration. We need to reconfigure this configuration. This is done by the following commands:

```
petalinux-config --get-hw-description=
<path containing the exported xsa file>
```

In the dialogue boxes shown subsequently, we set the machine name and change the Image Packaging Configuration Root Filesystem to SD card for it to boot from the same. We exit the configuration interface after saving.

2.4 Configure the Linux Kernel

We reconfigure the linux kernel by using the following command:

```
petalinux-config -c kernel
```

In this we unset the initramfs and ramddisk support.

2.5 Build the petalinux project

The petalinux project is built using the following command:

```
petalinux-build
```

2.6 Create and build the SL-DLA kernel module

We create and build the kernel modules required by the patched kernel to install the drivers for the hardware. This is done by the following command:

```
petalinux-create -t modules -n sl-dla-mpsoc --enable
```

2.7 Modify the system user device tree

The following is added to the ¡path to project¿/project-spec/meta-user/recipes-bsp/device-tree/files/system-user.dtsi file.

```
/include/ "system-conf.dtsi"
/ {
    reserved-memory {
    #address-cells = <2>;
    #size-cells = <2>;
    ranges;
    sl_dla_reserved: buffer@0 {
        no-map;
        reg = <0x0 0x60000000 0x0 0x10000000>;
        };
    };
};
&cpu_opp_table {
    compatible = "operating-points-v2";
    opp-shared;
    opp04 {
        opp-hz = /bits/ 64 <1299999988>;
        opp-microvolt = <1000000>;
        clock-latency-ns = <500000>;
    };
    opp05 {
        opp-hz = /bits/ 64 <149999997>;
        opp-microvolt = <1000000>;
        clock-latency-ns = <500000>;
    };
    opp06 {
        opp-hz = /bits/ 64 <99999997>;
        opp-microvolt = <1000000>;
        clock-latency-ns = <500000>;
    };
};
```

The project is then rebuilt to build using the newly created recipe.

3 Create SL-DLA Platform Assets Container

PAC is used to package one or more sets of custom boot assets. For this we can either use the provided PAC file structure or create our own by writing a manifest.yaml and bootgen.bif files and placing them in the shown file heirarchy.

Listing 1: manifest.yaml

```
name: sl-dla-platform
desscription: Boot assets for the SandLogic DLA Design - 2021.1
revision: 1
assets:
    zcu104: zcu104
```

Listing 2: bootgen.bif

```
the_ROM_image:
{
    [bootloader, destination_cpu=a53-0] zynqmp_fsbl.elf
    [pmufw_image] pmufw.elf
    [destination_device=pl] system.bit
    [destination_cpu=a53-0, exception_level=el-3, trustzone] bl31.
        elf
    [destination_cpu=a53-0, load=0x00100000] system.dtb
    [destination_cpu=a53-0, exception_level=el-2]/usr/lib/u-boot/
        xilinx_zynqmp_virt/u-boot.elf
}
```

Listing 3: file structure for PAC

```
SL-DLA-PAC
    |---- hwconfig
        |---- DLA-SMALL-256
            |-- manifest.yaml
            |-- zcu104
                |-- zynqmp_fsbl.elf
                |-- system.dtb
                |-- system.bit
                |-- pmufw.elf
                |-- bl31.elf
                |-- bootgen.bif
```

4 Installation and Activation of SL-DLA PAC

We mount the SD card from the board and place our created PAC files in the SD card filesystem. We create the following file paths and place our PAC there:

```
/boot/firmware/xlnx-config/<pac-name>
/usr/local/share/xlnx-config/<pac-name>
```

4.0.1 Activating the PAC

We boot the board using the SD card and run the following command to query the PL config:

```
xlnx-config -q
```

This returns a table containing available PACs and their respective status. Now we need to activate our PAC. For this we run the following command and then reboot the board:

```
sudo xlnx-config -a sl-dla-platform
sudo reboot now
```

5 Setting up X2GO Server

This is not related to the patched kernel building process but we need this to be able to get a remote display into the board. This sets up a display manager on the board os and we can connect to it using th evnc protocol according to the SandLogic documentation.

6 Patching the Target Linux Kernel

We now can patch the kernel for the SL-DLA kernel drivers. For this we clone the ubuntu kernel repository and install the linux kernel build dependencies. The build dependencies are the packages required for building the kernel. In the git repository we checkout the branch that we want to work on. In this case it was the Ubuntu-xilinx-zynqmp-5.4.0-1017.20.

Next we configure the build environment by exporting the required environment variables as follows:

```
export ARCH=arm64
export $(dpkg-architecture -aarm64)
```

After this we need to introduce the changes required for the kernel drivers. This is the actual patching since the set of changes required for the custom kernel is called a patch. This can be done using the following command if the .patch file is available:

```
git apply <patch file name>.patch
```

If the patch file is not available, as in this case, the changes need to be introduced manually. When this is done, we need to build the kernel. We first clean the build environment and then start the build process. This takes a lot of time. The corresponding commands are as follows:

```
fakeroot debian/rules clean
fakeroot debian/rules binary
```

After building we need to install the kernel. For this we do the following:

```
    cd ..
    sudo dpkg -i *.deb
```

After this we reboot the board and we have the board runnning the patched kernel with the required drivers installed and ready.