Parallella Linux - quickstart guide

Antmicro Ltd

June 13, 2016

Contents

CHAPTER 1

Introduction

This document is a quickstart guide describing how to run Ubuntu 12.04 LTS (long-term support) Linux on the Parallella board.

The Building the FPGA bitstream and FSBL section describes how to build the FPGA bitstream with HDMI support. The Building the Linux BSP section describes how to build the bootloader, kernel and OS image (filesystem). The Usage section describes how to put all the necessary elements onto the Parallella and run the Linux image.

While the first two chapters of the guide feature information how to build your own software/fpga stack from scratch, you can proceed straight to last chapter if you download precompiled binaries, as described in the *appropriate section of that chapter*.

If you do want to change the software/fpga stack you are running, you will also need Xilinx tools:

1.1 Xilinx tools

1. PlanAhead is required to view and reimplement exemplary project. Version used for creating exemplary project is 14.4, thus at least this version (or newer) is required to work with. If you do not plan to build your own hardware project you do not need this tool.
2. Xilinx Software Development Kit (SDK) is required for generating and building the First Stage Boot Loader (FSBL). If you do not want to make any changes to the hardware configuration (which replacement of FSBL you do not need this tool.
3. Xilinx Microprocessor Debugger (XMD) is required to connect with the Zynq CPU via JTAG. If you do not plan do load or debug software this way you do not need this tool.
4. Bootgen is required for preparing a bootable image for the Zynq CPU. If you do not plan to change the default bootloader you do not need this tool.

If any of the abovementioned cases applies to you, you need to download and install Xilinx ISE Design Suite. Select “Embedded edition” as the installation type.

Note: Xilinx software is not open source and may need extra licensing.

1.2 Version information

Author	Content	Date	Version
Michael Gielda	Initial version	2013-08-26	0.1.0
Karol Gugala	Software guide draft	2013-10-01	0.2.0
Michael Gielda	Corrections	2013-10-09	0.2.1
Karol Gugala	Hardware guide draft	2013-10-14	0.3.0
Michael Gielda & Karol Gugala	Fixes after feedback from AO	2013-10-18	0.3.1

CHAPTER 2

Building the FPGA bitstream and FSBL

This section describes the preparation of the Parallella hardware (=FPGA) project. It is based on and example project containing Processing System (PS) configuration and its connection to Programmable Logic(PL). Epiphany link and HDMI IP cores are placed in PL and connected with PS via the AXI bus. See the following subsection for details.

2.1 Prerequisites

To view or modify the example project Xilinx PlanAhead (version 14.4 or newer) is required. See *Xilinx tools* for details.

2.2 Project

The project was implemented in Verilog (top level and Epiphany link) and VHDL (HDMI IP core) using planAhead software (version 14.4). The Project can be divided into two main parts:

1. CPU embedded project (containing the PS configuration and the HDMI IP core)
2. Epiphany link (providing the interface between the CPU and Epiphany)

Both parts are instantiated in the top level file (parallella_7020_top.v).

2.3 Project structure

1. Parallella__7020_top.v - Top file
2. Parallella.v - Epiphany link
3. System_stub.v - CPU embedded project wrapper

2.3.1 Getting the project

The project can be cloned from the Parallella GitHub as follows:

```
git clone https://github.com/parallella/parallella-hw
```

2.4 Processing System Configuration

2.4.1 Cortex A9 configuration

Cortex-A9 peripherals and connections:

Peripheral	connection
UART1	MIO 8..9
QSPI	MIO 1..6
SD1	MIO 10..15
USB0	MIO 28..39
USB1	MIO 40..51
I2C0	EMIO
Enet0	MIO 16..27

Note that the I2C0 peripheral is connected to the EMIO IO controller, thus attached to the PL part of the Zynq chip. The only purpose of doing so is to be able to route this interface through PL to the desired chip pins. To do so signals from this interface (SCL and SDA pins) must be set as external ports in the Embedded Processor project.

In order to connect the Epiphany link to the CPU it is required to derive the AXI bus interface outside the Embedded processor project. To do so the following interfaces have to be enabled and set as external ports:

1. Master AXI interconnect (M_AXI_GP1)
2. High performance AXI slave interconnect (S_AXI_HP1)

For audio data transfer it is required to enable the DMA (DMA0) controller and connect it to the axi_spdif_tx peripheral in Progammable Logic.

Note: Note that CPU configuration isn't written to the bitstream, thus the Processing System is not configured during the bitstream download process. It can only be done with software - generally with the *FSBL*.

Warning: I2C is used during the boot process to set proper voltage in the power management chip. Since I2C is routed through Programmable Logic it must be assure that the bitstream loaded by the FSBL has this feature. If the proper voltage is not set during boot, the Parallella board will consume more power, which may lead to overheating.

2.4.2 Clock configuration

Zynq input clock frequency is 33.333333 MHz, while the CPU clock ratio is set to 6:2:1. Clock configurations are listed in the tables below:

CPU clocks

Device	Clock source	Clock Frequency [MHz]
CPU	ARM PLL	666.666666
DDR	DDR PLL	400.000000
QSPI	ARM PLL	200.000000
Ethernet	IO PLL	125.000000
SD	IO PLL	50.000000
UART	PLL	50.000000

PL clocks

Clock name	Clock Source	Clock Frequency [MHz]	Peripherals driven
FCLK_CLK0	IO PLL	100.000000	AXI interconnect
FCLK_CLK1	IO PLL	200.000000	AXI Video DMA
FCLK_CLK2	IO PLL	200.000000	HDMI reference clock
FCLK_CLK3	IO PLL	40.000000	Epiphany Link

2.5 Programmable Logic Configuration

2.5.1 HDMI support

HDMI support is provided by an IP core from Analog Devices (Analog Devices GitHub). IP cores needed for HDMI support are located in the `cf_lib` folder. Peripherals used in the Parallella Exeample project, their purpose and connections are listed below.

1. axi_clkgen (v1.00a) - programmable reference clock for the HDMI transmitter. It is connected to the main AXI bus and provides the reference clock for the `axi_hdmi_tx_16b` peripheral. It requires a 200MHz input clock (FCLK_CLK2 in the example project)
2. axi_vdma - dma controller for video data. It is connected to the main AXI bus as slave and to the secondary one as master.
3. axi_hdmi_tx_16b - video signals generator for the ADV7513 chip. It generates video synchronization signals (HSYNC and VSYNC), pixel clock and delivers video data. It is connected to the main AXI bus and requires a reference clock with proper frequency for the chosen resolution. Video data is transferred to this peripheral with DMA.
4. axi_spdif_tx - digital audio signal generator. For correct operation it is required to deliver a 12.288135 MHz clock signal to this component. It is connected to the main AXI bus and audio data is delivered using DMA.

To provide the proper clock signal to the spdif peripheral a Xilinx clock generator IP core can be used.

2.6 FSBL

The First Stage Boot Loader code is generated from the Xilinx Software Development Kit.

See the Xilinx wiki and the documents it refers to for more details.

CHAPTER 3

Building the Linux BSP

This section describes how to download, configure and build the Linux BSP for the Parallella board.

3.1 Prerequisites

ARM toolchain

Xilinx tools come with `arm-xilinx-eabi-` and `arm-xilinx-linux-` toolchains, either of them can be used.

If you do not need or want to use Xilinx tools, any other ARM toolchain (e.g. Sourcery CodeBench lite edition) can also be used for compiling the BSP.

U-Boot tools

U-Boot tools are required to generate a loadable image of the Linux kernel.

On Debian/Ubuntu distributions this software can be obtained with:

```
apt-get install u-boot-tools
```

3.2 U-Boot

3.2.1 Getting the source

Official U-Boot source code for the Parallella board can be found at: https://github.com/parallella/parallella-uboot on the `parallella-gen0` branch. In order to get it use:

```
git clone https://github.com/parallella/parallella-uboot
git checkout parallella-gen0
```

3.2.2 Building

```
export ARCH=arm
export CROSS_COMPILE=<your_toolchain_prefix> #e.g. arm-xilinx-eabi-
export PATH=</path/to/your/toolchain>:$PATH
make parallella_config
make -j<X> #X is typically no. of threads on your system +1
```

3.3 Linux

3.3.1 Getting the kernel source

Official Linux kernel source code for the Parallella board can be found at: https://github.com/antmicro/linux-parallella on the `parallella-linux3.9` branch. In order to get it use:

```
git clone https://github.com/antmicro/linux-parallella
git checkout parallella-linux3.9
```

3.3.2 Building the kernel

```
export ARCH=arm
export CROSS_COMPILE=<your_toolchain_prefix> #e.g. arm-xilinx-eabi-
export PATH=</path/to/your/toolchain>:$PATH
make parallella_defconfig
make -j<X> uImage #X is typically no. of threads on your system +1
```

CHAPTER 4

Usage

This chapter covers how to prepare the necessary boot images and media based on binaries either built as described in the previous chapters (necessary only if you intend to change the hardware configuration) or downloaded as *described below*.

Precompiled boot images are also available from the same source.

4.1 Precompiled binaries

Repository file list:

- fsbl.elf - compiled First Stage Boot Loader
- u-boot.elf - compiled U-Boot
- uImage - Linux kernel image (u-boot loadable format)
- devicetree.dtb - compiled device tree for the Parallella board
- parallella.bit - Zynq bitstream
- parallella.bit.bin - Zynq bitstream (u-boot loadable format)

4.2 Preparing a Zynq boot image

This section describes preparation of a Zynq boot image that can be written into the onboard flash memory of the Parallella board. A Zynq boot image can be created with the bootgen tool (see *Xilinx tools*). Bootgen requires a `.bif` input file, which is just a text file describing boot image layout. An example `.bif` file is listed below:

```
the_ROM_image:
        {
        [bootloader]/path/to/fsbl.elf
        /path/to/u-boot.elf
        }
```

With this configuration, a simple boot image containing only the first and second stage bootloader (FSBL and U-Boot) can be built. This is done as follows:

```
bootgen -image /path/to/image.bif -o i parallella.bin
```

The resulting file (`parallella.bin`) is a bootable image that can be programmed into the Parallella onboard flash memory. More information about using bootgen and creating Zynq boot images can be found in Zynq-7000 All Programmable SoC Software Developers Guide.

4.3 Bootloader deployment

There are two ways of running new bootloader software on the Parallella board:

1. Program the flash with a new Zynq boot image file see *Programming the flash*.
2. Download and run the bootloader via JTAG connection see *Using JTAG*.

4.3.1 Using JTAG

Prerequisites

- *Xilinx tools*
- Xilinx Platform Cable

Note: JTAG deployment requires the Xilinx Platform Cable to be connected to the Parallella board.

This method allows you to run new software on the Parallella board without reflashing it, or recover it after flashing with an improper boot image.

1. The first step of the JTAG deployment procedure is running Xilinx Microprocessor Debugger tool. It is able to connect to Zynq CPU and provides a gdb server. Moreover, using this tool you are able to program the Zynq chip with a new configuration file, but it isn't always necessary (if not, the `fpga` command in the following procedure can be skipped).

The important step when setting the JTAG connection is the proper configuration of the Zynq chip (especially when it was flashed with an improper boot image). This is done by running tcl procedures from `ps7_init.tcl` script (TODO: ref to repository). The `stub.tcl` script sets the Zynq CPU into debug mode.

```
    connect arm hw
    fpga -f </path/to/your/>bitstream
    source </path/to/your/>ps7_init.tcl
    ps7_init
    init_user
    source </path/to/your/>stub.tcl
    target 64
```

2. After configuring the Zynq chip, a software application (e.g. U-Boot) can be loaded onto it. This can be done either with xmd:

```
dow </path/to/your/>u-boot.elf
con
```

or via gdb:

```
target remote localhost:1234
file </path/to/your/>u-boot.elf
load
c
```

4.3.2 Programming the flash

Warning: Reprogramming the flash with an incompatible Zynq boot image may result in breaking the Parallella board. Moreover, during flashing, a stable power supply must be assured.

Note: If the board was programmed with an improper boot image or there was some other problem during flashing, it still can be bring up using JTAG (see *Using JTAG*).

The U-Boot delivered with Parallella can be used to re-flash the board. The binary available from the repository (*Precompiled binaries*) or built (*U-Boot*) from github source also has this functionality. If you changed the default bootloader and it does not provide this feature, you can still run the default official Parallella U-Boot via JTAG (*Using JTAG*) or load it with your current bootloader.

To re-flash the Parallella board using the official U-Boot follow these steps:

1. Remove the SD card from the slot.
2. Power up the board - U-Boot should start, and the lack of and SD card will prevent it from booting Linux.
3. Put the SD card into the slot.
4. Run the following commands in the U-Boot prompt.
   - Initialize the mmc subsystem

```
mmcinfo
```

   - Load the Zynq boot image from the first partition (FAT-formatted) into RAM (make sure the image file is present on the SD card)

```
fatload mmc 0 0x4000000 <boot_image_name>
```

Warning: Be careful about the fatload address; if an address outside RAM is given (e.g. 0x40000000 instead of 0x4000000), the command will hang without a warning.

- Initialize the SPI flash subsystem

```
sf probe 0 0 0
```

- Erase the entire flash memory

```
sf erase 0 0x1000000
```

> **Note:** This process can take a long time to complete, do not interrupt it.

- Program the flash memory

```
sf write 0x4000000 0 0x$filesize
```

> **Note:** The $filesize variable holds the size of the previously loaded file, so if you had loaded some other file in U-Boot in the meantime, this will not be the size you want and you have to provide the right value by hand.

5. Power cycle the board.

Note: The boot image length is displayed when the file is loaded into RAM from the SD card.

4.4 Booting Linux

This section discusses a few (out of the many possible) ways of booting Linux on the Parallella board. If you are interested in general boot procedure of Zynq based device refer to Zynq-7000 All Programmable SoC Software Developers Guide.

4.4.1 Booting from SD card / USB drive

Copy `uImage`, `parallella.bit.bin` and `devicetree.dtb` onto the first partition of a >= 2GB SD card (FAT-formatted), insert card into the board and power it up. Unpack the Linux root file system onto the USB drive or the second partition of SD card (ext formatted).

Note: Remember to set the proper boot device in the Linux kernel bootargs (`/dev/sdaX` for USB drive boot or `/dev/mmcblk0pX` for SD card)

4.4.2 Default Boot sequence

Default Boot sequence on the Parallella board is as follows:

1. After power-up, the internal Zynq BootROM finds the boot image in onboard flash, copies the FSBL from it into the On-Chip RAM and runs it.
2. FSBL finds the boot image on the onboard flash, copies U-Boot from it into RAM and runs it.
3. U-Boot searches the first (FAT formated) partition on the SD card for the Linux kernel image (uImage), devicetree (devicetree.dtb) and Zynq configuration (parallella.bit.bin). If found they are copied into RAM.
4. U-Boot configures the Zynq chip, and boots Linux kernel passing the devicetree to it.
5. The Linux kernel boots into the rootfs according to bootargs passed in the devicetree.