**Report on how to build a full system on the ZCU104 board with RPU running**

Author: Greg Smart, Kutu Systems

Date : 13/05/2019

**Introduction**

The Xilinx MPSoc is a very complicated device that is difficult to come to terms with. The reason why is that it has so many components required to build a complete system. A normal system will include a PSU and I/O configuration, ATF firmware, PMU firmware, an FPGA block diagram, custom IP’s, a Linux build on the main A53\_64 processors and also bare-metal software on the RPU. The key to simplifying this is to use a modular development process

There are many application notes by Xilinx that cover various aspects of a system design. Petalinux but there isn’t anything that describes how to put together a complete system in simple steps. This procedure builds a Linux system with an X-windows configuration on a ZCU104 which contains some custom IP blocks with drivers. It also includes a simple RPU application.

The purpose is not to describe the individual processes, but the process for building a system that is easy to manage. The download for this this project is only around 300Kbytes. This compares to over 1GByte for some of the Xilinx BSP’s. The underlying principle is that anything you put in your git repository, you have to manage. By working with the tools effectively, you only need to manage the data that is specific to your system. This improves efficiently enormously.

Building an MPSoc system involves layers of scripts under the hood. This procedure is deliberately limited (at this point) so the highest level script doesn’t exist, but it is built at the next layer down. The reason for this is so the actual processes are visible to get a better understanding of how the end output is generated. It also makes it clear how to modify each component individually to allow a modular development process.

**Investigation Procedure**

The initial process was to create a Vivado project for the ZCU104, and then add in the various system components. The end goal was to simplify the development process for my customers.

The project doesn’t contain any confidential information, and is available at [https://github.com/KutuSystems/zcu104\_rpu.git](https://github.com/KutuSystems/zcu102_gpio.git)

**1. Build Vivado Project**

The first thing that needs to be done is to build the Vivado project. The output is normally several hundred Mbytes of data.

Before the Petalinux system can be built, the Vivado project must be built and exported to hardware. The steps to build the FPGA design are:

1. Open Vivado 2018.2

2. cd to <github root>/zcu104\_rpu/zcu104\_rpu/hardware/ZCU104\_RPU

3. type source ./create\_all\_projects.tcl

At this point Vivado will generate the custom IP blocks from source and then generate the top level block diagram. When complete a block diagram of the system will be displayed. If modifications are to be made, they are made at this stage. If the block diagram is changed, then the block diagram is re-exported using the menu item “File→Export→Export Block Design”. This should always be done using the GUI.

The next stage is to build the project

4. click on “Generate Bitstream” (bottom left of IDE)

When the bitfile generation is complete you get asked if you want to open implemented design.

5. Select “open implemented design” and click “ok”

6. when the design has been opened go to “File” menu (top left). Go down to “Export”. In the Export menu select “Export hardware ...”. A small dialog appears. Ensure “Include bitstream” is selected and click “ok”.

After this stage is complete you can exit Vivado. Vivado is no longer required, all subsequent steps to build a Linux system are done using Petalinux.

**2. Build Petalinux**

The Petalinux init script has to be run before the system can be built.

source ~/petalinux-v2018.2/settings.sh. This assumes Petalinux was installed in your home directory.

The Petalinux project is built using the following commands.

1. cd zcu104\_rpu/zcu104\_rpu/

2. petalinux-config --get-hw-description=hardware/ZCU104\_RPU/ZCU104\_RPU.sdk

You will then enter a config menu. Just exit out, configuration is complete.

3. petalinux-build

The Petalinux build process is a Yocto system that adds a few extra Xilinx specific steps. It automatically reads in the hardware description file to generate the device tree. The device tree nodes for the custom IP blocks are exported from Vivado in the hdf file. The petalinux-build process builds the ATF and PMU firmware, the first stage bootloader, the Linux kernel and the root file system.

If there is no RPU firmware, then the boot image, boot.bin, can be built with the following command:

3. petalinux-package --force --boot --fsbl images/linux/zynqmp\_fsbl.elf --fpga images/linux/system.bit --pmufw images/linux/pmufw.elf –u-boot

when complete the files boot.bin and image.ub are in the images/linux

These 2 files are then copied to the SD card. The ZCU104 can can then be started, it doesn’t require the RPU firmware to be included in the build.

**3. Build RPU**

If there is RPU firmware, then the firmware must be built using the Xilinx SDK. We do this at the command line using XSCT, but the output projects can still be debugged and tested using the SDK tool.

To use the SDK, you must first source the tools. Petalinux doesn’t do this.

source /opt/Xilinx/SDK/2018.2/settings64.sh

The RPU firmware is built using the following steps:

1. cd hardware/ZCU104\_RPU/ZCU104\_RPU.sdk

2. xsct ../scripts/build\_xsct.tcl

Petalinux doesn’t support inclusion of the RPU firmware, so the BOOT.BIN file must be built using the XSDK bootgen utility. The build\_xsct.tcl script performs using the SDK bootgen utility.

The build\_xsct.tcl script uses the command line version the SDK to create and build the project. The script creates an empty project and then imports the source code. The output.bif file describes how bootgen builds BOOT.BIN

setws ./

createhw -name R5\_0\_bsp -hwspec top\_ZCU104\_RPU.hdf

createapp -name R5\_0\_app -app {Empty Application} -proc psu\_cortexr5\_0 -hwproject R5\_0\_bsp -os standalone

importsources -name R5\_0\_app -path ../../R5\_0\_app/src

projects -build

exec bootgen -arch zynqmp -image ../../ZCU104\_RPU/scripts/output.bif -w -o ../../../images/linux/BOOT.BIN

The BOOT.BIN and image.ub file are then copied to the SD card on the ZCU104 board.

**4. Results**

The intial results were with the default Vivado configuration (USB3.0 transceiver enabled)

The results of the investigation showed some issues which demonstrated the problem. The internal linux kernel and the external kernal have the same md5 checksum, but the output files are a different size. This indicates that the internal kernel is being configured differently than the external kernel but the differences are hidden from the user.

**Conclusion**

The conclusion is that the Xilinx kernel modifications do not address the issue, and that currently the only reliable workaround is to provide a 26MHz clock on transceiver 2 and configure the USB port as USB3.0.