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 MPSoc 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's. 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 provides much of the process 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 bare-metal application.

The purpose is not to describe the individual processes or how to use the tools. The aim is to develop a 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 efficiency enormously.

Building an MPSoc system involves layers of scripts under the hood. In this note, the 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 provide a modular development process.

System Build 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

The process assumes that Vivado is installed in /opt/Xilinx, and petalinux is installed in ~/petalinux-v2018.2/

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. In this case we use the GUI to run the build script so the process is graphically visible. 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 RPU firmware in the system then go to section 3.

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 of 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

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

4. ZCU104 Board Testing

There are 2 available serial ports on the ZCU104. The RPU application has been copied from Xilinx’ UG1209 application note. The first serial port is used for the Linux console, and the second serial port is used for the RPU “hello world” application. Both ports are configured as 115200,8,1. The Linux system also has a basic X-windows configuration using XCFE (which is why you get all the GTK messages at bootup). The RPU application prints a message when a key is pressed.

To test the build, connect 2 terminal programs to the 2 uarts and power up the ZCU104. On one terminal, the linux console shows the boot process, and on the other terminal the “Hello World from R5-0” is shown.

The system components (Vivado project, Petalinux, RPU build), can all be worked on individually, and then the project updated.

5. Conclusion

The purpose of the ZCU104 build is to create a base system that can be used as a reference for building a new project. A higher level script file can be created to build the entire system with a single command script. The Vivado project is created within the GUI in this case to show the process, but it can also be run from the command line. Using a build environment that is modular and well structured is important for a commercial project. It ensures that the build is a reliable and repeatable process and that components can be developed independently.
Most of the work is done using Petalinux which is intended to simplify this process, but with the addition of the RPU’s in the MPSOC, and also soft-core processors, the build system needs to use Petalinux as a separate tool as part of a larger system. The Vivado/Petalinux/SDK suite of tools are complex to use. Creating an environment that simplifies using the tools can significantly reduce costs.
The main reason for writing this application note was to create a template for a system where only the user code is maintained in the source repository. This frees the designer from maintaining code and data that is not part of the system.