SYSTEM ON CHIP DEVELOPMENT PLATFORM FOR SOFTWARE DEFINED RADIO

Dan DUŞTINŢĂ, Alexandra STANCIU

Transilvania University of Braşov, România (dan.dustinta@student.unitbv.ro, alexandra.stanciu@unitbv.ro)

DOI: 10.19062/1842-9238.2018.16.1.9

Abstract: *In a rapidly growing domain, Software Defined Radio platforms prove to be a sustainable and efficient way of testing and developing new features. By using the Xilinx Zynq-7000 programmable SoC (System on Chip), based on an ARM processor coupled with a FPGA (Field-programmable Gate Array) integrated circuit allows us to combine unique features into flexible and versatile implementations. The platform benefits from the large amount of existing support that a Linux based open-sourced software can bring to an embedded system with the highly reconfigurable logic blocks that can be used with the FPGA. This paper presents the method for setting up an Software Defined Radio work environment around Zynq processing system and Xilinx programmable logic*

Keywords: *SDR, development, platform, open-sourced, FPGA*

1. INTRODUCTION

As technology has moved forward, software development has begun to be highly dependent on the reusability and portability of existing source code. Many successful stratagems are based on this approach, Linux which was developed initially for personal computers has been ported to numerous platforms, including embedded systems. Although traditionally Linux is not a RTOS (Real Time Operating System) it's use in embedded platforms is fueled by great device support, network connectivity, file systems and of course it's UI (User Interface).

On a small embedded system with limited resources Linux is usually not viable, but on a more powerful system like the Zynq-7000 SoC the advantages become increasingly obvious. One could take advantage of the dual core ARM processor to virtualize certain non-critical and not real time dependent functionality on core which uses an embedded Linux operating system. The other core can be used as a "hard" real time operating system for functional relevant features or real-time applications. OpenAMP (Open Asymmetric Multiprocessing) allows multiple processing cores to interact between each other on a single SoC. Usually these software environments can run independently of each other and provide heterogeneous functionalities to the system. Moreover, system like Zynq-7000 SoC gives you the possibility to combine the software programmability of an ARM-based processor with the hardware reconfigurability of an FPGA, enabling hardware acceleration while integrating CPU, DSP and mixed signal functionality on a single device. The features listed above make the Zynq-7000 a good platform for Software Defined Radio (SDR) implementation.

The aim of this paper is to describe the method for setting-up an SDR work environment using the Linux ARM co-processor and the method to implement a minimal System on Chip – SoC using Xilinx Vivado at FPGA(Zync) level, using as radio interface the Analog Devices AD-FMCOMMS3-EBZ board.

This environment will be further used for creation and testing of SoCs in the telecom domain, taking benefit of the existing IP Cores in the Xilinx Vivado library like for example: Coder/decoder 3GPP LTE, Convolution Encoder, LTE FFT, LTE PUCCH Receiver, LTE RACH Detector, 3GPP LTE MIMO Encoder/Decoder.

2. SOFTWARE DEFINED RADIO IMPLEMENTED ON ZYNQ

Software Define Radio (SDR) is a popular prototyping technology for wireless communication systems, software programmability providing flexibility and enhanced utility. In an SDR system, all the signal computations are made on the host computer and then the waveforms are send to the RF front end [1]. Nowadays the signal processing is so computationally expensive, making it very difficult to find a properly host computer. In this paper we intend to show how a platform like Zynq-7000 SoC could be the host computer for an SDR due to its rich architecture. This product integrates a dual-core ARM cortex A9 based processing system (PS) and Xilinx programmable logic in a single device.

This paper intends to be a starting point for an SDR implementation using the Digilent Zedboard and the AD-FMCOMMS3-EBZ radio platform. By using the AD-FMCOMMS3-EBZ evaluation board, the SDR development platform can be used to implement both access network concepts and core network concepts. The ideea of using Zynq-7000 is to run computationally intensive algorithms on the FPGA and compute the rest of the chain using the ARM processors. Beside the existing Vivado IP cores (e.g. Coder/decoder 3GPP LTE, Convolution Encoder, LTE FFT, LTE PUCCH Receiver, LTE RACH Detector, 3GPP LTE MIMO Encoder/Decoder) new IP cores might be implemented in hardware description languages HDL for telecom mobile data networks [2], enhancing the idea of processing the signals into hardware in order to gain high speeds and low latencies.

One option for wireless communications implementations would be the use of the opensource GNU Radio toolkit as part of the Linux system running in ARM, but this would not take complete benefit of the Zynq-7000 programmable logic.

Thus, the optimum implementation would be to balance the computational tasks between the ARM processors and the FPGA based programmable IP cores, based on a common SoC platform. The three steps which will lead to an SDR application on the Zynq-7000 plus the FMCOMMS3 RF extention are summarized below:

1. create a System on Chip (SoC) hardware platform
2. setup the Linux on ARM processors
3. create the software application, using a mix of software implemented elements (e.g. in GNU Radio) and hardware blocks/IP cores.

The process for setting up the mixed platform is presented in the following sections, as follows. An example of hardware platform is presented in Section 3. A typical radio application will require an operating system. PetaLinux software development kit (SDK) has the complete environment for to building, developing, testing and deploying Linux on custom embedded systems [3]. Section 4 describes in detail how PetaLinux may run on the Zynq-7000 hardware platform. The software application could use the GNU Radio software toolkit which provides signal processing blocks to implement software radios [4].

Depending on the SDR power and efficiency requirements, it may be chosen which part of signal processing is implemented in software and which algorithms are developed in hardware.

3. SYSTEM ON CHIP HARDWARE PLATFORM FOR SDR

The hardware platform should contain at least the Zynq processor system and an interface between the processor system and the FCOMM3-EBZ, as we have implemented in the SoC that may be seen in Figure 1. The interface between the processor system and the analog device is implemented as an IP core in hardware description languages. The interface contains an AXI interface in order to be configured by the ARM cores and a digital interface for sending data to the analog device peripheral. The digital interface consists of 12 bits of DDR data and supports full duplex operation in all configurations up to 2x2. The transmit and receive data paths share a single clock. The data is sent or received based on the configuration (programmable) from separate transmit and to separate receive chains. The internal structure of the IP core is presented in Figure 2 [5]. The IP core may be found as a reference design on [5] and the AXI interface with the processor system could be generated using Vivado.

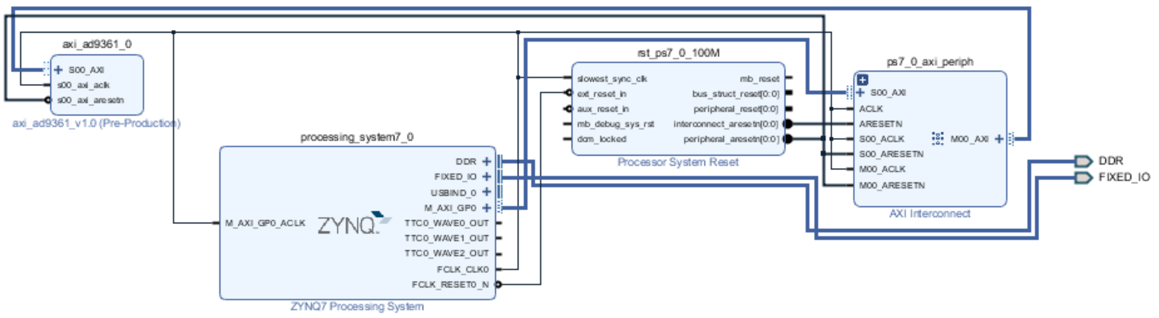


FIG. 1. ZYNQ Processing system implemented in Vivado

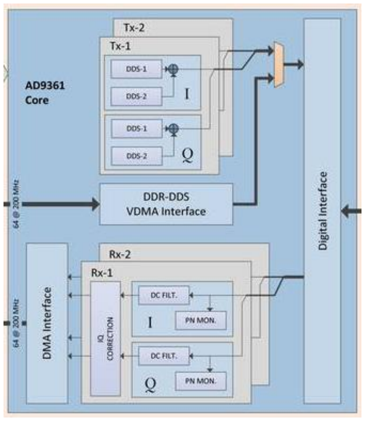


FIG. 2. AD9361 IP Core [5]

The hardware system may contain other peripherals included in Zynq processor system or implemented in the programmable logic resources in order to interact with other elements of the platform, like the HDMI controller, USB Keyboard, USB Mouse, Ethernet controller.

4. LINUX SETUP ON ZYNQ-7000 SOC

The Xilinx SDK(Software Development Kit) can be used to create the boot image for the system. The U-Boot(Universal Bootloader) is usually used in combination with the Linux platform. U-Boot provides the ability to start the application image, e.g. Linux [11]. Right away the advantages of using open source tools become obvious, as the development effort is greatly reduced.

In order to create the Linux image firstly the FSBL (First Stage Bootloader) must be created. This software component is responsible for the early system initialization and handing over control to the U-Boot. In order to build this component, the hardware designs must be available [6]. U-boot is an open-source primary bootloader that ca be used in embedded systems to start the system's kernel. In order to build U-boot for the Zynq-7000 platform the source files, the DTC (Device Tree Compiler) and the Xilinx SDK must be available [7]. Essentially the bootloader must handle the setup and initialization of the RAM, detect the machine type, setup the kernel, load the "initramfs" (root filesystems) and finally call the kernel image[8].The rootfs (root file system) must contain everything needed by our Linux system. The Zynq uses an "initramfs" which is extracted when the kernel boots up. Depending on the used Linux version (BusyBox, Linaro, etc.) the filesystem can be either persistent or it can be cleared on power-off [9]. The Device tree is used to describe the underlying hardware system. It is a data structure that can be read by the operating system. The kernel image is the one that will be called by the U-Boot at the end of the startup phase and is the heart of the embedded Linux system. In order to build the kernel, a cross compiler toolchain must be installed. In this case the boot image contains the above described components, U-boot is used to load the Linux image [13]. After formatting the SD card and copying the above described components, the SD card can be inserted into the Zedboard. All the steps are summarized in Fig. 3-8.

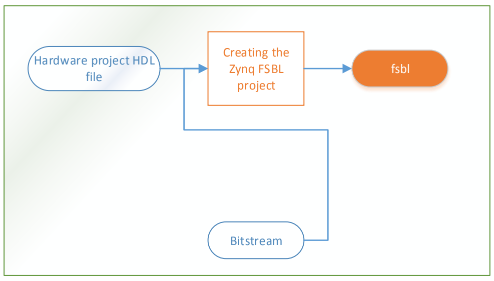


FIG. 3. FSBL development workflow

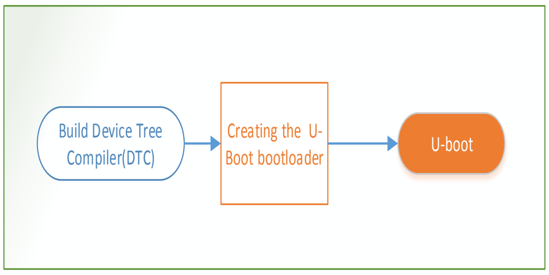

Build Device Tree Compiler(DTC)
Creating the U-Boot bootloader
U-boot

FIG. 4. U-boot development workflow

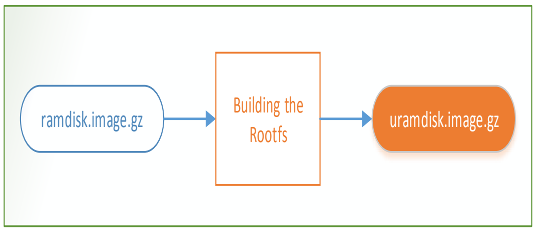


FIG. 5. Root File System development workflow

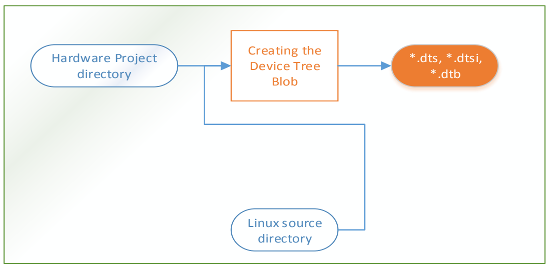


FIG. 6. Device Tree Blob development workflow

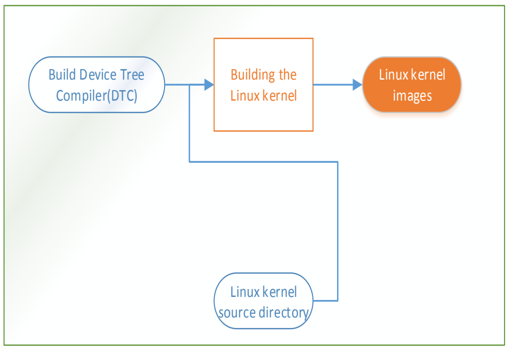

Build Device Tree Compiler(DTC)
Building the Linux kernel
Linux kernel images
Linux kernel source directory

FIG. 7. Linux Kernel development workflow

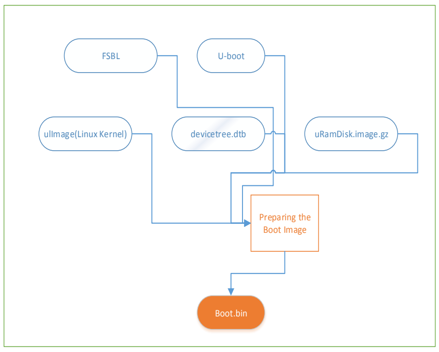


FIG. 8. Boot Image development workflow

In order to test our system, the mode jumpers must be set in the following configuration [8]:
– MIO[2]/Boot_Mode[3] sets the JTAG mode
– MIO[5:3]/Boot_Mode[2:0] select the boot mode
– MIO[6]/Boot_Mode[4] enables the internal PLL
– MIO[8:7]/Vmode[1:0] are used to configure the I/O bank voltages, however these are fixed on ZedBoard and not configurable

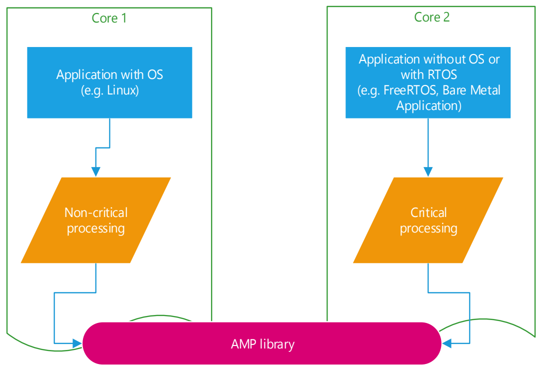
If the Linaro distribution was used, then HDMI output support exists and we can easily see the boot operation. Additionally if the Zedboard is used together with the AD-FMCOMMS3-EBZ development board then the ADI IIO Oscilloscope can be used to verify if the daughter board was recognized and if it is working correctly.

5. SOFTWARE APPLICATION

The AMP concept illustrated in Fig. 9. can be implemented on the Zedboard such as the Cortex-A9 processors run independent software environments in different contexts. Using the Xilinx SDK the Core running Linux can be configured as the system master and will be responsible for general system initialization, controlling the other core's startup, communication between the two cores and with the user [10]. Great care must be taken to control the private and shared resources of the processor, such as caches, timers, DDR memory, etc. One possible scenario is that the Linux master is in charge of controlling the shared resources (Fig. 10)

The software can be divided in three major software components, the FSBL(First Stage Bootloader), the Linux operating system and the Bare-metal application [12]. The FSBL is the first software component that is run after a system power-on or reset. This software component is responsible for loading the Application into the DDR memory and executing it. The FSBL must be modified so that the AMP configuration is supported.

To create a AMP Linux image the boot-image, device tree, root file system ramdisk must be modified.


Core 1
Application with OS (e.g. Linux)
Non-critical processing
Core 2
Application without OS or with RTOS (e.g. FreeRTOS, Bare Metal Application)
Critical processing
AMP library
FIG. 9. AMP concept

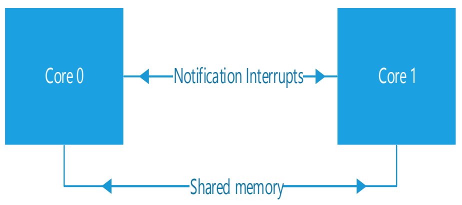


FIG. 10. Inter Core Communication

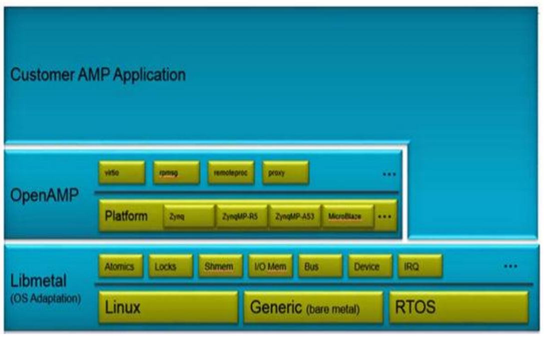


FIG. 12. Libmetal architecture

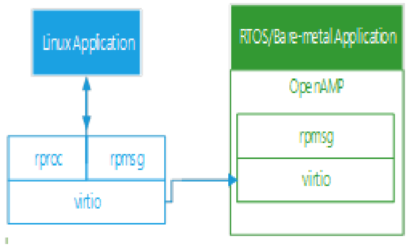


FIG. 13. Implementation in kernel

The basic concepts that are the foundation of AMP principles are the shared memory between the two different software environments and the notification interrupts that allow communication between the two contexts. OpenAMP uses the libmetal library to provide an API(Application Programming Interface) in order for the software components to be able to use the processors resources, and to handle the device's interrupts[9]. OpenAMP provides three important components. virtIO is used for managing the shared memory, this is a virtualization standard used for network and disk device drives. Remoteproc which provides life cycle management capability of the remote processor, it is used to allocate resources and to create virtIO devices. RPMsg is used for inter process communication, it allows different software environments running independently to communicate in the AMP system. OpenAMP uses the libmetal library to create an abstraction layer to the low level devices and to allow access to the shared memory. It provides a standardized way to access resources and to handle interrupts.

6. CONCLUSIONS

We have developed a working environment for Software Defined Radio using ZedBoard development platform and the FMCOMMS3 RF interface.

The created System on Chip using the Xilinx Vivado tool is integrating ARM microprocessors and the AD9631 IP Core as interface to the radio element. We have described the created SoC and the method for setting up the Linux environment at the ARM ptocessor level. The SoC is fully reconfigurable at both hardware level and software level.

As future work we intend to implement an SDR system based on FPGA programmable logic elements such as DSPs and software IP cores (Microblaze, Picoblaze).

REFERENCES

[1] X. Cai, M. Zhou, X. Huang, "Model-Based Design for Software Defined Radio on an FPGA", IEEE Access, Volume 5, April 2017.

[2]T. Balan, D. Robu, F. Sandu, "LISP Optimisation of Mobile Data Streaming in Connected Societies", Mobile Information Systems, Vol. 2016

[3] S. Gvozdenovic, Software Level Design of Software-Defined Radio Platform, Worcester Polytechnic Institute

[4] GNU Radio, Online: https://wiki.gnuradio.org/index.php/Main_Page

[5]AD9361 HDL References Design, Online: https://wiki.analog.com/resources/eval/user-guides/ad-fmcomms2-ebz/reference_hdl

[6] Booting ARM Linux, Online: https://www.kernel.org/doc/Documentation/arm/Booting

[7]Kernel Documentation File System, Online: https://www.kernel.org/doc/Documentation/filesystems/

[8] Xilinx Prepare Boot Image, Online: http://www.wiki.xilinx.com/Prepare%20Boot%20Image

[9]Xilinx, Libmetal and OpenMP for Zynq Devices User Guide, Online: https://www.xilinx.com/ support /documentation/sw_manuals/xilinx2017_3/ug1186-zynq-openamp-gsg.pdf

[10]Xilinx Application Note, Simple AMP Running Linux and Bare-Metal System on Both Zynq SoC Processors, Online: https://www.xilinx.com/support/documentation/application_notes/xapp1078-amp-linux-bare-metal.pdf

[11] Xilinx, Zynq-7000 All Programmable SoC Data Sheet, Online: https://www.xilinx.com/support/ documentation/data_sheets/ds190-Zynq-7000-Overview.pdf

[12] Xilinx Build U-Boot, Online: http://www.wiki.xilinx.com/Build%20U-Boot

[13] Xilinx Build FSBL, Online: http://www.wiki.xilinx.com/Build%20FSBL