GSoC'13: GnuRadio FPGA Co-processing with the Xilinx Zynq System-on-Chip

Jonathon Pendlum (jpendlum@gmail.com)

May 3, 2013

1 Introduction

Many signal processing blocks in GnuRadio exhibit parallelism and can be efficiently mapped to the architecture of a FPGA. However, GnuRadio has been slow to adopt FPGA hardware acceleration, likely due to a lack of suitable hardware and difficulty of HDL design. Recently, FPGA vendor Xilinx released the Zynq, a System-on-Chip (SoC) that tightly couples programmable logic with a dual core Cortex A9 ARM processor [1]. It features low latency, high throughput, and cache-coherent communication between the programmable logic and the ARM processor cores. Due to this feature, the Zynq SoC has gained interest in the GnuRadio community as a viable platform for FPGA hardware co-processing. This project proposes adding a framework to GnuRadio to support FPGA co-processing with Xilinx's Zynq SoC.

2 Benefits to GnuRadio

- **Improved performance**: GnuRadio utilizing FPGA co-processing on the Zynq SoC will realize substantial performance improvement versus other ARM processor implementations.
- **Expands GnuRadio into new research fields**: This framework would allow GnuRadio users a foundation to experiment with ideas not currently possible such as low power, small form factor SDRs, FPGA co-processing, FPGA partial reconfiguration for SDRs, and low latency SDR designs. For instance, an enterprising user could interface a radio front end with the Zynq SoC and use the framework to provide GnuRadio ADC sample data with very low latency. This could spur further experimentation and protocol design in GnuRadio based Cognitive Radio, whose algorithms benefit from reduced data delay.
- **Access to low cost hardware**: Xilinx has released the Zedboard as a low cost development platform for the Zynq [2].

- **Support for upcoming devices**: This work is a relevant step towards interfacing GnuRadio with other Zynq based devices such as the Parallela many core computing platform [3] and Ettus Research's next generation embedded series USRP.

3 Design

The project will implement a FPGA co-processor block to GnuRadio Companion and the framework necessary to support the new block. The framework consists of a kernel device driver and VHDL interface block for utilizing the Zynq's Accelerator Coherency Port. The overall framework will be demonstrated by replacing the low pass filter block in the GnuRadio Companion wideband FM demodulator demo with a FPGA accelerated decimating CIC filter. Figure 1 shows the system level approach from GnuRadio Companion to the VHDL processing blocks and how the project deliverables will be divided into sections.

3.1 GnuRadio Companion Modifications

The GnuRadio Companion block in figure 1 shows one deliverable, a modified wideband FM demodulator demo replacing the low pass filter block with a new FPGA co-processor block. The new block will provide an interface between GnuRadio and the kernel device driver for sending and receiving control / sample data. In the demo, the control data will be parameters for setting the CIC filter characteristics such as decimation rate. The use of mmap()[2] will allow GnuRadio and the device driver to exchange data without the overhead of copying data between user and kernel space.

3.2 Accelerator Coherency Port Linux Kernel Device Driver

The device driver will implement several functions to facilitate communication between GnuRadio and the FPGA co-processors.

- Allocate separate ring buffers for GnuRadio and the ACP Interface VHDL block to fill with sample data
- Allocate separate buffers for GnuRadio and the ACP interface VHDL block to fill with control data
- Handle interrupts that occur when the ACP Interface VHDL block finishes filling a buffer
- Implement the mmap() interface for GnuRadio to read the ring buffers from user space

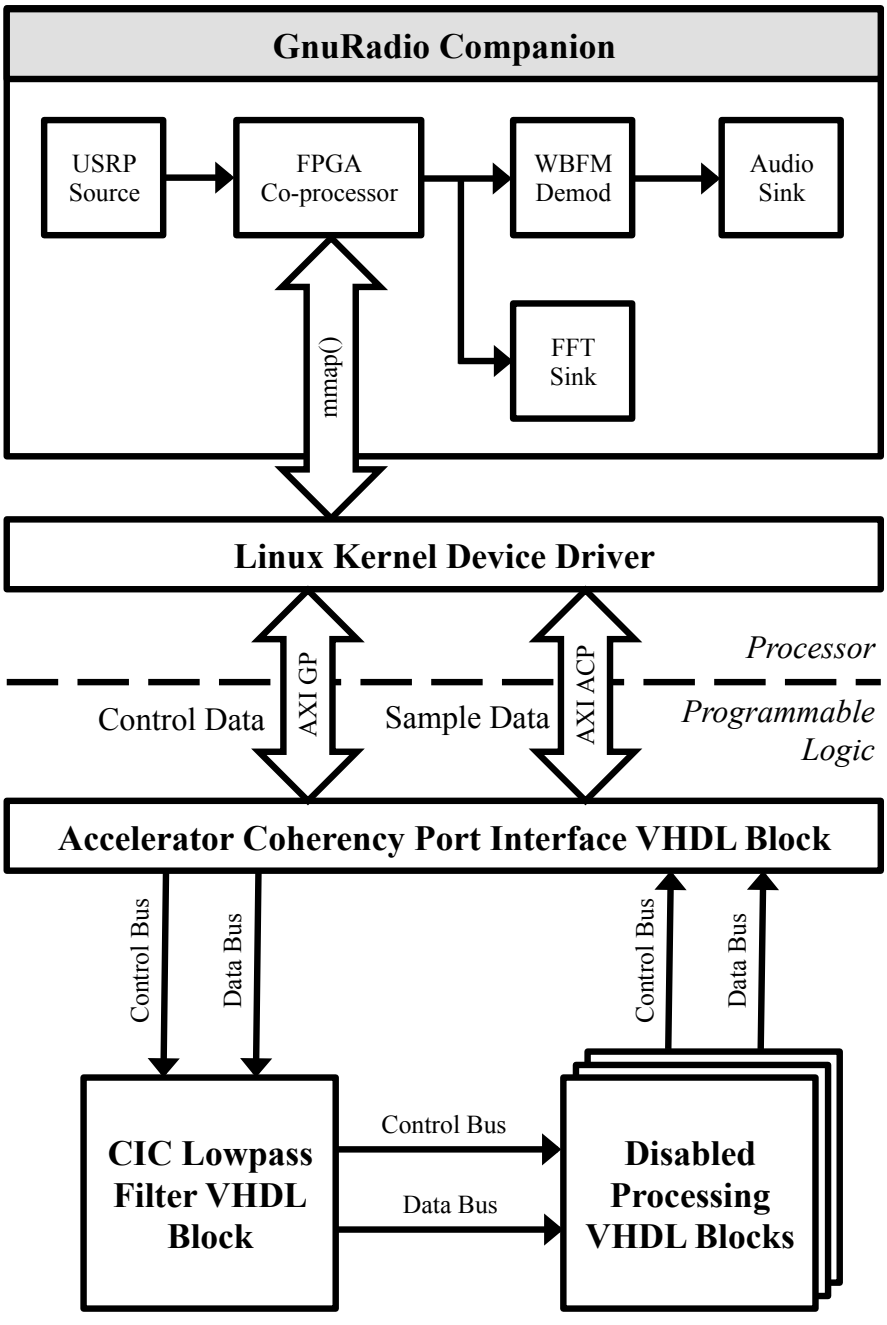

Figure 1: Block Diagram of Proposed FPGA Co-processor Framework

3.3 Accelerator Coherency Port Interface VHDL Block

The Accelerator Coherency Port Interface VHDL block provides the means for transferring data between the FPGA co-processor blocks to the ARM processor cores. The Accelerator Coherency Port (ACP a.k.a. AXI ACP)[3] provides the FPGA fabric a high throughput (1200 MB/s) 64-bit bus with direct write and read access to the L1/L2 caches. This port will be used to transfer sample data. Writes to the cache will be cache-coherent and the kernel device driver will provide memory addresses to the read/write ring buffers. The general purpose AXI interface provides the FPGA fabric a 32-bit bus similar to the ACP port. This interface will be used for lower speed control data. Sample data and control data transfers will be initiated by the ACP Interface VHDL block and trigger a kernel interrupt on completion.

3.4 FPGA Co-processor VHDL Blocks

The FPGA co-processor blocks reside in the Zynq's programable logic. Each block has input control and sample data buses and output control and sample data buses. To prevent excessive fanout that could reduce the clock speed, all blocks are connected in a daisy chain configuration. Since users many not want to use all the available blocks in the FPGA fabric, a mechanism through the control bus will allow blocks to be disabled. Blocks configured to not participate in the processing will forward data to the next block in the chain. This will reduce the need for the generally time consuming process of resynthesizing the FPGA bitstream. Only the first and last blocks in the chain are interfaced with the ACP Interface VHDL block.

For the project example demo, a parameterized decimating CIC filter VHDL block will be written for the wideband FM demodulation example. The CIC filter[4] is a good choice as the filter parameters, such as decimation rate, can be adjusted without resynthesizing the logic.

4 Deliverables

Each block outlined in the design section will have a design file and associated test bench / unit test. One special case is the ACP interface VHDL block, which will have an additional standalone C program to verify transfers without an operating system. This will be helpful for isolating bugs between the kernel driver and the VHDL code. Below is a preliminary list of expected design files:

- CIC Filter VHDL Block (VHDL, cic_filter.vhd)
- CIC Filter VHDL Block Test bench (VHDL, cic_filter_tb.vhd)
- Generic FPGA Co-processor Template VHDL block (VHDL, fpga_coproc_template.vhd)
- Generic FPGA Co-processor Template VHDL block Test bench (VHDL, fpga_coproc_template_tb.vhd)

- Accelerator Coherency Port Interface VHDL block (VHDL, zynq_acp_intf.vhd)
- Accelerator Coherency Port Interface VHDL block Test bench (VHDL, zynq_acp_intf_tb.vhd)
- Accelerator Coherency Port Interface Transfer Verifier Program (C, zynq_acp_verif.c)
- Accelerator Coherency Port Linux Device Driver (C, zynq_acp_driver.c)
- GnuRadio FPGA Co-processor Block (C++, fpga_coproc.cc)
- GnuRadio FPGA Co-processor Block Unit Test (Python, fpga_coproc.py)
- GnuRadio Companion FPGA Co-processor XML Description file (XML, fpga_coproc.xml)
- GnuRadio Companion wideband FM example flowgraph using the CIC Filter VHDL block (GRC, fpga_cic_fmdemod_example.grc)

Besides the files listed above, the project will provide a makefile capable of calling Xilinx's software tools necessary for synthesizing the FPGA portion of the design. Documentation will be written to guide a GnuRadio user with no background on the Zynq SoC on the how to install Linux and GnuRadio on the Zynq, create custom FPGA co-processing blocks, generate the necessary FPGA bitstream, and download the bitstream.

5 Timeline

The project aligns with my own graduate research, so I will begin work before the official start date and continue afterwards. I plan to spend 40 hours a week on the project, but likely more as this project is ambitious and will require a lot of time. Also note that my timeline includes time on the first week to vet the design with my mentor and the GnuRadio community. I believe this is important as a well thought out foundation will make future development easier and improve the chances for long term success.

- **Before GSoC official start date (3 weeks)**: Get a working development and test environment setup for the ZC706. This includes setting up the Xilinx software tools and building a Linux image for the Zynq with GnuRadio and GnuRadio Companion. Document installation procedures and any necessary workarounds.
- **Week 1**: Get to know mentor, discuss current status of project. Solicit advice from mentor and GnuRadio community on overall design. Finalize design with input from mentor and the community. Code Generic FPGA Co-processor Template VHDL block and CIC filter VHDL block. Verify CIC filter VHDL block with test bench.

- **Week 2-4**: Research and code Accelerator Coherency Port Interface VHDL block, test bench, and standalone ACP transfer verification C program.
- **Week 5**: Verify Accelerator Coherency Port Interface VHDL block and write documentation.
- **Week 6-7**: Research and code kernel device driver.
- **Week 8-10**: Partially verify system with kernel device driver, Accelerator Coherency Port Interface VHDL block, and CIC filter VHDL block. Write documentation.
- **Week 11-12**: Research and code GnuRadio FPGA Co-processor Block.
- **Week 13-15**: End to end system testing with modified wideband FM demodulation example in GnuRadio Companion.
- **Week 16**: Write makefiles and finalize documentation.

6 Personal Background / Qualifications

I am currently pursuing a PhD in computer engineering at Northeastern University in Boston Massachusetts and expect to graduate in 2017. I am a member of the Reconfigurable Computing Laboratory led by my advisor, Professor Miriam Leeser. My current research project is CRUSH[5], a Cognitive Radio research platform that allows off the shelf FPGA development boards, such as Xilinx's Zedboard and ZC706, to directly interface with the USRP via the MICTOR debug port. By using the debug port, CRUSH allows a user to have low latency (< 1 μs) access to the USRP transmit and receive sample data. As well, the user can utilize the full dynamic range and bandwidth of both the ADC and DAC, which is not possible through UHD. These features are desirable to Cognitive Radios as their spectrum sensing algorithms require wide bandwidths and low data latency. If interested, all of my CRUSH code is available on Github under the GPLv3 license.

I became familiar with GnuRadio while studying at Purdue, my undergraduate university. I spent two semesters in a software defined radio research group working on GnuRadio blocks for a FM receiver and Galileo receiver. After I graduated, I worked for 5 years in the defense industry writing embedded software and designing FPGA HDL code for communication systems and software defined radios.

This project excites me as I am interested in the performance gains of tightly coupled processor - programmable logic devices, especially when applied to software defined radios. It relates directly with my own research, CRUSH, as I had planned to port it to the Zynq to exploit the low latency access. After the

completion of this project, I want to continue adding more FPGA co-processing to GnuRadio, such as including the option for FPGA co-processing in VOLK.

7 Conclusion

The Zynq opens new possibilities for FPGA acceleration in software defined radios and GnuRadio. The addition of FPGA co-processing to GnuRadio will provide performance improvements and reduced data latency. From a big picture perspective, this project is the first step in GnuRadio and GnuRadio Companion to seamlessly support FPGA co-processing that is transparent to the user.

8 References

[1] Zynq-7000 All Programmable SoC, *http://www.xilinx.com/content/xilinx/en/products/silicon-devices/soc/zynq-7000.html*

[2] Linux Device Drivers, *Colbert J, Rubini A, and Kroah-Hartman G, 2005*

[3] Zynq-7000 All Programmable SoC Technical Reference Manual, *http://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf*

[4] Cascaded integrator-comb filter, *http://en.wikipedia.org/wiki/Cascaded_integrator-comb_filter*

[5] CRUSH: Cognitive Radio Universal Software Hardware, *http://jpendlum.github.io/crush/*