A Low-Cost Embedded SDR Solution for Prototyping and Experimentation

Christopher R. Anderson[1], George Schaertl[1], and Philip Balister[2]
[1]Wireless Measurements Group
Electrical and Computer Engineering Department
United States Naval Academy, Annapolis, MD 21402
[2]OpenSDR, Blacksburg, VA 24060
{canderso@usna.edu, gschaertl@gmail.com, philip@opensdr.com}

ABSTRACT

The release of a variety of embedded processor platforms over the past few years has sparked an interest in developing SDRs that are capable of operating on moderate-performance hardware. Currently, the two popular SDR frameworks-the SCA and GNU Radio-have been employed almost exclusively on general purpose processors. However, to utilize one of these frameworks, embedded platforms must both maximize the efficiency of the software as well as take advantage of co-processors (such as FPGA's) to perform computationally intensive tasks.

This paper presents the design and initial development of a low-cost SDR platform based around the Texas Instruments Beagleboard OMAP3 embedded processing platform along with off-the-shelf radio transceiver boards Initial performance results demonstrate basic functionality of both the FPGA interface board as well as the Beagleboard SDR engine. The platform can not only be used for embedded SDR applications, but is also valuable for low-cost SDR prototyping and experimentation.

1. INTRODUCTION

Over the past decade, software-defined radios have evolved from esoteric systems used by a handful of researchers or hobbyists into mainstream communication systems that are operated by and relied upon by government, military, and commercial end-users. These systems have been able to take advantage of the increasing amounts of power and performance offered by advanced ADC's, DSP's, GPP's, and FPGA's in order to meet the ever-increasing demands placed on SDR systems. A wide variety of vendors produce both general-purpose SDR platforms for commercial or military use (e.g., [1]–[5]), and details on numerous academically developed SDR platforms are available in the literature.

The current generation of SDR systems, however, are being severely challenged by a new set of user requirements that has arisen over the past few years—small size and weight, limited power consumption, and long battery lifetimes. These requirements are particularly poignant when developing SDR's for man-portable radios that may need to last for an 8-12 hour mission or wireless sensor networks that may require months or years of battery life. Meeting these demands going forward will require hybrid architectures that combine the best features of DSP's, FPGA's, ASIC's, and GPP's with designs that are optimized for power efficiency rather than raw processing performance. In [6], the authors investigate the tradeoffs between DSP, FPGA, GPP, and ASIC in the hardware design of SDR's, as well as hybrid solutions that combine the best features of different devices. The authors observe that one of the primary limitations to incorporating the power and flexibility of FPGA's and DSP's in SDRs are the complex design tools and highly specific instruction sets which do not always integrate into a C/C++ development environment.

One example of a hybrid system is [7], where the authors implemented a combined FPGA/DSP architecture to provide them with an extremely flexible platform for creating an SDR for receiving and decoding Global Navigation Satellite Signals. Their system utilizes the FPGA to process high-speed sample data by performing digital downconversion, carrier removal, and code correlation. The DSP can then focus on tracking and baseband processing. Another hybrid system example is [8] where the authors implement an SDR capable of MIMO processing by using one DSP and four FPGA's. On their board, the FPGA's are used to demodulate the input signals, which frees up the DSP to run smart antenna or other MIMO processing algorithms.

Perhaps the most well-known hybrid SDR is the Universal Software Radio Platform (USRP) and USR-

PII produced by Ettus Research [9]. Both versions of the USRP utilize an FPGA to perform pre-processing of data and either a USB or Gigabit Ethernet connection to send sample data to a GPP running an SDR architecture. These platforms have become widely used by industry, government, and academia, however, the architecture is based around using a very powerful GPP (such as an Intel Core2, AMD Athlon, or Cell Broadband Engine). Both versions of the USRP provide a relatively portable platform, fairly low power consumption, and–in conjunction with GNU Radio—have become the *de facto* standard for academic research platforms.

In order to meet the demands of smaller size and lower power consumption, a key focus area in future SDR development will be maximizing the efficiency of the hardware platform both through reducing software overhead and by utilizing co-processors (such as FPGA's) to perform computationally intensive tasks. Furthermore, cost is a major concern for SDRs deployed in applications such as wireless sensor networks, where thousands or tens of thousands of radios may be deployed in an area. To address these issues, this paper presents the design and initial development of a low-cost SDR platform based around the Texas Instruments Beagleboard OMAP3 embedded processing platform [10] along with off-the-shelf radio transceiver boards.The platform can not only be used for embedded SDR applications, but is also valuable for low-cost SDR prototyping and experimentation.

2. THE BEAGLEBOARD AS AN SDR PLATFORM

The OMAP3 family from Texas instruments [11] provides a low power processor that combines an ARM general purpose processor (GPP) with a Texas Instruments digital signal processor (DSP). The combination of GPP/DSP processors and an external FPGA provide the basis for a low cost/low power software defined radio.

The introduction of the Beagleboard [10] in 2008 provided a low cost platform for experimenting with the OMAP3 processor. Although the Beagle Board does not expose the optimum interfaces available on the OMAP3 for high speed data transfer to/from a FPGA, it does provide a platform for proof of concept applications. As discussed below, the hardware floating point support allows easy use of software developed on a desktop PC, without converting to a fixed point implementation. The DSP permits the designer to compare DSP implementations with GPP and FPGA implementations.

The OMAP3 processor contains an ARM Cortex-A8 GPP, an Image and Video Accelerator (IVA) based on the TI C64x+ DSP and numerous support peripherals. Specific part versions provide various combinations of DSP and graphics accelerators, including parts available with only the Cortex-A8 processor and not the DSP.

The Cortex-A8 processor [12] provides the armv7-a instruction set with the thumb-2 extensions. Thumb-2 allows the use of shorter instructions that still execute almost as fast as the conventional 32 bit instructions. This leads to smaller code size, this reduces the amount of non-volatile memory required for program storage. The OMAP3 also contains a coprocessor that implements the VFP light and advanced SIMD instruction (NEON) instructions. The NEON instructions operate on a variety of data types including short vectors of single precision floating point numbers and various size integers. These are valuable instructions for SDR applications.

The C64x+ processor is a VLIW design that supports eight functional units [13]. The eight functional units include 2 16x16 multipliers and six ALU's. This architecture allows the processor to execute eight instructions per clock cycle including two 16x16 multiplies per clock cycle. The DSP can access memory via a MMU. This allows the DSP and GPP to operate with hardware protection for their local memory. Unfortunately, the current TI dsplink software does not make use of this feature.

For this project, our interest was focused on the peripheral interfaces with the OMAP3 processor, which can support cameras, LCD panels, various serial interfaces, USB, different types of memory (such as DDR, NAND and NOR flash), graphics accelerators and JTAG interfaces. The Beagle board provides a 28 pin expansion connector. This connector exposes several interfaces, including McSPI (Serial Peripheral Interface) and MMC (Multi Media Controller). Although the MMC interface has the ability to transfer 4 or 8 bits per clock cycle, the first implementation of the interface to the FPGA uses the McSPI interface. The McSPI controllers are multi-channel SPI interface. SPI is a four wire serial interface with dedicated transmit and receive lines. The OMAP3 controller can clock the interface at up to 48 MHz. With 16-bit complex samples, the McSPI interface can transfer up to 1.5 MHz of sampled bandwidth between the FPGA and OMAP3. Newer Beagle Boards provide an EHCI USB host interface that allows the connection of the Ettus Research USRP. The EHCI provides another possible hardware interface, although the FPGA on the USRP does not have much room for custom FPGA processing.

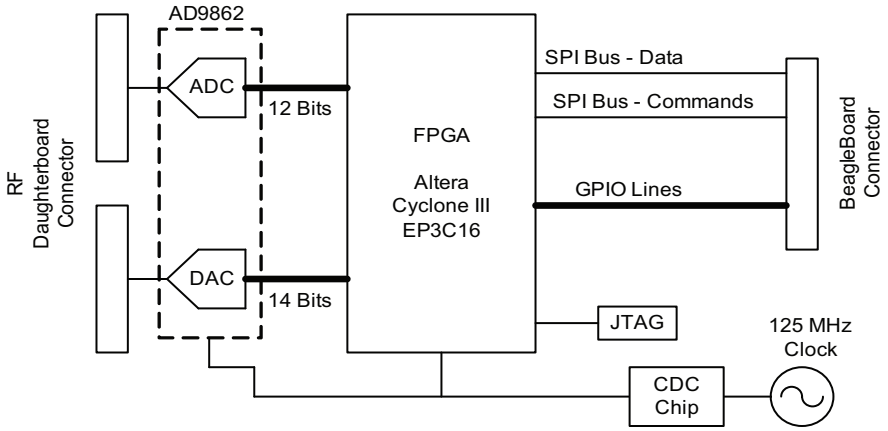

Fig. 1. A block diagram of the Beagleboard SDR interface board.

3. Interface Board Hardware Design

A basic block diagram of the embedded SDR system is shown in Figure 1. The system consists of a commercially-available analog RF front end, a mixed-signal front-end processor, FPGA, and digital interface to the OMAP3 processor. To take advantage of the existing open-source hardware and software resources, the RF front end was designed to accept any of the commercially available USRP Daughtercards [9]. The RF front end then interfaces with an Analog Devices AD9862 mixed-signal processor. Received sample data is then input to an Altera Cyclone III FPGA.

The FPGA communicates to the OMAP3 processor through two SPI (Serial Peripheral Interface) ports on the Beagleboard expansion header. One of the SPI ports is configured to stream transmit and receive sample data between the FPGA and Beagle at a maximum rate of 48 Mbps full-duplex. While the AD9862's has a 12-bit ADC and 14-bit DAC, the OMAP3 SPI controller requires data to be framed in 16-bit words, and adds overhead in the form of delays between words. As a result, the system is limited to handling a maximum signal bandwidth of 1.0 MHz. Bandwidths of up to approximately 6 MHz are possible by utlizing the Multi-Media Controller (MMC) interface (which supports 384 Mbps half-duplex), however, implementing a controller for the MMC interface is non-trivial and adds significant complexity to the system.

As a result of the bandwidth limitations of the SPI interface, the FPGA is primarily responsible for performing processor-intensive routines, such as digital up/downconversion, filtering, decimation or interpolation, or any signal processing routines desired by the user. Because of the expanded role of the FPGA compared to the USRP, the FPGA interface board uses a Cyclone III EP3C16, which provides faster operation and more logic gates than the Cyclone I EP1C12 FPGA on the USRP.

The FPGA interface board operates with a master clock which runs at a frequency of 125 MHz. A clock distribution chip divides the clock frequency in half and routes the clock signal to the AD9862, FPGA, and an off-board connector (for synchronizing additional peripherals). Additionally, the clock distribution chip will accept an external clock signal input at frequencies of up to 128 MHz.

One of the major design goals and design challenge of the FPGA interface board was to implement the board for a bill-of-materials cost of less than $500. The initial design, however, was relatively complex, requiring a 6-layer PCB with surface-mount components on both sides, and had a bill-of-materials cost of approximately $1,000. A second revision was able to more efficiently implement the system, resulting in a reduction of the bill-of-materials cost to approximately $600. All design and documentation files for the board are available on [14].

4. GNU Radio on the Beagleboard

GNU Radio is a free software runtime environment and set of DSP libraries for developing SDR systems. Although GNU Radio is platform-independent, it is designed to run on a desktop computer's powerful general-purpose processor. It makes heavy use of hardware-accelerated floating point math and includes support for the x86 architecture's SSE/SSE2 SIMD instructions. Hooks are provided to implement filtering routines in hand-optimized assembly for other architectures. [15] Where optimized routines do not exist for a given architecture, a generic implementation (written in portable C) is compiled and linked into the GNU Radio

libraries. On the BeagleBoard's OMAP3 processor, a 256-tap FIR filter with real coefficients ran between 20 and 25 times faster using a dot product routine written in ARMv7 assembly with NEON SIMD extensions than the same filter using the generic routine. Ongoing work focuses on including optimizations for ARMv7 and NEON in the `fftw` FFT library, as well as porting code to the OMAP3's on-chip floating point DSP.

GNU Radio can be compiled for the BeagleBoard, along with the rest of a complete Linux distribution, using the OpenEmbedded build system. OpenEmbedded consists mainly of a set of "recipes" telling the `bitbake` build automation tool how to download, configure, (cross-)compile, and package the binaries for individual software packages, as well as for meta-packages composing part or all of a complete system. Once a recipe is created for a new package, it may be cross-compiled in an environment already set up by OpenEmbedded, then installed on top of a running Linux system on the BeagleBoard. With an Ethernet adapter or USB mass storage device connected to the BeagleBoard, this allows code to be revised, recompiled, and tested without powering off the BeagleBoard, loading the new binaries onto its MMC card, and rebooting after every revision.

Once GNU Radio and any custom blocks have been compiled and installed on the BeagleBoard, using them is relatively simple. After importing the GNU Radio libraries into Python, a class representing the top-level flow graph is instantiated. The classes corresponding to each block are instantiated next, then connected to the flow graph. Processing is started or paused by calling the flow graph's `start()` or `stop()` methods. Blocks may be reconfigured on the fly if they provide mutator methods, for example, to change the taps on an FIR filter. If the flow graph is temporarily stopped, blocks may be added, deleted, or connected differently. Although GNU Radio includes a set of blocks that may be used to create a GUI with the wxWidgets library, OpenEmbedded does not currently include a recipe to build the Python bindings for wxWidgets.

5. Device Drivers and Interfacing

The OMAP3 provides several ways to interface with an FPGA, including: MMC, SPI, and GPMC. For this project we used the McSPI interface for simplicity and availability of the needed pins. Once the physical interface was decided, a device driver was needed to control the hardware and provide an interface to the radio software.

The McSPI controllers on the OMAP3 provide a four wire bi-directional serial interface. Additional lines are available as chip selects so multiple devices on a single bus may be controlled. The standard lines are slave in master out (SIMO), slave out master in (SOMI), clock and ground. The maximum clock rate available from the OMAP3 controllers is 48 MHz. Because the GPP runs Linux, the remainder of this discussion is focused on Linux device drivers.

The Linux kernel driver interface is the traditional Unix model with open, close, read, write, ioctl, etc [16]. The challenge for writing a driver for a SDR system is to map the needed functions into this model. Another possibility would be to write the driver using the UIO framework [17], although at this time the UIO framework does not support using the system DMA controllers. The DMA controller is very useful for transferring data from the FPGA to processor RAM without involving the GPP.

The Linux kernel contains a driver for the SPI controllers, greatly simplifying the code required for the FPGA interface driver. This code provides a standard API for all SPI controllers supported by Linux. By using this API, the driver can easily operate on different hardware providing SPI controllers without requiring changes to the driver code.

The hardware interface to the FPGA uses two of the MCSPI controllers. One controller transfers data to/from the FPGA for the signal processing chain. The other is used to read and write control register in the FPGA and the daughter boards. In the future, provisions will be made for sending some control signals embedded within the transmitted/received data to provide synchronized setup and status with the data samples.

The current version of the Linux MCSPI driver does not support the hardware buffering function or the use of DMA. Patches exist to enable these features, but they require more work before they can be integrated in to the Linux kernel. This leads to a measured transfer rate lower than the theoretical maximum rate. At this time we have not tested them to see how much performance improvement the use of these features provides.

6. Data transfer

An FPGA image created for testing the Linux device driver loaded data into a FIFO. The output side of the FIFO converted a 32 bit word to serial for transmission to the SPI master on the OMAP3. The input side of the FIFO was clocked by a free running counter and loaded count data into the FIFO. The output side is clocked by the SPI clock from the OMAP3 SPI controller. Finally, the FPGA used an OMAP3 GPIO pin to generate an interupt for the

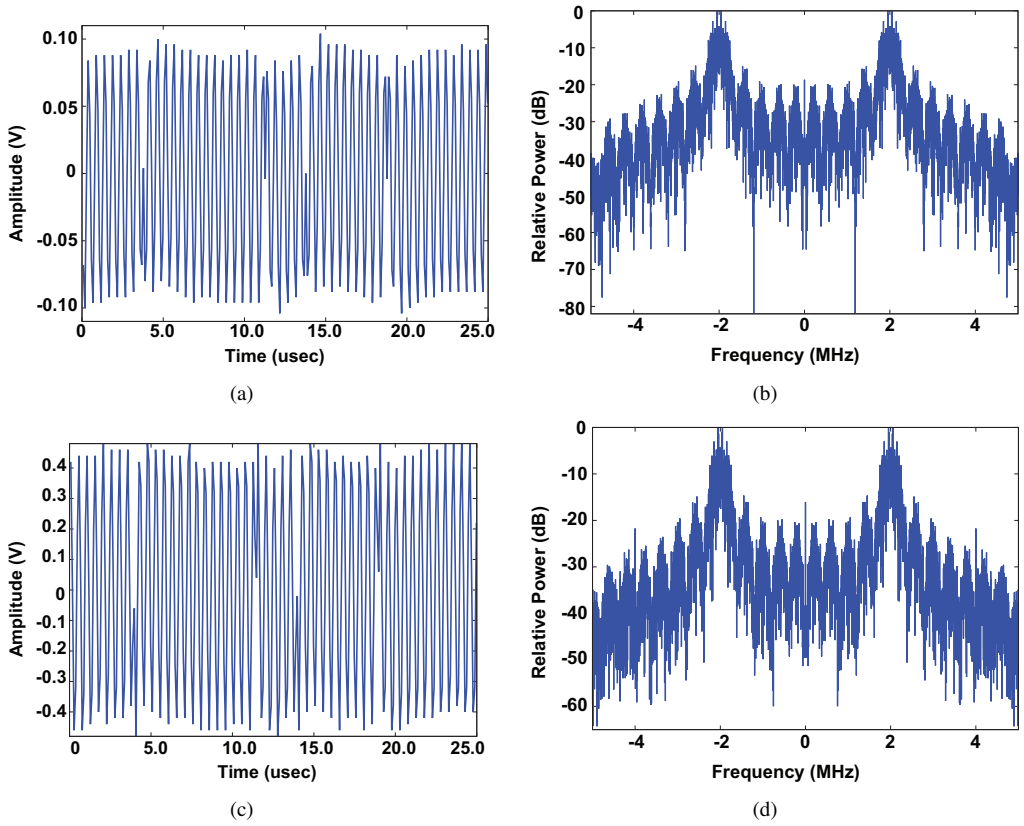

Fig. 2. A 2 MHz carrier with an amplitude of -10 dBm, BPSK-modulated with a pseudo-random bit sequence at 200 kbps, was connected to the board through the LFRX daughterboard. The FPGA was configured to connect the ADCs digital output directly to the DACs digital input. (a) and (b) show the input signal in the time and frequency domains; (c) and (d) show the DACs output through the LFTX daughterboard.

device driver when the FIFO contained 4096 bytes of data; this triggered a SPI read cycle that read 4096 bytes.

A test program on the Beagleboard read data from the device driver and did limited testing for dropped samples. We observed that the driver started dropping samples at about 3.5 Mbytes per second. Using 16 bit complex samples, the interface supports signals with a 890 kHz bandwidth.

We expect that the transfer rate could be improved by optimizing the SPI driver in the Linux kernel, however, even the un-optimized performance is adequate for a variety of signals.

We also compared the amount of processor time used by the driver with a similar program used to test the Universal Software Peripheral(USRP) from Ettus Research. The USRP connects to a host computer via a USB high speed interface. We attached the USRP to the Beagleboard's EHCI host USB port and used the `test_usrp_standard_rx` program to transfer the same amount of data at a rate close to 1 Mbytes pers second. Measuring the user and system times (using the time command) showed that the SPI driver used much less system time than the USB based USRP drivers. The data transfer required a total of 132 seconds. The SPI based interface used 0.42 seconds of system time and the USB driver used 8 seconds of system time. The reason for the performance difference is that the SPI interface is significantly simpler than the USB interface. The USB driver is receiving packets that could be from many different devices and must direct them to the proper destination. On a typical USRP based system, the USRP is connected to a fast desktop processor, however, for a small, low power system, it is important to minimize processor time spent on all aspects of the radio.

7. Initial Performance Results

To evaluate the performance of the Beagleboard as an SDR platform, a GNU Radio block, `spi_srcsink_ss`, was created for the BeagleBoard, allowing GNU Radio applications to access the FPGA using the SPI ports exposed on the BeagleBoard's expansion header. The block has been successfully implemented in a GNU Radio flow graph, sending

16-bit samples approximately 1.25 MHz (full-duplex). This will support a transmit and receive bandwidth of up to 625 kHz to and from the signal processing logic on the FPGA.

For the first test, the FPGA was loaded with a VHDL design that connected the ADC's digital output directly to the DAC's digital input, while making the ADC's control registers accessible to the BeagleBoard over a SPI port. A signal generator and digital oscilloscope were connected to the board using the LFTX and LFRX daughterboards available from [18]. The signal generator was configured to create a BPSK waveform at a frequency of 2.0 MHz using a 200 kbps pseudo-random bit sequence. Figure 2 shows the time and frequency domain plots of the signal input to the LFRX receiver, as well as the time and frequency domain plots of the output of the LFTX transmitter. From the figures, we can observe that the Beagleboard was able to successfully control the ADC by sending commands to the ADC control register through one of the SPI ports.

To evaluate the performance of the total system, two other tests were performed: basic transmission of an AM waveform and reception of an FM waveform. For the AM test, audio samples stored on an SD card connected to the Beagleboard were read into GNU Radio, which generated a simple AM waveform at a center frequency of 300 kHz. The Beagleboard then transmitted the samples to the interface board which broadcast the resulting waveform using the LFTX daughtercard; an analog RF front end then upconverted the rsulting signal to a center frequency of 905 MHz and broadcast the resulting signal over the air. A commercially-available radio was then used to verify reception of the signal.

For the FM reception test, a commercially available radio front-end was use to downconvert a signal from the FM Radio Band to an IF of 10.7 MHz. The resulting signal was amplified, filtered, and input to the LFRX daughtercard. The interface board digitally downconverted the received waveform to a center frequency of 300 kHz and transmitted the resulting samples to the Beagleboard. GNU Radio (running on the Beagleboard) was then used to downconvert, demodulate, and output the baseband audio data in real time.

Performance evaluation of the fully integrated system with digital signals, such as FSK and BPSK, is currently ongoing, and will utilize the second generation of the interface board and latest revision of the Beagleboard.

8. CONCLUSIONS

In this paper, we have described the design, development, and initial results of an embedded software defined radio platform based around the OMAP3 processor. The key hardware component of the system is a low-cost FGPA interface board that can receive and transmit signals via a USRP RF daughtercard. The system is designed to perform the bulk of the signal processing in the FPGA (filtering, downsampling, digital downconversion, etc), before streaming baseband samples to the OMAP3. Initial performance tests utilized a full-duplex SPI interface between the OMAP3 and FPGA, demonstrating that up to 1.5 MHz of spectrum could be transfered in each direction. The performance tests also demonstrated the ability of GNU Radio running on the OMAP3 to control the ADC and FPGA, as well as the ability to transmit and receive simple waveforms. Overall, the platform provides a very low-cost and low-power SDR platform for research, experimentation, and prototyping.

REFERENCES

[1] [Online]. Available: http://www.lyrtech.com
[2] [Online]. Available: http://www.gdc4s.com
[3] [Online]. Available: http://www.thales.com
[4] [Online]. Available: http://www.harris.com
[5] [Online]. Available: http://www.spectrumsignal.com
[6] M. S. Safadi and D. L. Ndzi, "Digital hardware choices for software radio (SDR) baseband implementation," in *IEEE Information and Communication Technologies Conference*, Damascus, Syria, 2006.
[7] M. Spelat, F. Dovis, G. Girau, and P. Mulassano, "A flexible FPGA/DSP board for GNSS receivers design," in *IEEE Research in Microelectronics and Electronics*, Ontario, Canada, 2006.
[8] Y. Li, X. Zhu, and L. Hu, "General multiple antenna evaluation platform," in *IEEE Conference on Mobile Technology, Applications, and Systems*, Guangzhou, China, 2005, p. Digest of Papers.
[9] [Online]. Available: http://www.ettus.com
[10] [Online]. Available: http://www.beagleboard.org
[11] [Online]. Available: http://www.ti.com/omap3530
[12] [Online]. Available: http://www.arm.com/products/CPUs/ARM_Cortex-A8.html
[13] T. Instruments, *iOMAP35x Applications Processor IVA2.2 Subsystem*, 2009.
[14] [Online]. Available: http://www.opensdr.com
[15] N. McCarthy, E. Blossom, N. Goergen, T. OShea, and C. Clancy, "High-performance sdr: Gnu radio and the ibm cell broadband engine," in *Virginia Tech Wireless Personal Communications Symposium*, June 2008, p. Digest of Papers.
[16] Sreekrishnan Venkateswaran, *Essential Linux Device Drivers*. Prentice Hall Open Source Software Development Series, 2008.
[17] [Online]. Available: http://www.kernel.org/doc/htmldocs/uio-howto.html
[18] [Online]. Available: http://www.ettus.com/