Cognitive Radio Kit Framework : Experimental Platform for Dynamic Spectrum Research

Khanh Le[†], Prasanthi Maddala[†], Craig Gutterman[†], Kyle Soska[†], Aveek Dutta[‡], Dola Saha[‡], Peter Wolniansky[*], Dirk Grunwald[‡], and Ivan Seskar[†]
[†] WINLAB, Rutgers University, NJ
[‡] Department of Computer Science, University of Colorado, Boulder
[*] Radio Technology Systems, NJ

ABSTRACT

This paper presents an overview of a Cognitive Radio Kit, an open software defined radio framework developed specifically to enable experimental research in cognitive radio and dynamic spectrum techniques. Currently available open software defined platforms are limited by performance and bandwidth constraints, and inadequate frequency tuning range at the RF front-end.

The proposed platform addressed those limitations by providing the ability to dynamically add hardware based acceleration for baseband processing, coupled with up to four wide-tuning range RF front-ends. The challenge resides in defining the architecture and programming model for the platform. All those considerations along with an application example are discussed and presented in this paper.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design - Wireless Communication

General Terms

Design, experimentation.

Keywords

Wireless, Software-Defined Radio, Cognitive Radio, Dynamic Spectrum Access, Architecture, Platform.

1. INTRODUCTION

The presented Cognitive Radio Kit (CRKIT) platform is intended to overcome performance and usability limitations of existing software defined radio devices to enable real-world dynamic spectrum access and cognitive radio networking experiments. Dynamic spectrum technologies are strategically important to the wireless community because of the need to alleviate spectrum congestion resulting from ongoing exponential growth in mobile data usage. A range of so-called dynamic spectrum access or cognitive radio techniques such as frequency agility, spectrum sensing, non-contiguous channels, etiquette protocols, Internet-based spectrum coordination and spectrum subleasing have been proposed as potential solutions, and taken together have the potential of providing order-of-magnitude improvements in overall spectrum efficiency.

While a great deal of theoretical work has already been done on dynamic spectrum techniques, larger scale experimental evaluations have yet to be conducted. The lack of experimental evidence of potential gains suggests that currently available open platforms are constrained with respect to some of following requirements for dynamic spectrum and cognitive radio applications :

1. Low latency processing,
2. Wide-band radio,
3. Wide-tuning range radio,
4. Fast frequency switching time.

The current open platforms have different strategies in providing programmable radio solutions. GNURadio[1] and SORA[2] platforms provides complete flexibility through software programmability. Those platforms are nevertheless somewhat constrained due to limited hardware acceleration capabilities. GNURadio bandwidth is typically in the order of 10MHz. While the Python scripting language provides for an easy programming environment, it also sets a limit on how much performance one can extract from the system. SORA, on the other hand, operates at higher bandwidth by distributing baseband functions on multiple General Purpose processor cores. The programming model of SORA as such is more elaborated as users need to have a good understanding of the underlying processor architecture and feature sets. For example, certain baseband functions are written such that they minimize cache misses. This translates to non-trivial optimization work from the user side to achieve the best performance.

To avoid these limitations, the WARP[3] and AirBlue[4] platforms trade off some flexibility with low latency processing in hardware. As for programming model, WARP platform uses MATLAB Simulink tool for baseband designs, whereas AirBlue relies on Bluespec[5]. However, both WARP and AirBlue frameworks are rather monolithic in the sense that all baseband functions are centered around the processor which also provides the interconnect subsystem, and all traffic whether control or data must therefore request access to the bus. Scalability may become an issue as more PHYs are added onto the bus, hence the bus bandwidth will become a bottleneck.

CRKIT was designed specifically for Dynamic Spectrum and Cognitive Radio applications in mind, and similar to the WARP and AirBlue approaches, the CRKIT baseband processing is FPGA-based to address the low latency requirement. Furthermore, CRKIT incorporates additional features at the architecture level which allowed for ease of programming and modularity. Obviously, the radio part is an essential component of the overall framework. The CRKIT radio permits for a wide-tuning range and relatively wideband e.g. 36MHz. The CRKIT baseband architecture permits for multiple PHYs and other Cognitive Radio functions such as spectrum sensing, in addition to the support of up to four simultaneous full duplex wide-tuning range RF front ends. All these capabilities are integrated into a single environment. This platform serves as a proof-of-concept for further development of more advanced functions, as will be presented in subsequent sections.

In section 2, the wide-tuning range radio (WDR) transceiver and FPGA-based baseband processing hardware are described. Section 3 talks about the FPGA-based System-on-Chip (FSoC) framework with pluggable application (APP) modules. The CRKIT transport layers will be detailed along with the concept of static FSoC and dynamic APPs . Section 4 talks about the system programming model with emphasis on design methodology and user interfaces. In section 5, a CRKIT APP example is used to demonstrate how two uncoordinated cognitive radios can be synchronized using a Rendezvous algorithm. Finally, the paper is concluded in section 6, followed by a discussion on future work, including a platform development road map.

2. CRKIT HARDWARE PLATFORM

The CRKIT platform consists of WDR transceiver daughter cards mounted on an FPGA-based baseband processor board. The baseband motherboard is a commodity off-the-shelf board, whereas the WDR transceiver card was custom designed and manufactured according to our specifications. Up to 4 full-duplex WDR modules can be stacked on the baseband processor motherboard, where they can be operated independently using FSoC framework. The actual CRKIT hardware platform is shown in Figure 1 with two WDRs. Two more radios can be stacked on top, adding to a total of four independently tuned radios.

2.1 Radio Module

The radio provides a wide RF tuning range from 100 to 7500 MHz using heterodyne downconversion and sharp IF filtering for excellent adjacent channel rejection, allowing operation as close as 5-10MHz from strong interferers. The frequency tunable step-size is 0.5Hz using Direct Digital Synthesizer, and nominal frequency switching time is 50μs with some provisions for very fast switching e.g. in the order of 1μs. The radio can be configured to operate in either half or full duplex mode over entire tuning range allowing any combination of uplink and downlink frequency sets. The transmitter baseband bandwidth is rated for 48MHz, whereas the receiver baseband bandwidth is 36MHz. The

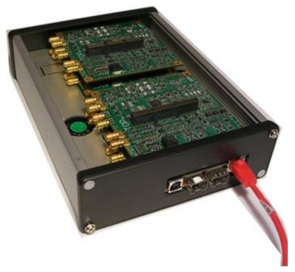

Figure 1: CRKIT hardware platform with enclosure, dual WDRs mounted on baseband processor.

receive path also includes dual 12-bit, 50MSPS ADC, and the transmit path includes dual 12-bit, 200MSPS DAC. The sampling rate reference clocks can be provided either by the baseband or generated locally on the radio module itself.

2.2 Baseband Processor

An off-the-shelf FPGA board is used for signal and baseband processing. The board includes a Xilinx Virtex5 SX95T FPGA and multitude of high-speed interfacing options such as GigE, USB2.0, and PCIe. The WDR modules are mounted on EXP connectors, where the radio Serial Peripheral Interface (SPI) control busses, ADC and DAC I/Q channels and corresponding reference clocks are routed to and from the baseband FPGA.

Moore's law states that the transistor counts and densities are to double approximately every two year, therefore the CRKIT hardware platform can benefit from this technology advancement. More importantly, new generations of FPGA baseband boards can be used with little additional engineering cost. An "upgrade path" is in place to allow the radio transceiver design to be used on increasingly capable (or cheaper) signal processing boards, including emerging SoC-FPGA [6]. The baseband FSoC framework design should therefore be easily upgradeable to newer and more powerful hardware platforms, or in terms of FPGA higher speed and density. To support this future upgradability option, the FSoC framework must be modularized and be FPGA technology independent to the greatest extent.

3. SYSTEM-ON-CHIP FRAMEWORK

Designing and building a real-time, large and complex baseband FSoC from the ground up is not a trivial endeavour, especially for our targeted audience consisting of wireless communication researchers and students. It requires extensive hardware design skills, usually only afforded by a relatively large engineering design team. Chances are the targeted user base does not have access to such resources. Even if present, it is unlikely that serious engineering design resources will be readily available for ad-hoc experimentation purposes.

Innovation requires prototyping and ad-hoc experimentations to complete the full innovation cycle as illustrated in Figure 2. The first two phases of the innovation circle are considered as the creative processes (*Idea* and *Algorithms/Models*), whereas the last two are mostly engineering driven processes (*Build Radio* and *Live Experiments*).

The *Build Radio* step constitutes a substantial barrier to entry for many users. Building a complete working radio re-

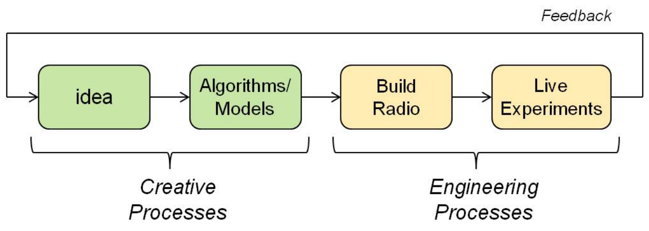

Figure 2: Innovation Cycle

quires cross-disciplinary expertise in areas such as RF, hardware, software, networking, communication and so forth. Therefore, with CRKIT, users should concentrate more on the creative aspects of the wireless problem, less on the complex engineering problems associated with building a radio. In other words, focus on creativity rather than engineering complexity. This can be achieved by dividing the baseband framework into two domain spaces - *Static* and *Dynamic*, as shown in Figure 3.

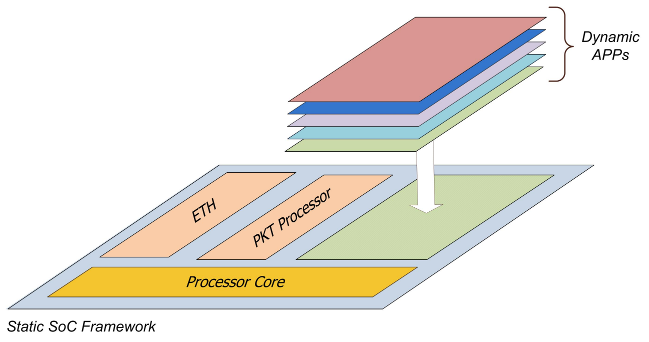

Figure 3: Downloadable Dynamic Application.

The dynamic APP modules are associated with the creative processes, whereas the static FSoC framework refers to the complex engineering problems. Intuitively, CRKIT users would become APP developers, while the static FSoC framework is maintained by framework developers. Depending on the experiments, different APPs can be linked onto the static framework. The FSoC framework contains a complete system with a 32-bit processor core, along with Ethernet, Packet Processing and RF interfacing modules. This framework is fully open sourced, it should therefore be feasible for experienced users to modify and experiment with lower level FSoC framework design as well. However, in most cases it is anticipated that users will confine themselves inside the APP domain.

3.1 Framework Architecture

The top level framework architecture is shown in Figure 4. The block functionality is summarized as follows :

- *Ethernet Port (static)* - provides interfacing to Gigabit Ethernet port. Inbound Ethernet frames synchronization. Outbound ethernet frames formatting.

- *Packet Processor (static)* - simple packet classification/forwarding scheme based on IP/UDP. Control packets get routed to *Processor Core*, whereas Data packets

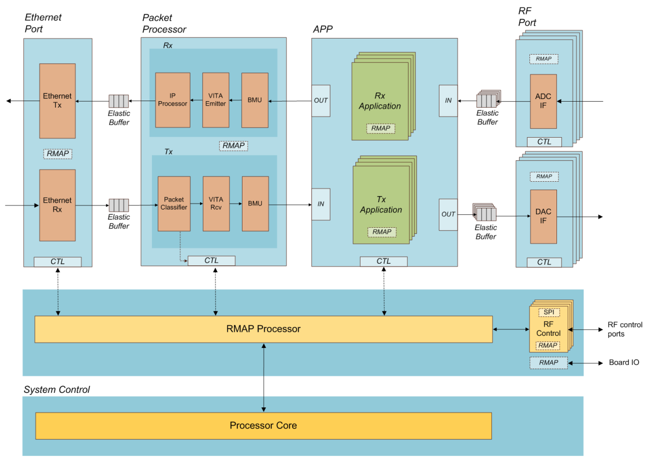

Figure 4: CRKIT FPGA Framework - static SoC and Dynamic APPs.

are forwarded to corresponding *APP* for further wireless layer processing. Support a subset of VITA Radio Transport protocol [7]. APP data buffer management.

- *APP (dynamic)* - user specific application, could be simple QPSK/QAM, OFDM and so forth. Support up to 4 APPs simultaneously (one APP per RF), APPs can be swapped as needed by users depending on the application.

- *RF Port (static)* - interfacing to ADC/DAC.

- *RMAP Processor (static)* - general sub-system interfacing and control, provides processor interfacing, address decoding and RF SPI control functions.

- *Processor Core (static)* - 32-bit RISC-based embedded processor, and bus interconnect with interfaces to external 32MBytes DRAM and 16MBytes FLASH.

The baseband data path does not require access to the system bus, eliminating potential bottlenecks imposed by the interconnect subsystem. All static domain modules were designed using VHDL hardware description language, and validated according to the rules of good chip design practices. The APPs can be designed using either VHDL/Verilog or Mathworks Simulink tool, and verified using the CRKIT APP development environment available for Simulink. In principle, any other design methodologies, such as Bluespec[5], are applicable for APP designs. The key point is users should remain creative in developing APPs, and not be weighed down by complex engineering considerations at the FSoC level.

The framework design is modularized such that individual modules can be upgraded or replaced without major disruptions to the overall system architecture. In particular, the *processor core* is considered as a "black-box", in the sense that it may be swapped out in future revision of the framework. The current processor core (PCORE) is based on Xilinx 32-bit Microblaze softcore processor and CoreConnect PLB bus, this processor subsystem was selected primarily for conveniency, as it is well integrated into the Xilinx tool chain environment. An Open Source 32-bit RISC-based processor from the OpenCores community, OpenRISC [8], could have been used instead, or a hard macro ARM-based processor and AMBA bus [6].

3.2 Transport Layers

Three distinct data paths are defined for traffic flows through the system :

1. *APP/PCORE* to outbound ethernet port
2. Inbound ethernet port to *APP*
3. Inbound ethernet port to *PCORE*

The inbound traffic flow, i.e., from network to CRKIT, is as shown in Figure 5. The left-hand side protocol stack resides within the static framework domain, whereas the *User Specific Layers* are within the dynamic APP domain. The framework protocol stack is only defined up until the VITA Radio Transport (VRT) layer [7]. VRT is an emerging standard for SDR applications, it provides interoperability between diverse SDR components by defining a common transport protocol to convey signal data and radio parameter settings.

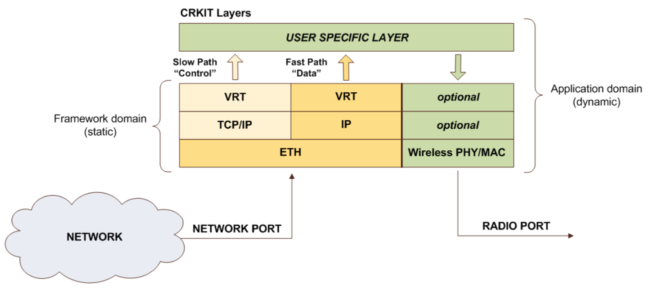

Figure 5: CRKIT Transport Layers

Each protocol layer has specific processing requirements as summarized below.

- *ETH* - Ethernet Physical layer only, no MAC. Only Ethernet frames with Broadcast MAC or matching destination MAC addresses are forwarded to IP layer.
- *IP (Fast Path)* - Hardware based implementation, only a subset of IP and UDP functions. This fast track is reserved for data related traffic requiring fast and direct access to APP domain. Data IP packets are routed to the fast track based on specific UDP port numbers.
- *IP Layer (Slow Path)* - Software based implementation, does support TCP as this is done in software. This slow track is reserved mostly for control related traffic such as CRKIT hardware configuration (register map access) and RF control. Any IP packets not destined for the fast track will by default be routed to the slow track.
- *VRT* - VRT layer is optional, it can be bypassed if not used. VRT is particularly useful to mux multiple radio streams to a single pipe, and demux at the other end. Two types of packets are defined by VRT protocol : 1) *Data* for signal data transmission, could be digitized I/Q samples. 2) *Context* for control information such as frequency, power, bandwidth settings and so forth.

Once in the *User Specific Layers*, users have unrestricted freedom to implement any additional protocol layers as required. Typical inbound packet processing flow is illustrated in Figure 6. Inbound IP packets are parsed by the *Packet Classifier* which consists of a simple IP filtering block. If IP traffic is of type UDP with port numbers 1000-1004 (configurable), then data gets routed to the *VITA Receiver*, otherwise the IP packet is forwarded to PCORE for further parsing. The IP-to-VITA path is defined as the *fast track*, whereas the IP-to-PCORE path is the *slow track*. Furthermore, UDP port number 1000 is reserved for VITA based traffic, whereas UDP ports 1001-1004 are reserved for non-VITA traffic. The four non-VITA UDP ports are available for tunneling data to potentially four different transmit APPs. The association between UDP ports and corresponding APPs are made using a programmable look-up table. Each APP is given a specific Port Identifier (PID), hence a virtual link between UDP port number and specific APP (i.e. PID) can be generated. This UDP-to-APP mapping look-up table is available within the *VITA Processor* block, and PCORE software configures this look-up table as part of the initial configuration settings. Once the UDP-to-APP mapping sequence is completed, the "raw" data is temporarily stored in the *Buffer Management Unit (BMU)*, APP then pulls the data for further processing.

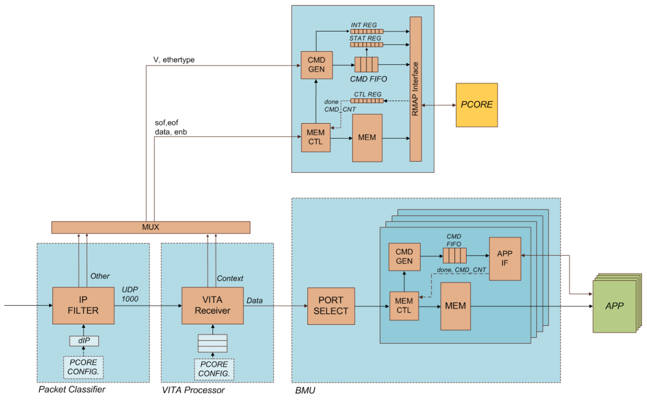

Figure 6: Inbound Packet Processing Flow

The typical outbound traffic has a similar data flow but in reverse order as illustrated in Figure 7. Here, the traffic flows from either APP or PCORE to the ethernet network port. APP/PCORE data are temporarily stored in the *BMU*, and multiplexed into a single stream for processing further down the pipeline i.e. *VITA Emitter*, *IP Processor* and *Ethernet Port*. Data from APPs or PCORE is selected in a round-robin fashion. The traffic encapsulation scheme is user programmable on a per flow basis, the following three encapsulation options are available :

1. *Ethernet, IP and VITA* - "raw" data is encapsulated with VITA headers, followed by IP and then Ethernet frame formatting.
2. *Ethernet and IP* - IP encapsulation, followed by Ethernet frame formatting.
3. *Ethernet only* - Ethernet frame formatting only.

Similar to the inbound traffic, each outbound flow is associated with a PID. For traffic originating from PCORE,

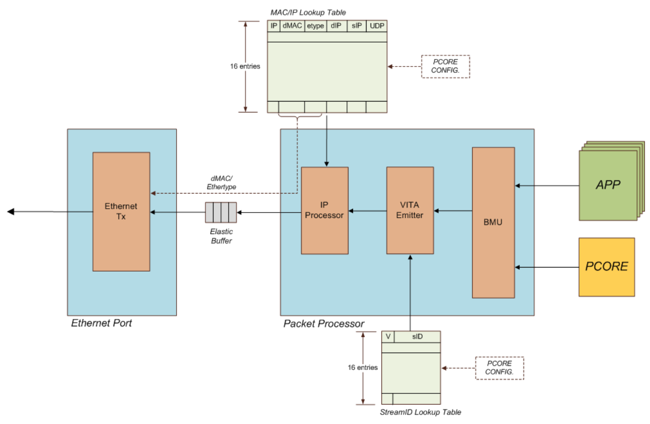

Figure 7: Outbound Packet Processing Flow

different encapsulation schemes based on the traffic type can be configured. For example, DHCP and ARP traffic would be configured as *Ethernet Only* mode i.e. no hardware based VITA and IP encapsulations are required since the lower level data formatting is being done by software. Hardware only adds the Ethernet header and tail.

The encapsulation scheme is computed dynamically as traffic flows down the pipeline using the associated PID. The PIDs are essentially pointers to the two lookup-tables, *StreamID* and *MAC/IP*. At the *VITA Emitter* block, using the PID for current data flow, the V-flag bit is fetched from the *StreamID* look-up table, and if set then VITA formatting is enabled. VITA header is appended to current data stream using information from *StreamID* table. Similarly, the *IP Processor* block checks the IP-flag bit, if set then IP packet encapsulation scheme is enabled. For each PID, there is an associated data tuple i.e. IP destination/source addresses, UDP port number, destination MAC address, and Ethernet Ethertype field. Using this information, the IP and Ethernet headers can be added dynamically to the data stream. The look-up tables are configured by software during initial system configuration or when a new flow is being setup.

The described inbound and outbound data flow provide a highly pipelined and flexible architecture. Since processing is configurable on a per flow basis, a flow can easily be added or torn down, thus creating a virtual processing link from source to destination. APP developers can just link the designed dynamic APPs to the framework, configure the virtual processing path, and traffic should start flowing from APP to ethernet port, and then to a networked Host machine. All these functions are available as part of the FSoC static framework.

4. SYSTEM PROGRAMMING MODEL

The CRKIT programming model consists of the following three processes :

- CRKIT APP development
- CRKIT embedded software development
- Host software development

A Host machine could be any machine on the network which transmits and receives data to and from the CRKIT hardware platform. The Host machine and CRKIT platform combo essentially forms a client-server computing model. In such model, the Host provides the higher level supervisory algorithmic functions, whereas lower level hardware-based acceleration are available at the CRKIT platforms. In this respect, the CRKIT static framework forms a virtual link between client *(APP)* and server *(Host)*.

4.1 APP Development

The individual steps necessary for APP development and eventual integration into the overall framework are as shown in Figure 8. The APP can be designed using either VHDL/ Verilog or Mathworks Simulink combined with Xilinx System Generator tools. For novice APP developers, it is recommended to use the graphical Simulink environment as this reduces the complexity and learning curve associated with the APP development process. The Simulink graphical environment is more intuitive to use, and a plethora of Simulink tools are available such as filter design, state flow charts, waveform viewers, channel modeling and so forth.

The dynamic APP is verified using the CRKIT Simulink testbench environment in which the static framework data flows are emulated, including PCORE read/write accesses. This design environment allows for the development of communication APPs independently from the overall FSoC. This reduces the users focus area to APP domain only, hence minimizing both design and verification time.

Following the APP Validation phase, the APP is compiled to produce a binary file which is then linked to the overall framework binary file. The complete framework is then built to generate a downloadable FPGA bit file. Finally, the bit file is loaded onto baseband FPGA, and the system boots up. At this point, PCORE proceeds with the execution of the CRKIT embedded software.

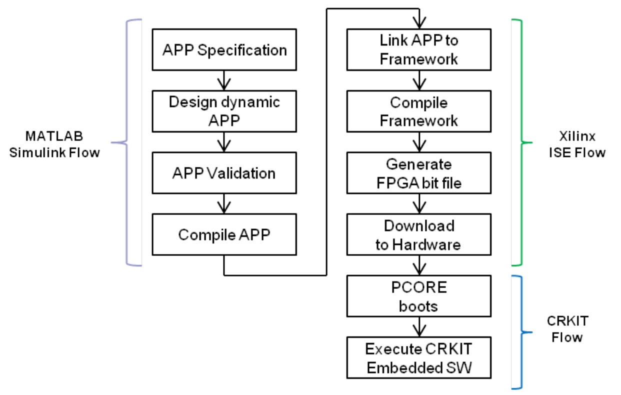

Figure 8: APP Development and Integration

4.2 Software Development

The current embedded software is relatively rudimentary and written entirely in C-language. This basic software is mostly used to bring up the system and configure the RF and virtual connections between APPs and Host. There are plans to port Linux i.e. uClinux to the current platform to take advantage of the networking stack available for Linux. The embedded software functions are as shown in Figure 9.

Upon system boot, the CRKIT hardware is initialized to some default settings. The PCORE-to-EthernetPort virtual processing path is configured to permit transmission of net-

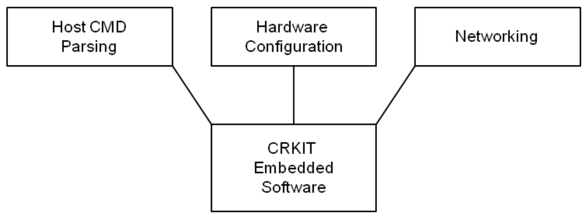

Figure 9: CRKIT Embedded Software Functions

work related traffic such as DHCP and ARP packets. This processing flow is set to *Ethernet Only* mode of operation via the *StreamID* and *MAC/IP* look-up tables. The Ethernet payload content is generated by the embedded software according to the network protocol. As part of the initial hardware configuration, any RF default settings pertaining to the operation of the WDR are also programmed through the RF Control SPI bus interface. For example, duplexing mode, sampling rates, center frequency, power level and so forth. Once the default hardware configuration is done, software initiates DHCP to retrieve an IP address from the network, followed by a link up with the Host machine.

Dynamic APPs require their own set of configuration options, and the initialization of those APP settings is performed by Host application software. Host may query CRKIT for currently installed APPs, and send dedicated Ethernet/IP commands (CMD) to configure those APPs. Host has full access to all register maps available on the CRKIT i.e. control, status, and interrupt registers, similar to what is available to PCORE. Host can configure the RF, APPs, and static framework dynamically while the system is running live, just as if Host software is executed locally on CRKIT. This configuration scheme requires atomic level register read/write ethernet-based CMDs being sent from Host. Another option is to use VITA transport protocol format to package multiple CMDs into a single IP packet. The VITA context message is parsed and processed at CRKIT, this reduces the amount of configuration related traffic between Host and CRKIT. Furthermore, VITA can be used as interfacing protocol between CRKIT and GNU Radio software platform, thus leveraging a substantial amount of available signal processing blocks from GNU Radio. In this respect, CRKIT provides hardware acceleration functionality.

Host application software can be developed using any available programming language of choice. The current Host software was developed using Java and C# to create GUI-based application for system debugging and testing purposes, while C-language was used for algorithmic efficiency.

5. APPLICATION EXAMPLE

The capabilities of CRKIT platform are best illustrated by a system design example. It is outside the scope of this paper to provide a thorough theoretical foundation, but a dynamic spectrum access technique is demonstrated using two CRKITs, each with a dual RF front-ends. The two radios are initially uncoordinated but need to establish a wireless communication link in the presence of interference by opportunistically utilizing available spectrum. For this particular example, a blind rendezvous [9] algorithm was chosen to synchronize the two radios on a RF channel. No common control channel is used to exchange information. The only available information to the radios is respective spectrum sensing data. Using this information combined with the rendezvous algorithm, the radios should attempt to synchronize as described in [10]. Both radios operate in half-duplex mode, and once rendezvous is successfully achieved, they take turn to communicate. If the communication channel becomes inefficient i.e. high bit error rate, the radios initiate a new rendezvous search to select a better communication channel.

5.1 Rendezvous Algorithm

The number of possible RF channels is limited to 16 for this initial experimental validation of the rendezvous algorithm. The algorithm ranks each of the 16 channels based on spectrum information provided by the CRKIT Spectrum Sensing APP. Each channel is ranked based on energy level e.g. the higher the energy level, the lower rank it gets. Therefore, ranking is based on quantized energy level, and using threshold settings a channel can be flagged as either *free* or *occupied.* This channel occupancy map is used by the rendezvous process to hop through the *free* channels in search of the other radio. Both radios perform the same operation independently, they each have their own channel occupancy map and chances are that some of the *free* channels are overlapped between the two radio occupancy maps. Theoretical work have shown that the jump-stay based hopping sequence guarantees rendezvous in finite time [10]. This finite rendezvous time is one of the interesting results to be obtained through this experiment.

At each *free* channel, the rendezvous algorithm sends a Beacon, waits for some time for a response, and if none is received within the time-out period, the algorithm hops to the next *free* channel and repeats the process again. The radios are always in listening mode, except for Beacon transmissions. A successful rendezvous is assumed when an ACK is received within the time-out period after Beacon transmission. This completes the rendezvous search process, and the communication process takes over for traffic transmission between the radios. The radio communications are enabled using a simple slotted ALOHA multiple access protocol, where MAC level processing is done by Host, whereas PHY layer processing is performed by CRKIT.

5.2 Rendezvous Application Modules

To support the rendezvous and communication processes, the following CRKIT functions are necessary : 1) Spectrum Sensing for channel ranking purpose and 2) half-duplex Quadrature Phase Shift Keying (QPSK) Modulation transceiver for communication. These functions were developed using three dynamic APP modules, as shown in Figure 10. The outbound data flows are generated by the Spectrum Sensing and QPSK Receiver APPs, whereas the inbound traffic flow is absorbed by the QPSK Transmitter APP. Spectrum Sensing and QPSK transceiver APPs are connected to RF module 1 and 2, respectively.

The Spectrum Sensing APP performs a 64-point FFT on the IQ samples coming in at a rate of 25MSPS, and generates the power spectral density over the 16 channels, i.e., 4-point per channel. The power levels are then averaged over multiple FFT computations and sent to Host for further post-processing by the rendezvous algorithm.

The QPSK transmitter APP functional block diagram is illustrated in Figure 11. Here, digital bits from Host are

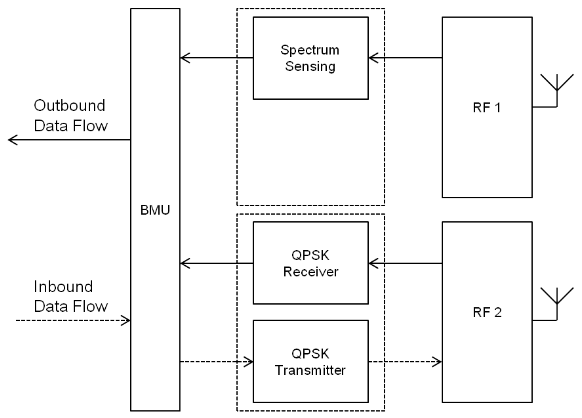

Figure 10: Application Modules for Rendezvous Process

mapped to QPSK symbols. These symbols are upsampled by 32 resulting in a 25MSPS sampling rate, and pulse shaped using a Root Raised Cosine (RRC) filter. This is followed by Frequency Translation and Digital-to-Analog conversion. The Frequency Translation converts baseband signal to IF, equivalent to one of the 16 possible communication channels as required for the rendezvous process.

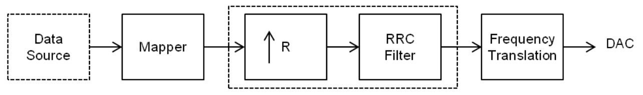

Figure 11: QPSK Transmitter APP diagram

As shown in Figure 12, the received IF signal is converted back to baseband by the Frequency Translation block. The signal is passed through a RRC matched filter, downsampled by 32, and the symbols demapped to digital bits. Furthermore, carrier frequency/phase offset errors are compensated using Costas Loop, whereas symbol synchronization is achieved using Maximum-Likelihood timing estimation. Frame Detector determines the start-of-frame based on Barker sequence preamble. The detected frame is then passed further down the outbound data processing pipeline.

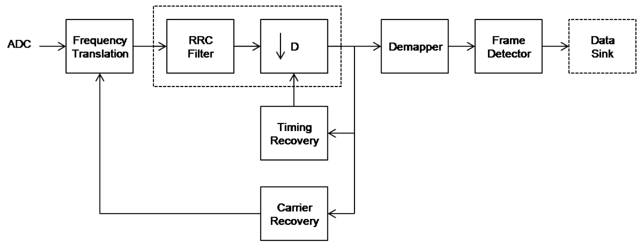

Figure 12: QPSK Receiver APP diagram

All APPs were designed using Mathworks Simulink environment, and ported to the external framework design. A Simulink based implementation example of QPSK Receiver is shown in Figure 13. From the left, the ADC I/Q data are pulled from the ADC *Elastic Buffer* (refer to Figure 4) and fed into the *Frequency Translation* and *Automatic Gain Control* (AGC) modules. The subsequent processing steps are as explained previously for Figure 12. On the right hand side, the *Packet Processor Interface* provides the interfacing to the BMU, whereas the *Clock Domain Transfer* module enables clock domain crossing between APP and framework domain spaces. Finally, the *Register Map* module provides PCORE interfacing ability, allowing software to configure and monitor the APP functions.

The FPGA resource utilization level for this rendezvous experiment is shown in Table 1. From APP domain perspective, the Spectrum Sensing function use the least amount of Flip-Flops (FF), but the most number of block RAMs (BRAM). The 64-point FFT operation requires more storage and arithmetic than logic operations i.e. multiplication and addition. These arithmetic operations use more of Xilinx DSP48E slices, and less of Logic slices. The QPSK Receiver has a larger FF count than Transmitter due to the additional Carrier and Timing recovery blocks. The static framework contains the highest amount of FF i.e. ~17% and BRAMs (for PCORE instruction/data caches and boot code, and BMU storage). Ideally, the remaining 83% of the FPGA FF resources can be used for APP development, obviously this will be reduced due to limited routing resources.

The complete FSoC FF utilization for this particular experiment is 30.37% which can easily be placed and routed on the physical FPGA. The fraction of static framework utilization level can dramatically be reduced if migrating to newer FPGA technologies, hence leaving substantial resources for APP development.

Table 1: Xilinx Virtex-5 SX95T FPGA Resource Utilization

Module	FF	Util.	BRAM	Util.
Spec. Sen. APP	338	0.57%	7	2.87%
QPSK Tx APP	2,798	4.75%	2	0.82%
QPSK Rx APP	4,834	8.21%	3	1.23%
Static Framework	9,912	16.83%	35	14.34%
Complete FSoC	17,882	30.37%	47	19.26%

5.3 ORBIT Integration

The rendezvous experiments are facilitated by CRKIT platform integration into the ORBIT [11] wireless network emulator. The CRKIT platform acts a physical layer extension to the current radio nodes, allowing ORBIT users to experiment with lower wireless physical layers as well as the higher protocol and application layers. In particular, the wide RF tuning range and real-time aspects of the platform will prove to be fundamental to experimental research in Cognitive Radio and Dynamic Spectrum Access techniques. The current rendezvous experiments are performed on ORBIT Sandbox 6 (SB6). SB6 consists of two nodes, each having a single CRKIT hardware platform. Each node acts as Host machine for a CRKIT platform, where the rendezvous application software is executed. This rendezvous experimentation platform is currently open to ORBIT users, the eventual goal is to create a library containing a variety of APPs for research and development purpose in Cognitive Radio and Dynamic Spectrum Access techniques.

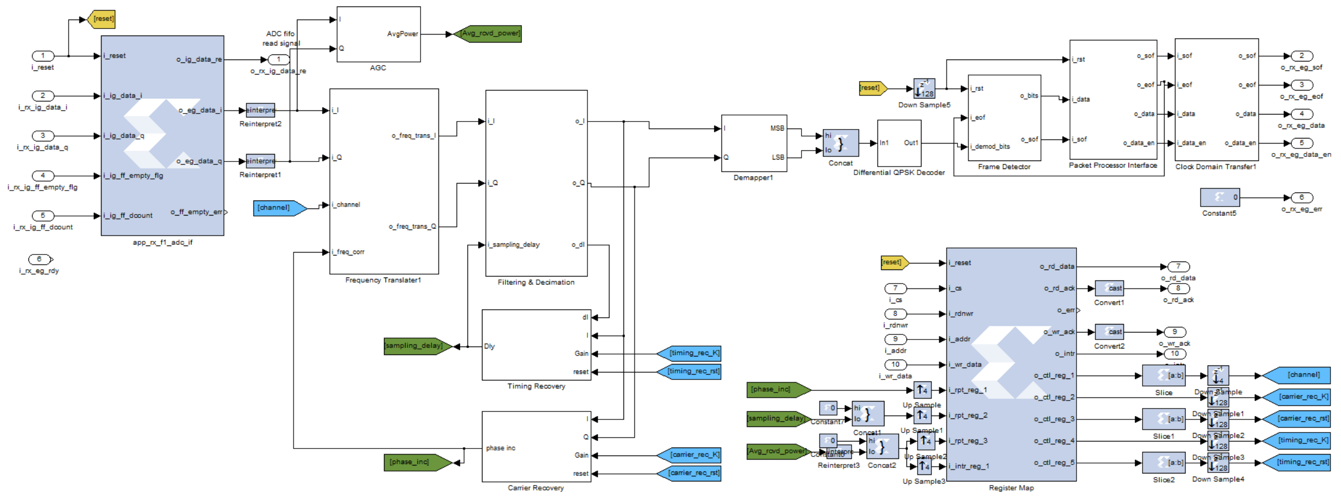

Figure 13: Simulink QPSK Receiver APP

6. CONCLUSION

In this paper, the CRKIT platform was presented as an advanced radio system enabling experimental research in Cognitive Radio and Dynamic Spectrum Access techniques. The powerful combination of wide-tuning range radio and flexible baseband processing was discussed, including the FSoC framework architecture. Here, the concept of Static and Dynamic domain spaces was elaborated, and emphasis was made on APP development for creativity and productivity, whereas framework development for engineering complexity. A system design example based on rendezvous algorithm was presented to illustrate the CRKIT development concepts and integration into ORBIT wireless network emulator.

7. FUTURE WORK

A multi-pronged effort is planned for further development of CRKIT platform. First, the APP library will be extended to include OFDM-based waveforms, this can be used for sub-carrier bandwidth allocation research work. Second, the static framework architecture will be upgraded to support sampling rates and APPs run-time reconfigurability. The goal is to enable network programmable APPs, allowing users to select and upload APPs from library to "live" CRKIT. Futhermore, compute intensive blocks can dynamically be pushed to CRKIT. On the software side, plans to port linux to PCORE at the CRKIT side, as well as fully integrating CRKIT into the ORBIT Management Framework (OMF). This should facilitate CRKIT user friendliness experience and experimentation scalability.

Finally on the hardware side, the radio will be upgraded to being truly wideband capable of 800MHz bandwidth, and up to 1GSPS, 8-bit ADC (per rail) and 1.25GSPS, 16-bit DAC (per rail). In principle, the spectrum sensing function should be able to capture an instantaneous 800MHz band, and with four such radios the platform can capture a 3.2GHz instantaneous band. To support such extreme capabilities, the baseband processing board needs to be upgraded to newer and higher performance FPGA technologies. The current FSoC framework design should be portable to this newer platform.

8. REFERENCES

[1] GNU Radio. `http://gnuradio.org`.

[2] K. Tan, J. Zhang, J. Fang, H. Liu, Y. Ye, S. Wang, Y. Zhang, H. Wu, W. Wang, and G. M. Voelker. Sora: high performance software radio using general purpose multi-core processors. In *Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, pages 75–90. USENIX Association, 2009.

[3] K. Amiri, Y. Sun, P. Murphy, C. Hunter, J. R. Cavallaro, and A. Sabharwal. Warp, A Unified Wireless Network Testbed for Education and Research. In *Proceedings of IEEE MSE*, 2007.

[4] M. C. Ng, K. E. Fleming, M. Vutukuru, S. Gross, Arvin, and H. Balakrishnan. Airblue: a system for cross-layer wireless protocol development. In *Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, 2010.

[5] Bluespec Inc. `http://www.bluespec.com`.

[6] Xilinx. Zynq. `http://www.xilinx.com/products/silicon-devices/epp/zynq-7000/index.htm`.

[7] The VITA Radio Transport (VRT). `http://www.pentek.com/tutorials/17_2/VITA.cfm`.

[8] OpenCores. OpenRisc 1200. `http://www.opencores.org/openrisc,or1200`.

[9] L. A. DaSilva and I. Guerreiro. Sequence-based Rendezvous for Dynamic Spectrum Access. In *Proceedings of IEEE DySPAN*, pages 1–7, 2008.

[10] Z. Lin, H. Liu, X. Chu, and Y.-W. Leung. Jump-Stay Based Channel-Hopping Algorithm with Guaranteed Rendezvous for Cognitive Radio Networks. In *Proceedings of IEEE INFOCOM*, 2011.

[11] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh. Overview of the orbit radio grid testbed for evaluation of next-generation wireless network protocols. *Wireless Communications and Networking Conference*, pages 1664–1669, 2005.