2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines

Reducing Processing Latency with a Heterogeneous FPGA-Processor Framework

Jonathon Pendlum, Miriam Leeser, Kaushik Chowdhury
Dept. of Electrical and Computer Engineering
Northeastern University, Boston MA
Email: {jpendlum, mel}@coe.neu.edu, krc@ece.neu.edu

***Abstract*—Both Xilinx and Altera have released SoCs that tightly couple programmable logic with a dual core Cortex A9 ARM processor. These SoCs show promise in accelerating applications that exploit both the FPGA's parallel processing architecture and the CPU's sequential processing. For example, before accessing a wireless channel, a cognitive radio does spectrum sensing to detect channel occupancy and then makes a decision based on spectrum policies. Spectrum sensing maps well to FPGA fabric, while spectrum decision can be implemented with a CPU. Both algorithms are highly sensitive to latency as a faster decision improves spectrum utilization. This paper introduces CRASH: Cognitive Radio Accelerated with Software and Hardware – a new software and programmable logic framework for Xilinx's Zynq SoC targeting cognitive radio. We implement spectrum sensing and the spectrum decision in three configurations: both algorithms in the FPGA, both in software only, and spectrum sensing on the FPGA and spectrum decision on the CPU. We measure the end-to-end latency to detect and acquire unoccupied spectrum for these configurations. Results show that CRASH can successfully partition algorithms between FPGA and CPU and reduce processing latency.**

***Keywords*—FPGA; Heterogeneous Computing; Data Latency; Software radio; Spectrum Sensing; Cognitive Radio; System on Chip**

I. Introduction

Many applications employ a mix of algorithms that require both sequential and parallel processing. Previous research has found that a heterogeneous computing system [1], [2] can reduce computation time by partitioning algorithms to processing units (CPU, GPU, or FPGA) best suited for the particular workload. System-on-Chips from Xilinx and Altera with both FPGA fabric and ARM processors on a single die show promise in accelerating applications like cognitive radio that utilize both parallel and sequential processing but have tight timing requirements.

Sect. III presents the design of CRASH: Cognitive Radio Accelerated with Software and Hardware, a framework that addresses these challenges on Xilinx's Zynq SoC. CRASH provides the capability to partition designs between processing blocks in the FPGA fabric and user programs running in Linux on the ARM processors.

CRASH targets accelerating the key enabling functions of cognitive radio, an emerging field concerned with alleviating the congested wireless spectrum by using alternate licensed frequencies. Since most cognitive radio functions, including spectrum sensing and spectrum decision, are implanted in software, inherent parallel structures that exist in their execution chains can be easily exploited on FPGAs. Faster spectrum sensing allows the cognitive radio to react quickly to changing spectrum availability. Also, spectrum decision algorithms must have a fast response time to prevent adverse interference with licensed users. The CRASH framework on the Xilinx Zynq SoC enables effective implementation of both these kinds of processing.

This paper makes the following contributions:

- Introduction of the CRASH framework that exploits the Xilinx Zynq's heterogeneous computing characteristics to enable low latency processing with both FPGA and CPU resources.
- Implementation of spectrum sensing and spectrum decision algorithms with CRASH.
- Quantification of the sources of latency, both with and without CRASH.

II. Background

A. Heterogeneous FPGA-CPU Devices

Xilinx and Altera have both released System-on-chips, the Zynq SoC [3] and the Cyclone V SoC [4], that pair programmable logic with a dual core ARM Cortex A9 processor. Both families of devices have similar capabilities in processor speed and FPGA resources. Their processor and programmable logic are interconnected via ARM's Advanced Microcontroller Bus Architecture (AMBA) using the Advanced eXtensible Interface (AXI) [5]; an interface designed for high throughput transfers between the CPU and peripheral devices. The CRASH framework is implemented on the Xilinx Zynq SoC. CRASH uses AXI ACP and a General Purpose AXI Port for communication between the ARM processors and the programmable logic.

B. Cognitive Radio

1) Software Functions and Hardware Platforms: The wireless spectrum is increasingly crowded as more devices gain wireless connectivity. Cognitive radios seek to opportunistically use licensed frequencies without interfering with the licensed or primary users of these frequency bands. This goal is achieved by first measuring the activity on the licensed channels, called as *spectrum sensing*, followed by selecting which spectrum to use among the available options, called as *spectrum decision*.

While various time-domain and frequency domain-based spectrum sensing techniques exist, we focus on the latter,

978-1-4799-5111-6/14 $31.00 © 2014 IEEE
DOI 10.1109/FCCM.2014.13

where a Fast Fourier Transform (FFT) is applied on the sample data stream from the radio front-end [6]. This allows a fine-grained analysis of spectrum usage, as compared to simple energy measurements undertaken across a channel bandwidth.

Spectrum decision may use spectrum sensing data, network topology and routing metrics, adaptive machine learning algorithms, and regulatory policies to decide which vacant channel should be ultimately used. Thus, this step can occur within the front end using a simple metric like *least used channel* or a complex high level policy that runs at the host computer.

Software defined radios serve as the building blocks of cognitive radios, by moving as much signal processing as possible from fixed hardware to software or reconfigurable hardware. In this study, we specifically describe our implementation using the Universal Software Radio Peripheral (USRP) [7] family of software defined radios. The USRP contains the transmitter and receiver circuitry along with a small FPGA to send and receive sample data over Ethernet to the host computer. This host performs the bulk of the processing through a software framework such as GNU Radio.

2) Latency in Cognitive Radios: Spectrum sensing and the spectrum decision involve finite computational time, and there are additional delays introduced by the Ethernet communication between the USRP and the host computer. Any delay in the spectrum decision process can cause poor spectrum utilization and possible interference with other users.

C. The CRUSH Platform

The USRP uses the TCP/IP protocol over Ethernet which adds additional header information to the data stream. This overhead can add up to milliseconds of time. CRUSH [8] reduces the interface latency with the USRP with a direct high speed serial interface to a FPGA board using a custom circuit board and a standard MICTOR cable. It allows the USRP to stream its Analog to Digital Converter (ADC) samples at full rate with no overhead, enabling spectrum sensing with the FPGA fabric. CRASH extends CRUSH by adding the capability to transmit as well as receive, and by adding the capability to stream Digital to Analog Converter (DAC) samples at full rate.

D. Related Work

Other works have targeted the Zynq for heterogeneous computing. [9] pairs the USRP with the Zynq to make a software defined radio. They port the SDR library Iris to the ARM processor, and present methods to identify blocks for potential acceleration in the Zynq's programmable logic.

III. CRASH Framework

CRASH runs under Linux on the ARM processor and uses a kernel device driver to facilitate communication between user programs and FPGA processing blocks. Fig. 1 shows an overview of the CRASH framework.

HDL. CRASH allows the user to place processing blocks in the FPGA fabric, connected to an common AXI-Stream interconnect that can support up to 16 processing blocks in a full crossbar. Each processing block can transfer data at up to 1.2 GB/sec (150MHz x 64 bits). All blocks in the

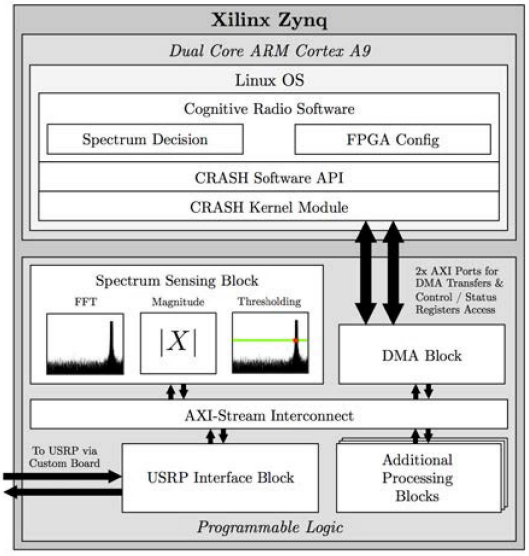

Fig. 1. The CRASH Framework

FPGA fabric operate at 150 MHz. Each block's interface includes both the AXI port and an interface to memory mapped I/O control and status registers, which are accessible by the processor via a dedicated AXI slave port. Control and status registers have an access time in Linux on the order of 100 – 200 ns. The Direct Memory Access (DMA) block shuttles data from other blocks to the CPU over the AXI Accelerator Coherency Port (ACP) bus. Transfers over AXI ACP can directly read and write to the processor cache which maintains cache coherency and avoids the latency of RAM access. User programs in the ARM processor setup DMA transfers between the programmable logic and CPU by configuring the DMA block's control registers. Once configured, the DMA block will initiate memory reads and writes without further intervention.

CRASH sends and receives sample data with the USRP via the USRP DDR interface block. The USRP's ADC and DAC both sample at 100 million-samples/sec and use complex data. Since CRASH runs at a different clock rate than the USRP, this block includes logic and FIFOs to synchronize data between the clock domains, as well as programmable decimation and interpolation filters for increasing or reducing the sample rate of transmit and receive samples. Optional fixed to floating point data conversion is also implemented here.

The Spectrum Sensing block implements spectrum sensing and spectrum decision in single precision floating point. All processing in this block is in single precision floating point. Spectrum sensing uses a variable size FFT followed by magnitude calculation for frequency-domain based energy detection. The spectrum decision algorithm consists of comparing the output of spectrum sensing against a threshold. If the magnitude falls below the threshold, the block communicates to the processor that a transmit decision has been reach.

Table 1 shows CRASH's FPGA resource utilization on a Zynq 7045, including spectrum sensing and USRP DDR interface blocks.

TABLE I. FPGA RESOURCE UTILIZATION OF THE CRASH FRAMEWORK ON THE ZYNQ 7045

Resource	Used	Available	Percent Used
Slice Registers	38,270	437,200	8%
Slice LUTs	29,772	218,600	13%
DSP48s	216	900	24%

Software. In CRASH, the ARM processors run Linux. Linux based libraries such as GNU Radio can therefore be easily run. Linux employs memory protection with virtual memory, preventing unprivileged user programs from directly accessing specific memory address. However, user programs need such addressing capability to access the processing blocks embedded in the FPGA fabric. CRASH uses a kernel driver for communication between the user program and the FPGA processing blocks. After kernel driver initialization, user programs can access control and status registers as an array in memory. The driver provides contiguous memory buffers for DMA transfers and handles interrupts from the FPGA fabric.

Hardware. The hardware used for this paper was: a Xilinx ZC706 development board, USRP N210 used as the radio front end, and the high speed serial interface to the USRP (Fig. 2).

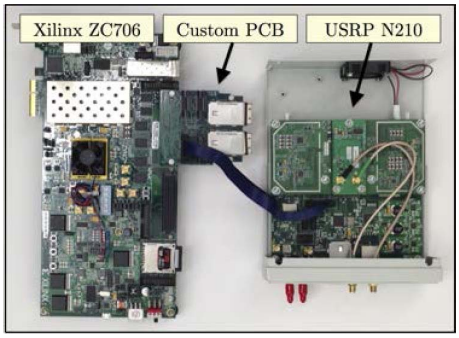

Fig. 2. Hardware used with the CRASH Framework

IV. EXPERIMENTS: EVALUATING PROCESSING LATENCY WITH CRASH

We implemented the Spectrum Sensing and Spectrum Decision algorithms in three configurations:

Exp. 1: Spectrum Sensing and Spectrum Decision in FPGA Fabric. In this experiment, the USRP DDR Interface block receives samples and routes them via the AXI-Stream interconnect to the Spectrum Sensing block, which performs both spectrum sensing and the spectrum decision. When the block determines the spectrum is unoccupied, it triggers the USRP DDR Interface block to send a buffered transmit waveform to the USRP.

Exp. 2: Spectrum Sensing in FPGA Fabric, Spectrum Decision in ARM. Here, sample data enters CRASH through the USRP DDR Interface block and is transferred to the Spectrum Sensing Block. However, the output of the FFT and magnitude calculation pipeline is transferred to the ARM processor via the DMA block for thresholding. This moves *spectrum decision* to the ARM processor. After it is determined that the spectrum is unoccupied, the ARM processor triggers the USRP DDR Interface block to transmit the waveform.

Exp. 3: Spectrum Sensing and Spectrum Decision in ARM. Here the FPGA fabric simply interfaces with the USRP and performs filtering. This experiment provides data to compare the processing time of spectrum sensing on the ARM processor versus the FPGA fabric.

A. Measuring Processing Latency

We decided to treat the system as a black box and measure the *turnaround time* between providing a spectrum hole and the cognitive radio transmitting, as shown in the oscilloscope display in Fig. 3. The turn around time has these qualities: 1. It is a "black box" measurement independent of the experiment configuration. 2. It is the aggregation of many latencies including delay through the USRP, transferring data from the USRP to CRASH, filtering in the USRP DDR Interface block, spectrum sensing, the spectrum decision, and transferring data from the FPGA fabric to the ARM processors. 3. It is a valuable indicator of responsiveness of the cognitive radio. Turnaround times of 10-30 μs indicates the system could meet 802.11 MAC layer timing requirements – a desirable trait in a cognitive radio.

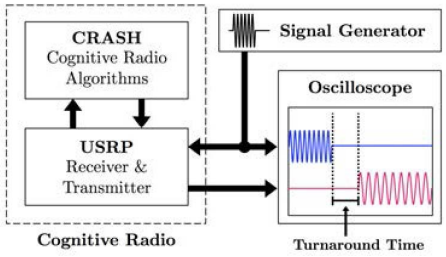

Fig. 3. Measurement of Turnaround Time

B. Measurement of Individual Latencies in CRASH

Three methods are used to breakdown the turnaround time into the individual component latencies: Clock counting, program timers, and RX/TX loopback. Clock counting uses Xilinx's Chipscope to observe internal FPGA signals and count clock cycles of events within the FPGA fabric. . For processing on the ARM, system timers are used to measuring execution time. To determine the USRP delay, the receive sample data is looped back to the transmit path and measured using the same technique as the overall system delay.

V. RESULTS

For each of the three configurations, turnaround times were measured while varying the FFT size. Fig. 4 plots the average turnaround times. The standard deviation is an artifact of the employed spectrum sensing algorithm: frequency-domain energy detection. The signal generator in our experiment transmits the RF pulse independently of CRASH's FFT sample window. Therefore, the sensed spectrum energy will vary depending on when CRASH began buffering samples relative to the RF pulse.

As shown in Fig. 4, as FFT size increases so does the turnaround time. This agrees with theory, as the FFT execution time grows at $O(\log N)$ in the parallel implementation and

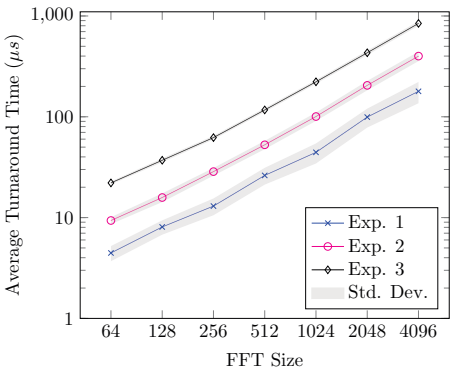

Fig. 4. Average Turnaround Time for each Experiment

$O(N\log(N))$ with sequential processing. Second, the latency increases as less work is performed in the FPGA. Experiment 1 has the best performance – 120% faster than experiment 2 and 300-400% faster than experiment 3. Offloading spectrum sensing to the FPGA fabric (experiment 2) versus executing both algorithms in software (experiment 3) is approximately 120% faster. Finally, the larger turnaround time in the third configuration relative to the other two illustrates the performance degradation when performing spectrum sensing in software versus reconfigurable hardware.

TABLE II. AVERAGE LATENCY OF INDIVIDUAL COMPONENTS FOR EACH EXPERIMENT

Latency Component	Exp. 1	Exp. 2	Exp. 3
USRP RX & TX Paths	0.6 μs	0.6 μs	0.6 μs
USRP - CRASH Interface	0.7 μs	0.7 μs	0.7 μs
Spectrum Sensing			
64 Point FFT	2.6 μs	2.5 μs	13 μs
4096 Point FFT	149 μs	149 μs	747 μs
Spectrum Decision			
64 Point FFT	0.8 μs	3.3 μs	2.0 μs
4096 Point FFT	28 μs	203 μs	121 μs
FPGA-CPU DMA Delay			
64 Point FFT	N/A	2.3 μs	2.2 μs
4096 Point FFT	N/A	34 μs	41 μs
Total			
64 Point FFT	4.7 μs	9.4 μs	19 μs
4096 Point FFT	178 μs	387 μs	910 μs

CRASH's Effect on Latency. Table II presents a breakdown of the individual component latencies. The USRP RX & TX Path delays show a lower bound of approximately 600 ns for turnaround time – where the RF input to the USRP is looped back for immediate transmission. For the three experiments, the majority of the overall delay (60-95%) comes from spectrum sensing and spectrum decision. Conversely, in experiments 2 and 3 the DMA transfers only contributes 5-25%. This is a favorable outcome as we want to maximize the time performing computations and minimize the time spent transferring data. The performance of experiment 3 suffers the most compared to the first two. Even with the use of FFTW with optimizations, spectrum sensing in experiment 3 contributes an overwhelming 70-80% of the overall latency. This exemplifies the need to accelerate certain functions in the FPGA fabric. By offloading spectrum sensing, the time spent spectrum sensing is reduced to 25-40%.

CRASH's Performance as a Cognitive Radio A turnaround time of less than 30 μs is desirable to meet 802.11 MAC layer timing requirements [11]. From Fig. 4, we can see that all three experiments can meet that goal for a FFT size of 64, but only experiments 1 and experiment 2 can meet that goal for FFT sizes of 128 and 256. This shows that a cognitive radio utilizing the CRASH framework could implement a sophisticated spectrum decision algorithm in the ARM processor and still make strict timing requirements.

VI. CONCLUSIONS AND FUTURE WORK

CRASH is a versatile heterogeneous computing framework for the Xilinx Zynq SoC. Our experiments show that CRASH can successfully segment two cognitive radio algorithms, spectrum sensing and the spectrum decision, between the Zynq's FPGA fabric and ARM processors. More importantly, turnaround times specified in the 802.11 specification can be met while providing the designer with flexibility in implementing important processing components.

The CRASH framework is generic enough to be used in many applications requiring a mix of FPGA acceleration and CPU processing. In the future we plan to exploit the platform's low latency processing capabilities to study Cognitive Radio algorithms when operating with other 802.11 wireless devices. We also plan to expand the platform to support easily moving processing blocks between software and reconfigurable hardware, possibly incorporating run-time reconfiguration.

REFERENCES

[1] M. Reichenbach, R. Seidler, and D. Fey, "Heterogeneous computer architectures: An image processing pipeline for optical metrology," in *Reconfigurable Computing and FPGAs (ReConFig)*. IEEE, 2012, pp. 1–8.

[2] P. Meng, M. Jacobsen, and R. Kastner, "FPGA-GPU-CPU heterogenous architecture for real-time cardiac physiological optical mapping," in *Field-Programmable Technology (FPT)*. IEEE, 2012, pp. 37–42.

[3] Zynq-7000 all programmable soc overview. Available at: http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf.

[4] Cyclone V device overview. Available at: http://www.altera.com/literature/hb/cyclone-v/cv_51001.pdf.

[5] Xilinx axi reference guide. Available at: http://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf.

[6] T. Yucek and H. Arslan, "A survey of spectrum sensing algorithms for cognitive radio applications," *Communications Surveys Tutorials, IEEE*, vol. 11, no. 1, pp. 116–130, 2009.

[7] Universal software radio peripheral n series product overview. Available at: http://www.ettus.com/content/files/kb/Ettus_Networked_Series.pdf.

[8] G. Eichinger, K. Chowdhury, and M. Leeser, "Crush: Cognitive radio universal software hardware," in *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*. IEEE, 2012, pp. 26–32.

[9] J. van de Belt, P. D. Sutton, and L. E. Doyle, "Accelerating software radio: Iris on the zynq soc," in *Very Large Scale Integration (VLSI-SoC)*. IEEE, 2013, pp. 294–295.

[10] A. Filgueras, E. Gil, C. Alvarez, D. Jimenez, X. Martorell, J. Langer, and J. Noguera, "Heterogeneous tasking on smp/fpga socs: The case of ompss and the zynq," in *Very Large Scale Integration (VLSI-SoC)*. IEEE, 2013, pp. 290–291.

[11] I. C. S. L. M. S. Committee *et al.*, "Wireless lan medium access control (mac) and physical layer (phy) specifications," 2012.