High Throughput Channel Tracking for JTRS Wireless Channel Emulation

Dajung Lee*, Janarbek Matai†, Brad Weals‡, Ryan Kastner†
*Electrical and Computer Engineering, †Computer Science and Engineering
University of California, San Diego
‡Toyon Research Corporation
Email: {dal064, jmatai, kastner}@eng.ucsd.edu*†, bweals@toyon.com‡,

***Abstract*—Testing and verifying wireless systems in a real world environments is a challenging but an important problem. This is particular true for the Joint Tactical Radio System (JTRS) where the modulation techniques are optimized towards environments that are difficult to reproduce (e.g., ship to plane, plane to satellite communications). Such cases necessitate a wireless channel emulator to facilitate testing in the laboratory as the protocols are being developed. Furthermore, the increasing complexity of communications protocols and highly variable network scenarios force the channel emulator to support an accurate and complicated channel model that can scale to handle a large number of radios that operate across a wide frequency spectrum. We developed a unique channel impairment emulator prototype to meet these requirements. It maximizes the scalability and performance, operating in a frequency range of 2 MHz to 2 GHz. Moreover, our emulator design accommodates radio operation that use unknown frequency hopping techniques, which is increasingly common in JTRS systems. This key feature to this system is a high throughput channel tracker module that handles high bandwidth intermediate frequency (IF) signals while providing the scalability to handle a large number of channels.**

I. Introduction

Evaluating a wireless system in an early stage of development is crucial for the success of the system. The performance of wireless network significantly depends on physical environment, which is affected by various factors. Moreover, as wireless devices become more common in our lives, the network scenarios in which they are used are increasingly congested and complicated. Researchers have to find a solution to evaluate the wireless systems under development in an efficient and effective manner. Field-testing provides a real user environment, but it is expensive and almost impossible to create repeatable tests. As such, channel emulators are used as a virtual platform to simulate radio environment for testing in laboratory. A channel emulator has a number of challenging requirements in order to accurately construct a complete and realistic wireless network.

The Joint Tactical Radio System (JTRS) is a standard for software defined radio (SDR) systems. It has an advantage of implementing a wide range of communication schemes in a flexible manner. The standard operates over the 2 MHz to 2 GHz frequency band and provides next generation voice, video, and data links.

Our system was built to support the demands of US Navy applications for military scenarios based on SDR. They required an extensible and programmable wireless channel emulator for evaluating their current and future JTRS network systems quickly. The requirements specified in two ways : (1) the emulation of RF channels for a minimum of 8 full duplex radios with the ability to expand to 100 full duplex radio nodes and (2) the operation across 250 MHz bandwidths while capable of covering the 2 MHz throughout 2G Hz spectrum to support frequency hopping. To meet these requirements we developed a software defined radio channel emulator using an FPGA. We focused on the design of a single FPGA system that can handle 8 radios. However, the system was designed in a modular fashion, and it can be expand to handle additional radios using multiple channel emulator modules connected via a high speed digital link. In addition, JTRS waveforms often switch their carrier frequency randomly among many frequency channels. We need a *channel tracker* which enables our channel emulator to quickly detect signals across the large 2 MHz to 2 GHz band. The *channel tracker* detects the frequency hopping by analyzing its power spectrum using a correlation-based linear prediction method. The rest of this paper will focus on describing the entire system operation and the detailed hardware architecture of the channel tracker.

We designed the channel tracker using a state-of-the-art high-level synthesis tool, Xilinx Vivado HLS, which allowed for quick design space exploration and the agility to interface with changing demands from other parts of the system.

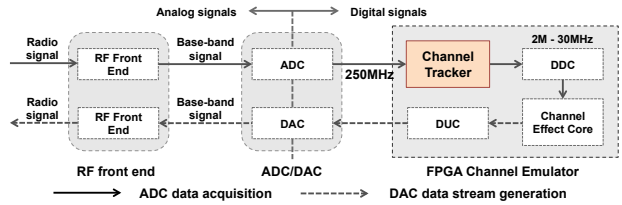
In this paper, we describe the channel emulator requirements; we present our hardware design process using HLS tools; and we provide a thorough design space exploration using a variety of different optimization methods. The primary contributions of our work are:

- Providing a system overview for a highly scalable channel emulator including the interface between the analog and digital components, and its network model to expand to larger system.
- Designing an optimized hardware module to track a carrier frequency in given signal spectrum based on linear prediction for high resolution spectral analysis.
- Developing a methodology for creating HLS designs with high throughput constraints.

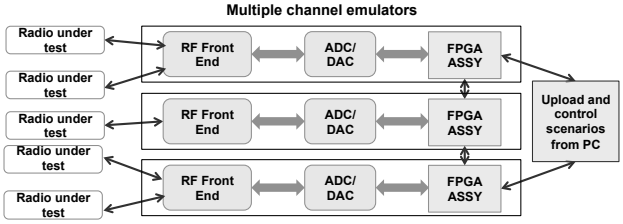
The rest of this paper is organized as follows: We review related work in Section II. Section III provides an overview of the channel emulation system. Section IV presents a hardware design of the channel tracker focusing on two optimization approaches targeting throughput and latency. Section V shows the results, and we conclude in Section VI.

II. Related Work

There are several projects related to developing wireless channel emulators. Borries et al. presents a channel emulator for 210 independent channel models between 15 nodes, but they limits their emulation model in a specific frequency range only for 802.11 wireless radios [1]. Eslami et al. compares their FFT-based emulator in frequency domain to a



(a)



(b)

Fig. 1. We employ an off-the-shelf board from Pentek, which has ADCs, DACs, and an FPGA. (a) The channel tracker receives data from the ADC and informs the digital down converter (DDC) to current location in the spectrum of the signal. This portion of the spectrum is then feed into a core that simulates the channel. The resulting data is feed into a digital up converter (DUC) which is provided to the DAC. (b) An extended system of chained multiple channel emulators with a host computer. This larger system is controlled by the host PC, and channel emulators are connected with high speed digital links: Xilinx Aurora

conventional approach based on finite-impulse response (FIR) filters in time domain [2]. They suggest a frequency domain analysis to handle the complexity of temporal analysis in high order network arrays. However, FFT analysis depends on the number of sample points in FFT, which limits the flexibility in the frequency range of analysis. Buscemi et al. presents a cluster-based wireless channel emulator prototype in [3]. They decentralize computations of networks to minimize the complexity in previous wireless channel emulators. Based on this research, [4] introduces an expanded system and a scaling study using FPGA clusters to support 1250 wireless devices.

We employ a statistical channel model making our design is more scalable than previous works and flexible to accommodate broader bandwidths. The key feature of our system, a high throughput *channel tracker* analyzes streaming data to find its carrier frequency in the frequency hopping condition without prescheduling and delivers them to the subsequent modules, which is used as a preprocessing step for our previous work of channel emulator core design [5].

III. System Overview

We employed a commercial off-the-shelf digital signal processor board, Pentek Cobalt PCIe hosted reference board. Its key features are including a Xilinx Vertex 6 series FPGA (XC6VSX315TFF1156-2), two 500 MHz analog to digital converters (ADCs), two digital down converters (DDCs), two 800 MHz digital to analog converters (DACs), and a digital up converter (DUC) with PCI Express (PCIe) interface.

Figure 1(a) describes an end-to-end flow for a single path of simulated channel in our channel emulator. It is divided into two path ways: ADC data acquisition for the transmit path, and DAC data stream generation for the receive path. In data acquisition path, an RF module takes radio signals and converts them into a base band signal. An ADC module converts these analog signals into digital data stream running at 250 MHz. A *channel tracker* module in front of DDC generates control signals for DDC by analyzing this 250 MHz data stream. A DDC module accepts and downconverts it to run in a channel effect core which operates at much lower rate. The emulated signals from the output of the channel effect core are converted into final radio output signals in the DAC data stream generation path. A DUC translates them to an intermediate frequency (IF) signal (250 MHz), and a DAC converts these high sample rate signals to analog ones.

We can expand our prototype into a larger system by making connections between multiple channel emulators. The platform consists of a host PC for control unit and multiple FPGA DSP boards. The enclosure system has eight PCIe slots for FPGA boards. Each single board is for one channel emulator, and one channel emulator can connect equal or more than eight test radios. Multiple boards can communicate to each other through directly connected synchronous interfaces outside of PCIe and act as one. We use Xilinx Aurora interface to link them in a ring arrangement. They transfer downconverted signal data in 30 MHz streaming along these connections. The host PC controls this network using Ethernet.

IV. Channel Tracker

The *channel tracker* module focuses on accepting accepts high throughput data stream and detecting the presence of a signal somewhere within the data. Intermediate frequency (IF) signals, input to the *channel tracker* module come from the ADC. The ADC and the FPGA assembly share common clock resources, and the FPGA part can accept this high sample rate data directly from the DAC.

The *channel tracker* informs the programmable DDC module how to convert the 250 MHz signal to a 30 MHz signal which is easier for the channel effect core to process. There are many small bandwidth slots with different center frequencies in the initial bandwidth of 250 MHz. The data is capable of changing its carrier frequency over 100 MHz between the range of 3 MHz to 245 MHz.

A. Spectral Analysis Algorithm

Our goal is to detect a carrier frequency of incoming signal that is hopping without prescheduling. We use a linear prediction method to predict the current power spectrum by calculating the correlation values and a linear combination of sample streams in a current window. A Fast Fourier transform (FFT) analysis is more frequently used for power spectral analysis since it is more intuitive and understandable in frequency domain. However, modern power spectral analysis uses a linear prediction method to handle high resolution spectrum. It is more accurate and makes less complex FPGA design because it is a window filtering based operation in time domain.

The *channel tracker* module makes an average of correlation values in the window and estimates an angular frequency by calculating an arctangent of averaged complex number. It is transferable into a linear frequency by multiplying a constant ($2\pi f = w$). This output value will be used to control the DDC. The input signal includes noise and will switch between active and inactive modes periodically. The *channel tracker* module

Algorithm 1 Latency Optimization

while $Input$ **do**
  $temp_1 \leftarrow task1(Input, temp_1)$
  if $Condition1(temp_1)$ **then**
    $temp_2 \leftarrow task2(Input, temp_1, temp_2)$
    if $Condition2(temp_2)$ **then**
      $temp_3 \leftarrow task3(Input, temp_1, temp_2, temp_3)$
      if $Condition3(temp_3)$ **then**
        $temp_4 \leftarrow function1(Input, temp_4)$
        $Output_1 \leftarrow function2(temp_4)$
        $Output_2 \leftarrow 1$
      else
        $temp_4 \leftarrow function1'(Input, temp_4)$
        $Output_1 \leftarrow previous\ Output_1$
        $Output_2 \leftarrow 0$
      end if
    end if
  end if
end while
return $Output_1, Output_2$

Algorithm 2 Throughput Optimization

while $Input$ **do**
  Module a($Input$){
  $temp_{1-1} \leftarrow task1_1(Input)$
  $temp_{1-2} \leftarrow task1_2(Input)$
  }
  Module b($temp_{1-1}$){
  $temp_2 \leftarrow function1(temp_{1-1}, temp_2)$
  $Output_1 \leftarrow function2(temp_2)$
  }
  Module c($temp_{1-2}$){
  $Output_2 \leftarrow task_{1+2+3+\alpha}(temp_{1-2})$
  }
end while
return $Output_1, Output_2$

calculates the signal power to choose the valid signal and reject transmit energy from the receive path.

B. Hardware Implementation

The DDC module has complex signal inputs, in-phase(I) and quadrature (Q), coming from the ADC, and output narrows down them into 30 MHz. It is controlled by a set of control registers, mainly tuned by two parameters: tuning frequency (center frequency) and decimation rate (sampling frequency). The *channel tracker* module taps two input ports and analyzes them to estimate the carrier frequency, which decides one of these two parameters, the tuning frequency. The control registers have another link from PCIe interface, and it is used to make a direct communication between a host PC and channel emulator.

We used HLS tools to perform design space exploration and implement our hardware design. We targeted two different optimization approaches, *latency optimization* and *throughput optimization*.

Latency optimization is similar to a single threaded implementation (**Algorithm 1**). In this structure, if conditions and parameters are nested over several layers with significant data dependency. The innermost functions, $function1$ and $function2$, generate final outputs, but run only when it holds several conditions in the behavior analysis. It would be more efficient to reserve incoming data and calculate them when it is necessary based on given conditions. However, this dynamic operation makes scheduling difficult in hardware design that requires high throughput performance.

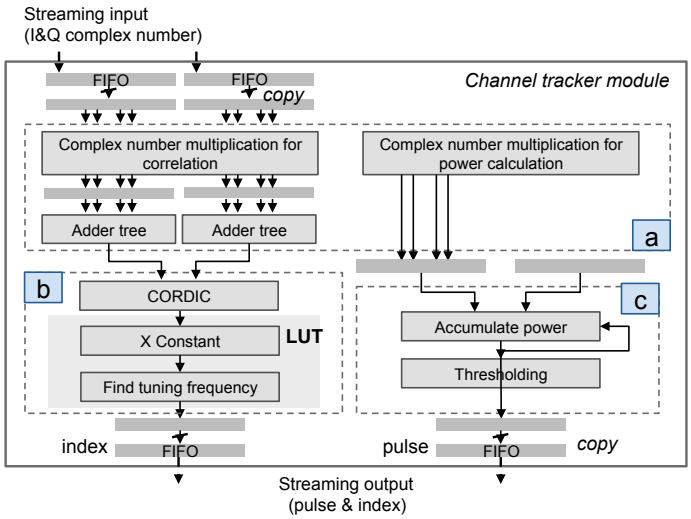


Fig. 2. A diagram of the *channel tracker* module. Part (a) performs correlation and power analysis through complex number multiplication. Correlation also requires a summation of the multiplied signals. Part (b) uses *CORDIC* to determine the angular frequency and a *finding tuning frequency* module that generates a 6 bit index value corresponding the linear frequency. Part (c) calculates the power of the signal and is used as a threshold in order to reject cases when there is noise.

For the high throughput operation, the computational units should run concurrently: in a parallel or pipelined manner. *DATAFLOW* is one of pragmas Vivado HLS provides to optimize and specify it, particularly pipelining in functional level. **Algorithm 2** shows the transformed structure of **Algorithm 1** for pipelined design to use *DATAFLOW* pragma. This altered design minimizes data dependency and rearranges the procedures to force them run in parallel. The operation of key parts does not depend on the if conditions. We utilize $Output_2$ signal in **Algorithm 1** as a flag. All of the modules are run regardless of the condition, and the $task_{1+2+3+\alpha}()$ module generates a flag that indicates when the output is valid.

To achieve the target performance in pipelined design, the clock frequency should be at least 250 MHz in *channel tracker* in order not to miss any samples, and the modules should run in pipelined way to keep up with the inputs. The module with the largest latency decides the overall performance in pipelined operation. Our optimization process focuses on minimizing the clock period while iteratively optimizing the module with the longest latency. Figure 2 shows all operations of *channel tracker* in detail. We assigned twenty samples per window in our experiment, and the system pipeline is also based on the window size.

1) Module a (Complex multiplications): Two FIFO memories accept I&Q streaming inputs and transfer them to partitioned buffers.The partitioned buffer in HLS is a set of registers, and each of these elements can be accessed simultaneously. That makes the first module for complex multiplications accesses these registers and calculations correlations in parallel. They are averaged by adder trees and the final output is translated into angular frequency by *arctangent* operation.

2) Module b and Module c: Mathematically, the arctangent result is converted into the linear frequency value, and it finds

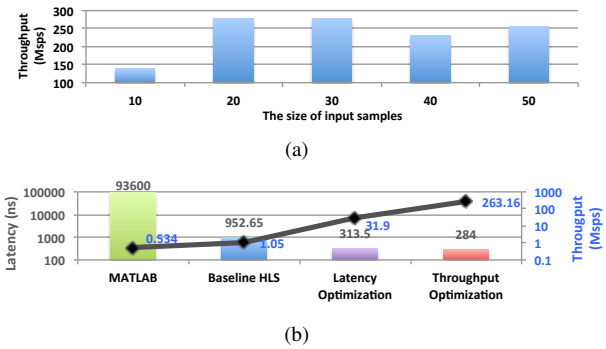


Fig. 3. (a) A pattern of throughput with the different size of input samples (b)A comparison of performances in different implementations: MATLAB, Baseline, *Latency Optimization*, and *Throughput Optimization*

its best matching frequency from a fixed frequency table. This index is a final output, *index* in Figure 2. We optimized this searching process by implementing our own CORDIC module and using a LUT. CORDIC produces one digit output, 1/0 for $+/-$, in every iteration while minimizing the precision error. We used this output digit sequence as an address of the LUT that provides the 6 bit index of the frequency table. The frequency table is another LUT to translate a 6 bit number into 32 bit data corresponding to the center frequency. It sets up a tuning frequency to control the DDC. Also, this module produces the power sum to make a pulse, which will be used as an enable signal in control registers.

V. Results

We synthesized our RTL design using Xilinx Vivado HLS 2012.3 and implemented using ISE 14.3. We focused on two things when optimizing the hardware architecture to achieve high throughput: *minimizing the bottleneck module latency* and *maximizing the clock frequency*.

We can express the final throughput of entire pipelined system as

$$(throughput) = \frac{w}{c \times f(w)} = \frac{w}{c \times (w-1)} \quad (1)$$

where w is the number of samples in one window, c is a clock period achieved, and $f(w)$ is the number of clock cycles in a bottleneck module, which is a function of size of the window and decides the performance of whole system. In HLS design, buffering needs at least $w-1$ cycles, which sets the minimum latency for the overall design. We have optimized all of the other modules to achieve this minimum latency.

In our design we set $w = 20$, and therefore our target is to achieve the latency in each module less than 19 cycles. The modules copying inputs and outputs from FIFOs to buffers and vice versa takes 19 cycles (*Input buffering* and *Output buffering* modules). It takes 10 cycles for complex number multiplication and adder tree operations (*Module a*) and 18 cycles for CORDIC operation and searching from frequency table index (*Module b*). Accumulating power values for energy thresholding spends 19 cycles (*Module c*). All modules are optimized to accomplish the 250 MHz clock frequency. As a result, *channel tracker* module achieves 250 Msps throughput running in 250 MHz, and its total latency is 71 clock cycles or 284 ns.

Figure 3(a) shows a pattern of throughput results with the different size of inputs. The size of the buffer in theory effects the accuracy of the tracker. However the buffer size also changes the performance. The buffering module is the bottleneck. Therefore, the throughput keeps around 250 Msps with the larger number of input samples. The 40 and 50 input sample throughput is lower than 20 and 30 due to a larger clock period. Also, as the window size is smaller, buffering module does not decide the overall performance anymore, and *Module b* would be the bottleneck that spends 18 cycles of latency. Therefore, only for the case of the small window, 10, it shows the different throughput, 138 Msps.

Figure 3(b) shows the performance results from different software and hardware designs. The *throughput optimization* is superior compared to all others in terms of throughput. When comparing the latency results, the *latency optimization* design could show less clock cycles, but it takes more latency than *throughput optimization* on average because the bottleneck module behavior depends on conditional statements.

TABLE I
Area usages of the difference implementations: baseline HLS, latency design, throughput design

	Baseline HLS	Latency Optimization	Throughput Optimizatoin
SLICE	2613	1873	940
LUTs	6299	4914	2104
FFs	8081	5860	2338
DSPs	51	33	112
BRAMs	4	4	3

VI. Conclusion

Wireless channel emulators are an attractive way to test radio systems under complex scenarios. However, the increasing complexity of wireless systems forces significant performance constraints on the emulator. In particular, the operating frequency of the radios can span a width range making a high throughput channel tracker a key component for digital channel emulation systems. We implemented a programmable and extensible channel emulator based on an FPGA hardware. It can track the digital baseband data stream in 250 MHz to accommodate the wide band frequency hopping. For high resolution power spectral analysis, we estimate the power spectrum based on linear prediction method. This module is the most challenging part for a real-time system, which required design space exploration to accomplish the target throughput.

References

[1] K. C. Borries, G. Judd, D. D. Stancil, and P. Steenkiste, "Fpga-based channel simulator for a wireless network emulator." in *VTC Spring*, 2009.

[2] H. Eslami, S. V. Tran, and A. M. Eltawil, "Design and implementation of a scalable channel emulator for wideband mimo systems," *Vehicular Technology, IEEE Transactions on*, vol. 58, no. 9, pp. 4698–4709, 2009.

[3] S. Buscemi and R. Sass, "Design and utilization of an fpga cluster to implement a digital wireless channel emulator," in *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*. IEEE, 2012, pp. 635–638.

[4] S. Buscemi, W. Kritikos, and R. Sass, "A range and scaling study of an fpga-based digital wireless channel emulator," in *Field-Programmable Custom Computing Machines (FCCM), 2013 IEEE 21st Annual International Symposium on*. IEEE, 2013, pp. 137–144.

[5] J. Matai, P. Meng, L. Wu, B. T. Weals, and R. Kastner, "Designing a hardware in the loop wireless digital channel emulator for software defined radio," in *FPT*, 2012, pp. 206–214.