Designing a Hardware in the Loop Wireless Digital Channel Emulator for Software Defined Radio

Janarbek Matai, Pingfan Meng, Lingjuan Wu, Brad Weals, and Ryan Kastner
Department of Computer Science and Engineering,
University of California, San Diego
Toyon Research Corporation

December 11
FPT’2012
Motivation

- **Problem**
  - Wireless system testing and verification → Difficult

- **Current wireless system testing methods**
  - *Field testing* → Expensive, time consuming and difficult to repeat
  - *Simulation* → limited by fidelity, excessive run time

- **Wireless Channel Emulator (WCE)**
  - Fills the gap left between simulation and field testing
  - Repeatability, high-fidelity, and the opportunity to test complete radio
  - Software implementation is not feasible due to amount of computation for large network
Hardware in the Loop Wireless Channel Emulator (WCE)

- Hardware in the loop (HWIL) **Wireless Channel Emulator (WCE)** implementation on an **FPGA** platform is purposed using **High-Level Synthesis** Tool.

![Top level concept of WCE](image)
Scenario

The signal takes multiple paths each with a different Path delay and Path gain.
Conceptual Multipath Channel Modeling in FPGA

- Each channel composed of multiple summed paths
- Delay, path gain, fading, and Doppler modeled in each path
- In this work, we focus on implementing a single channel emulator with multi path
Single Channel Emulator Model

- **Single channel emulator model**
  \[ s_{o_t} = \sum_{i=0}^{n} s_{i_{t-i\tau}} w(t)_i \]
  - \( s_i \): Previous \( n+1 \) input complex samples
  - \( s_o \): Complex sample output at present time
  - \( w(t) \): Dynamically changing set of weights (gain and delay)

- **Complex sample calculation (channel function)**
  \[
  \text{tapline}_{ij} = \text{delayline}_{index}
  \text{taps}_i = \sum_{i,j=0}^{N, K} \text{tapline}_{ij} \cdot \text{weight}_{ji}
  \text{so_r}_i = \sum_{i=0}^{N} \text{gains}_r \cdot \text{taps}_r_i - \text{gains}_i \cdot \text{taps}_i_i
  \text{so_i}_i = \sum_{i=0}^{N} \text{gains}_r \cdot \text{taps}_r_i - \text{gains}_i \cdot \text{taps}_r_i
  \]
  - Path delays
  - Path gains
  - channelFunction

Background
Target performance

- pathDelays and ChannelFunction: 30 Mhz
- pathGains: 40kHZ
High Level Synthesis

- High Level Synthesis
  - Creates an RTL implementation from C level source code
- Why use HLS?
  - A good digital WCE has to handle a wide range of dynamically changing parameters such as Doppler effect, fast fading, and multipath
  - HLS provides easy design space exploration with different parameters
    - E.g., varying number of paths in a channel

Courtesy to Xilinx
Process of WCE design with HLS

1. **Baseline design**
   - Synthesizable C code

2. **Restructured design**
   - Manually optimizing C code for HW

3. **Bit accurate design**
   - Bit-width optimization

4. **Pipelining, Unrolling and Partitioning (PUP)**
   - Parallelizing computation
1. Baseline design

- **Main goal:**
  - Synthesizable

- **Things to be done**
  - Matlab → Initial C
  - Initial C → Optimized C
  - Remove dependencies
  - Remove dynamic memory...

![Performances Chart](image)
1. Baseline - Results

- **Baseline**
  - PathDelays: 3180X slower than target (30 Mhz)
  - PathGains: 17X slower than target (40 kHz)
2. Restructured Design

- Two goals:
  - To optimize the code itself without using any HLS pragmas
  - To write a “C code” targeting the architecture
- E.g.,
  - Loop merging
  - Expression balancing
  - Loop unrolling
  - ...
2. Restructured Design - Example

- Clock cycle reduction of pathGains module
  - Loop merging
    ```c
    for (int p = 0; p<N; p++){
        for(int i=0;i<SIZE; i++){
            t1=t1+js[i]*FIRin[p][i][0];
        }
    }
    ```
    ```c
    for (int p = 0; p<N; p++){
        for(int i=0;i<SIZE; i++){
            t1=t1+js[i]*FIRin[p][i][0];
            t2=t1+js[i]*FIRin[p][i][1];
        }
    }
    ```
    50215 → 41250

  - Expression balancing
    ```c
    t1=t1+js[i]*FIRin[p][i][0];
    t2=t1+js[i]*FIRin[p][i][1];
    ```
    41250 → 29730

  - Loop unrolling
    ```c
    for (int p = 0; p<N; p++){
        for (int i = 0;i<SIZE; i++){
            for (int j = 0;j<SIZE2; j++){
                FIRinputs[p][i][j] = ...
            }
        }
    }
    ```
    29730 → 20785
2. Restructured Design – Results

- Restructured design vs. Target
  - PathDelays: 523X slower than target (30 Mhz)
  - PathGains: 7X slower than target (40 kHz)
  - ChannelFunction: 229X slower than target (30 Mhz)
3. Bit accurate design

- By default, HLS C/C++ have standard types
  - E.g., char (8-bit), int (32-bit), ...
- Minimizing bit widths will result in smaller & faster hardware
  - E.g., ap_fixed and ap_int
- Bit accurate design of PathGains module
  - 55 types are set to use fixed point type

```c
typedef ap_fixed<8, 4, AP_RND > AP_FIXED8_4;
typedef ap_fixed<15, 5, AP_RND > AP_FIXED15_5;
typedef ap_fixed<22, 16, AP_RND > AP_FIXED22_16;
typedef ap_fixed<18, 16, AP_RND > AP_FIXED18_16;
typedef double ap_fixed<31, 1, AP_RND > AP_FIXED31_1;
typedef double ap_fixed<27, 1, AP_RND > AP_FIXED27_1;
typedef double ap_fixed<24, 1, AP_RND > AP_FIXED24_1;
typedef double ap_fixed<21, 1, AP_RND > AP_FIXED21_1;
typedef double ap_fixed<18, 1, AP_RND > AP_FIXED18_1;
```
3. Bit accurate design - Results

- Bit accurate design vs. Target
  - PathDelays: 47X slower than target (30 Mhz)
  - PathGains: 3X slower than target (40 kHz)
  - ChannelFunction: 133X slower than target (30 Mhz)
4. Pipelining and Partitioning

- On top of bit accurate design, PUP is applied

Pipeline

- Improves throughput
- Default: Target initiation interval (II) of 1
  II=2, II=3,...

Partition

- BRAMs limit pipelining → Partition large BRAMs into smaller BRAMS or into registers
4. Pipelining and Partitioning - Example

- **pathDelays**
  - Optimizations: Partition: 5 BRAM, Pipeline: II=1
  - DSP48: 19 → 30 (57%),
  - FF: 424 → 786 (85%),
  - LUT: 563 → 4230 (651%)
  - Throughput: 47X than bit accurate design

- **pathGains**
  - Optimizations: Partition: 12 BRAM of 42, Pipeline/Unroll
  - DSP48E: 47 → 86 (82%),
  - FF: 20399 → 34421 (68%),
  - LUT: 22121 → 38893 (75%)
  - Throughput: 15X than bit accurate design
4. Pipelining and Partitioning - Results

- Bit accurate design vs. Target
  - PathDelays 0.6X slower than target (30 Mhz)
  - PathGains 0.2X slower than target (40 kHz)
  - ChannelFunction 1.1X slower than target (30 Mhz)
Final Results

Design Space Exploration

PathDelays
-106X
-17X
-1.5X
47X
Baseline (ns)
Restructured (ns)
Bit-width (ns)
PUP (ns)
SW Optimized (ns)

PathGains
-13X
-5X
-2X
6.4X
-4.8X
-2X

ChannelFunction
42X

NS (log scale)
DSE of WCE for Different Number of Paths

- HLS allows easy DSE of WCE for different parameters
- E.g., number of paths 1, 2, 3, 4, 5
Results for Five Paths

Resource (xc6vlx240t)

<table>
<thead>
<tr>
<th>PathDelays</th>
<th>Slices</th>
<th>LUT</th>
<th>FF</th>
<th>DSP48E</th>
<th>BRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PathGains</td>
<td>1131</td>
<td>3469</td>
<td>1798</td>
<td>40</td>
<td>0</td>
</tr>
</tbody>
</table>

Performance

<table>
<thead>
<tr>
<th>PathDelays</th>
<th>Clock Cycles</th>
<th>Clock Period (ns)/ Frequency</th>
<th>Latency(ns)/Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>PathGains</td>
<td>4</td>
<td>5.394 /184 Mhz</td>
<td>21 / 47 Mhz</td>
</tr>
<tr>
<td>ChannelFunction</td>
<td>6</td>
<td>6.62 /151 Mhz</td>
<td>37 /26 Mhz</td>
</tr>
<tr>
<td>ChannelFunction</td>
<td>501</td>
<td>9.97 / 100 Mhz</td>
<td>4994 /0.2 Mhz</td>
</tr>
</tbody>
</table>
## Design Effort

<table>
<thead>
<tr>
<th>Design</th>
<th>Days spend</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>25.9% (28 hours)</td>
<td>Understanding the code, Converting matlab to C++, Removing library dependency, Writing HLS synthesizable code</td>
</tr>
<tr>
<td>Restructured code</td>
<td>22.2% (24 hours)</td>
<td>Manually loop merging, Expression balancing, Loop unrolling</td>
</tr>
<tr>
<td>Bit Accurate Design</td>
<td>29.6% (32 hours)</td>
<td>Calculation of 57 fixed point type widths (pathGains: 36, PathDelays:19, ChannelFunction: 2)</td>
</tr>
<tr>
<td>Optimized Design</td>
<td>7.4% (8 hours)</td>
<td>Optimizing using directives</td>
</tr>
<tr>
<td>Collecting Results/DSE/Presenting</td>
<td>14.8% (16 hours)</td>
<td>DSE, Collecting results, Presenting</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>~108 hours</td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

- Designed single channel wireless emulator using HLS tool.
  - HLS provides easy parameterization of WCE design.
- We plan to extend this work to multiple channel emulator and make end-to-end system
- Lessons Learned
  - Achieving target performance and area depends
  - Writing a “C Code” targeting architecture is essential
  - Application and code size
  - 2 optimization pragmas (pipeline, partition out of 33) + Restructured code+ Bit Width ➔ Target goal