# 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

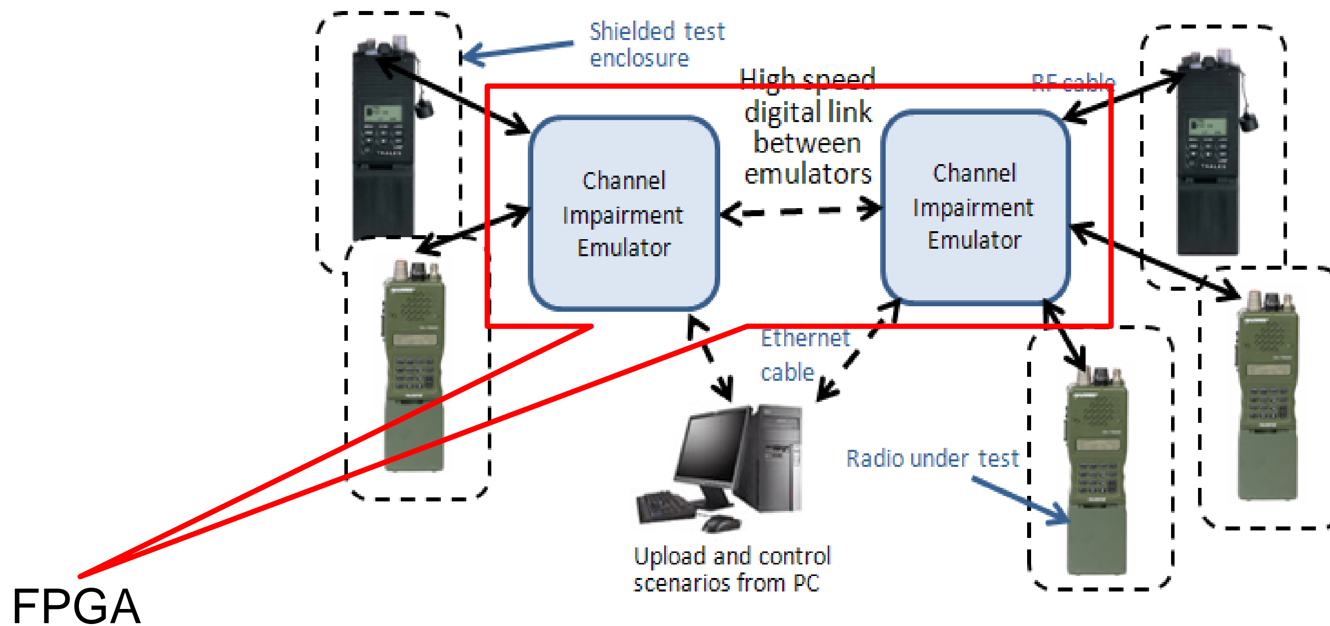
FPT'2012

# 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)

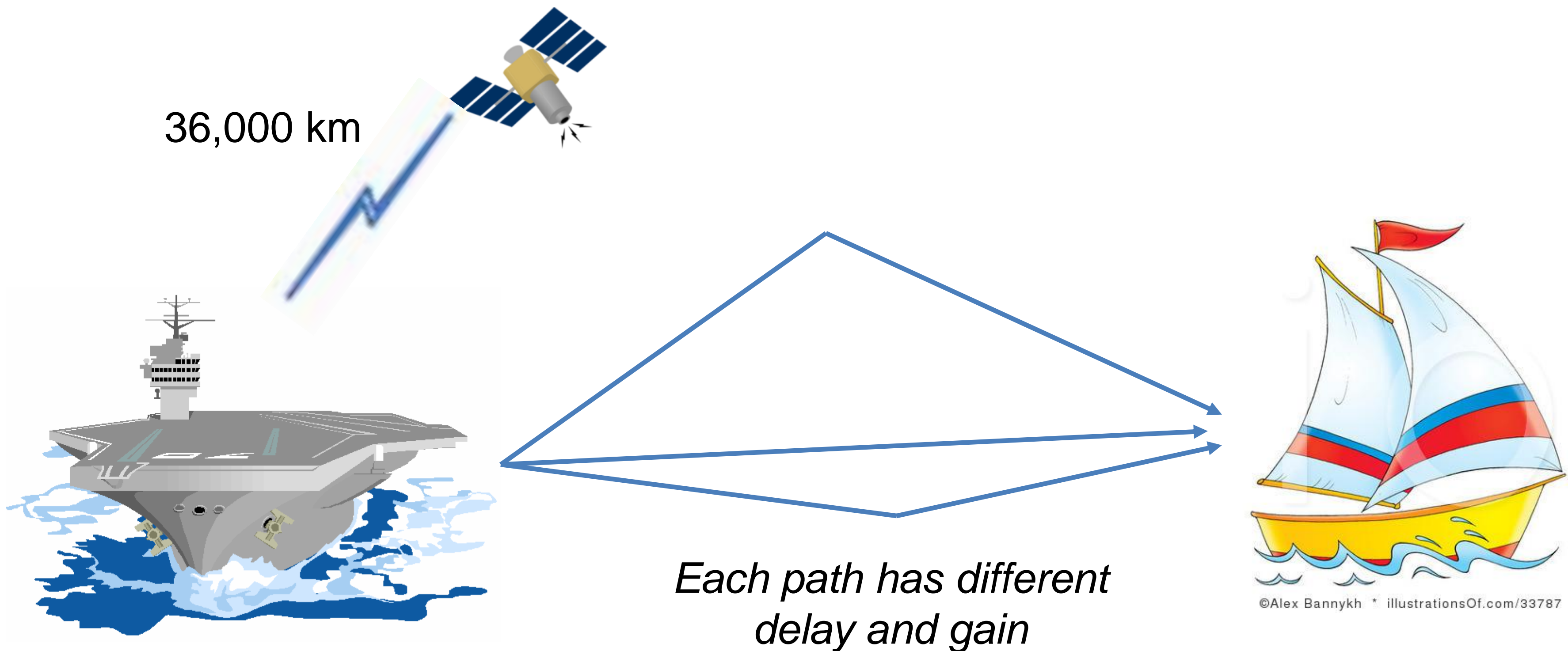
- Hardware in the loop (HWIL) **Wireless Channel Emulator (WCE)** implementation on an **FPGA** platform is purposed using **High-Level Synthesis** Tool.



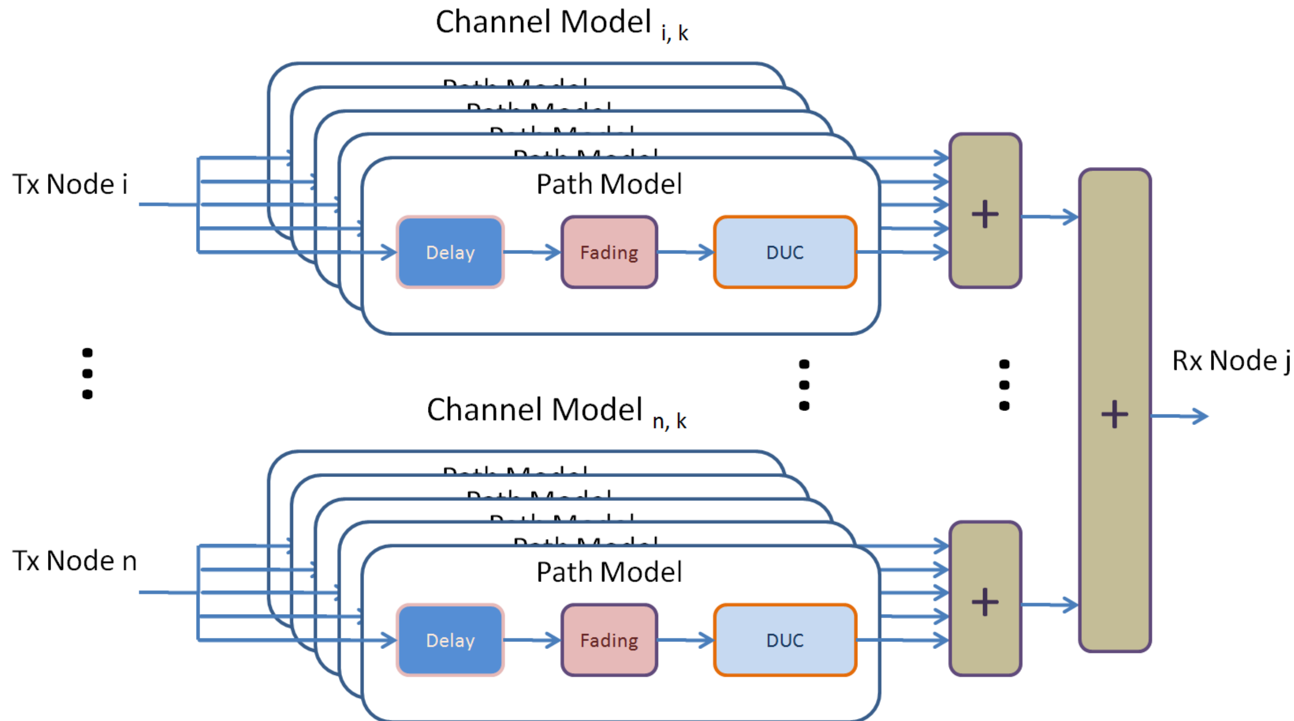

Top level concept of WCE

# 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

$$so_t = \sum_{i=0}^{n} si_{t-i_\tau} w(t)_i$$

  - *si*: Previous n+1 input complex samples
  - *so*: Complex sample output at present time
  - *w(t):* Dynamically changing set of weights (gain and delay)

- Complex sample calculation (channel function)

$$\mathrm{tapline}_{ij} = \mathrm{delayline}_{\mathrm{index}}$$

$$\mathrm{taps}_i = \sum_{i,j=0}^{N,K} \mathrm{tapline}_{ij} * weight_{ji}$$

$$\mathrm{so_r}_i = \sum_{i=0}^{N} gains_r_i * taps_r_i - gains_i_i * taps_i_i$$

$$so_i_i = \sum_i gains_r_i * taps_i_i - gains_i_i * taps_r_i$$

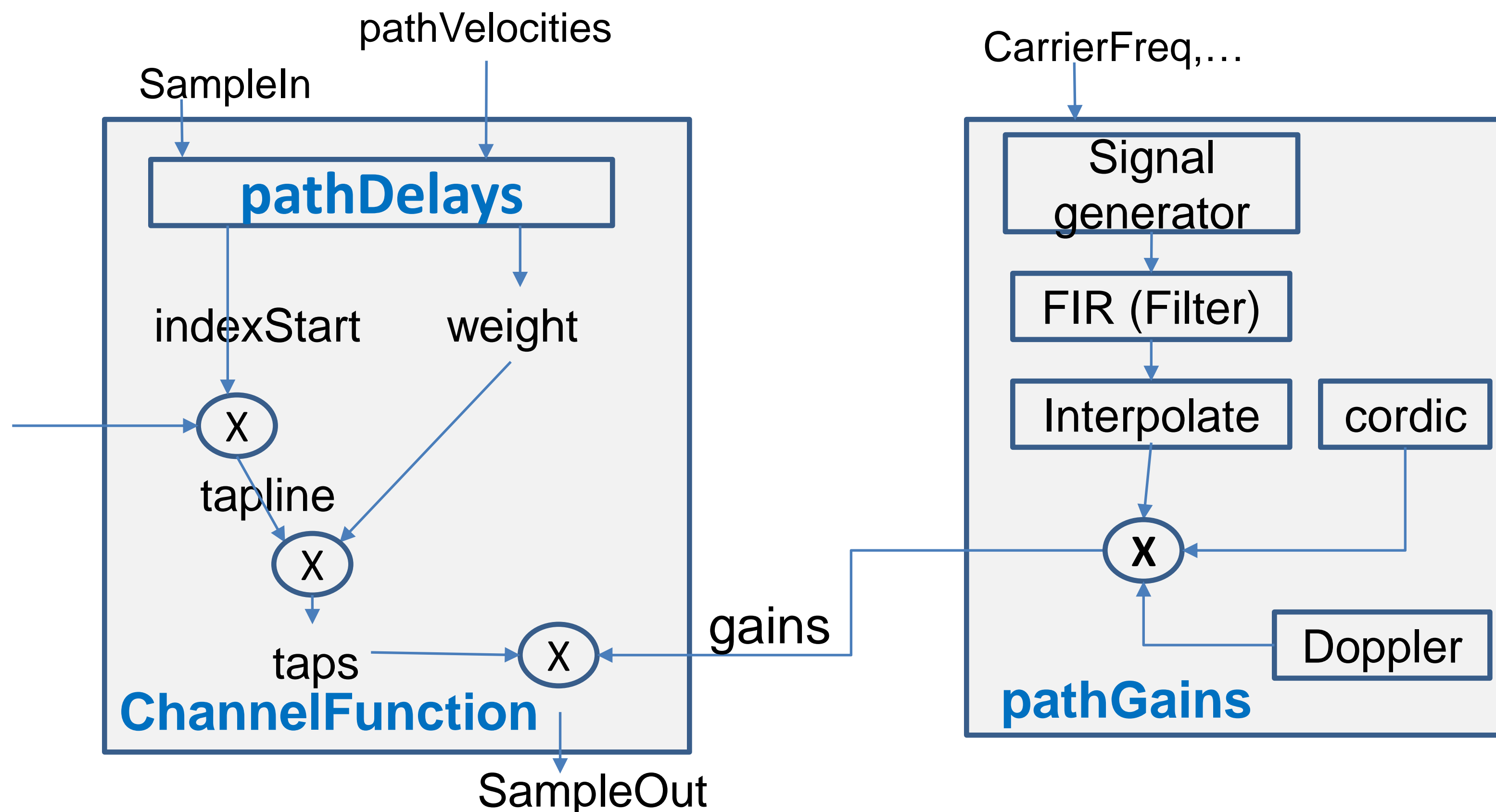
Path delays

Path gains

channelFunction

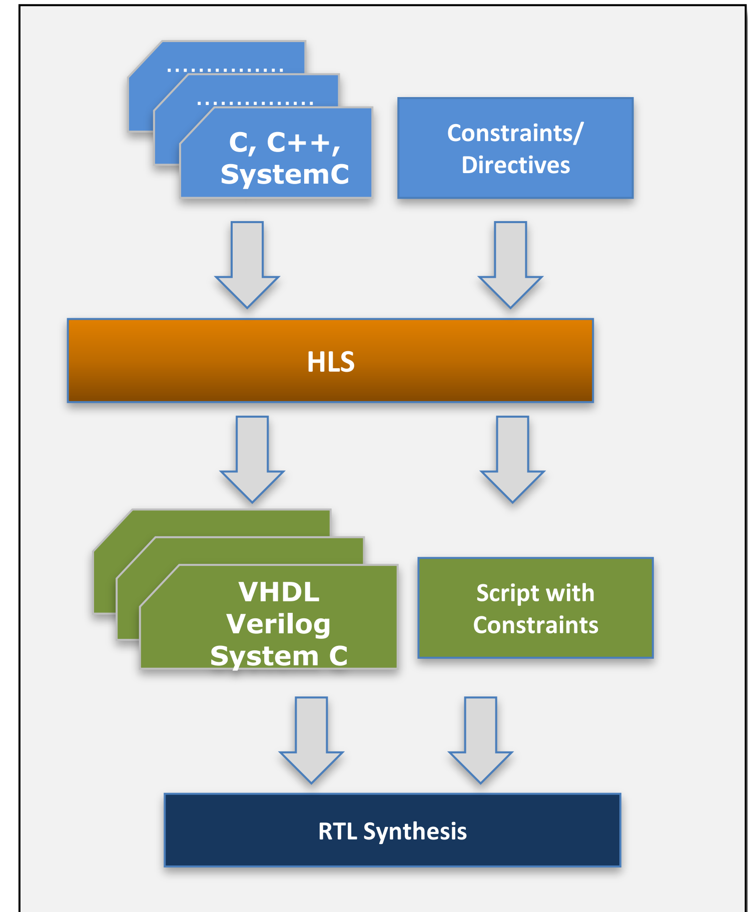
# Block Diagram of WCE

- 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 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



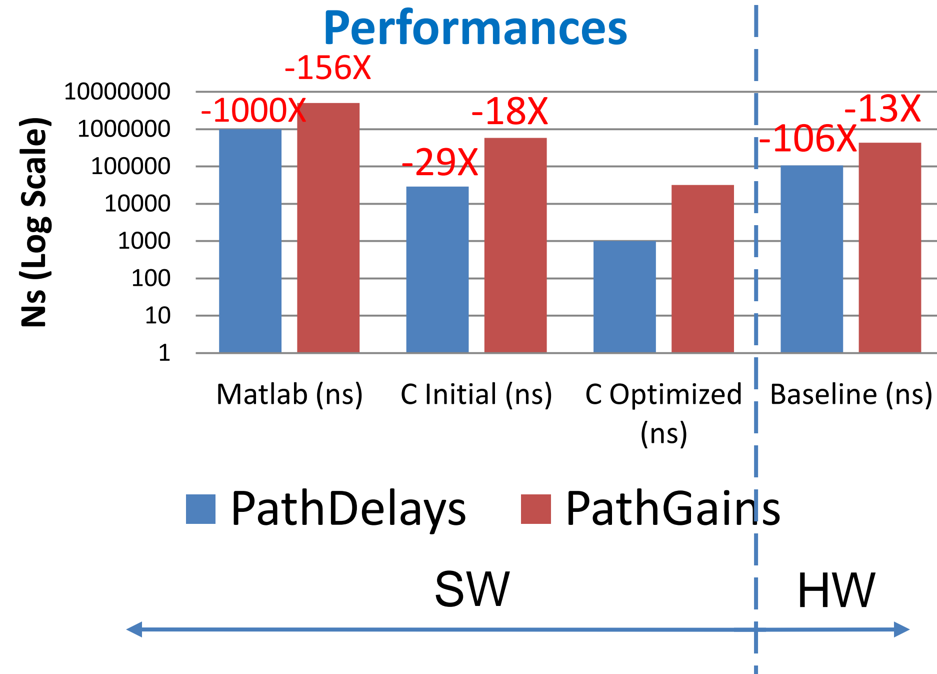

*Courtesy to Xilinx*

# 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…



# 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

```
for (int p = 0; p<N; p++){
  for(int i=0;i<SIZE; i++){
    t1=t1+js[i]*FIRin[p][i][0];}

  for(int i=0;i<SIZE; i++){
    t2=t1+js[i]*FIRin[p][i][1];}
}
```

⇒

```
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

```
t1=t1+js[i]*FIRin[p][i][0];
t2=t1+js[i]*FIRin[p][i][1];
```

⇒

```
t1_1=js[i]*FIRin[p][i][0];
t1=t1+t1_1;
t2_1=js [i]*FIRin[p][i][1];
t2=t2+t2_1;
```

41250 → 29730

  - Loop unrolling

```
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]  = …
        }
    }
}
```

⇒

```
for (int p= 0;p<N; p++){
  for (int i = 0;i<SIZE; i++){
    FIRinputs[p][i][0]  = …
    FIRinputs[p][i][1]  = …
  }
}
```

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

```
#ifdef BIT_ACCURATE
typedef ap_uint<6> AP_UINT6;
typedef ap_uint<12> AP_UINT12;
typedef ap_uint<33> AP_UINT33;
typedef ap_uint<32> AP_UINT32;
typedef ap_int<32> AP_INT32;
typedef ap_uint<20> AP_UINT20;
typedef ap_ufixed<15,2,AP_RND > AP_UFIXED20_3;
typedef ap_fixed<12,10,AP_RND > AP_FIXED12_10;
typedef ap_fixed<10,9,AP_RND > AP_FIXED10_9;
```

```
#else
typedef unsigned int AP_UINT6;
typedef unsigned int AP_UINT12;
typedef unsigned long long int AP_UINT33;
typedef unsigned long int AP_UINT32;
typedef int AP_INT32;
typedef unsigned int AP_UINT20;
typedef double AP_UFIXED20_3;
typedef double AP_FIXED12_10;
typedef double AP_FIXED10_9;
```

```
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 ap_ufixed<33,1,AP_RND > AP_UFIXED33_1;
typedef ap_ufixed<32,1,AP_RND > AP_UFIXED32_1;
typedef ap_ufixed<27,1,AP_RND > AP_UFIXED27_1;
typedef ap_ufixed<20,1,AP_RND > AP_UFIXED20_1;
typedef ap_fixed<10,1,AP_RND > AP_FIXED10_1;
typedef ap_ufixed<6,1,AP_RND > AP_UFIXED6_1;
typedef ap_ufixed<6,5,AP_RND > AP_UFIXED6_5;
typedef ap_ufixed<20,15,AP_RND > AP_UFIXED20_15;
typedef ap_ufixed<18,14,AP_RND > AP_UFIXED18_14;
```

```
typedef double AP_UFIXED45_1;
typedef double AP_UFIXED33_1;
typedef double AP_UFIXED32_1;
typedef double AP_UFIXED27_1;
typedef double AP_UFIXED20_1;
typedef double AP_FIXED10_1;
typedef double AP_UFIXED6_1;
typedef double AP_UFIXED6_5;
typedef double AP_UFIXED20_15;
typedef double AP_UFIXED18_14;
#endif
```

# 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

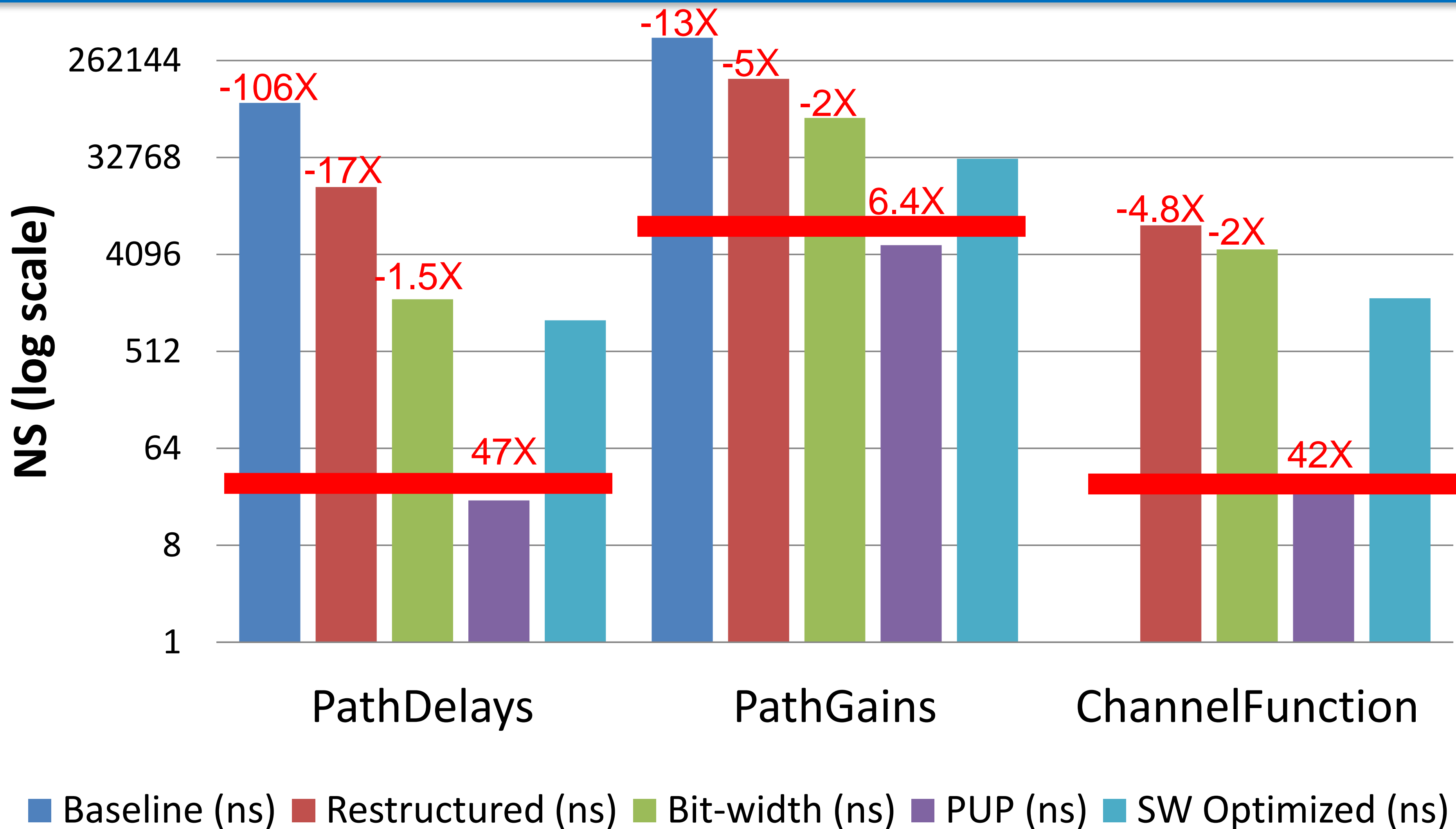
# 4. Pipelining and Partitioning - Example

- **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

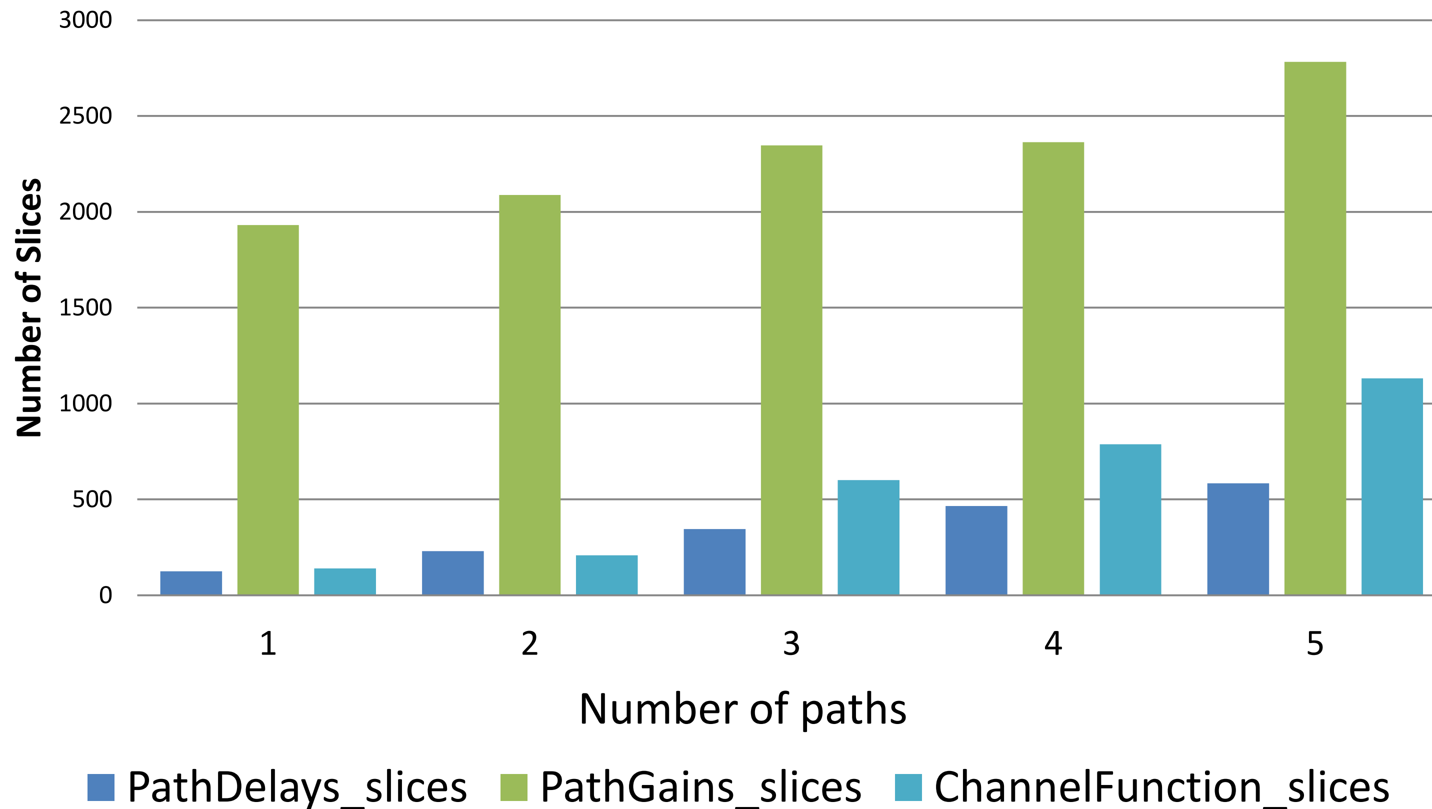
- Bit accurate design vs. Target
  - PathDelays 0.6X slower than target (30 Mhz)
  - PathGains 0.2X slower than  target (40 kHz)
  - ChannelFunction 1.1X slower than target (30 Mhz)

# Final Results



# 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)

| | Slices | LUT | FF | DSP48E | BRAM |
|---|---|---|---|---|---|
| PathDelays | 584 | 1843 | 411 | 30 | 0 |
| PathGains | 2783 | 8756 | 7044 | 53 | 30 |
| ChannelFunction | 1131 | 3469 | 1798 | 40 | 0 |

## Performance

| | Clock Cycles | Clock Period (ns)/ Frequency | Latency(ns)/Throughput |
|---|---|---|---|
| PathDelays | 4 | 5.394 /184 Mhz | 21 / 47 Mhz |
| PathGains | 501 | 9.97 / 100 Mhz | 4994 /0.2 Mhz |
| ChannelFunction | 6 | 6.62 /151 Mhz | 37 /26 Mhz |

# Design Effort

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

# 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