A Scalable Generator of Massive MIMO Baseband Processing Systems

Yue Dai

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-38
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-38.html

May 10, 2021

Acknowledgement

I first thank my advisor, Professor Borivoje Nikolic. His supports at all steps in this degree make me a well-rounded engineer. His knowledge gives me inspirations in my research. I additionally thank my committee Professor Yakun Sophia Shao for the valuable advice. The basis of this project is built on the prior work of Antonio Puglielli and Greg Lacaille. I greatly appreciate the guidance of Greg Lacaille on this project, and the help of Harrison Liew and James Dunn on the generator design. I thank the JUMP ComSenTer program for funding this project and providing me with the opportunity to learn from other universities.

A Scalable Generator of Massive MIMO Baseband Processing Systems

by Yue Dai

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Borivoje Nikolic
Research Advisor

5/10/2021

(Date)

* * * * * * *

Professor Yakun Sophia Shao
Second Reader

5/31/2021

(Date)

A Scalable Generator of Massive MIMO Baseband Processing Systems

by

Yue Dai

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Engineering - Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Borivoje Nikolić, Chair
Assistant Professor Yakun Sophia Shao

Spring 2021

The thesis of Yue Dai, titled A Scalable Generator of Massive MIMO Baseband Processing Systems, is approved:

Chair _____ Date _____

_____ Date _____

_____ Date _____

University of California, Berkeley

A Scalable Generator of Massive MIMO Baseband Processing Systems

Abstract

A Scalable Generator of Massive MIMO Baseband Processing Systems

by

Yue Dai

Master of Science in Engineering - Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Borivoje Nikolić, Chair

This thesis describes a scalable, highly portable, and power-efficient generator for massive multiple-input multiple-output (MIMO) uplink baseband processing. This generator is written in Chisel HDL, and produces hardware instances for the distributed processing in a scalable massive MIMO system. The generator is parameterized in both the MIMO system and hardware datapath elements. The performance of several generator instances with different parameter values are validated by emulation on a field-programmable gate array (FPGA), demonstrating both functionality and scalability, and operation up to 6.4Gb/s data throughput.

Contents

List of Figures

List of Tables

Acknowledgments

I first thank my advisor, Professor Borivoje Nikolic. His supports at all steps in this degree make me a well-rounded engineer. His knowledge gives me inspirations in my research. I additionally thank my committee Professor Yakun Sophia Shao for the valuable advice. The basis of this project is built on the prior work of Antonio Puglielli and Greg Lacaille. I greatly appreciate the guidance of Greg Lacaille on this project, and the help of Harrison Liew and James Dunn on the generator design. I thank the JUMP ComSenTer program for funding this project and providing me with the opportunity to learn from other universities.

Chapter 1

Introduction

1.1 Motivation

With the increasing number of mobile devices and the increase of consumption and production of media-rich content, the demand for higher data rate is growing dramatically. Innovations in the capacity-achieving codes and efficient modulation techniques have brought spectral efficiency of wireless point-to-point systems close to the theoretical Shannon limit [1]. MIMO wireless technology is widely considered as an energy-efficient, secure and robust approach to increasing the overall channel capacity and to reducing the effects of interference, [8, 16]. By increasing the spatial resolution with hundreds to thousands of antennas at the base station (BS), massive MIMO can support numerous users in the same time-frequency resource by providing each user with their interference-free, high-capacity link to the base station (BS) [11]. The integration of many antennas can be enabled by operating at millimeter-wave (mm-wave) frequencies due to the small antenna size. [4].

While the massive MIMO is attractive, implementing the system in a cost-efficient and energy-efficient way is challenging. To achieve a higher cost-efficiency and energy-efficiency, a scalable MIMO architecture needs to be applied, and the system should also be able to reconfigurable and portable easily for different MIMO system designs and different platforms. For these reasons, this work uses a scalable and modular MIMO system, where the beamforming operation is divided into 2 stages - the frequency flat maximum-ratio combining (MRC) beamforming and the frequency-selective decorrelation stage. The operation of the MRC beamforming can be distributed to multiple panels, and each panel perform the MRC beamforming independently. All panels are chained up, and panel MRC beamformed signals are summed up along the daisy-chain. The summed beamformed signal is sent to the decorrelation stage.

Energy efficiency can be achieved through hardware specialization; On the other hand, designing the specialized hardware is costly. The approach of designing hardware generators, as opposed to instances, enables reuse and thus reduces design cost [14, 15]. For many signal processing tasks, reusing the algorithm implementation enables faster development

of application-optimized hardware [17, 14]. As a result, instead of implement the MRC beamformer using dedicated instances in hardware, a generator is built using Chisel [3] to improve the reconfigurability and portability of the hardware implementation.

1.2 Prior Work

Massive MIMO testbeds

Centralized processing architecture

Argos testbed [19] is implemented with 64 antennas, and it has the capability of serving 15 users at the same time through zero-forcing and conjugate multi-user beamforming. It is integrated with commercial radio modules to realize the prototype. Argos testbed includes the central controller, 16 modules with 4 radios on each, the Ethernet switch, a clock distribution board, and a transimission synchronization board. The central controller, which is the host PC, use MATLAB to transmit data, weights, and control commands to the radio modules. The baseband processing of Argos use the centralized processing architecture, where all data are collected from antennas and processed in the centralized processor.

Another massive MIMO testbed is the LuMaMi [10], which is proposed by Lund University. The LuMaMi utilizes up to 100 base station antennas, and can serve up to 12 users on the same time/frequency resources. The testbed is integrated with software-defined-radios (SDRs), board switches, co-processors and higher layer control processor. SDRs enable the local processing and work as the interface between the digital and radio-frequency domain. The centralized beamforming operation is implemented on the FPGA of the co-processors. The higher layer processor controls the system and configures the radios.

However, one disadvantage of the above testbeds is that those architectures highly depend on the interconnection bandwidth. The data have to be collected from the frontends and sent to the centralized processor to perform beamforming. With the increase of number of antennas integrated on the testbed, the processing throughput will be limited by the interconnection bandwidth. As a result, more efforts are required to design more complicated, and higher-throughput routers or networks. To solve the interconnection bottleneck and improve design scalability, the massive MIMO system has to be distributed.

Decentralized processing architecture

An implementation of the decentralized baseband processing for a massive MIMO system in the graphic processing unit (GPU) has been proposed by Li [9], which partitions antennas into clusters and detects data locally to reduce inter-cluster communication bandwidth. The distributed beamforming algorithm uses alternating direction method of multipliers and conjugate gradients based method. The interconnection bandwidth requirement for this distributed GPU implementation is much lower than the centralized processing designs. But this distributed beamforming algorithm cannot avoid loops during computation, which

increases the hardware implementation complexity. Beside the decentralized GPU implementation, ArgosV2 also implements distributed processing based on Argos, which is the previous version of this testbed and implemented using SDRs [18].

Although the GPU and the software-defined radio (SDR) enable flexibility for experimentation with algorithms, they are not practical power-efficient solutions. Compared with FPGAs and application-specific integrated circuits, the power consumption of GPUs and SDRs is much higher. To make the massive MIMO system power-efficient, the hardware specialization is the solution.

Generators

Due to the high cost of specialized hardware design, to optimize the energy efficiency as well as reduce design cost, design generators instead of instances can achieve this goal [7]. In this work, Chisel is used to implement the generator. Chisel is a domain-specific extension to the Scala programming language [3]. Hardware designers can create their own libraries with the hardware primitives given by Chisel. After designers finish the hardware design, the Chisel compiler translates them into synthesizable Verilog representations, which makes the hardware design procedure much easier.

A memory-based, runtime-reconfigurable FFT engine generator proposed by Wang [20] is a good example of the generator design using Chisel. Although the generator targets SDR, the flexibility and portability of the generator help this work support a wide range of applications. The FFT engine can be generated easily with different design parameters and different design purposes.

1.3 Thesis Organization

The paper is organized as follows. Chapter 2 gives the overview of the scalable massive MIMO uplink baseband processing system. Chapter 3 discusses the design of the generator for the distributed processing part in the system. Chapter 4 details the "golden" model simulator and FPGA emulation setup for the generator evaluation. The functionality and performance are demonstrated and analyzed in Chapter 5.

Chapter 2

System Overview

In this chapter, the massive MIMO system, which includes both the uplink BS architecture and the signal packet format, will be discussed in detail. The BS consists of three parts: Heads, Spines, and the Tail. The Head is the mm-wave (or radio frequency) front-end. The Spine contains a maximum-ratio combining (MRC) beamformer for a sub-array. All Spines are chained up through the daisy-chain architecture. The Tail receives the MRC beam-formed signal, and performs the frequency-selective decorrelation to remove inter-user interference. Fig. 2.1 shows the uplink BS architecture.

2.1 Two-stage beamforming

Linear beamforming for wideband channels needs to consider the frequency-selective propagation effect. To improve energy-efficiency of computation, a two-stage beamforming algorithm has been proposed[7].

MRC beamformer

Theoretically, in flat-fading channels, MRC can maximize the output SNR [2], which is the purpose of the first-stage beamformer. Assume a multi-user MIMO (MU-MIMO) system containing K users and M antennas at the BS; let $\mathbf{H} \in \mathbb{C}^{M\times K}$ be the channel matrix, $\mathbf{s} \in \mathbb{C}^K$ be the transmitted symbols from users and $\mathbf{y} \in \mathbb{C}^M$ be the signal received at the BS. Then

$$\mathbf{y} = \mathbf{H}\mathbf{s} + \mathbf{n}, \tag{2.1}$$

where $\mathbf{n} \sim \mathcal{CN}(0, \sigma^2\mathbf{I}_M)$ is independently and identically distributed (i.i.d) zero mean complex Gaussian noise. Let $\tilde{\mathbf{y}}_{MRC} \in \mathbb{C}^K$ be the MRC beamformed signal at BS. Then the MRC beamformer gives

$$\tilde{\mathbf{y}}_{MRC} = \mathbf{H}^H\mathbf{y} = \mathbf{H}^H\mathbf{H}\mathbf{s} + \mathbf{H}^H\mathbf{n}. \tag{2.2}$$

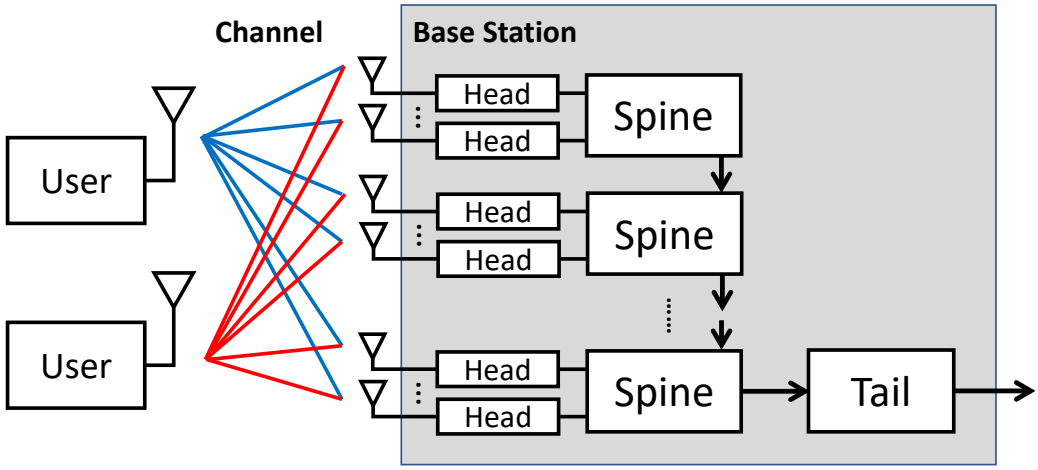

Figure 2.1: The scalable massive MIMO BS architecture for uplink.

The MRC beamformer can be split into multiple subarrays and then forms a distributed system by computing each sub-block locally, followed by summing the results through the daisy-chain. Let N equal to the number of sub-arrays, $\mathbf{B}_i \in \mathbb{C}^{M/N \times K}$ where $0 < i < N$, $\mathbf{H} = [\mathbf{B}_1^H \ \mathbf{B}_2^H \ ... \ \mathbf{B}_N^H]^H$, and $\mathbf{y}_i \in \mathbb{C}^{M/N}$ is the received signal at i-th sub-array. The previous equation can be expressed as

$$\tilde{\mathbf{y}}_{MRC} = \sum_{i=1}^{N} \tilde{\mathbf{y}}_{i,MRC} = \sum_{i=1}^{N} \mathbf{B}_i^H \mathbf{y}_i. \tag{2.3}$$

Frequency-selective decorrelation

For a MU system, the inter-user and inter-symbol interference are still present after the MRC beamforming. At this stage, a wideband channel is translated into multiple narrow subcarriers, and each narrow subcarrier performs the zero-forcing algorithm. Let N_s be the number of samples of the OFDM pilot, N_g be the number of subcarriers. Then each subcarrier contains N_s/N_g samples. Let $\tilde{\mathbf{Y}}_{i,MRC} = FFT(\tilde{\mathbf{y}}_{MRC})[i]$ be the i-th sample of the MRC beam-formed signal in the frequency domain, and $\mathbf{H}_i$ be the channel matrix for the i-th sample. The zero-forcing estimator gives

$$\hat{\mathbf{x}}_i = IFFT((\mathbf{H}_i^H \mathbf{H}_i)^{-1} \tilde{\mathbf{Y}}_{i,MRC}). \tag{2.4}$$

The estimation of $\mathbf{H}_i$ is the average of all estimations of $\mathbf{H}_j$ within the same subcarrier.

After the two-stage beamforming, the inter-user interference is removed and signal-to-interference-plus-noise ratio (SINR) increases.

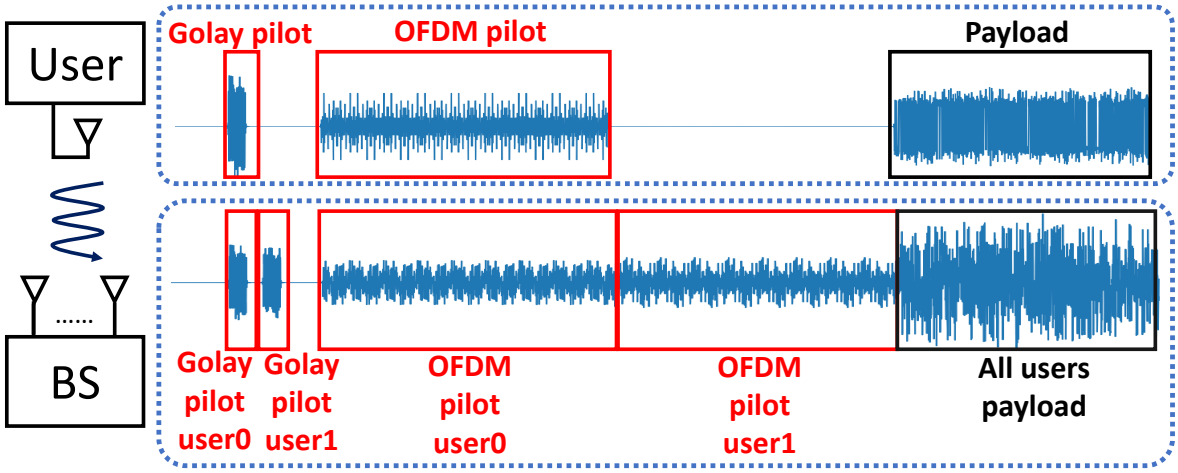

Figure 2.2: The signal packet format. The upper inset is the user TX signal packet. The lower inset is each BS channel received signal packet.

2.2 Signal packet format

Fig. 2.2 shows the individual user's signal packet and the signal packet received by each BS channel, respectively. Users send time-interleaved Golay pilots first, followed by time-interleaved OFDM pilots, to help perform the channel estimation in the two-stage beamformer. Golay pilots are used for the estimation of a frequency-flat channel matrix in the MRC beamformer, which consists of Golay complementary pairs. The BPSK-modulated OFDM pilots are used for the frequency-selective channel estimation in the decorrelation stage. Both Golay pilots and OFDM pilots include guard intervals. After all users send both pilots, they send quadrature amplitude modulated (QAM) payloads simultaneously. To synchronize the BS and users, a beacon signal is propagated to all users from the BS at the beginning of each packet to work as a time reference.

Chapter 3

Generator Design

Generators are modular, highly parameterizable register-transfer level (RTL) hardware designs implemented in the higher-level programming language to enable parametrization and design reuse via extension. The Spine generator is written in Chisel [3], which is a hardware construction language that facilitates parameterized circuit generation for both ASIC and FPGA digital logic designs. It generates the MRC beamformer stage in this scalable massive MIMO BS architecture. The Spine generator is parameterized in

- the number of channels per Spine
- the number of users in the system
- the oversampling rate
- the number of root-raised cosine (RRC) filter taps
- the Golay pilot design, including the code length, code seeds, and the guard interval
- the Lagrange polynomial order
- the datapath bitwidth
- the datapath parallelism

The Spine generator is integrated with multiple generators, which are designed for the general digital signal processing (DSP) by using the reusable hardware library described in [21]. The Spine datapath is shown in Figure 3.1. The inputs are M channels of in-phase and quadrature (IQ) signals which are parallelized at the configuration time. The outputs are K users MRC beam-formed signal summed with the signal coming from the upper neighboring Spine in the daisy-chain. The outputs are sent to the lower neighboring Spine. The parameters of each module generator are shown in the yellow boxes.

3.1 Signal Correction

The signal correction group includes a 65-tap RRC filter, an IQ signal synchronizer, and a direct current (DC) offset cancellation for each channel. The RRC filter is generated by the tree-reduced strength-reduced FIR filter generator. The number of taps is parameterizable, and the coefficient array is configurable. Fig. 3.2 shows an example of a FIR filter generated by the FIR generator with 6 taps and parallelized by 2. Each valid cycle, way-0 and way-1 shift registers right by one, and multiply with the corresponding coefficient. Way-0 and way-1 multiplication results are accumulated through way-0 and way-1 pipelined tree adders, respectively. Assume the number of taps in a RRC filter is N_T, because the RRC filer coefficients are symmetric, then to reduce the number of multipliers in the design, the input of the FIR filter is the summation of samples i and $N_T - i - 1$ for $i \in [0, N_T)$.

To balance the front-end impairments, the IQ synchronizer and the DC offset cancellation are included in the datapath set with manually calibrated IQ impairment and DC. The IQ imbalance comes from two parts:

- different gains, g_i and g_q, for I and Q channels.
- phase offsets, θ_i and θ_q, that make I and Q channels not orthogonal.

Let $y = y_i + j \cdot y_q$ be the signal without IQ imbalance for a single channel, and $y_{im} = y_{im,i} + j \cdot y_{im,q}$ be the signal with IQ imbalance due to the frontend. Then y_{im} can be expressed as

$$\begin{bmatrix} y_{im,i} \\ y_{im,q} \end{bmatrix} = P \begin{bmatrix} y_i \\ y_q \end{bmatrix} = \begin{bmatrix} g_i \sin\theta_i & g_q \cos\theta_q \\ -g_i \cos\theta_i & g_q \sin\theta_q \end{bmatrix} \begin{bmatrix} y_i \\ y_q \end{bmatrix} \tag{3.1}$$

To correct the IQ imbalanced signal, P^{-1} should be multiplied with the imbalanced signal. Then

$$\begin{bmatrix} y_i \\ y_q \end{bmatrix} = P^{-1} \begin{bmatrix} y_{im,i} \\ y_{im,q} \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} y_{im,i} \\ y_{im,q} \end{bmatrix} \tag{3.2}$$

Based on the equation, the architecture of IQ synchronizer is shown in Fig. 3.3.

Parameters of the IQ synchronizer are the coefficient matrix, ranging from $(-1, 1)$.

3.2 Frequency-flat channel estimation

The MRC beamforming stage and the frequency-selective decorrelation stage perform the channel estimation separately. In the MRC beamformer stage, the Golay pilot is used for the frequency-flat channel estimation; while in the frequency-selective decorrelation stage, the OFDM pilots aid the channel estimation. The detailed design of the frequency-selective channel estimation can be found in [7].

The Golay pilot contains two complementary sequences with the same length L, where L is a power of 2. The sum of the aperiodic autocorrelation functions of the sequence pair

is a Kronecker delta [6]. Let $h_{m,k}$ represent the mth-row kth-column element in the channel matrix $\mathbf{H}$, and $\mathbf{g}_k$ be the Golay pilot of user k. Because the Golay pilot is time-interleaved, the pilot received at the channel m is given by $\tilde{\mathbf{g}}_{k,m} = h_{m,k}\mathbf{g}_k + \mathbf{n}$, where $\mathbf{n}$ is the additive white Gaussian noise. Let the auto-correlation of $\tilde{\mathbf{g}}_{k,m}$ be $R_{k,m}$. The estimation of $h_{m,k}$ is

$$\hat{h}_{m,k} = \frac{1}{2L} R_{k,m}. \tag{3.3}$$

According to [6], the Golay sequence with length $2L$ can be generated by a sequence of delayed values $\mathbf{D}$ and a sequence of seeds $\mathbf{W}$ with length $log_2(L)$. In this design, $\mathbf{D}$ is any permutation of the set $\{2^0, 2^1, ..., 2^{log_2(L)-1}\}$, and the seed $w \in \{-1, 1\}$. The Golay sequence pair is generated using the following algorithm:

Algorithm 1 Golay sequence pair generation

Input: L, **D**, **W**
Output: **g**

```
Create two empty arrays g_A and g_B with sizes are log2(L)+1 by L
g_A[0,0] ← 0, g_B[0,0] ← 0
for 0 ≤ m < L do
  for 0 ≤ n < log2L do
    if m ≥ D[n] then
      g_A[n+1,m] ← W[n]g_A[n,m] + g_B[n,m − D[n]]
      g_B[n+1,m] ← W[n]g_A[n,m] − g_B[n,m − D[n]]
    else
      g_A[n+1,m] ← W[n]g_A[n,m]
      g_B[n+1,m] ← W[n]g_A[n,m]
    end if
  end for
end for
g ← Concatenation of the reverse of g_A[log2L] and the reverse of g_B[log2L]
return g
```

We designed a pipelined parallelizable Golay correlator generator based on the Algorithm 1. Because the reverse of a Golay sequence pair is also a Golay sequence pair, the Golay correlator is equivalent to the Golay generator of the reverse sequence. Fig. 3.4 shows the generated architecture of an oversampling-2, length-4, parallelism-4 Golay correlator. The proposed design doesn't contain multipliers, which makes it energy-efficient. The upper path generates $\mathbf{g}_B$ and the lower path generates $\mathbf{g}_A$ in Algorithm 1. The datapath is pipelined by $log_2(L)$ stages. The amount of parallelism changes the delay register connection and the input of the adder by input look-ahead. The adder input can be either from the delay registers or the pipeline registers. The architecture is hardware parameterized for $\mathbf{D}$, but can be configured for $\mathbf{W}$ on the fly. The sequencing controller controls the input of $\mathbf{W}$ at each user's Golay pilot time interval.

The channel delay estimation contains two parts: coarse delay estimation and fine delay estimation. The coarse delay estimation is based on the correlation norm peak of Golay pilots in Eq. 3.3. The correlation norm peak detection contains two parts: parallel peak detection and global peak detection. The parallel peak detection detects the maximum value in each parallel lane, and the global peak detection detects the peak among all lanes. The finite state machine for parallel peak detection contains 3 states:

- **Wait**. In this state, the parallel peak detector tracks the average power P_{avg} of the correlation result for a certain window size WS. If the input correlation power is larger than **threshold** $\times$ P_{avg} and larger than **lowerbound** value, then update P_{max} to be the input correlation power and maximum power buffer index i_{max} to be $WS-1$, and go to **Detect** state; Otherwise, stay in this state.

- **Detect**. Track the maximum power for the window size $WS-1$. If the input correlation power is larger than current P_{max}, then updates P_{max} and i_max. After finishing tracking $WS-1$ number of correlation power, go to **Done** state.

- **Done**. In this state, check whether the input correlation power is larger than the current P_{max} or not. If yes, output this correlation power; Otherwise, output P_{max}. Then go back to **Wait** state.

The finite state machine for the global peak detection contains 3 states as well:

- **Wait**. Use the shift register to keep records of the most recent $WS\times$ number of lanes correlation results and their power. If one of lanes detects the peak, go to **Detect & Output** state; Otherwise, stay in the current state.

- **Detect & Output**. Use tree-reduced comparator to find the maximum power among the correlation powers in current and the previous cycle in all lanes. If the index of the maximum power in the buffer is larger than or equal to the number of lanes, output the correlation result corresponding to the maximum power and its adjacent $2l$ correlation results, shift the buffer, and go back to **Wait** state; Otherwise, go to **Output** state.

- **Output** state. Output the correlation result corresponding to the maximum power and its adjacent $2l$ correlation results, and go back to **Wait** state.

However, due to the ADC skew and the sampling phase mismatch, the signal may not be sampled at the maximum power, which does not maximize the correlation norm peak. To resample the signals with maximum power, Lagrange polynomial $\mathcal{L}(x)$ is used to interpolate the correlation norm based on $2l+1$ correlation norm samples $c(i)$ around the peak, where $i \in \{-l, ..., 0, ...l\}$. The fine delay estimation $\hat{s}_0$ is

$$\hat{s}_0 = \arg\max_x \mathcal{L}(x) = \arg\max_x \sum_{i=-l}^{l} \left[c(i) \prod_{-l, j\neq i}^{l} \frac{x-j}{i-j} \right]. \tag{3.4}$$

Let $y(t)$ be the skewed signal and $\hat{y}(t)$ be the deskewed signal, which is given by

$$\hat{y}(t) = \sum_{i=-l}^{l} \left[y(t+i) \prod_{-l, j \neq i}^{l} \frac{\hat{s}_0 - j}{i - j} \right]. \tag{3.5}$$

In software design, to find $\hat{s}_0$, one needs to take the derivative of $\mathcal{L}(x)$, and use Newton method to find the maximum point. Loops cannot be avoided when using Newton method to find the zero point. In order to avoid loop iterations in hardware, the following algorithm is used to calculate the maximum $\hat{s}_0$ for a given resolution r.

Algorithm 2 Fine delay estimation

Input: **c**, r
Output: $\hat{s}_0$
  $s_q \leftarrow q \cdot r$, where $q \in \{-1/r, ..., 0, ..., 1/r\}$
  for q from $-1/r$ to $1/r$ **do**
    Calculate all coefficients $cf_i \leftarrow \prod_{j=-l, j\neq i}^{l} \frac{s_q - j}{i - j}$, where $i \in [-l, l]$
    $\hat{y}_q(0) \leftarrow \sum_{i=-l}^{l} y(i) \cdot cf_i$
  end for
  $\hat{s}_0 \leftarrow \arg\max_q \hat{y}_q(0)$
  return $\hat{s}_0$

Let the datapath bitwidth to be B. The threshold of the peak detector ranges from $[0, 2^{B-1})$ with precision 0.5, and the lower bound of the peak detector ranges from $(-1, 1)$ with precision $2^{-(B+1)}$. The fine delay estimation ranges from $(-1, 1)$ with precision r, and the fine delay synchronization coefficient ranges from $(-1, 1)$ with precision $2^{-(B+1)}$.

The Golay correlator generator is tested against the match filter result using the same Golay pilot. The Golay pilot is generated by random seeds. The coarse delay estimation and fine delay estimation are tested against the result produced by the Python golden model, which will be discussed in detail in Chapter 4, using the same input signal. The fine delay synchronizer is implemented using the FIR generator. It is tested with randomly generated data and coefficients, and the result is compared with the software implementation of the same FIR filter.

3.3 MRC beamformer

The MRC beamformer is generated by the pipelined systolic-array-based matrix multiplier generator, with parameterized matrix dimensions and datapath parallelization. The matrix multiplier uses the weight-stationary architecture. Each processing element (PE) consists of a width-parameterized complex, fixed-point multiplier and an adder. For a massive MIMO system with K users and M channels per Spine, the matrix multiplier dimension is $M \times K$

for a single path. In order to meet timing requirement for different platforms or technology, the number of pipeline stages along K dimension is also parameterized.

Fig. 3.5 shows the beamformer architecture. The weight inputs are the estimated frequency-flat channel matrix, and the input to each row is the channel's fine synchronized signal.

Parameters K, M, and parallelism can be any integer larger than 0. The input, coefficients, and the output have the same data range and precision as the datapath. The MRC beamformer is tested using randomly generated numbers, and the result is compared with the ground truth produced by the software implementation of the matrix multiplier using the same test data.

3.4 Sequencing Controller

The sequencing controller is designed for: 1) providing the time reference for each processing module; 2) channel synchronization.

At the beginning of each signal packet, the sequencing controller sends the beacon signal to all modules to reset registers and state machines. For Golay correlators, the sequencing controller also controls the change of W for different users. Because Golay pilots are sent in TDD mode, the sequencing controller will set up the W sequence in all Golay correlators to be user k's W when it is the time slot for that user.

Due to different user propagation delays and the front-end mismatches, the signal arrival times are mismatched, even for the same user. Let the time delay created by the user i be $T_{u,i}$ and the time delay created by the channel j be $T_{c,j}$ where $i \in [1, K]$ and $j \in [1, M/N]$. We can assume $T_u \sim \mathcal{N}(\mu_u, \sigma_u^2)$. Then the time delay between user i and channel j in the same Spine module is $T_{u,i} + T_{c,j}$. To synchronize the channel before the MRC beamformer, a channel time delay estimator is needed. This can be achieved by averaging the time delay among all users signal received by the same channel.

$$\mathbb{E}\big[\frac{1}{K}\sum_{i=0}^{K}(T_{cj} + T_{ui})\big] = T_{cj} + \mu_u. \tag{3.6}$$

In the sequencing controller, both integer and decimal delay estimations for all users are accumulated for each channel. The averaged integer delay estimation will be sent to channel coarse synchronizers, and the averaged decimal delay estimation will be sent to channel fine synchronizers. In channel coarse synchronizers, all other channels are synchronized to the channel that has the longest delay. In channel fine synchronizers, a l-tap FIR filter is used, whose l taps are generated using Eq.3.5.

Due to the slow change of channel delays and the long critical path of the channel delay estimation, the channel delay estimation of the current packet will be used for the channel synchronization of the next packet.

The user time delay needs to be estimated after the MRC beamforming. The user time delay estimation uses the same algorithm as the channel delay estimation. The downsampling phase selection is determined by the user coarse time delay estimation.

The Spine Generator

The Spine Generator is integrated with the generators described above. The datapath is shown in Fig.3.1. The ADC input in each channel firstly goes through the signal correction module, which includes the RRC filter, IQ synchronizer and DC cancellation, to correct the frontend impairment. Then the corrected pilot is used to estimate the frequency-flat channel matrix and the channel delay in the channel estimation module. Channel delays are collected in the sequencing controller module. The sequencing controller calculates the delay, and sets both the channel coarse delay synchronization module and the channel fine delay synchronization module for the next packet. The synchronized signals of all channels then go into the MRC beamformer. After beamforming, each user's delay is estimated in the user delay estimation module, and the estimated delay is set to the user fine delay synchronization module. The sequencing controller calculates the payload start position of each user based on the user delay estimation result, and then controls the down sampler. The down sampled signal is sent to the panel sum module, and summed with the accumulation result transmitted from the upper neighbor panel in the daisy-chain architecture. The accumulation output is then passed to the lower neighbor panel.

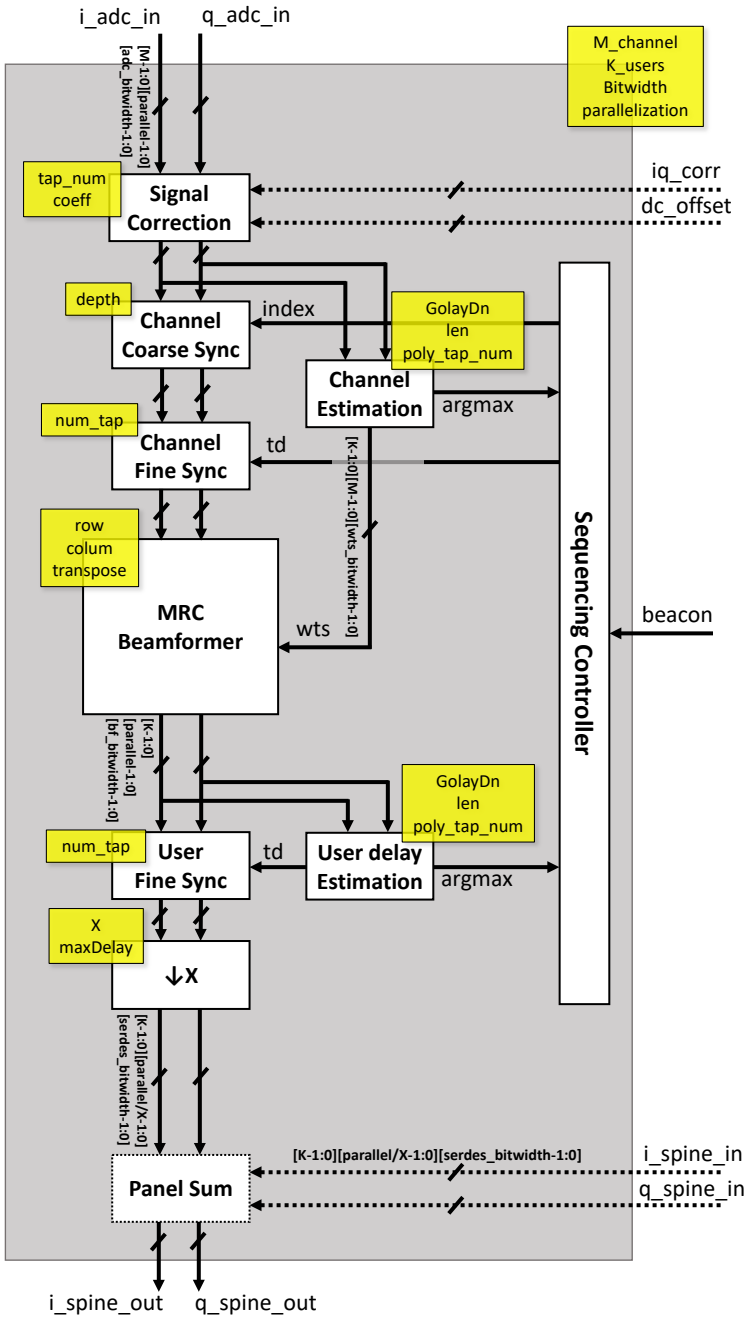

Figure 3.1: The Spine generator datapath.

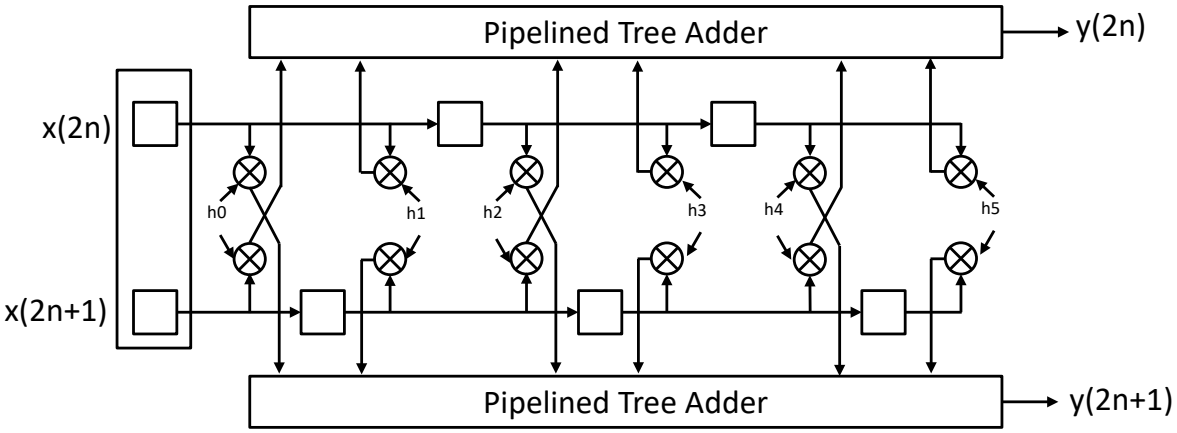

Figure 3.2: FIR Filter Generator.

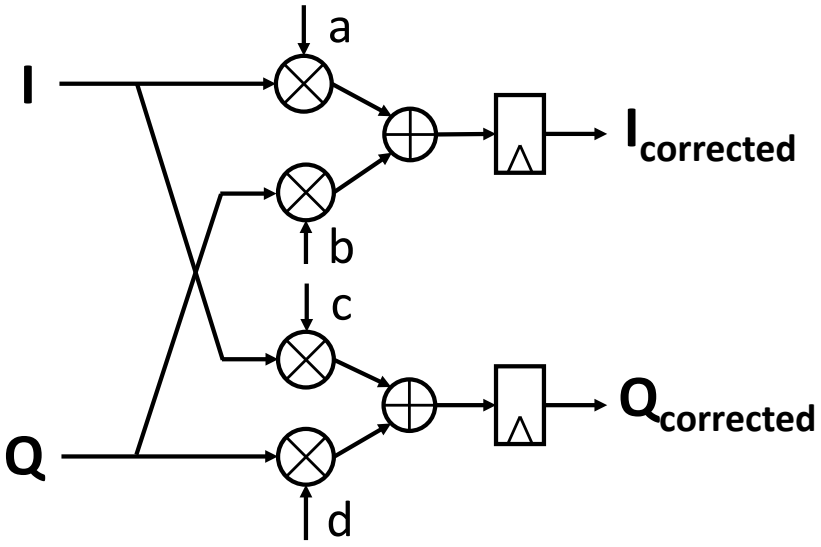

Figure 3.3: IQ Correction.

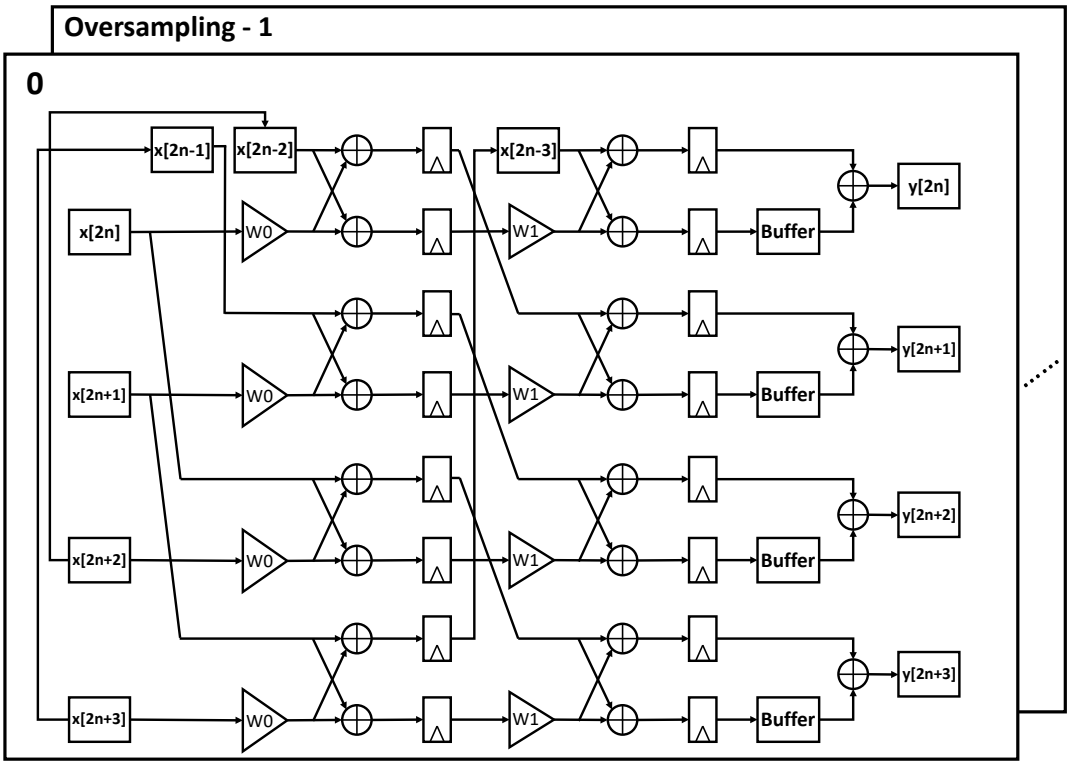

Figure 3.4: Golay Correlator.

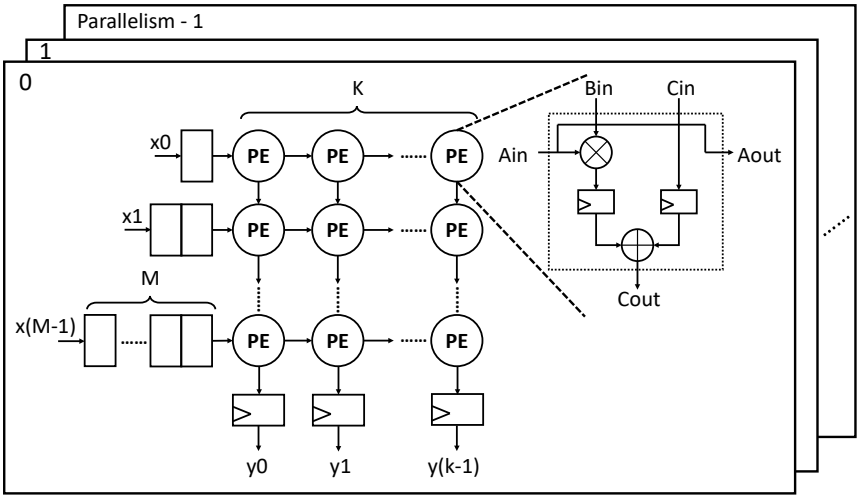

Figure 3.5: MRC Beamformer.

Chapter 4

FPGA Emulation

4.1 Simulator

A Python-based system simulator is built to simulate an end-to-end massive MIMO system, including the transmission packet generation, various channel models, front-end impairments, and modular baseband signal processing [7]. We treat this simulator as a "golden" model and evaluate the performance of this generator against it.

The transmission packet is parameterized in terms of the QAM modulation order, the pilot length, the guard interval length, and the number of sub-carriers for the OFDM pilot. All users shared the same **D** sequence, which is the same as the hardware design, but with different **W** when generating the Golay pilot. User payloads are generated separately. All user payloads are stored for later evaluation.

Channel models included in the simulator are the frequency-flat i.i.d channel, the frequency-flat line-of-sight channel, and the Rician channel. It is also parameterized on the SNR. The simulator also includes the carrier frequency offset (CFO) simulation, the sampling frequency offset (SFO) simulation, the channel skew simulation, and the front-end impairment simulation. Since the Spine generator implements the frequency-flat MRC beamformer, the performance is evaluated on: 1) frequency-flat i.i.d channel; 2) Rician channel with $K = 10dB$.

The reference simulator includes the Spine and the Tail modules. The Spine algorithm shown in Fig. 3.1 is simulated in floating-point data format. The Tail is simulated with the frequency-selective decorrelation algorithm on the beamformed signal to eliminate inter-user interference and equalize inter-symbol interference [7]. In the Tail module, CFO and SFO are corrected using OFDM pilots. The corrected signal is transformed to the frequency domain, and decorrelation algorithm is applied on each sub-carriers. After the decorrelation stage, the signal is transformed back to the spatial domain, and then demodulated.

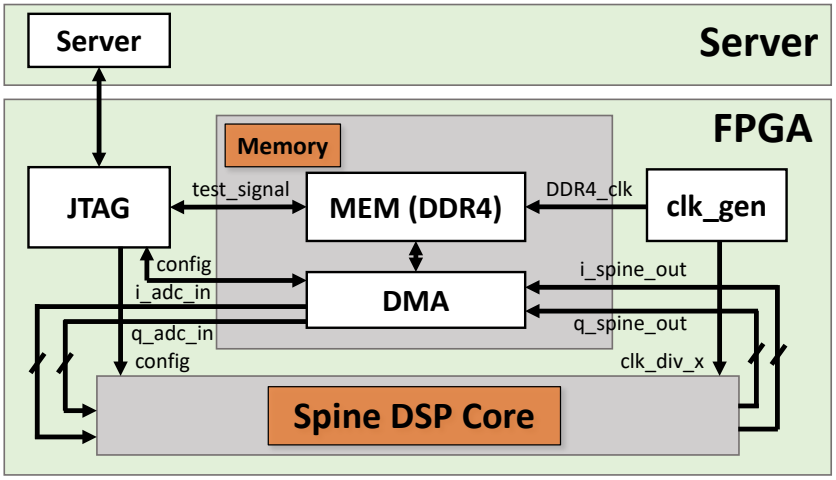

Figure 4.1: The scalable massive MIMO uplink baseband processing system emulation architecture.

4.2 FPGA Emulation

The Spine generator is implemented on a Xilinx VCU118 FPGA. The FPGA emulation system design is shown in Fig. 4.1. The set up is shown in Fig. 4.2. The FPGA contains:

- the Spine digital signal processing (DSP) core generated by the Spine generator but without the panel summation module
- the memory system
- the clock generators for two clock domains

The server transfers the data to the FPGA, and simultaneously configures the Spine DSP core and DMA through JTAG. Because the panel summation module maintains precision, the emulation performance should be the same as if the summation is implemented on the FPGA. The emulation system parameters are shown in Table. 4.1. The MIMO system contains 32 channels and 2 users with signal bandwidth 200MHz. Due to the resource limitation of the FPGA, each FPGA is implemented with 4 channels, 8-bitwidth, and 8-parallelization on the datapath. The baseband clock frequency is 50MHz.

The JTAG connection bitwidth is 64. The on-board DDR4 memory bitwidth is 512 without error correction code. The DDR4 is connected using the DDR4 interface provided by Xilinx.

The emulation data flow is shown in Fig. 4.3. The Python simulator generates user packets and simulates the channel. The Tcl script generator converts data to the 8-bit fixed-point format, and generates the Tcl scripts for the data transmission control and the Spine

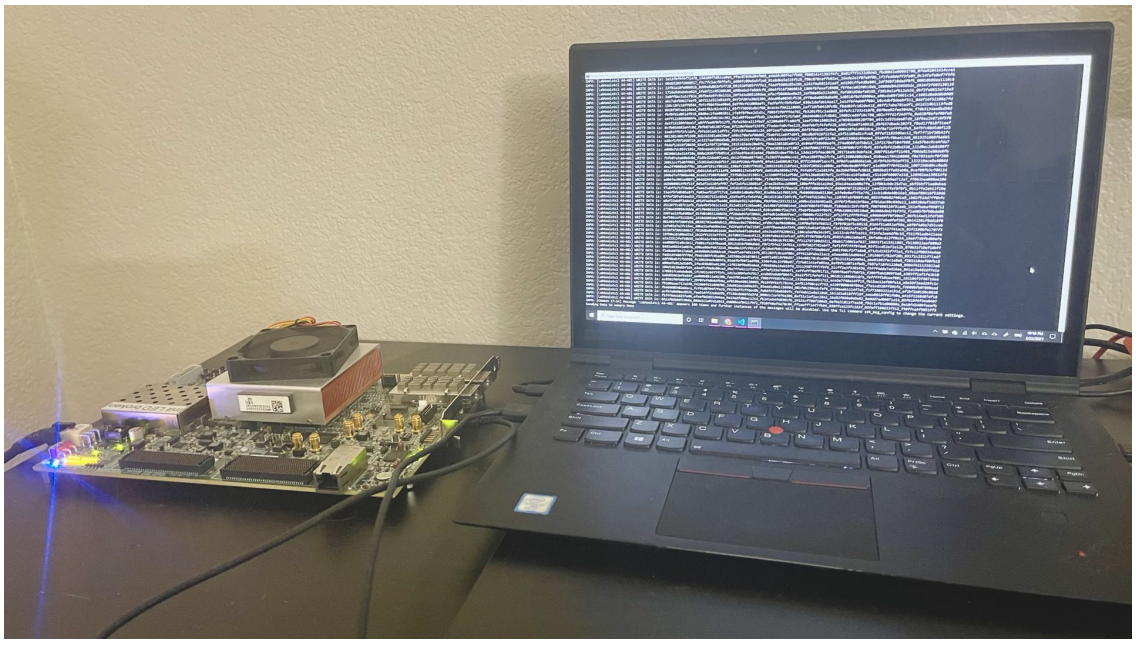

Figure 4.2: The scalable massive MIMO uplink baseband processing system set up.

Table 4.1: FPGA emulation system parameter

	Parameter	Value
System parameter	MIMO system parameter	32 channels, 2 users
	Signal bandwidth	200MHz
	Oversampling rate	2
	Modulation scheme	QPSK, 16-QAM
FPGA parameter	FPGA Type	Xilinx VCU 118
	Number of channels per Spine	4
	Datapath bitwidth	8
	Datapath parallelization	8
	Baseband clock freq	50MHz

DSP core configuration. The data transmission is implemented with Tcl axi commands. Because the number of channels per Spine is 4, then every 4 channels data are grouped together, and reshaped by the channel order. For example, the order of the data can be channel 0 sample 0, channel 1 sample 0, channel 2 sample 0, channel 3 sample 0, channel 0 sample 1, etc. Each Tcl command sends 512 bits from the server to the FPGA memory. The DSP core configuration includes:

- Set up DMA with the load and store base addresses and length, and the start signal.

- The beacon signal to indicate the start of each packet.
- The number of users in the system.
- Each user's **W** sequence for the Golay pilot.
- The thresholds for the channel and user correlation peak detector.
- Set up debug ports.

During the signal processing in the FPGA, the input data stream are loaded from the memory through the DMA, then processed in the DSP core, and stored back to the memory through DMA. When load and store conflicts, store has a higher priority. After the FPGA finishes data processing, the data are loaded from the memory back to the server through JTAG. This procedure is also implemented with Tcl commands. Each command loads 2048 bits from the FPGA memory to the server. When the first 4 channel data finishes processing, the next 4 channel data will follow the same procedure, and process the data in the FPGA. This procedure will be repeated till all 32 channel data are processed. All of the communication between the PC and the FPGA are through JTAG using AXI4 protocol. The emulation data are then reformatted to the floating-point format and simulated by the Tail simulator in PC.

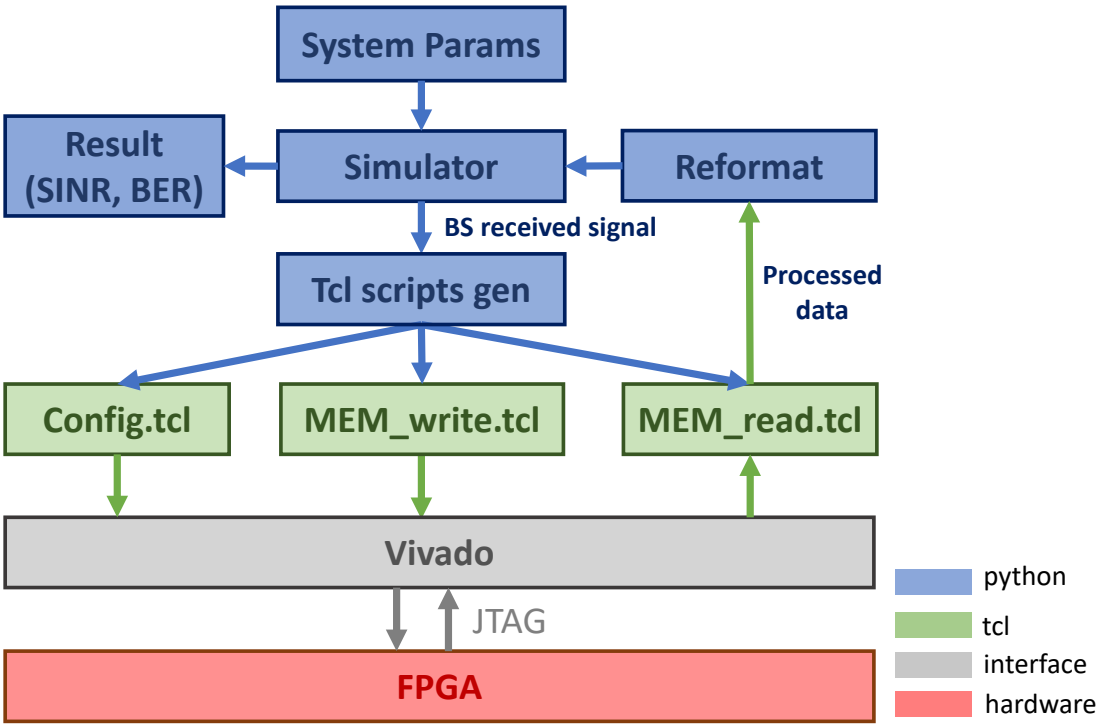

Figure 4.3: Emulation system control flow chart.

Chapter 5

Results

5.1 Performance

The Spine generator demodulation bit-error rates (BER) are evaluated under different SNR scenarios. The channel models are the flat i.i.d channel and the Rician channel with $K = 10dB$. The results of BER vs. SNR for the FPGA emulation system is shown in Fig. 5.1. The figure shows the BER ranging from 10^{-6} to 10^{-3}. The inset a) shows the BER of the quadrature phase-shift keying (QPSK) modulation, and the inset b) shows that of the 16-QAM. The result shows that at SNR 11.9dB for the QPSK modulation and 19dB for the 16-QAM modulation, the BER reaches 10^{-3} in the flat i.i.d channel. For both QPSK and 16-QAM modulation scheme, the emulation BER is very close to the simulation BER produced by the floating-point golden model, validating the functionality of the Spine generator. The small difference between the simulation and emulation performance comes from the residual channel estimation and the datapath quantization errors.

The blue lines in Fig. 5.2 show the normalized mean-square error (MSE) of the true channel matrix and the frequency-flat channel estimation in the FPGA versus SNR for different Golay pilot lengths in the flat i.i.d channel. The normalized MSE of the channel estimations decreases as the Golay pilot length increases. From the figure, one can also notice that when the SNR is lower than 17 dB, 15 dB, and 13dB for the lengths of 8, 16, and 32, respectively, the MSE increases faster than higher SNR in log scale. The change of the slope is due to the increase of pilot synchronization failure. However, by increasing the Golay pilot length to 64, the channel estimation MSE reduces to lower than 3×10^{-2} for the SNR even lower than 15dB, and the possibility of pilot synchronization failure becomes much lower.

The red lines in Fig. 5.2 show that when the SNR is lower than 23 dB, 19 dB, 17 dB, and 13dB for pilot lengths of 8, 16, 32, 64, respectively, the normalized channel estimation MSE of the simulator and the FPGA begins to increase. The normalized MSE converges to 5×10^{-4}. To fit into the 8-bit datapath, a 0.2 gain is applied at the emulation input. Then the normalized MSE floor corresponds to about 0.5 quantization bit, which is the quantization

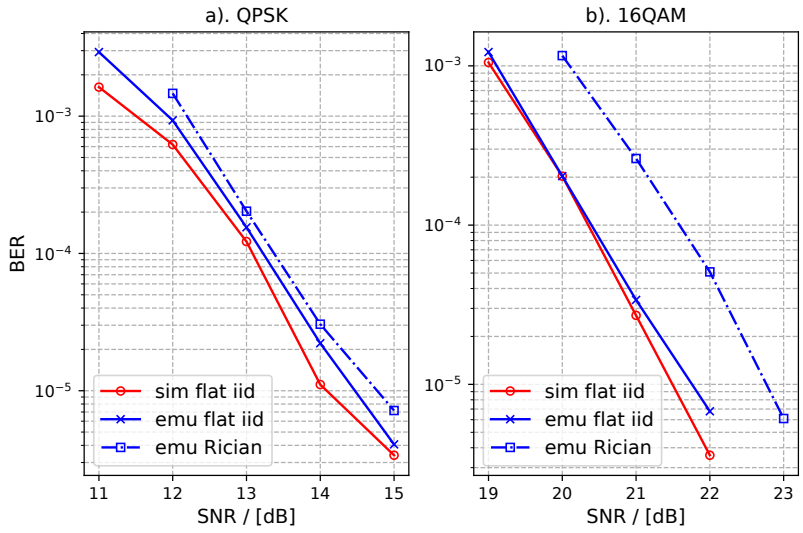

Figure 5.1: BER vs SNR for 32 antenna 2 users simulation and emulation results for a). QPSK modulation scheme and b). 16-QAM scheme.

error. Emulated design adopts the Golay pilot length of 64 because of its adequately low MSE. In Fig. 5.3, the SINR difference for the QPSK slightly increases as the SNR decreases, which agrees with the MSE measurements from the channel estimation. For 16-QAM, the SINR for the simulation and emulation is nearly the same.

Fig. 5.1 also shows the BER versus SNR for the QPSK and 16-QAM in a low-multipath Rician channel. For the QPSK modulation, the performance is similar to the flat i.i.d channel. For the 16-QAM modulation, the performance degrades by about 1 dB. Compared with the QPSK modulation, the 16-QAM modulation is more sensitive to the imperfect channel estimation in the Rician channel model, which leads to the larger performance degradation than that of the QPSK modulation.

Emulation time for all presented BER measurements is 64 hours.

5.2 Throughput

The throughput of the Spine DSP core depends on the number of users in the system K, baseband clock frequency f, oversampling rate x, datapath bitwidth b, and datapath parallelization p. Because both I and Q have the same bitwidth, the throughput of a single Spine is $2fbpK/x$ bits per second. By increasing datapath parallelism, which is a generator parameter, the throughput can scale up. For the specific implementation in this paper, the throughput is 6.4Gb/s.

Notice that due to the daisy-chain connection of Spines, the final throughput of the MRC beamforming stage doesn't scale up with the number of channels in the system, and is the

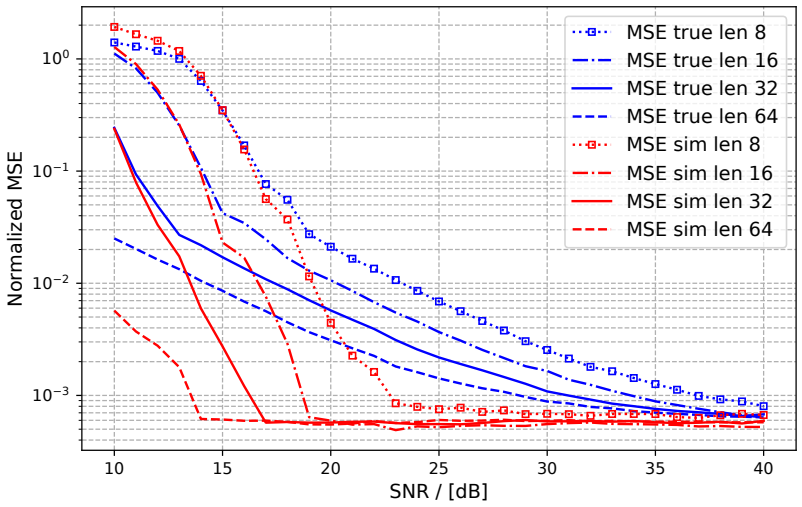

Figure 5.2: Frequency flat normalized channel estimation MSE vs SNR with different Golay pilot length. Blue lines show the normalized MSE of the true channel matrix and the channel estimation in the FPGA; red lines show the normalized channel estimation MSE of the simulator and FPGA.

same as a single Spine. Compared with the LuMaMi whose interconnection bandwidth is proportional to the number of antennas at the front-end [10], our work sigifficantly relaxes the interconnect requirements.

5.3 Power estimation

The power of the implementation on Xilinx VCU118 is estimated using Xilinx Power Estimator. The estimated power for each single Spine FPGA implementation is about 5.4W. The power breakdown is shown in Fig.5.4. I/O, signals, logic, and DSP contribute to most of the dynamic power consumption. This design can be mapped onto an ASIC for further power savings.

5.4 Summary

Table 5.1 summarizes several state-of-the-art massive MIMO BS implementations. Unlike [13] and [10] which include proprietary external FPGA modules, the design, the Spine generator is written in Chisel and is open-source [5]. This leads to its high portability to different hardware platforms, either ASIC or FPGA, by easily changing the system and hardware parameters. Compared to other narrowband OFDM designs in [10] and [9], our work supports much higher signal bandwidth, since the bandwidth of the carrier recovery algorithm

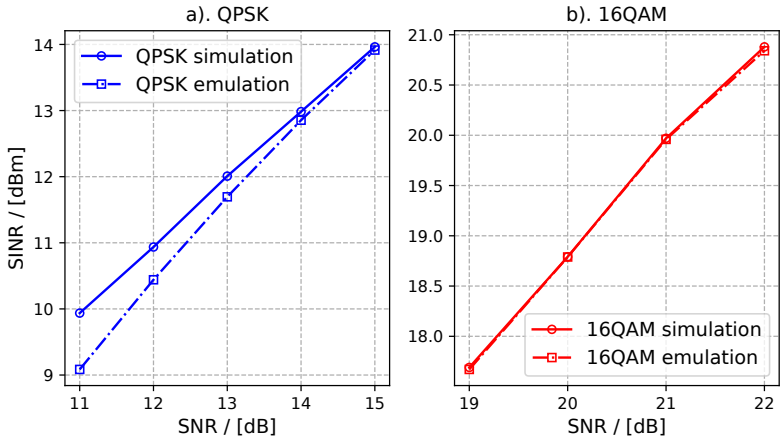

Figure 5.3: SINR versus SNR for 32 antenna 2 users system simulation and emulation results for a). QPSK modulation scheme and b). 16-QAM scheme.

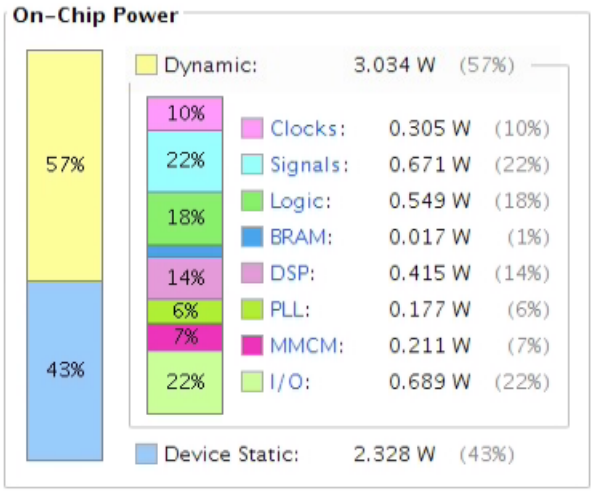

Figure 5.4: Power breakdown.

in the Tail is not limited by the OFDM subcarrier bandwidth by using QAM payloads. The distributed processing and daisy-chain architecture in our work relaxes the interconnection requirements and improves the system scalability. Compared with the GPU implementation in [9] which involves amount of memory access overhead, our work is specialized and much more power-efficient.

Table 5.1: Comparison

	This work	LuMaMi[10]	RIVF'19 [13]	JETCAS'17 [9]
Platform	FPGA/ASIC	FPGA	FPGA	GPU
Portable	Yes	No	No	No
Modulation	QAM	OFDM	QAM	OFDM
Distributed	Yes	No	No	Yes
Bandwidth	200MHz[1]	20MHz	935MHz[2]	40MHz
Method	Two-stage BF	MRC/ZF/RZF	SOR	ADMM
CHEST[3]	Yes	Yes	No	Yes
Power	5.4W[1]	-	-	235W[4]

[1] The signal bandwidth is the FPGA implementation bandwidth. The power estimation are based on the FPGA implementation of a single Spine with parameters shown in Table 4.1.
[2] This is the maximum FPGA implementation clock frequency, not the signal bandwidth. The signal bandwidth isn't given.
[3] Short for channel estimation.
[4] This is the thermal design of the Tesla K40 GPU with fully utilization.

Chapter 6

Conclusion

6.1 Conclusion

This paper presents the design and the emulation results of a scalable, highly portable, and power-efficient massive MIMO uplink baseband processing generator - the Spine generator. The Spine generator is parameterized across a range of MIMO system and the datapath hardware parameters, and specific instances were emulated on the FPGA.

The Spine generator is parameterizable on the number of users, the number of channels, Golay pilots design, delay estimation precision, and datapath bitwidth and parallelism. An 4-channel 8-bitwidth 8-way parallelism Spine is implemented on VCU118, and integrated with the python golden model for controlling and evaluaton. The FPGA emulation supports up to 200MHz signal bandwidth. The BER and SINR under different channels with different modulation schemes are evaluated. Both the frequency flat i.i.d channel and the Rician channel with QPSK and 16-QAM modulation schemes are evaluated. The results show the performance of channel estimation with different Golay pilot lengths, and the functionality of the Spine generator. At SNR 11.9dB for the QPSK modulation and 19dB for the 16-QAM modulation, the BER reaches 10^{-3} in the flat i.i.d channel. The system can operate up to 6.4Gb/s with power estimation 5.4W. The parallelization and pipelining of the configurable datapath and the distributed processing architecture enable the high throughput of the Spine without putting pressure on the clock frequency or scaling up with the number of channels in the system.

6.2 Future Work

Channel estimation improvement

The channel estimation evaluation result shows that when SNR is low, channel estimation residuals increase drastically due to both channel noises and the quantization error. This leads to the performance degradation of the Spine generator. In order to improve the channel

estimation performance, some other channel estimation algorithms can be tried. Beamspace channel estimation (BEACHES) can be applied to achieve this goal. BEACHES is a novel algorithm which adaptively denoises the channel vectors in the beamspace domain using an adaptive shrinkage procedure [12]. After integrating BEACHES to the Spine generator, the channel estimation performance should be improved.

System integration

Currently, the Spine generator emulation only uses the data generated by the python golden model. In the next step, the Spine generator needs to be integrated into the discrete Hydra testbed [7], which includes receiver modules with antenna, ADCs, and FPGAs. All FPGAs are connected with the daisy-chain architecture with high speed serial lanes. All FPGAs are connected to the host through the Ethernet. By integrated the Spine generator with the discrete Hydra testbed, the functionality and the performance of the Spine generator can be evaluated using real-time data.

Bibliography

[1] A.Goldsmith. "5G and beyond: What lies ahead for wireless system design". In: *PIMRC*. 2014.

[2] K. S. Ahn et al. "Performance analysis of maximum ratio combining with imperfect channel estimation in the presence of cochannel interferences". In: *IEEE Transactions on Wireless Communications* 8.3 (2009), pp. 1080–1085.

[3] J. Bachrach et al. "Chisel: Constructing hardware in a Scala embedded language". In: *Design Automation Conference*. 2012, pp. 1212–1221.

[4] S. A. Busari et al. "Millimeter-Wave Massive MIMO Communication for Future Wireless Systems: A Survey". In: *IEEE Communications Surveys Tutorials* 20.2 (2018), pp. 836–869.

[5] *Chisel3*. `https://github.com/chipsalliance/chisel3`. Accessed on 2020-02-20.

[6] Enrique García et al. "Efficient filter for the generation/correlation of Golay binary sequence pairs". In: *International Journal of Circuit Theory and Applications* 42.10 (2014), pp. 1006–1015.

[7] G. LaCaille et al. "Design and Demonstration of a Scalable Massive MIMO Uplink at E-Band". In: *2020 IEEE International Conference on Communications Workshops*. 2020, pp. 1–6.

[8] E. G. Larsson et al. "Massive MIMO for next generation wireless systems". In: *IEEE Communications Magazine* 52.2 (2014), pp. 186–195.

[9] Kaipeng Li et al. "Decentralized Baseband Processing for Massive MU-MIMO Systems". In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 7.4 (2017), pp. 491–507.

[10] S. Malkowsky et al. "The World's First Real-Time Testbed for Massive MIMO: Design, Implementation, and Validation". In: *IEEE Access* 5 (2017), pp. 9073–9088.

[11] T. L. Marzetta. "Noncooperative Cellular Wireless with Unlimited Numbers of Base Station Antennas". In: *IEEE Transactions on Wireless Communications* 9.11 (2010), pp. 3590–3600.

[12] Seyed Hadi Mirfarshbafan et al. "Beamspace Channel Estimation for Massive MIMO mmWave Systems: Algorithm and VLSI Design". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 67.12 (2020), pp. 5482–5495.

[13] C. Nhat Cuong et al. "Hardware Implementation of the Efficient SOR-Based Massive MIMO Detection for Uplink". In: *2019 IEEE-RIVF International Conference on Computing and Communication Technologies*. 2019, pp. 1–6.

[14] B. Nikolic et al. "Generating the Next Wave of Custom Silicon". In: *IEEE 44th European Solid-State Circuits Conference*. 2018, pp. 6–11.

[15] B. Nikolić. "Simpler, more efficient design". In: *IEEE 41st European Solid-State Circuits Conference*. 2015, pp. 20–25.

[16] F. Rusek et al. "Scaling Up MIMO: Opportunities and Challenges with Very Large Arrays". In: *IEEE Signal Processing Magazine* 30.1 (2013), pp. 40–60.

[17] O. Shacham et al. "Rethinking Digital Design: Why Design Must Change". In: *IEEE Micro* 30.6 (2010), pp. 9–24.

[18] Clayton Shepard, Hang Yu, and Lin Zhong. "ArgosV2: A Flexible Many-Antenna Research Platform". In: *MobiCom* (2013), pp. 163–166.

[19] Clayton Shepard et al. "Argos: practical many-antenna base stations". In: *MobiCom* (2012).

[20] A. Wang, J. Bachrach, and B. Nikolié. "A generator of memory-based, runtime-reconfigurable 2N3M5K FFT engines". In: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2016.

[21] Angie Wang et al. "ACED: A Hardware Library for Generating DSP Systems". In: *Proceedings of the 55th Annual Design Automation Conference*. DAC '18. San Francisco, California, 2018.