Spring 4-25-2022

A Reconfigurable FPGA Overlay Architecture for Matrix-Matrix Multiplication

Zihao Chen

Washington University in St. Louis

McKelvey School of Engineering

Department of Computer Science and Engineering

Thesis Examination Committee:
Roger Chamberlain, Chair
Anthony Cabrera
Shantanu Chakrabartty

A Reconfigurable FPGA Overlay Architecture for Matrix-Matrix Multiplication

by

Zihao Chen

A thesis presented to the McKelvey School of Engineering
of Washington University in St. Louis in partial fulfillment
for the degree of Master of Science

May 2022
Saint Louis, Missouri

Table of Contents

List of Tables

List of Figures

Acknowledgments

My wonderful parents Xiaosong Chen and Rong Li have been the constant inspiration and source of power throughout my life. I am grateful for what they have done for me.

I would like to thank Dr. Roger Chamberlain, my academic advisor throughout my undergraduate and graduate study here at WashU and, for his guidance and support when I lost direction. I also appreciate the support from Dr. Shantanu Chakrabartty and my lab mates during this last semester. I could not accomplish this without their help.

A special thank goes to my college friends. I could not imagine how hard these past two years would be without you. I hope you all the best.

I want to dedicate this work to my grandfather, Shouqin Chen, who passed away during the pandemic. The memories we shared will never fade.

Zihao Chen

Washington University in St. Louis
May 2022

ABSTRACT OF THE THESIS

A Reconfigurable FPGA Overlay Architecture for Matrix-Matrix Multiplication

by

Zihao Chen

Master of Science in Computer Engineering

Washington University in St. Louis, 2022

Research Advisor: Professor Roger Chamberlain

The increasing popularity of deep learning in workloads across vision, speech, and language has inspired many attempts to develop hardware accelerators for matrix-matrix multiplication. Both application specific integrated circuits (ASICs), and field-programmable arrays (FPGAs) are used for this purpose. However, a trade off between the two platforms is that ASICs provide little flexibility after they are manufactured while designs on FPGAs are flexible but application development on FPGAs is more time-consuming.

In this work, we aim to find the balance between reconfigurability and development efficiency by designing a reconfigurable systolic architecture as an overlay on the FPGA. Our contribution to the reconfigurable systolic architectures is a multiplexer based crossbar network that interconnects every processing element in the network. The crossbar network grants user run-time reconfigurability of the topology of the systolic array, enabling the user to specify the shape and size of the systolic architecture on-the-fly. The proposed overlay architecture achieves similar computational hardware resource usage and maximum clock frequency compared to the baseline designs.

Chapter 1

Introduction

General matrix-matrix multiplication (GEMM) operations are the key computational kernel in many modern day workloads, especially in the deep learning domain. Many attempts have been made to build hardware accelerators for GEMM operation ranging from application-specific integrated circuits (ASICs), such as the Nvidia Volta GPU [16] and Google's TPU [6] to field-programmable arrays (FPGAs) [1, 15]. Most of the state-of-the-art GEMM accelerators resemble traditional systolic array architectures, due to their high efficiency in processing parallel and regular data flow. However, types of GEMM workloads in real world workloads such as convolutional neural networks (CNN) are not always regular. They vary in size and sparsity from instance to instance. While ASICs usually are more performant and energy-efficient [12], they lack the flexibility to accommodate a fast-changing data flow. Designs based on FPGAs are reconfigurable, but its development cycle is slow. Coarse grain reconfigurable arrays [12] and FPGA overlays are state-of-the-art solutions that mitigate the slow development cycles on FPGAs while retaining the flexibility to some extent. Our work aims to utilize the FPGA overlay idea to build a dynmically reconfigurable systolic architecture.

Many FPGA-based overlay designs [5, 13] grant users the ability to reconfigure the circuit at run-time after loading the bit file onto the FPGA. We adopt a similar idea in developing our overlay systolic architecture that allows a user to dynamically adjust the topology and function of the systolic array. We design a crossbar interconnect network that replaces the neighbor-to-neighbor connectivity in traditional systolic arrays. The crossbar network can provide arbitrary connection from one processing element to another, and the user can transfer a new topology to the crossbar network quickly at run-time.

This work designs an overlay systolic architecture that is specifically built for handling flexible matrix-matrix multiplication, and also details the structure of the key components of the overlay architecture. The performance of our overlay architecture is then compared against the performance of the traditional systolic array system, the timing characteristics, and hardware resource utilization. The contributions of this work are the following:

- we design a suite of baseline systolic array designs that are used as comparison to our overlay architecture;
- we design and implement the overlay architecture;
- we demonstrate that the overlay architecture consumes little resource overhead versus the traditional design while providing run-time reconfigurability;
- we describe an approach to mitigating the performance degradation of implementing run-time reconfigurability.

The outline of the thesis is as follows. Chapter 2 introduces background knowledge of the FPGA-overlay systems and related work. Chapter 3 describes the architecture of the baseline systolic arrays and the overlay architecture. Chapter 4 provides the hardware utilization and timing analysis that comprise the comparison between the baseline and the overlay architecture. Chapter 5 draws conclusions for our reconfigurable systolic architecture and points out directions for future work.

Chapter 2

Background and Related Work

We will first describe approaches to reconfigurability based on FPGA overlays. This is followed by a description of systolic arrays and their reconfiguration.

Wilson et al. [18] proposed a novel FPGA overlay-based application development approach called Seiba. Seiba integrates an HLS-generated fixed circuit with the reconfigurable overlay logic to grant a developer fast design iterations at run-time. The overlay architecture consists of three parts: a fixed, user circuit synthesized from HLS, a soft processor that provides reconfigurability on-the-fly, and an overlay execution manager. The HLS-generated user circuit executes the initial design with no dynamic reconfigurability and the execution manager only enables the user circuit. In later design iterations, when the user wants to incrementally modify the functionality of the design, the overlay execution manager enables the soft-processor and redirects the flow of execution to the programmable overlay to realize the incremental design changes dynamically. However, this study shows that supporting the run-time reconfigurability of both control and datapath incurs area overhead, which is also present in our design.

To reduce the circuitry redundancy that is necessary for run-time adjustments while preserving control of the configurability of the datapath, Coole et al. [2] introduce a family of overlay architectures called super nets that takes advantage of reusable netlists across multiple sources and a crossbar interconnect network that is similar to our design. The super net merges the netlists of the user circuits to formulate a superset of them. At run-time, user circuits are mapped onto the corresponding interconnects within the synthesized super nets network. Merging reusable circuits reduces the area cost of the overlay architecture. Additionally, the super net grants flexibility at run-time by integrating a secondary crossbar

interconnect network. Input netlists that cannot find a match in the super net interconnect network are routed through the crossbar network. The merge and match operation of the super net overlay along with the crossbar network can both reduce the area overhead of the synthesized circuit and expose dynamic flexibility to the users.

Other works demonstrate the wide variability of the FPGA overlay architectures but also show how FPGA-overlay systems are case-specific and thus, lack a universal standard for programmability, productivity and adaptability [12]. Jamal et al. [5] adopt a debug overlay on top of an HLS-generated circuit to provide functional flexibility for the developer to accelerate the debug cycles. Ng et al. [13] use a soft-processor overlay architecture to manage and control a hardware accelerator circuit, where a recompilation is needed to modify the hardware accelerator execution logic. Another work [19] provides flexible functionality and programmable datapath on-the-fly by reusing multiple modulated overlay building blocks recurrently. The above architectural designs of FPGA-overlay systems involves complex control and datapath designs which achieve very specific goals but suffer from loss of generality. This calls for a simple overlay design that both allows rapid development cycles and serves as building blocks for general computing workloads.

Systolic arrays map the data flow of high-performance computation onto hardware architectures, which leverages the advantages of both parallel processing and pipelining for data processing. Kung [8] introduces the advantage of systolic arrays in processing large amounts of regular and parallel data, and its reconfigurable datapath give rise to numerous interconnect topology that accommodates data flow of different types of workloads, as shown in Figure 2.1. Because of these benefits, systolic architectures have become popular basic computing blocks among many modern workloads such as HPC and DNN tasks. Recent research on FPGA overlay architecture specifically target systolic array-like circuits to expand functionality as well as flexibility.

Eyeriss [1] is a novel architecture inspired by traditional systolic arrays to accelerate convolutional neural network computation. In this work, they exploit the data locality in weights and activation matrices to minimize data movement among PEs and global memory. The layout of their PEs can adapt to different sizes of convolutional multiply and accumulate because the architecture utilizes the local storage of the PEs, the direct spatial connections between PEs, and PE-level parallelism. The design was prototyped on FPGA boards but

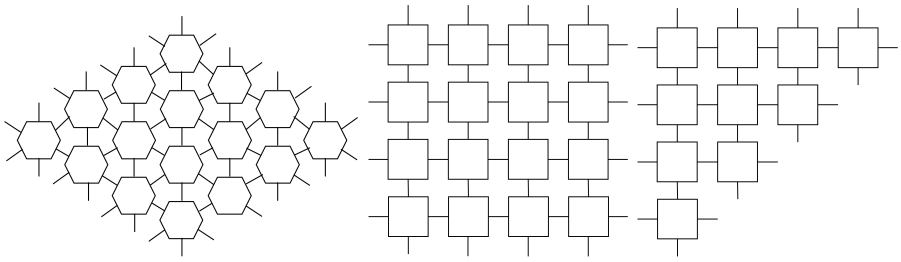

Figure 2.1: Three common types of systolic architectures.

later manufactured into ASICs. This differs from our reconfigurable architecture in that once it is manufactured, the datapath of the PE network becomes fixed logic circuit and thus, does not provide dynamic reconfigurability.

Another closely related work is by Qin et al. [15] on systolic architectures to accelerate DNN workloads. The authors argued that traditional systolic arrays can not perform general matrix-matrix multiplication on irregular or sparse matrices efficiently, which is a common operation in DNN workloads. They propose a novel accelerator architecture called SIGMA for handling irregular and unstructured sparse GEMM operations. Instead of the neighbor-to-neighbor connectivity on a traditional systolic array, SIGMA features a distribution network that can broadcast data over all PEs and a reduction network, called forwarding adder network, that performs the MAC operation based on the shape and sparsity of the original matrices. Although this design grants flexibility in the shape and sparsity of the input matrices, this work differs from our architecture in that the control knobs of the datapath are not exposed to the users.

Systolic arrays require a rhythmic data flow from the memory outside the computing fabrics, which calls for optimization of data movement outside of the systolic array. A recent study by Gao et al. [3] proposes an intra-layer parallelism together with inter-layer pipelining, where each layer can be a single systolic array. Their work aims to optimize the data flow outside the computation fabric that promotes data reuse and sharing as well as reduces pipeline delays. Since our design mostly focuses on implementing fast run-time reconfigurability of the computation units, this work may direct our future work to expand the current design to a full-on ASIC chip by integrating data flow optimization outside of the systolic array.

Chapter 3

Methods

To evaluate the resource utilization and timing characteristics of the proposed overlay architecture, we implement two baseline systolic arrays: a hexagonal array, described in [21], and a rectangular array, described in [17], as both a starting point and a control group to the proposed design. We implemented the baseline design and the overlay architecture separately onto the same target FPGA and examined the hardware utilization reports and timing analysis provided by the synthesis tool flow to evaluate the costs and benefits of the overlay design. The platform we use for the implementations of the baseline and the overlay architecture is the SP701 FPGA evaluation kit of Xilinx Spartan-7 family. To author the designs, we used the Vivado 2019.1 HLx Development Kit.

Algorithm 1 shows the naive matrix-matrix multiplication, where A and B are input matrices and C is the output matrix. The sequential execution flow of the original matrix-matrix multiplication algorithm does not take advantage of data-level concurrency and has repetitive loads on the same data in the memory. Systolic array architectures, on the other hand, map each iteration in the sequential execution onto a specific spatial and temporal location in the processing element network, which guarantees the correctness of the computation while enabling parallelism among iterations. Each data point loaded from the main memory gets utilized in each of the processing elements that it passes, thus preventing the bottleneck of fetching data from main memory.

Algorithm 1 Matrix Matrix Multiplication

```
1: int A[M][N], B[N][L]
2: int C[M][L]
3: for i ← 1 to M do
4:     for j ← 1 to L do
5:         for k ← 1 to N do
6:             C[i][j] += A[i][k] * B[k][j]
```

3.1 Building Blocks of Systolic Architectures

The fast reconfigurable systolic overlay architecture and the traditional systolic architectures share similar overall structures as shown in Figure 3.1. While the topology of the connectivity among processing elements varies between overlay and traditional architectures, there are circuits that remain unchanged, so we use several basic building blocks shared across the designs. Since this work primarily focuses on systolic array architectures for matrix-matrix multiplication, we use multiply-and-accumulate processing elements as one of the basic building blocks. The data transfer between the main memory and the systolic architecture, is outside the scope of this work, so we implemented linear feedback shift registers that generate pseudo-random numbers to emulate data fetched from the memory hierarchy. The designs of the processing elements and linear feedback shift registers are the same for both overlay and traditional systolic architectures for the sake of evaluation.

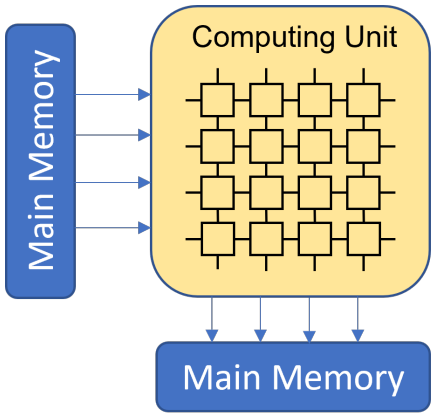

Figure 3.1: General architecture of systolic arrays

The Processing Element

The processing element is simply the multiply and accumulate unit that performs the

$$c_{ij} = \sum_{k=1}^{K} a_{ik} * b_{kj}$$

computation. The module definition of the RTL circuit for the processing element is shown in Listing 3.1. The parameter DATA_SIZE defines the width of the input and output elements for matrix A and B. The data width for matrix C is 2*DATA_SIZE + 1 to avoid overflow after multiply and accumulate operations. Inputs a_in, b_in, and c_in denote the data of matrix A, B, and C respectively that pass through the systolic array architecture into the processing element and outputs a_out, b_out, and c_out denote data of A, B, and C matrix after the processing element finished the computation.

```
module pe #(parameter DATA_SIZE = 16)
    (
        input clk, // input clk
        input reset_l, // reset signal
        // input values from neighboring PEs
        input [DATA_SIZE-1:0] a_in ,
        input [DATA_SIZE-1:0] b_in ,
        input [DATA_SIZE*2:0] c_in ,
        // output values to neighboring PEs
        output reg [DATA_SIZE-1:0] a_out ,
        output reg [DATA_SIZE-1:0] b_out ,
        output reg [DATA_SIZE*2:0] c_out
    );
```

Listing 3.1: Port declaration of processing element.

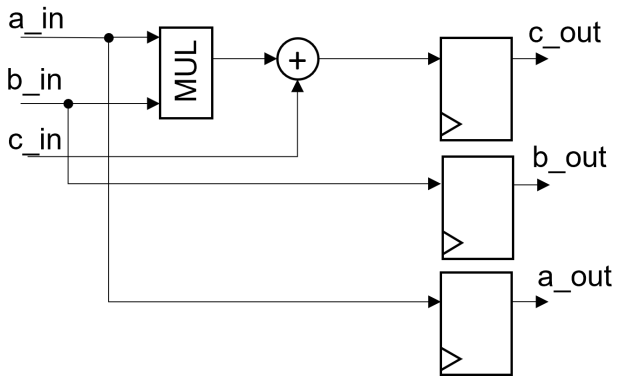

Figure 3.2: Block diagram of processing element.

```
always @(posedge clk) begin
    if (!reset_l) begin
        a_out <= 0;
        b_out <= 0;
        c_out <= 0;
    end
    else begin
        c_out <= c_in + b_in * a_in;
        a_out <= a_in ;
        b_out <= b_in ;
    end
end
```

Listing 3.2: RTL description of processing element.

The block diagram for the processing element is shown in Figure 3.2. When clock is at its rising edge and reset low signal is high, a_in, and b_in first pass through a multiplier. The intermediate product of a_in and b_in is then summed with c_in, which is then latched into c_out. a_in and b_in are latched into a_out and b_out without any changes applied to them. All three outputs will be set to zero when reset low signal is asserted. The RTL description of the circuit shown in Figure 3.2 is authored in Verilog as in Listing 3.2.

The Linear Feedback Shift Register

This work focuses on developing an innovative interconnect topology that grants traditional systolic architectures run-time reconfigurability. Implementing an interface with the main memory that enables direct data transfer to the computing unit is outside the scope of this

study. On the other hand, exposing the inputs and outputs of the systolic architectures significantly stresses the I/O resources on the FPGA board. Thus, for simulation purposes, we implemented a pseudo-random number generator through a simple linear feedback shift register circuit as shown in Figure 3.3.

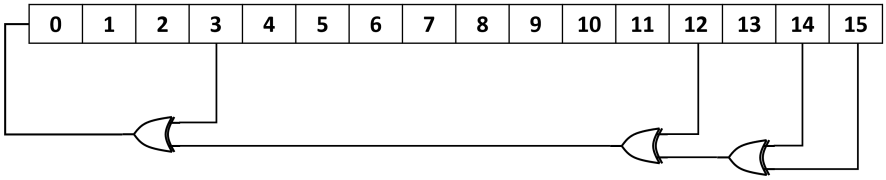

Figure 3.3: Block diagram of linear feedback shift register.

A linear feedback shift register is a shift register that, at the rising edge of the clock, advances the data through the register from one bit to the next most-significant bit. To form a feedback, some of the bits are wired through an exclusive-OR circuit. A linear feedback shift register takes the result of two or more registers after passing through the exclusive-OR and feeds it to one of the registers. The feedback mechanism of this kind of shift register generates a new equal-length signal at each clock cycle, if it is given a non-zero initial value. The exclusive-OR configuration also affects the number of possible numbers that a linear feedback shift register can generate. Peterson and Weldon [14] have compiled the exclusive-OR configurations that guarantee maximal-length linear feedback shift registers of different bit sizes, and we use the 16-bit configuration from their table in our implementation.

Note that in our implementation of the linear feedback shift register as shown in Listing 3.3 has an enable signal. Our overlay architecture enables transition between different sizes and types of systolic architectures, which may require different data transfer patterns. The spatial and temporal parallelism of the sequential execution flow of the traditional matrix-matrix multiplication can only be mapped onto systolic architectures when the correct data is clocked in at the correct time point and the correct processing element. Since the linear feedback shift register generates new data every clock cycle, we use the enable signal to control the pattern of the outgoing data flow. The enable signal is implemented in the top RTL module as a counter that accommodates the data rate of different configurations.

```
module lfsr#(parameter DATA_WIDTH = 16)
(
  input  clk,
  input  rst_l,
  input  en,

  output reg [DATA_WIDTH-1:0] data
);
wire newData ;

assign newData = data[15] ^ data[14] ^ data[12] ^ data[3];


always @(posedge clk) begin
    if (!rst_l || !en) begin
        data <= 16'b10086;
    end
    else begin
        data <= {data[15:1], newData} ;
    end
end

endmodule
```

Listing 3.3: RTL description of linear feedback shift register.

3.2 Baseline Systolic Architectures

Traditional systolic architectures are often two-dimensional, and they can be rectangular, triangular, or hexagonal to take advantage of higher degrees of parallelism [8]. Our overlay systolic architecture allows fast conversion between different types of traditional systolic arrays. We implemented two baseline systolic array architectures as benchmarks for the overlay architecture. One of them is the rectangular array, which is efficient at processing regular data flow. In terms of matrix-matrix multiplication, rectangular systolic architectures are used to compute dense and regular matrices [9]. Hexagonal systolic architectures are good at processing banded matrices, where the upper and lower triangular region are all zeros [10].

Rectangular Systolic Architecture

The execution flow of a 3 by 3 matrix-matrix multiplication on a rectangular systolic array is shown in Figure 3.4. Elements in matrix A and B are denoted as $a_{i,k}$ and $b_{k,j}$. At the first clock cycle, $a_{0,0}$ and $b_{0,0}$ enter the upper leftmost processing element, and the product of $a_{0,k}$ and $b_{k,0}$ is one of the three terms that sum up to $c_{0,0}$. As a result, the product of $a_{0,0}$ and $b_{0,0}$ is buffered in the processing element waiting for later data to come. At the second clock cycle, $a_{0,0}$ is passed to the current processing element's horizontal neighbor on the right and $b_{0,0}$ is passed to the bottom neighbor. Four new data are fetched on the same clock cycle into different processing elements: $a_{0,1}$ and $b_{1,0}$ to the upper leftmost, $a_{1,0}$ to the middle leftmost, and $b_{0,1}$ to the upper middle. Each of the three active processing elements that receives new data performs the same multiply and accumulate operation, where the middle leftmost computes $c_{1,0}$, and the upper middle computes $c_{0,1}$. Note that $a_{0,1}$ and $b_{1,0}$ get delayed for one clock cycle comparing to $a_{0,0}$ and $b_{0,0}$ to account for the pipeline delay. This ensures that every pair of signals that need to be multiplied together arrives at the same processing element at the correct clock cycle. The rest of the execution flow follows the description of the first clock cycle. After all the elements of matrix A and B pass through the systolic array, each of the processing elements will contain one element of the product matrix C, where the upper leftmost contains $c_{0,0}$, the upper middle contains $c_{0,1}$, the middle leftmost contains $c_{1,0}$, and likewise for the remaining processing elements.

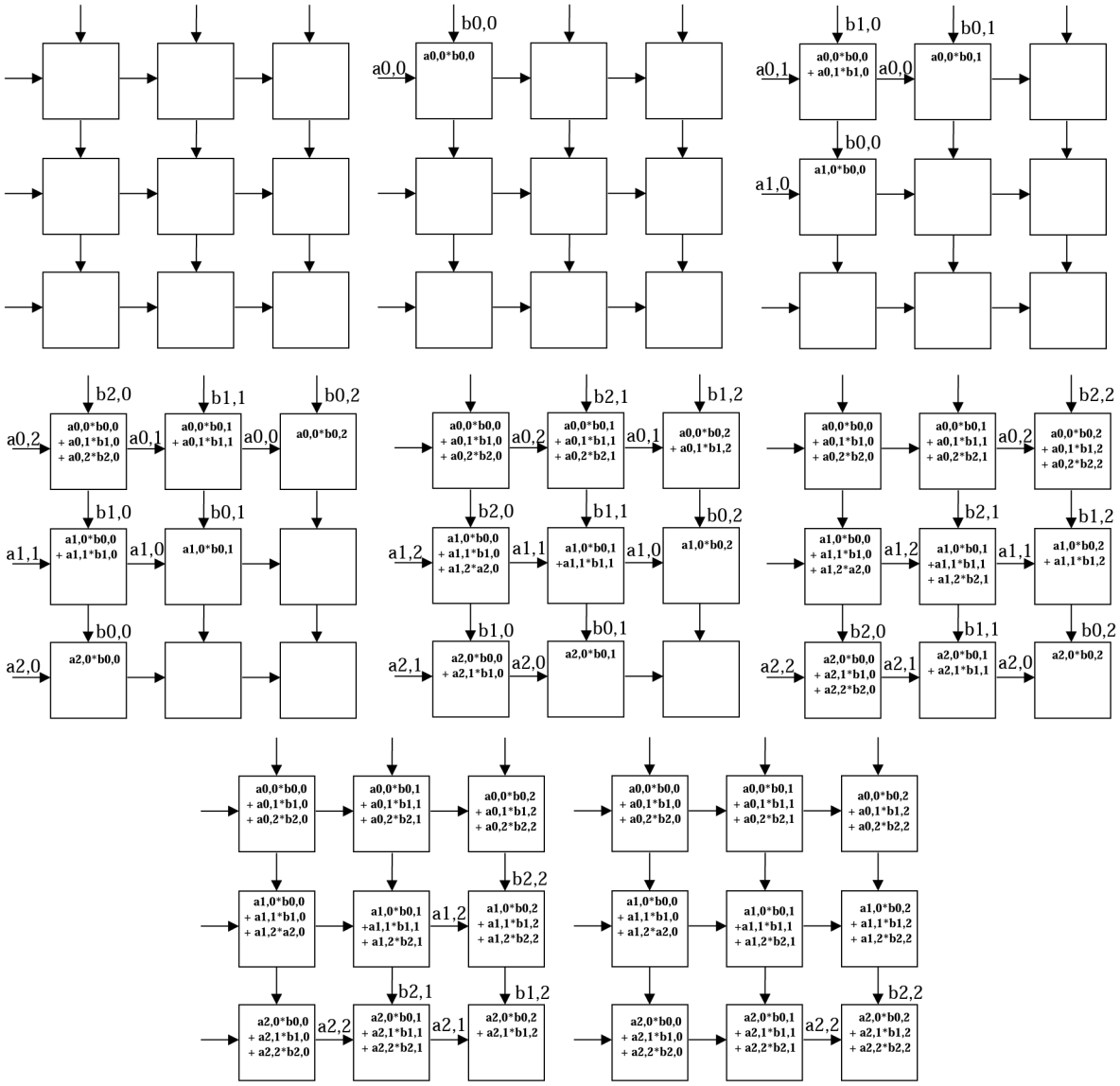

Figure 3.4: Data flow of a 3 by 3 rectangular systolic architecture[7].

Hexagonal Systolic Architecture

While rectangular systolic architectures are suitable for full and regular matrix-matrix multiplication, the strength of its hexagonal counterpart lies in banded matrix-matrix multiplication [10]. Figures 3.5 and 3.6 show the execution flow of two banded matrices A and B, each with bandwidth 3 multiplied together to get the product matrix C. Banded matrices differ from full matrices in that they have upper and lower triangular area equal to zero. To better illustrate the computing flow as shown in Figures 3.5 and 3.6, the matrix-matrix multiplication algorithm

$$c_{ij} = \sum_{k=1}^{K} a_{ik} * b_{kj}$$

can be mapped to a temporal recurrence that can be efficiently implemented in pipelines of the systolic array.

$$c_{ij}^{0} = 0,$$

$$c_{ij}^{t+1} = c_{ij}^{t} + a_{it} * b_{tj}$$

where the t indices, denoting the time instance of each of the computations, are the temporal mapping of the k indices in the original algorithm.

Different from the data flow of the rectangular systolic architectures, the A, B, and C matrices are pumped through the network in three distinct directions synchronously. In this study, the product matrix C is initialized with zeros and each of its elements passes through the systolic network from the bottom. However, each of the c_{ij} can be treated as partial sums from computation outside the systolic array, and does not necessarily have to equal zero.

The A matrix passes through the systolic array from the top left to the bottom right, B matrix passes from top right to bottom left, and C passes from bottom to top, as indicated from the arrows in Figures 3.5 and 3.6. At the first time instance, $a_{0,0}$, $b_{0,0}$, and $c_{0,0}$ are fetched from the memory and enter the systolic architecture. As shown in Figure 3.5, they reside in different processing elements so no computation is performed on them. At the next instance of time, two additional data points are fetched into the systolic architecture, $a_{1,0}$, $b_{0,1}$. Elements $a_{0,0}$, $b_{0,0}$, and $c_{0,0}$ also propagate to the next processing element along their respective direction, where they meet and perform the multiply and accumulate operation. At the third clock cycle, each of the data points that reside in the systolic array propagates along its designated

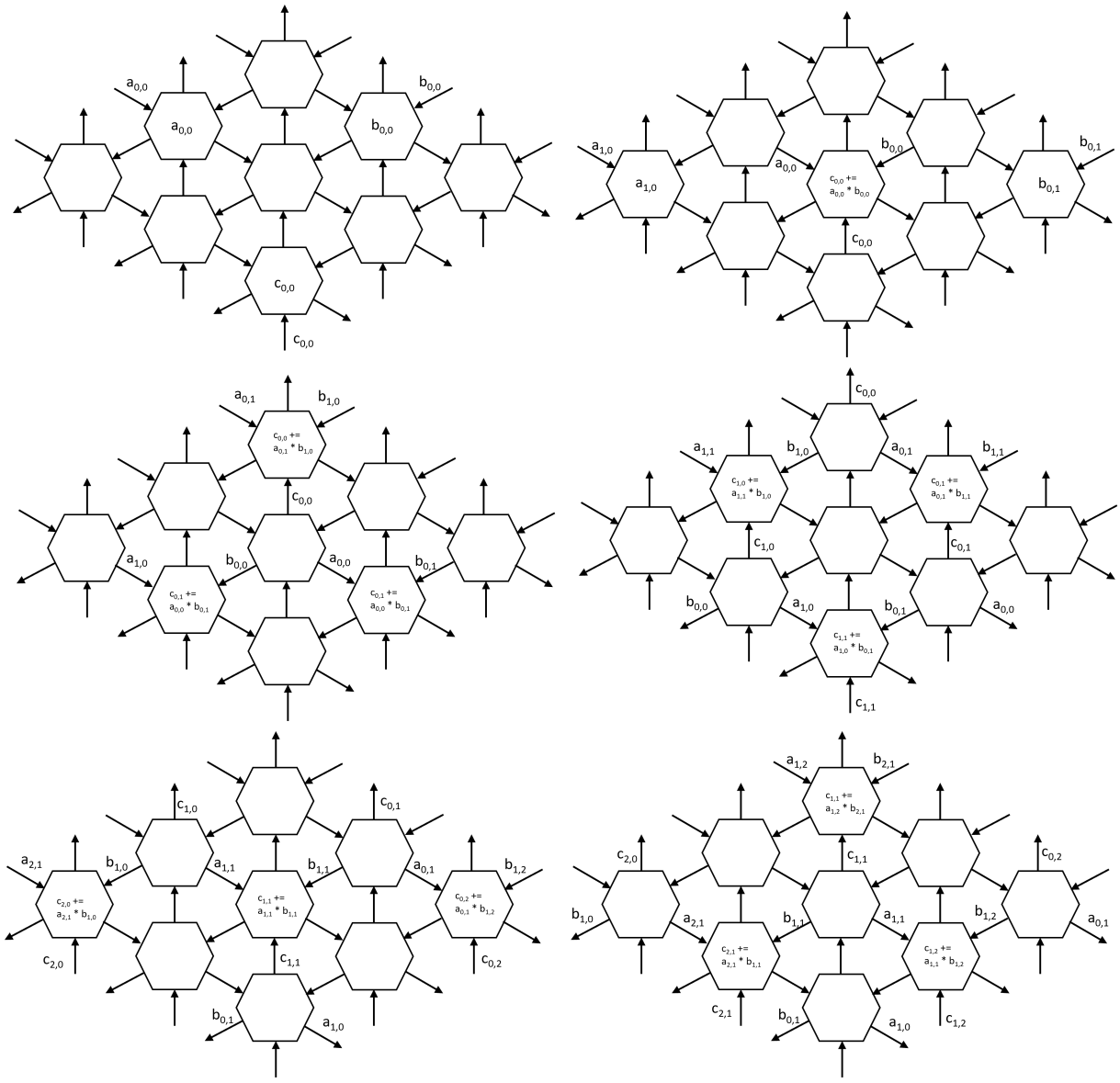

Figure 3.5: Data flow of a 3 by 3 hexagonal systolic architecture.

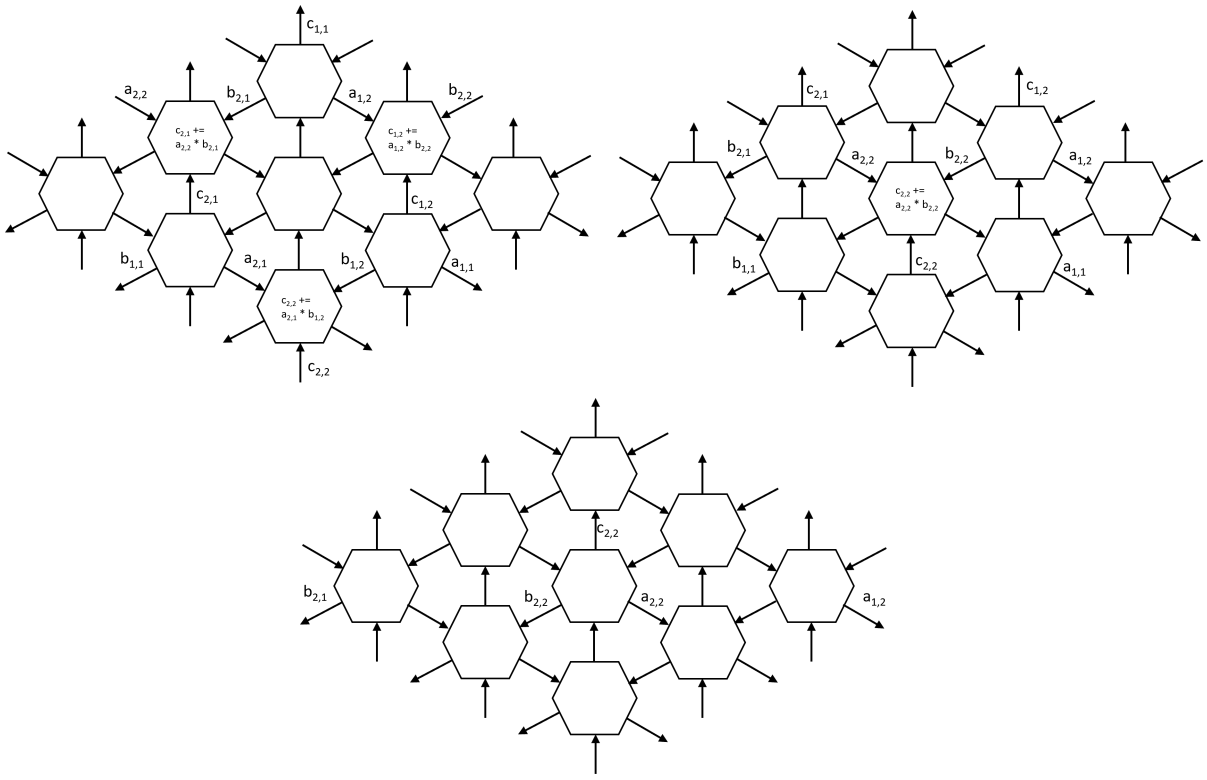

Figure 3.6: Data flow of a 3 by 3 hexagonal systolic architecture (cont.).

direction to the next processing elements and two additional data points $a_{0,1}$ and $b_{1,0}$ are fetched from the periphery. Three of the nine processing element are active and performing computation on the values received as shown in Figure 3.5. The rest of the data flow follows the pattern of the first three clock cycles. Different from the rectangular systolic array, when the computation finishes, the hexagonal architecture does not store the resulting C matrix in each of processing elements' local register but rather gets flushed out from the systolic array. For example, at the fourth clock cycle, $c_{0,0}$ obtains its final result and, as indicated by the upward arrow, is flushed into the memory outside the systolic architecture. Another difference from the rectangular systolic architectures is that the hexagonal architectures are not limited by the size of the matrices that passes through the array but the bandwidth of them, which grants additional flexibility.

3.3 Flexible overlay systolic architecture

Our overlay systolic architecture differs from the traditional systolic arrays in that it allows dynamic reconfiguration of the size and type of the systolic architecture after the bit file is loaded on the FPGA board. The data path of the traditional systolic architecture becomes fixed upon compilation, but our architecture contains a crossbar interconnect network that can take configuration bits at run-time to replace the fixed data path in normal systolic arrays. Thus, the topology of the interconnect network that we designed can be adjusted with user-defined configuration files dynamically. Since the building blocks of the traditional and our overlay systolic architecture are the same, changing the connectivity of the network that connects all processing elements is sufficient for changing the functionality of the systolic architecture. With 25 processing elements, our overlay architecture allows dynamic changes from five-by-five to a four-by-six array, from one five-by-five to one four-by-four and one three-by-three array, and from rectangular to hexagonal arrays.

Crossbar interconnect network

In our overlay systolic architecture, we implement a N-by-N crossbar network, where N denotes the number of processing elements. As shown in Figure 3.7, the crossbar network

allows arbitrary connectivity from one processing element to another. As we have shown in previous sections, data flow of the input matrices A and B and the output matrix C pass through the systolic array in three different directions. To realize N-to-N connectivity, three sets of the same N-by-N crossbar network are needed.

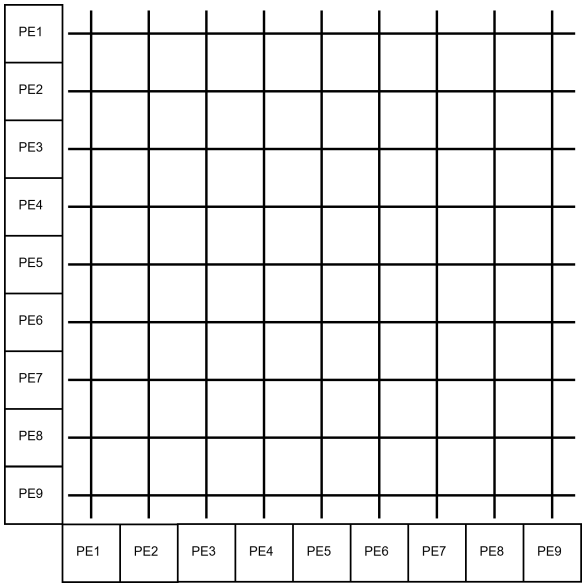

Figure 3.7: A 3-by-3 crossbar network.

The crossing points in the network serve as switches that permit data transfer between specific pairs of processing elements and avoid undesirable data passing from one processing element to another. The switches are implemented as N+M-to-one multiplexers as shown in Figure 3.8, where N denotes the number of processing elements used and M denotes the bandwidth of memory interface and the systolic architecture. The multiplexer takes inputs from the output ports of the processing elements 1 through N and the data directly from memory ports 1 through M. The value of M depends on the size and the topology of the systolic array. The edge processing elements receive data directly from the memory, so for a three-by-three processing element, the The reason for having the direct connectivity with the memory interface is that this overlay architecture grants arbitrary reconfigurability in terms of systolic array topology, so any processing element should be able to directly receive data from the memory. The select bits of the multiplexers are read from the configuration specified by the user, and the outputs of the multiplexers are fed to the input ports of the target processing elements. This circuit realizes connectivity from arbitrary processing

elements to the target processing elements based on the configuration bit. N sets of the multiplexer circuit implement the crossbar network.

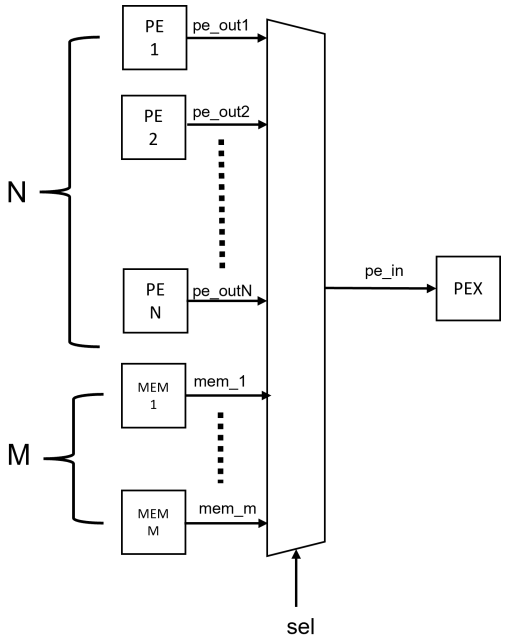

Figure 3.8: Implementation block diagram of the crossbar network.

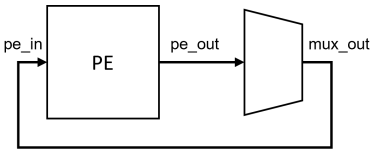

Figure 3.9: Feedback connection for C matrix data flow in rectangular configuration.

One thing worth noting about the multiplexer based crossbar network is the difference between rectangular and hexagonal topology. As mentioned earlier, three sets of crossbar network are used to implement full connectivity between processing elements. However, the rectangular systolic architecture differs from its hexagonal counterpart in that only input matrices A and B pass through the systolic arrays. As a result, for the rectangular topology, only two sets of the crossbar network are needed for the data transfer of matrix A and B. To accommodate this architectural discrepancy while preserving the ability to switch between rectangular array and hexagonal array, the overlay architecture still contains three sets of crossbar networks but when the user configures the overlay into rectangular arrays, the crossbar network for C matrix forms a feedback from the output port of the processing element

to the input port itself as shown in Figure 3.9. Thus, C matrix will not propagate through the systolic architecture as in the hexagonal configuration, and the latch in the processing elements will become a register that stores the result of the matrix-matrix multiplication.

Reading in setup bits

To configure the topology of the systolic architecture, we build a finite state machine for reading in the select bits for each of the multiplexer that implements the crossbar network. Inside the overlay architecture, the setup bits are stored in arrays of registers so that the topology remains unchanged unless the user specifies a new topology and flushes the old configuration. The simple finite state machine requires a simple data bus interface, which is used for data transfer from outside into the systolic architecture. The bus interface is shown in Listing 3.4

```
1 module flex_sysarray #(parameter DATA_WIDTH = 16, ARRAY_SIZE = 16)
2     (
3         input clk,
4         input reset_l,
5 
6         input [DATA_WIDTH-1:0] wrAddr,
7         input [DATA_WIDTH-1:0] wrData,
8         input wr
9     );
```

Listing 3.4: RTL description of the simple data bus interface.

Chapter 4

Results

In this chapter we compare the hardware utilization and the timing characteristics between the baseline systolic architectures and the overlay architecture. The results are based on a five-by-five overlay architecture, which can be dynamically reconfigured in both hexagonal and rectangular topology and transformed into one four-by-four and one three-by-three systolic arrays. The traditional systolic array baseline designs that are used to draw comparisons are designed accordingly.

4.1 Hardware Utilization

We compare the hardware resource utilization of the various designs we implemented as generated by Vivado HLx 2019.1 synthesis tool. We first report the hardware utilization of the baseline designs, including the three-by-three, four-by-four, and five-by-five traditional systolic array both in rectangular and hexagonal setup.

Table 4.1: The hardware utilization of three-by-three rectangular systolic array.

Resource	Utilization	Available	Utilization%
LUT	1	64000	0.01
DSP	13	160	8.13
IO	35	400	8.75
BUFG	1	32	3.13

Table 4.2: The hardware utilization of four-by-four rectangular systolic array.

Resource	Utilization	Available	Utilization%
LUT	1	64000	0.01
FF	1	12800	0.01
DSP	24	160	15
IO	35	400	8.75
BUFG	1	32	3.13

Table 4.3: The hardware utilization of five-by-five rectangular systolic array.

Resource	Utilization	Available	Utilization%
LUT	3	64000	0.01
FF	3	12800	0.01
DSP	37	160	23.13
IO	35	400	8.75
BUFG	1	32	3.13

Table 4.4: The hardware utilization of three-by-three hexagonal systolic array.

Resource	Utilization	Available	Utilization%
LUT	1	64000	0.01
DSP	10	160	6.25
IO	35	400	8.75
BUFG	1	32	3.13

Table 4.5: The hardware utilization of four-by-four hexagonal systolic array.

Resource	Utilization	Available	Utilization%
LUT	1	64000	0.01
FF	1	12800	0.01
DSP	18	160	11.25
IO	35	400	8.75
BUFG	1	32	3.13

Table 4.6: The hardware utilization of five-by-five hexagonal systolic array.

Resource	Utilization	Available	Utilization%
LUT	3	64000	0.01
FF	3	12800	0.01
DSP	28	160	17.5
IO	35	400	8.75
BUFG	1	32	3.13

The main difference across the different configurations is the number of digital signal processing units (DSPs) as shown in Table 4.1 through Table 4.6. When the number of processing elements increases in the design, the DSPs which are mostly used for synthesizing the processing elements increase in numbers. As shown in the tables, the number of DSPs generated by the synthesis tool is not exactly the number of the processing elements in the design. We determine that this overhead is caused by the implementation of linear feedback shift register because when we take out the pseudo random generator in our design and ran the synthesis, the number of DSPs consumed equals to the number of processing elements in the systolic array.

The hardware utilization of our overlay systolic architecture is shown in Table 4.7. As we demonstrated in Section 3.3, the main difference between the traditional systolic arrays and our overlay systolic architecture lies in the crossbar interconnect network, which is implemented as look-up tables by the synthesis tool. As a result, the number of LUTs synthesized is significantly higher than the number in the traditional systolic arrays. On the other hand, the number of DSPs also slightly increases comparing to the most resource-consuming traditional systolic arrays by 2% in terms of utilization on the board. Thus, we determine that our design is feasible in terms of computational resource usage comparing to baseline traditional systolic array systems. However, since we are using full crossbar networks in our overlay architecture, the other hardware resources such as LUTs and FFs are not scalable.

Table 4.7: The hardware utilization of five-by-five overlay systolic architecture.

Resource	Utilization	Available	Utilization%
LUT	7907	64000	12.35
FF	526	12800	0.41
DSP	40	160	25
IO	35	400	8.75
BUFG	1	32	3.13

4.2 Timing Characteristics

Since our overlay architecture encompasses some hardware redundancy and additional logic to realize run-time reconfigurability, we are interested in how the additional circuitry affects the timing characteristics of our overlay architecture comparing to traditional baseline systolic arrays. We base the timing analysis on the report timing summary of the Vivado synthesis tool. The target clock frequency we use in the implementations of all designs unless specified is 100MHz with a 10ns clock period. An expression for calculating the maximum clock frequency the specific design can reach is given by the manual provided by Xilinx [20], where T is the target clock period. The worst negative slack in the equation is given in the first column of the Tables. The number shows the latency for passing through maximum combinational logic path in the design within one clock cycle. Thus, we take the maximum between setup and hold worst negative slack time to calculate the maximum frequency.

$$f_{max} = 1/(T - WorstNegativeSlack) \tag{4.1}$$

Table 4.8: The timing characteristics of three-by-three rectangular systolic architecture.

Type	Worst Slack	Total Violation	Failing Endpoints	Total Endpoints
Setup	8.208ns	0	0	42
Hold	0.29ns	0	0	42
Pulse Width	4.5ns	0	0	10

We run the same timing summary report for our overlay systolic architecture with the same clock but found the worst negative slack violated the 10ns clock period constraints. So we

Table 4.9: The timing characteristics of four-by-four rectangular systolic architecture.

Type	Worst Slack	Total Violation	Failing Endpoints	Total Endpoints
Setup	6.813ns	0	0	64
Hold	0.29ns	0	0	64
Pulse Width	4.5ns	0	0	18

Table 4.10: The timing characteristics of five-by-five rectangular systolic architecture.

Type	Worst Slack	Total Violation	Failing Endpoints	Total Endpoints
Setup	5.178ns	0	0	162
Hold	0.228ns	0	0	162
Pulse Width	4.5ns	0	0	29

Table 4.11: The timing characteristics of three-by-three hexagonal systolic architecture.

Type	Worst Slack	Total Violation	Failing Endpoints	Total Endpoints
Setup	8.0588ns	0	0	192
Hold	0.478ns	0	0	192
Pulse Width	7.845ns	0	0	8

Table 4.12: The timing characteristics of four-by-four hexagonal systolic architecture.

Type	Worst Slack	Total Violation	Failing Endpoints	Total Endpoints
Setup	5.988ns	0	0	496
Hold	0.161ns	0	0	496
Pulse Width	4.5ns	0	0	18

Table 4.13: The timing characteristics of five-by-five hexagonal systolic architecture.

Type	Worst Slack	Total Violation	Failing Endpoints	Total Endpoints
Setup	4.737ns	0	0	930
Hold	0.148ns	0	0	930
Pulse Width	4.5ns	0	0	18

Table 4.14: The timing characteristics of five-by-five overlay systolic architecture.

Type	Worst Slack	Total Violation	Failing Endpoints	Total Endpoints
Setup	9.185ns	0	0	1566
Hold	0.261ns	0	0	1566
Pulse Width	24.5ns	0	0	543

relax the clock frequency to 20MHz with a 50ns period. The timing report with the relaxed target clock is shown in Table 4.14.

As shown in Table 4.8 to Table 4.13, the maximum clock frequency is given by equation 4.1 depending on the size of the systolic architecture. However, the maximum clock frequency for the five-by-five overlay architecture is 24.5MHz, which is $8\times$ slower than the five-by-five baseline design. We determined the performance degradation was due to the long logic path in the large multiplexers that are used to implement the crossbar interconnect network. Since they are synthesized from look up tables on the FPGA board, and the look up tables only have fixed size, so multiple look up tables are stacked together to build a single multiplexer, which is the factor that causes the worst negative slack exceeding the baseline design.

We propose to mitigate this performance drop by adopting the C-slow technique [11], inserting latches in the combinational logic to reduce the long paths. We use the C-slow technique to shorten the long paths in our multiplexers as shown in Figure 4.1.

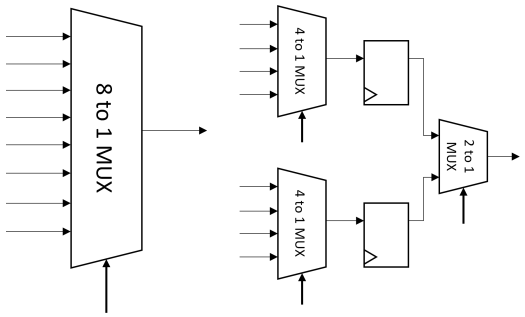

Figure 4.1: An 8-to-1 multiplexer vs. its C-slow equivalent multiplexer circuit.

The equation that determines the relationship between the achieved clock frequency and the number of layers of latches inserted is given by,

$$t_C = [t_{original}/C + 0.66 \times (\ln C)^{0.7}]ns \tag{4.2}$$

where t_C is the clock period after inserting C layers of latches, $t_{original}$ is the original clock period, and C is number of layers of latches [4]. We plot the relationship between f_C, which is given by $1/t_C$, and C with the achieved smallest clock period of our overlay architecture to demonstrate how the C-slow technique can mitigate the performance gap between our overlay design and the baseline designs.

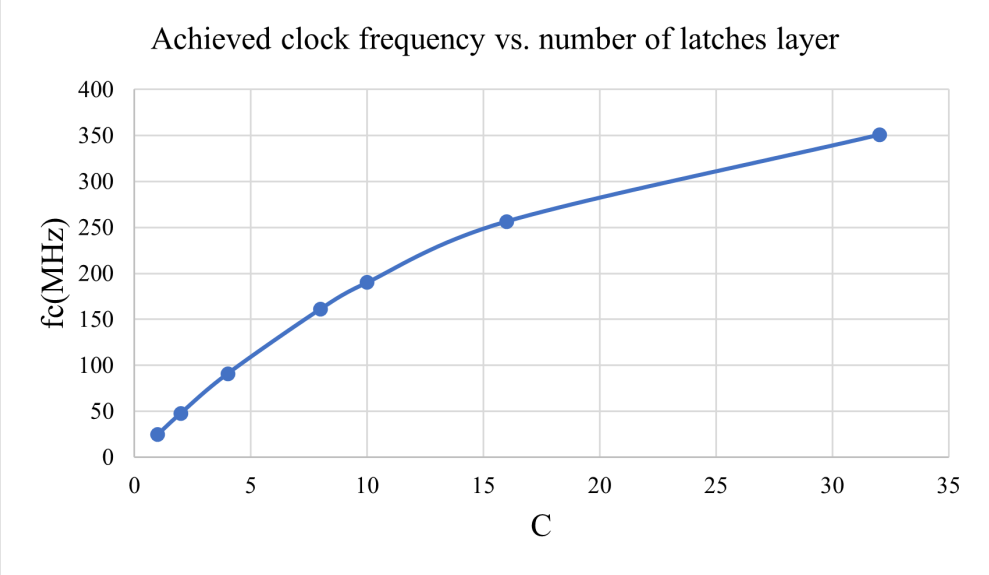

Figure 4.2: How frequency of the design changes with the number of C-slow layers added.

As shown in Figure 4.2, the frequency of the design increases with the number of C-slow layers inserted in the combinational logic. When the number of C-slow layers reaches 10, the performance gap between the baseline designs and the overlay architecture is mitigated. Inserting 10 layers of latches in the multiplexer design should yield a maximum clock frequency that is comparable to the baseline design. However, the C-slow technique induces additional clock cycles for the data to pass propagate through the multiplexer and the interconnect

network. We maintain the throughput of the design by interleaving multiple GEMM operations in our overlay architecture so that each GEMM propagates through the interconnect network in a pipeline-like fashion, and thus obtaining the same level of overall throughput.

Chapter 5

Conclusion and Future Work

This work describes our implementation of the proposed overlay systolic architecture that aims to grant users reconfigurability of the systolic topology at run-time. The dynamic flexibility of our overlay architecture lies mainly in the interconnect network that links the processing elements. The flexible interconnect network is implemented as a crossbar network that provides full connectivity from arbitrary processing element to another. The crossbar network reads in setup bits at run-time to adjust its topology that avoids going through the synthesis tool and re-loading a bit file onto the FPGA board.

We authored a suite of baseline systolic array designs with different shapes and sizes to compare the hardware utilization and timing characteristics to our proposed architecture. We found that our overlay architecture posed both resource and timing overhead. However, the hardware resources used for computation, namely digital signal processing units, exceeds the baseline design by only 2%, which is acceptable considering the run-time reconfigurability that the overlay architecture provides. The maximum clock rate for the overlay design was 8× slower than the baseline designs, but we mitigated the gap by using C-slow circuit [11] and interleaving multiple GEMM operations in parallel to maintain the throughput level.

Systolic array architectures require a rhythmic data flow to correctly operate on the data that passes through them. For simulation purposes, we implemented a linear feedback shift register that fed data to the systolic array to emulate the data flow from the main memory. For this works purpose of implementing run-time reconfigurability in traditional systolic array, we choose to not implement a memory controller interface that allows our overlay architecture to directly fetch data from the main memory. We conclude that the computational unit, namely the overlay systolic architecture, can reach comparable resource consumption

and timing characteristics comparing to the traditional designs. Additionally, a full crossbar interconnect network does not scale with the number of processing elements needed in the system. We choose to use the crossbar network because of its arbitrary connectivity comparing to other forms of interconnect network. As suggested above, further explorations can be made in two directions: designing a memory controller that serves as the interface between the computational fabric and the main memory, and designing application-specific interconnect network that is more area-efficient.

References

[1] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2017.

[2] James Coole and Greg Stitt. Adjustable-cost overlays for runtime compilation. In *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 21–24, 2015.

[3] Mingyu Gao, Xuan Yang, Jing Pu, Mark Horowitz, and Christos Kozyrakis. TANGRAM: Optimized coarse-grained dataflow for scalable NN accelerators. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, page 807–820, New York, NY, USA, 2019. Association for Computing Machinery.

[4] Michael J. Hall, Neil E. Olson, and Roger D. Chamberlain. Utilizing virtualized hardware logic computations to benefit multi-user performance. *Electronics*, 10(6), 2021.

[5] Al-Shahna Jamal, Jeffrey Goeders, and Steven J.E. Wilton. An FPGA overlay architecture supporting rapid implementation of functional changes during on-chip debug. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, pages 403–4037, 2018.

[6] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proc. of 44th International Symposium on Computer Architecture*, pages 1–12, 2017.

[7] Robert Keller. `https://www.cs.hmc.edu/courses/2001/spring/cs156/`. Accessed: 2022-4-4.

[8] H. T. Kung. Why systolic architectures? *Computer*, 15(1):37–46, 1982.

[9] H. T. Kung and Philip L. Lehman. Systolic (VLSI) arrays for relational database operations. In *Proceedings of the 1980 ACM SIGMOD International Conference on Management of Data*, SIGMOD '80, page 105–116, New York, NY, USA, 1980. Association for Computing Machinery.

[10] H. T. Kung and Charles E. Leiserson. Systolic arrays (for VLSI). In *Sparse Matrix Proceedings 1978*, volume 1, pages 256–282. Society for Industrial and Applied Mathematics, 1979.

[11] Charles E. Leiserson and James B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1–6):5–35, June 1991.

[12] Leibo Liu, Jianfeng Zhu, Zhaoshi Li, Yanan Lu, Yangdong Deng, Jie Han, Shouyi Yin, and Shaojun Wei. A survey of coarse-grained reconfigurable architecture and design: Taxonomy, challenges, and applications. *ACM Comput. Surv.*, 52(6), October 2019.

[13] Ho-Cheung Ng, Cheng Liu, and Hayden Kwok-Hay So. A soft processor overlay with tightly-coupled FPGA accelerator. `https://arxiv.org/abs/1606.06483`, 2016.

[14] W. Wesley Peterson and E. J. Weldon, Jr. *Error-Correcting Codes*. MIT Press, Cambridge, 2nd ed. edition, 1972.

[15] Eric Qin, Ananda Samajdar, Hyoukjun Kwon, Vineet Nadella, Sudarshan Srinivasan, Dipankar Das, Bharat Kaul, and Tushar Krishna. SIGMA: A sparse and irregular GEMM accelerator with flexible interconnects for DNN training. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 58–70, 2020.

[16] NVIDIA TESLA V100 GPU ARCHITECTURE. Technical report, NVIDIA, 2017. `https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf`.

[17] Mahendra Vucha and Arvind Rajawat. Design and FPGA implementation of systolic array architecture for matrix multiplication. *International Journal of Computer Applications*, 26, 07 2011.

[18] David Wilson and Greg Stitt. Seiba: An FPGA overlay-based approach to rapid application development. In *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–8, 2019.

[19] David Wilson, Greg Stitt, and James Coole. A recurrently generated overlay architecture for rapid FPGA application development. In *Proceedings of the 9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, HEART 2018, New York, NY, USA, 2018. Association for Computing Machinery.

[20] UltraFast Design Methodology Guide for Xilinx FPGAs and SoCs. Technical report, Xilinx, 2021. `https://docs.xilinx.com/r/en-US/ug949-vivado-design-methodology/No-Clock-and-Unconstrained-Internal-Endpoints`.

[21] Yun Yang, Wenqing Zhao, and Y. Inoue. High-performance systolic arrays for band matrix multiplication. In *2005 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1130–1133 Vol. 2, 2005.

Vita

Zihao Chen

Degrees B.S. Computer Engineering, May 2020
M.S. Computer Engineering, May 2022

May 2022