Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Martin Gundersen

Analyzing an FPGA Neural Network Accelerator Design for Implementation in an ASIC

Master's thesis in Computer Science
Supervisor: Magnus Själander
June 2019

NTNU
Norwegian University of Science and Technology

Martin Gundersen

Analyzing an FPGA Neural Network Accelerator Design for Implementation in an ASIC

Master's thesis in Computer Science
Supervisor: Magnus Själander
June 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

With the emerging applications of artificial intelligence, there is a growing interest in dedicated hardware accelerators for efficient computing of artificial neural networks. FPGA's provide configurable logic gates to create integrated circuits used for fast and power efficient computing, at little cost and development time. The development of specialized ASICs are potentially financially expensive but may provide a ten-fold performance increase and improved energy efficiency. The FPGA-based Bit-Serial Matrix-Multiplication Overlay [1] (BISMO) architecture utilizes the parallelism of matrix-matrix multiplications, which is a core computational kernel for convolutional neural networks, by distributing bit-serial operations on an array of multiple dot-product units, allowing run-time configurable precision. The mentioned properties make the architecture usefull for performing inference in neural networks, making it suitable for AI acceleration. This thesis describes the investigation of the different approaches required to implement the mentioned FPGA architecture in an ASIC. Necessary modifications of the original FPGA architecture are described, and a new memory scheme for ASIC implementation is suggested. The analysis of the ASIC implementation shows a theoretical potential for increased throughput, and the steps required for further investigation are discussed.

Sammendrag

Grunnet den økende interessen for anvendelse av kunstig intelligens har behovet for dedikerte maskinvare-akseleratorer som utfører effektive beregninger av kunstige nevrale nettverk økt. FPGAer tilbyr konfigurerbare logiske porter som kan kombineres til å lage integrerte kretser som kan utføre raske og energieffektive beregninger til en relativt lav kostnad, med en kort utviklingstid. Utvikling av en spesialisert ASIC kan medføre høyere kostnader, men forbedre ytelse og strømforbruk. Den FPGA-baserte arkitekturen *Bit-Serial Matrix-Multiplication Overlay* (BISMO) benytter seg av matrisemultiplikasjoners egenskap til å kunne beregnes samtidig. Matrisemultiplikasjon er en type beregning som utføres ofte i sammenheng med *convolutional neural networks*, som er en type kunstig nevralt nettverk. BISMO fordeler bit-serielle operasjoner over en formasjon med kryss-produktenheter, som gir muligheten til å definere antall gjeldende siffer under gjennomføring av en beregning. BISMO arkitekturens egenskaper gjør den egnet til akselerering av evalueringer innenfor kunstig intelligens. Denne oppgaven beskriver undersøkelsen av de forskjellige tilnærminger som kreves for å integrere BISMO arkitekturen i en FPGA og en ASIC. Nødvendige modifikasjoner av den opprinnelige arkitekturen blir beskrevet, og det blir foreslått et nytt minnesystem for å implementere arkitekturen i en ASIC. Undersøkelser av ASIC-implementasjoner tyder på et teoretisk potensial for økt utførelseshastighet, og det diskuteres steg som må tas for å fortsette undersøkelsene.

Abbreviations and Acronyms

AI Artificial Intelligence

ASIC Application-Specific Integrated Circuit

BISMO Bit-Serial Matrix Multiplication Overlay

BNN Binary Neural Network

BRAM Blocked Random-Access Memory

BSMMA Bit Serial Matrix Multiplication Accelerator

CLB Configurable Logic Block

DNN Deep Neural Network

EDA Electronic Design Automation

FIFO First In First Out

FPGA Field-Programmable Gate Array

GPU Graphics Processing Unit

GTECH General Technology library

HCL Hardware Construction Language

HDL Hardware Definition Language

IC Integrated Circuit

LAB Logic Array Block

LUT Look-Up Table

PS7 Processing System 7

QNN Quantized Neural Network

RTL Register-Transfer Level

SR Shift-Register

SRAM Static Random-Access Memory

List of Figures

List of Tables

Preface

The project reported in this thesis is a continuation of a literature review project carried out during the fall of 2018. The project took place from January to June 2019 and is the basis for a master thesis in computer engineering at NTNU in Trondheim. The project is under the Department of Computer Science, with some aid from the Department of Electronic Systems.

Acknowledgements

I would like to extend my gratitude to my supervisor, Magnus Själander, for providing me with his extensive experience in computer architecture through sagacious counseling and for allowing me the possibility to define and carry out a project, in which I could work with a high degree of autonomy.

I would also like to thank Lahiru at the Department of Computer Science, and Ove Joakim at the Department of Electronic Systems for enabling me to use EDA software at NTNU's servers.

Finally, I would like to thank my friends and family for enduring my fixation on the topic of hardware acceleration.

Contents

1 | Introduction

Deep learning is a field undergoing intense study due to the recent spike in interest of Artificial Intelligence (AI). Deep learning is a collection of methods used in representative learning and applies methods like Deep Neural Networks (DNNs) for machine learning. The demand for computational performance has resulted in the inclusion of specialized computational units for neural networks, like the Dual Neural Network Processing Unit included in the latest Huawei's Mate 20 [4], or the eight-core Neural Engine included in Apple's latest iPhones [5].

Deep Neural Networks are computational models that are inspired by the interactions between neurons in the biological brain, by using layers of computational functions represented by "cells," that eventually lead to an output. By interpreting the resulting output to improve the configuration of neurons in the network, the layers of neurons can be "trained" to yield more accurate output. Inference in the context of deep learning involves evaluating input using the information gained through training. The interactions between each layer of neurons typically consist of an abounding number of operations, making deep learning a computationally heavy model, and it is, therefore, sensible to make the computations as efficient as possible in terms of energy and time.

Matrix-matrix multiplication is a core computational kernel in neural networks. Matrix-matrix product computations are highly parallelizable, meaning that they can be spread out over many computational units at the same time, creating a potential for utilizing specialized massively parallel hardware accelerators to speed up each step of the computation process. There are already widely adopted techniques for software frameworks to utilize Graphics Processing Units (GPUs) in the computational process [6]. GPUs contain many small computational units with low complexity, making them desirable for a variety of parallelizable tasks. While GPUs are both powerful

and flexible, they are not the most efficient computational platform for all purposes. Handheld and embedded system-on-chip devices are particularly concerned with energy efficiency and restricted spatial dimensions. In many cases, training is performed externally in data centers, or "in the cloud." Inference can also be carried out in the cloud, but this is not desirable from the standpoint of communication, latency, and privacy. Instead, inference occurring locally on a device close to the input would be more beneficial [7].

By using reconfigurable logic, it is possible to set up an integrated circuit to perform specialized computations as an accelerator. Dedicated hardware restricts acceleration to specific problems, but in return requires less logic and overhead, leading to efficiency in terms of power consumption and speed. One special case of neural networks that may utilize specialized hardware is Quantized Neural Networks (QNNs). Research has shown that the full precision of floating point numbers is not necessary for performing inference. It has been demonstrated that the precision of integer operations can be pruned, in some cases even to a single bit [8]. Most hardware implementations use a fixed precision which causes a bad mapping between the computational need and the underlying hardware. Recent research projects have come up with integrated circuit architectures that can accelerate neural networks at a bit-wise level. Both the Stripes [9] on an ASIC, and Bit-Serial Matrix Multiplication Overlay (BISMO) [1] on an Field-Programmable Gate Array (FPGA) achieve a high number of operations per unit of power. BISMO is a bit-serial matrix multiplication accelerator which enables the precision to be defined dynamically at runtime. With BISMO as a core computational unit for neural networks, it would even be possible to adapt the precision for individual layers in the network.

The purpose of this project is to investigate the different design flows of the two types of integrated circuits. To gain knowledge about the two kinds of technology, an analysis of the BISMO FPGA architecture will be carried out, exploring the possible benefits and detriments of implementing the architecture in an ASIC using state of the art tools for circuit design.

2 | Background

2.1 Circuit Design Technologies

Application-Specific Integrated Circuits (ASICs) are designed to meet the requirements and constraints of a certain application. Electronic Design Automation (EDA) tools utilize a standard cell library provided by ASIC vendors, which contains information about the behavior of the cells. Metrics like delay, noise, and behavior under temperature changes are factors that need to be considered when designing a circuit. Specific constraints allow for the synthesis of a highly specialized circuit, in which there is as little unnecessary logic and overhead as possible, making the circuit able to run at higher frequencies, be more energy efficient and more compact, i.e., use less die area, than general purpose circuits. This does, however, come with a trade-off in terms of time and financial resources, which will be discussed later in this section.

FPGAs are integrated circuits manufactured for the purpose of allowing customers to configure the circuit's logic after deployment, and for developing hardware circuits without costly manufacturing in the form of an ASIC. The chip consists of an array of configurable interconnected logic blocks used to create reconfigurable logic that is able to replicate the function of virtually any digital circuit. Part of the array can be seen in figure 2.1, where the Configurable Logic Blocks (CLBs) are placed around a configurable interconnect that activate blocks and transfer data between blocks and off-chip. These blocks are referred to as CLBs, and Logic Array Blocks (LABs) by Xilinx and Altera, respectively [2]. Each, using Xilinx's naming convention, CLB contains what Xilinx refer to as slices, that contain one or more logic cells. Each logic cell contains one or more Look-Up Tables (LUTs) that in combination with multiplexers and flip-flops or latches allow any n-bit in-

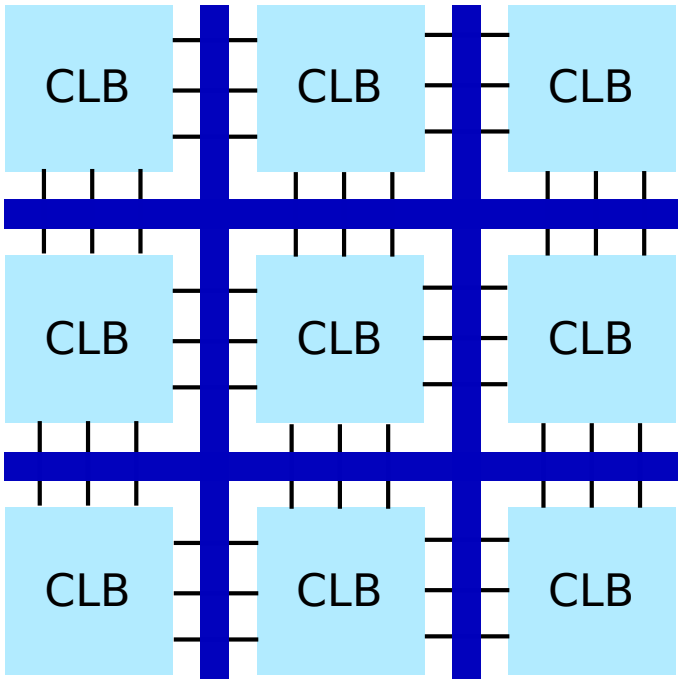


Figure 2.1: Configurable Array of CLBs

put LUT to express any rudimentary n-input boolean functions, by storing the output values in a truth table format in a small memory. Figure 2.2 shows a simplified overview of a basic CLB slice. Both logic cells contain one multifaceted LUT, meaning it could also serve the purpose of Static Random-Access Memory (SRAM) or a Shift-Register (SR). This type of functionality enables customers to change the behavior of the circuit to accommodate their specific needs.

In both FPGA and ASIC, memory is divided into off-chip and on-chip memory. In an ASIC design, on-chip SRAM is expensive and often affects the overall design. ASIC designers either get the macros from vendors or use SRAM generators to make arrays of memory cells and necessary connections which are combined into an "SRAM macro" [10]. The physical size, heat dissipation, and yield are all reasons why on-chip memory is costly, there is however a trade-off in moving off-chip on ASIC designs due to the devastating increase in access latency, and power consumption of off-chip memory [11]. Memory in an FPGA can be implemented in various ways, either through utilizing the gate array itself or using dedicated on-chip memory. A type of dedicated FPGA on-chip memory is Blocked Random-Access Memory (BRAM), developed by Xilinx, which is designed for large data seg-

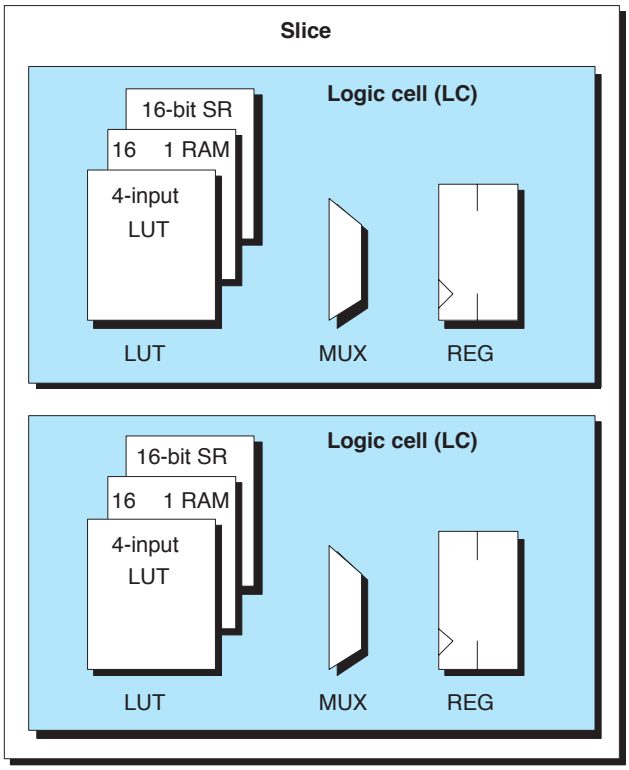


Figure 2.2: A basic CLB slice containing two logic cells
Reprinted from "FPGAs: Instant Access" [2]

ments, and supports multi-port and -clock capabilities [12]. Another method to store data is "distributed RAM", which is using the previously mentioned multifaceted properties of the FPGA's own LUTs to store information. This may be useful for small amounts of data, while it does use resources in the form of occupied LUTs that otherwise could be used to implement logic [13].

The design flow for FPGAs is similar to that of ASICs, there are however a few significant differences. FPGAs are, much like ASICs, often defined in an Hardware Definition Language (HDL) before being synthesized into an implementation by an EDA tool. With a Register-Transfer Level (RTL) design defined in Verilog or VHDL, the behavior of the circuit is described to the synthesis tool. In the case of ASIC implementation, the synthesis starts with building a netlist configuration with generic components based on the constraints and RTL design provided to the tool. Constraints may be a specific clock frequency, area confinement or power consumption. An equivalency check is performed to see if the configuration behaves like the RTL design is intended to. The design is then mapped into the real logic

Design flow	ASIC	FPGA
RTL Design	Designed in Verilog and/or VHDL	Same case as with ASIC
Behavioral simulation of gates	Specialized library cells from established manufacturers	Some differing features depending on manufacturer
Equivalency checking	Compare logic of synthesized design and RTL	Check that functions output the same in the configurable logic
Floor planning	Large designs need human intervention to optimize configuration	Leaves much of the control to the compiler
Placement and routing	Need manual optimization to fit constraints	Restricted to number of LUTs, and the pre-made interconnect grid
Clock Tree synthesis	Check design for timing violations	Uses a grid that makes sure all clocking is met during placement
Verification	Sign off to foundry to test design	Test directly on premade device

Figure 2.3: Design Flow for ASIC and FPGA

gates contained in the standard cell libraries provided by Integrated Circuit (IC) vendors. These cell libraries contain information about the components based on statistics gathered through testing [14]. The EDA tools for FPGA synthesis translates the RTL design, then maps it to the CLB array described by the characteristics of the targeted FPGA chip [15].

When the ASIC design is mapped to real-cell logic, it is ready for physical design in the form of floorplanning, placement, and routing. Floorplanning and placement consist of placing the modules in a way that conserves the spatial dimension of modules, and wire-length. The difference between floorplanning and placement is that in floorplanning the area of each module is known, but properties like height, width, and position of pins are not fixed. Floorplanning is commonly performed on larger modules like memories or larger computational units. Placement optimizes the dimensions and loca-

tion of the gates inside each module at a lower abstraction level.
Routing is the task of connecting all the pins with wires. The goal of routing is to minimize delay by using as short wires as possible while arranging the wires so they don't cause interference with each other. Interference between wires is called crosstalk. All the physical design steps are usually performed with EDA tools, however since the physical design of an ASIC is permanent, it is paramount to spend time on getting these steps right and they therefore need attention from field specialists [14].
Finally, there is a final verification process. In FPGAs, this is performed directly on the FPGA chip, and the results are consequently imminent. On an ASIC the binary files are sent off in Graphic Database System Mark 2 (GDSII), or equivalent format, to manufacturers for the final sign off. This cycle is time-consuming and often very expensive.

In figure 2.3, some of the design flow steps are described for both types of designs. In the case of an ASIC design, thorough timing analysis, floorplanning, and equivalency check is needed in order to sign off the implementation to a foundry. Many of these steps require significant manual intervention to assure that the tools perform as expected, depending on the design. A significant portion of the verification has also already been done at the design time of the FPGA itself, thus, making this step relatively much simpler. Since the FPGAs in contrast to fixed function circuits like ASICs allow modification after the logic is implemented, post-production errors are less costly to fix and therefore production costs are reduced. Manufacturers of integrated circuits can utilize this by testing and revising an architecture on an FPGA, before deploying the technology through foundries [16][17].

2.2 Hardware Construction Language

Chisel is a Hardware Construction Language (HCL) implemented in Scala by researchers at UC Berkeley. It provides powerful meta-programming techniques, simplifying the process of producing verbose Verilog code. Since the original Hardware Definition Languages (HDLs) like Verilog and VHDL were created for the purpose of simulating hardware, many of its constructs do not synthesize, whereas a HCL like Chisel uses a simple set of construction primitives that are domain specific for RTL design. By embedding the language into Scala, a multi-paradigm language, developers avoid the need to define a completely new language, while still being able to define abstract data types

and parameterized hardware generators [18].

2.3 Quantized Neural Networks

Quantized Neural Networks (QNNs) are special cases of Deep Neural Networks (DNNs) where weights, activations or gradients are represented by integers of a small number of bits. Other types of DNNs may require large amounts of memory in order to properly represent the model. Despite a possible degradation in predictive performance, quantization provides a potential solution to greatly reduce the model size and the energy consumption [19].

Algorithm 1 Bit-serial matrix multiplication on signed integers.

1: Input: $m \times k$ l-bit matrix L, $k \times n$ r-bit matrix R
2: Output: $P = L \cdot R$
3: **for** $i \leftarrow 0 \ldots l-1$ **do**
4: **for** $j \leftarrow 0 \ldots p-1$ **do**
5: $sgnL \leftarrow (i == l-1\ ?\ -1 : 1)$
6: $sgnR \leftarrow (j == p-1\ ?\ -1 : 1)$
7: $weight = sgnL \cdot sgnR \cdot 2^{i+j}$
8: # Binary matrix multiplication between $L^{[i]}$ and $R^{[j]}$
9: **for** $r \leftarrow 1 \ldots m$ **do**
10: **for** $c \leftarrow 1 \ldots n$ **do**
11: **for** $d \leftarrow 1 \ldots k$ **do**
12: $P_{rc} = P_{rc} + weight \cdot (L^{[i]}_{rd} \cdot R^{[j]}_{dc})$

Figure 2.4: Algoritm for Bit-serial GEMM
Reprinted from "BISMO: A Scalable Bit-Serial Matrix Multiplication Overlay for Reconfigurable Computing"

A special case of a QNN is a Binary Neural Network (BNN) where the weights and activations of the network are represented by a single bit at runtime. In addition to drastically reducing memory size and accesses, BNNs make it possible to replace most arithmetic operations with bit-wise operations, which is expected to improve power-efficiency [20] substantially. Bit-serial operations are inherently frugal since they only compute as many bits as specified by the precision of the operands. The mentioned frugality comes at the cost of accuracy. Thus, QNNs are required for more advanced applications.

Despite the attractive accuracy and computational properties, there is a chal-

lenge in reaping the benefits on QNNs on mobile devices with commodity processors. Three outstanding issues limit the benefits of QNN deployment on existing mobile CPUs: floating point parameters inside and between quantized layers, lack of native support for efficient few-bit integer matrix multiplications, and overhead of bit-masking operations for convolution lowering on few-bit activations. Thus, a need for mitigating these challenges with innovative methods occur, in order to utilize the potential of bit-serial computations [21].

Umuroglu and Jahre showed that by expressing a matrix multiplication as a weighted sum of binary matrix operations they were able to perform matrix-matrix multiplication with bit-serial computations. They use the specialized case of Binary General Matrix Multiplication (Binary GEMM), and use this mathematical kernel (line 9-12 in figure 2.4) to be able to perform few-bit integer matrix multiplications with a Bit-serial GEMM [8].

An example of bit-serial multiplication by expressing matrices as weighted sums of binary matrices can be seen in figure 2.5. The two matrices L and R, are factorized to compositions of powers of two. Using the distributive property of multiplication, the product of the two matrices can be calculated using the weighted sum of each binary product. Since the values range from zero to three, each number may be represented as two bits. In order to make the same work for numbers ranging zero to seven, another bit would have to be used, increasing the complexity of the calculation. This property is a cornerstone in the functionality of the BISMO architecture, elaborated in section 3.2.

$$L = \begin{bmatrix} 2 & 0 \\ 1 & 3 \end{bmatrix} = 2^1 L^{[1]} + 2^0 L^{[0]} = 2^1 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + 2^0 \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$$

$$R = \begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix} = 2^1 R^{[1]} + 2^0 R^{[0]} = 2^1 \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} + 2^0 \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$L \cdot R = (2^1 L^{[1]} + 2^0 L^{[0]}) \cdot (2^1 R^{[1]} + 2^0 R^{[0]})$$

$$= 2^2 (L^{[1]} \cdot R^{[1]}) + 2^1 (L^{[1]} \cdot R^{[0]}) + 2^1 (L^{[0]} \cdot R^{[1]}) + 2^0 (L^{[0]} \cdot R^{[0]})$$

Figure 2.5: Expressing a matrix-matrix multiplication as a sum of weighted binary dot products.

3 Modern Applications of DNN Hardware Acceleration

3.1 Applications in Consumer Electronics

Current technologies for accelerating DNNs have reached popular consumer devices. Designers of handheld devices have implemented existing applications of AI to new areas. The flagships of the top mobile phone manufacturers like Apple, Huawei, and Samsung, include the functionality of unlocking the device with the owners face. This feature is made possible by facial recognition software that uses AI methods to identify peoples facial structure. The nascent Kirin 980 SoC included in the latest range of Huawei products have utilized a dual Neural Processing Unit(NPU) architecture to process training and inference at higher throughput than through a traditional architecture [4]. Apple has its custom made A12-Bionic chip with a built-in eight-core neural engine, stated to perform at 5 trillion operations per second. Both of these AI accelerators are examples of the use of ASICs to improve the throughput of the system [5].

Google has recently introduced another type of AI accelerator. In addition to their AI cloud training services, they have presented a new portable ASIC called Edge. The device may be plugged into a computer, and be utilized by Google's own machine learning API called TensorFlow. The chip is purported to provide energy efficient inference through quantization. The Edge Tensor Processing Unit(TPU) operates with 8/16 bit integers, and use a Complex Instruction Set Computer (CISC) to control the computations [22]. Since the TPU is concerned with integers operations only, the computations are stated to be 83 times better than a CPU, and 29 times better than a GPU, when measuring performance per watt [23].

Because general-purpose processors such as CPUs and GPUs must provide good performance across a wide range of applications, they have evolved myriad sophisticated, performance-oriented mechanisms. As a side effect, the behavior of those processors can be difficult to predict, which makes it hard to guarantee a certain latency limit on neural network inference. In contrast, TPU design is strictly minimal and deterministic as it has to run only one task at a time: neural network prediction [23].

While the exact architecture of these chips is disclosed information, the concept of using specialized hardware accelerators for specific tasks, known colloquially as heterogeneous computing, is making its way to the market.

3.2 BISMO

The BISMO architecture utilizes the previously mentioned frugality of bit serial operations mentioned in section 2.3, by expressing a matrix multiplication as a weighted sum of binary matrix dot products. Fixed-precision operations need to support the largest precision number and keep this precision throughout the execution of an application where the required precision might vary. The BISMO architecture uses software hardware cooperation, where the hardware is fed instructions describing the size and precision of each matrix. The architecture comprises of a three-stage pipeline, where the matrix data is fetched from memory, executed, and storing the results back in memory. Between the stages, synchronization is performed by blocking reads and writes to synchronization First In First Out (FIFO) queues. All stage operations, including datapath control and synchronization, are controlled by instructions, which are fetched from instruction queues and executed in order. The architecture can be seen in figure 3.1.

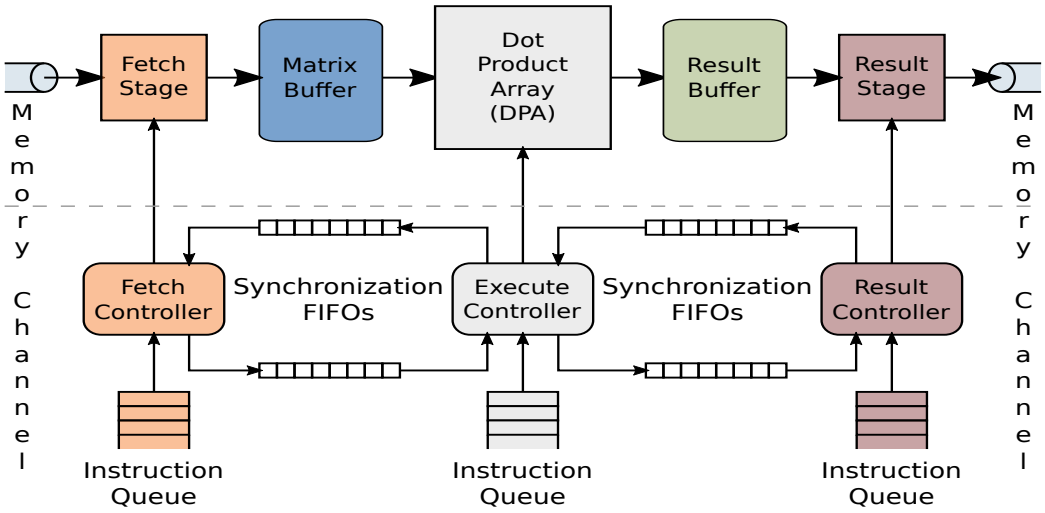


Figure 3.1: Overview of BISMO's' hardware architecture.
Reprinted from "BISMO: A Scalable Bit-Serial Matrix Multiplication Overlay for Reconfigurable Computing"

BISMO uses the Bit-serial GEMM algorithm described in figure 2.4, by buffering matrix computations from main memory and feeding them into an array of Dot Product Units (DPUs) called the Dot Product Array (DPA). The BISMO data flow can be seen in figure 3.2. The DPUs compute the binary dot product partial results by using a multi-bit bit-wise AND-gate, counting the resulting number of 1's using a PopulationCount unit, and a Left-Shift unit to include the weighting. The result can then optionally be negated, before being accumulated in an accumulation dedicated register. The DPU architecture can be seen in figure 3.3.

BISMOs parameterizable hardware architecture is designed in the HCL Chisel, with the use of some external Chisel libraries to utilize an Advanced eXtensible Interface (AXI) bus and BRAM for on-chip memory. The architecture is synthesized with the Xilinx Vivado software for FPGA design and evaluated on a PYNQ-Z1 FPGA board. The PYNQ-Z1 device uses a processing system called Processing System 7 (PS7), which controls the AXI bus and I/O for the programmable logic. The architecture of the processing system can be seen in detail in figure 3.4. The BISMO hardware architecture is design-time configurable, which is a key-feature of implementing the architecture on an FPGA. There is a finite amount of LUTs and BRAM per FPGA, so the authors have provided a cost model for the number of LUTs and amount of BRAM required for an overlay with a certain set of dimensions [1].

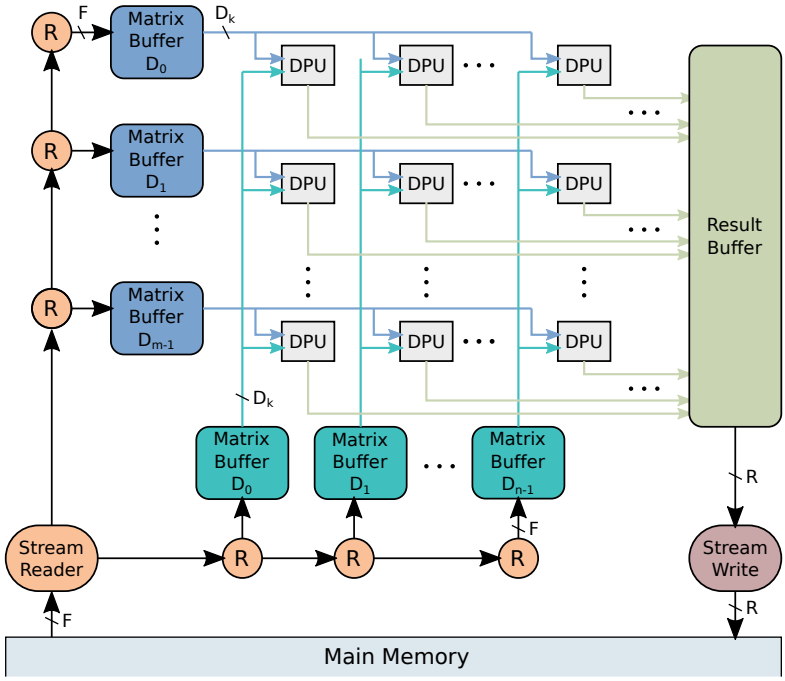


Figure 3.2: Overview of BISMO's Datapath.
Reprinted from "BISMO: A Scalable Bit-Serial Matrix Multiplication Overlay for Reconfigurable Computing"

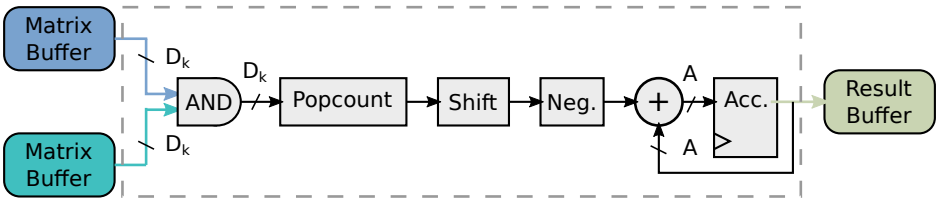


Figure 3.3: Dot Product Unit
Reprinted from "BISMO: A Scalable Bit-Serial Matrix Multiplication Overlay for Reconfigurable Computing"

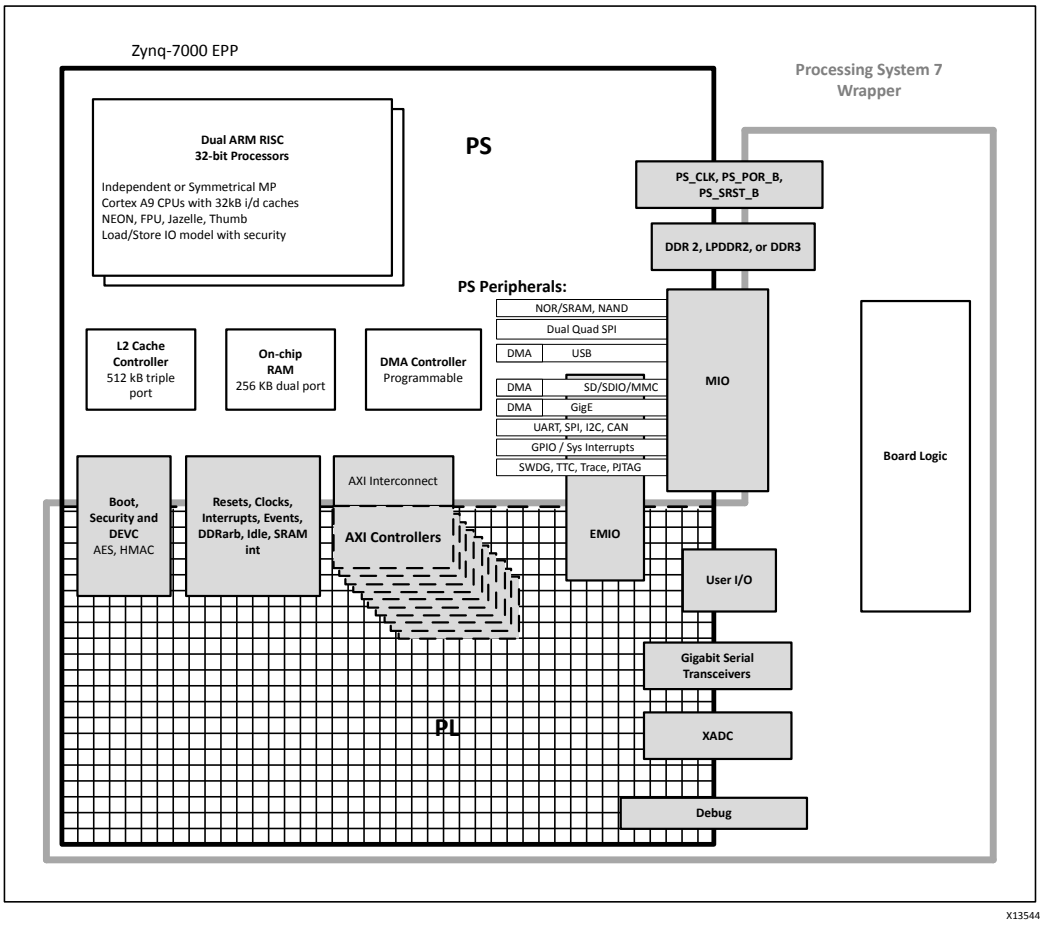


Figure 3.4: Processing System for ZYNQ 7000 [3]

4 | Methodology

The first step of the design process for both ASIC and FPGA development is to create a functional logical design at RTL level. The chisel code provided with the original FPGA BISMO project creates Verilog code that defines the RTL-level logic of the BISMO design. The resulting single-file Verilog code contains information about the wrapper that integrates the Bit Serial Matrix Multiplication Accelerator (BSMMA) into the PYNQ FPGA device. Info about the wrapper is otiose to the ASIC project, as the peripherals and bus organization is specific for the PYNQ device. The RTL design is organized hierarchically, allowing for the selection of a particular subcomponent in the design and separate it and its dependencies for independent synthesis and analysis.

4.1 Previous Work

In the original BISMO project, the chisel code contains parts that are included with an FPGA architecture in mind. In a preliminary study leading up to this project, the DPA part of the architecture was inspected using the Vivado synthesis tool and mapped into equivalent ASIC cells using Design Compiler. The results from the DPA synthesis is included in section 5.2.

4.2 Current Project

In this project, a complete analysis of the full BSMMA is performed and mapped into ASIC-equivalent technology. The first step being to inspect the altered architecture design inside the FPGA wrapper for the PYNQ device.

The altered design should also be implementable on an FPGA, considering that the FPGA LUTs are able to represent any logical function as long as the necessary amount of resources are available.

After confirming that the design works on the FPGA, the next step is to transfer the RTL-logic to tools intended for ASIC synthesis. For synthesis into a gate-level design in this project, Design Compiler by Synopsys is used as it is a popular tool for gate-level synthesis and, according to the company itself, has the largest market share for EDA design [24]. This project has customized and utilized an existing framwork created by Benjamin Bjørnseth, for ASIC synthesis with Design Compiler. The original project can be found at `https://bitbucket.org/benjambj/energy-model-synthesis`. Examples of the scripts utilized in this project is included in appendix B. The process was then planned and carried out using the Design Compiler User Guide [25]. A simplified flow chart from the user manual is shown in figure 4.1.

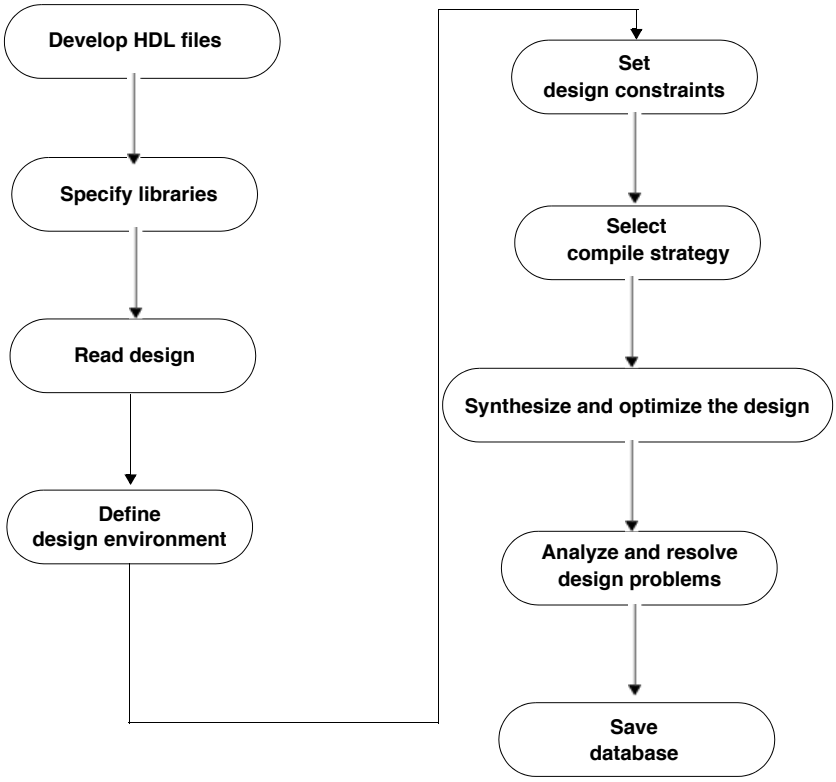


Figure 4.1: Design compiler synthesis flow

The design compiler first performs elaboration of the design with "presto compilation," which is Synopsys' term for creating a gate-level netlist of the

RTL-Verilog code. The resulting netlist comprises of components from a General Technology library (GTECH). The next step is to replace the GTECH components with a library provided by a company receiving approval from manufacturers. For the synthesis performed in this project, a 28nm technology cell library approved for manufacturing is used. The chosen library is set to operate under certain conditions, is supplied with a stable 1.0 voltage, and assumes a steady temperature of 25 degrees Celsius. The cell library available to the project does not contain information about wire capacitance, and so the synthesis is performed with a "zero wire load model," meaning that the synthesis assumes ideal wires.

5 Results

5.1 Design Characterization

When the design is synthesized using standard settings with Vivado, it is possible to inspect the resulting bitfiles that will be implemented on the FPGA. The notation used to describe the configuration of the BISMO design overlays in this project is MxKxN, where M and N are dimensions of the DPA, and K is the bit width of the DPUs.

Figure 5.1 shows how the FPGAs resources are utilized, with the teal colored parts of the image being LUTs occupied by the design, and the white bars utilized BRAM components. The orange blocks are part of the Processing System, controlling DDR-memory I/O and clocks. In figure 5.2, a more saturated FPGA device is shown. Due to the increase in both DPU bit width and DPA array dimensions, the 8x256x8 design occupies more LUTs than the 4x128x4 design. In figure 5.3 a graph shows the utilization of LUTs on a PYNQZ1 as the dimensions of the DPA are increased. The red bar shows the maximum number of LUTs available on a PYNQZ1. Design configurations requiring more LUTs than the maximum amount was, as expected, not able to be implemented on the FPGA device.

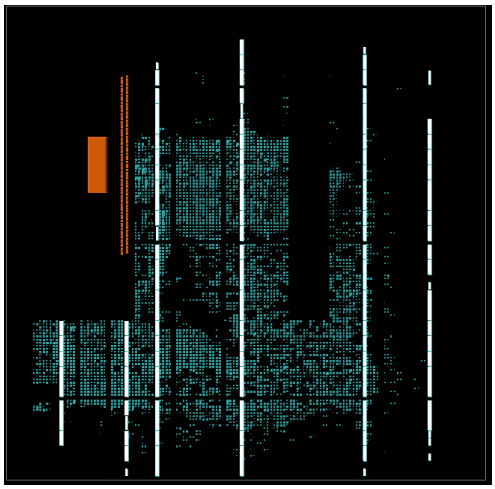


Figure 5.1: 4x128x4 FPGA overlay design.

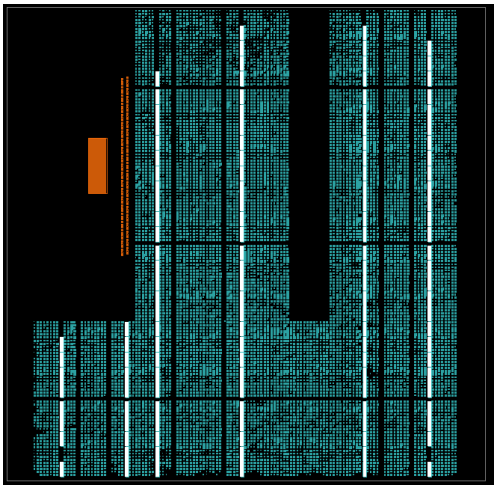


Figure 5.2: 8x256x8 FPGA overlay design.

5.2 DPA Synthesis

The results from the mapping of the DPA design to a cell library architecture provided insights into the scaling of the resource cost and complexity by increasing the size of the overlay. Although the results in the form of cell count and design area of the ASIC does not directly translate to the number of LUTs on the FPGA, they do represent a measure of complexity and resource usage on the different technologies. The fact that they scale similarly on

both technologies indicates that the full ASIC implementation might scale the same way as on the FPGA.

When implementing the BISMO architecture in an ASIC, making use of peripherals and memory might be more complex in ASIC design due to peripherals specific for FPGA platforms. Replacing dual-port BRAM with optimized SRAM requires the generation of foundry-specific memory macros. A synthesis and mapping of the DPA to a standard cell library was performed, and even though the metrics between the architectures are not directly translatable, the DPA showed similar scaling as the scaling on the FPGA implementation. There is also a theoretical potential for speedup in the form of about four times higher maximum frequency of the DPA compared to the implementation on an FPGA. The DPA results are included in appendix A.II.

5.3 BSMMA Synthesis and Memory Implementation

Through inspection of original chisel code and interpretation of resulting RTL-Verilog code, some changes were made to the original project to adapt it to an ASIC architecture. The included Verilog code for Dual port BRAM could not be compiled into plain generic ASIC components, as FPGA software will infer BRAM while ASIC synthesis software will not be able to interpret the logic into a valid design. An attempt to replace the inferred BRAM memory with ASIC architecture targeted Verilog code gave insight into the process of implementing memory in ASIC designs. Instantiating large memory devices with just flip flops or latches, and multiplexers is an exhaustive process in terms of virtual memory used by the synthesis software and total CPU time.

Attempting to replace BRAM on the FPGA with distributed RAM using the "ram_style" compiler directive resulted in a severe hike in resource use, and the Vivado software gave warning that the amount of memory in the design needed to be reduced. By manually reducing the memory used by each matrix buffer resulted in a synthesizable design, but the number of LUTs climbed to orders of magnitude higher than the original design, and no design was implementable on the PYNQZ1 FPGA.

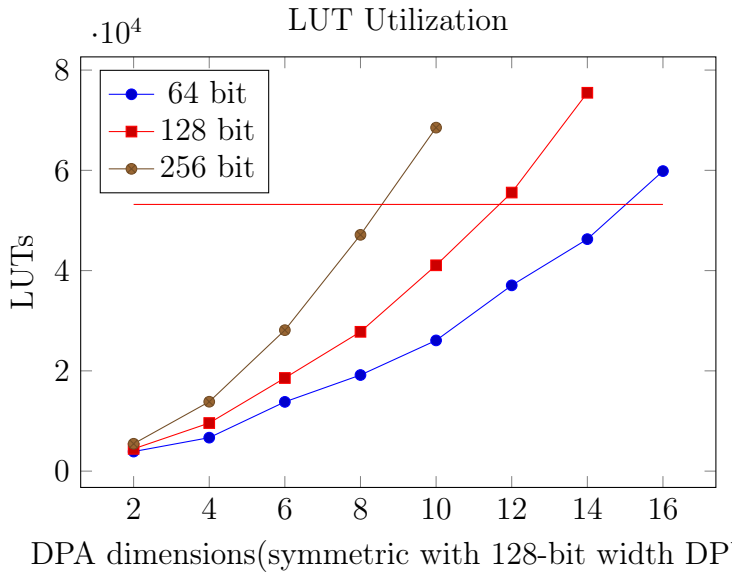


Figure 5.3: LUTs utilized in Vivado implementation on a PYNQZ1, red line indicates maximum number of LUTs on a PYNQZ1

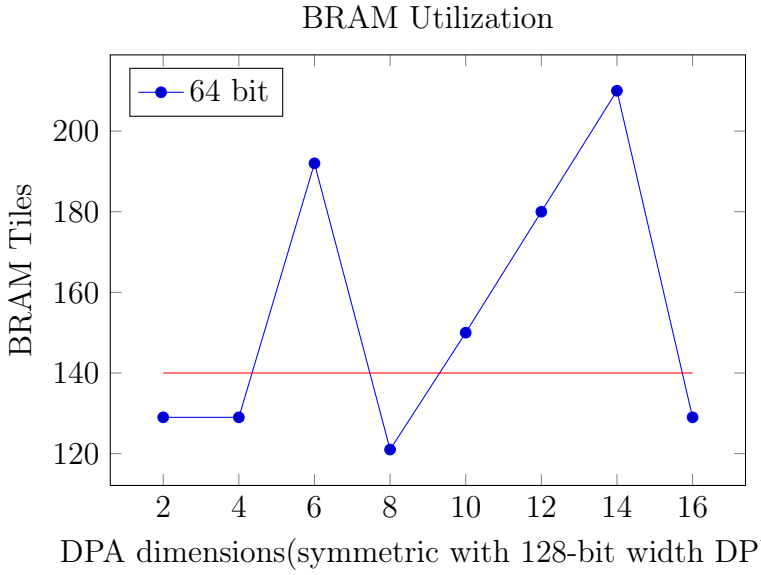


Figure 5.4: BRAM tiles utilized in Vivado implementation on a PYNQZ1, red line indicates maximum number of tiles on a PYNQZ1

The design did synthesize in Design Compiler, but even for a moderate 4x128x4 design with a relaxed 2ns clock constraint, the synthesis took several days. This rendered a proper analysis outside of the timeframe for this project. When the size of the buffers was reduced, the compile time was consequently also reduced. With the total amount of DPA buffer memory reduced to $\frac{1}{64}$ of the standard amount, the synthesis could be carried out and the critical path, cell count and cell area is plotted as *Reduced Queue ASIC* in figure 5.5, 5.6 and 5.7 respectively.

As an alternative to synthesizing memory, Design Compiler allows specific modules in the design to be described as "Black box". This means that Design Compiler has no knowledge of the internal functionality of the module, but can be supplied with the statistical parasitic information of the module from a vendor-generated library file. There exists open-source software that is intended for educational purposes, but in order to get a realistic replacement in the design to verify the functionality of the circuit, a library from a company licensed by the manufacturer is required. To acquire such a library, specific attributes of the SRAM need to be identified. An analysis of the BRAM usage provided the total amount of BRAM used by FPGA designs with represented in a graph in figure 5.4. The red line represents the maximum amount of BRAM available on the PYNQZ1 device, and the synthesized design overlay configurations requiring more BRAM than the device could not be implemented. The BRAM is mainly used as buffer FIFO queues between the Fetch and Execution stage, and a smaller FIFO queue as buffer for the result stage.

The design was compiled successfully with the memory modules instantiated as zero-load black boxes. The critical path length, cell count and design area of the BSMMA in contrast with previous DPA synthesis are graphed and displayed in figure 5.5, 5.6 and 5.7 respectively. The size of the DPA dimensions were incremented by two for each synthesis, and there was also synthesized a 32x128x32 overlay with black box memory, to see if the trajectory continued with relatively large designs.

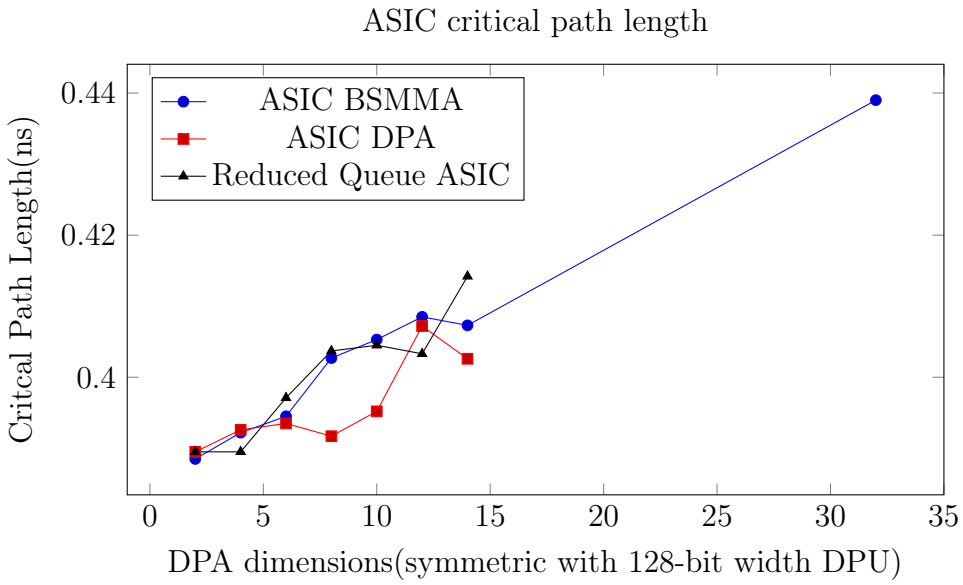


Figure 5.5: Critical path of ASIC designs with increasing dimensions of DPA.

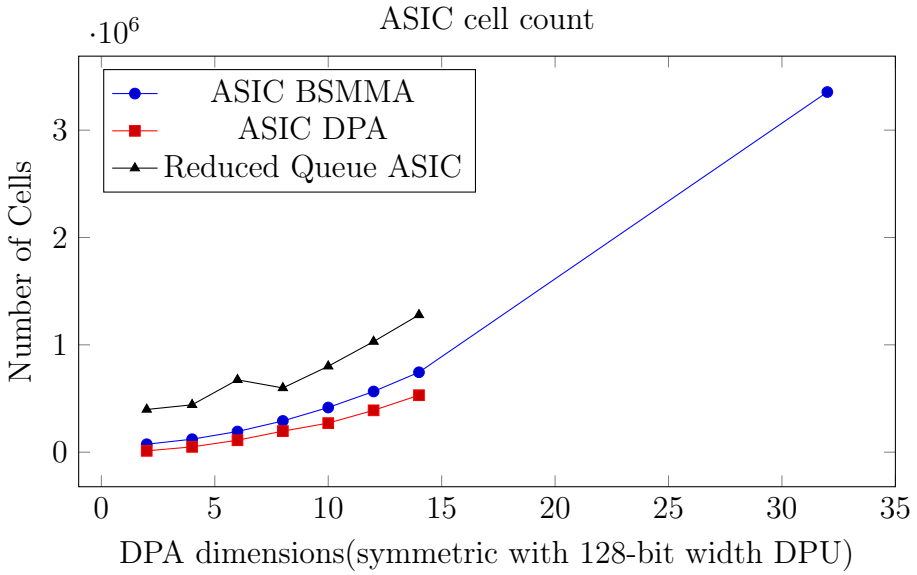


Figure 5.6: Cell count of ASIC designs with increasing dimensions of DPA.

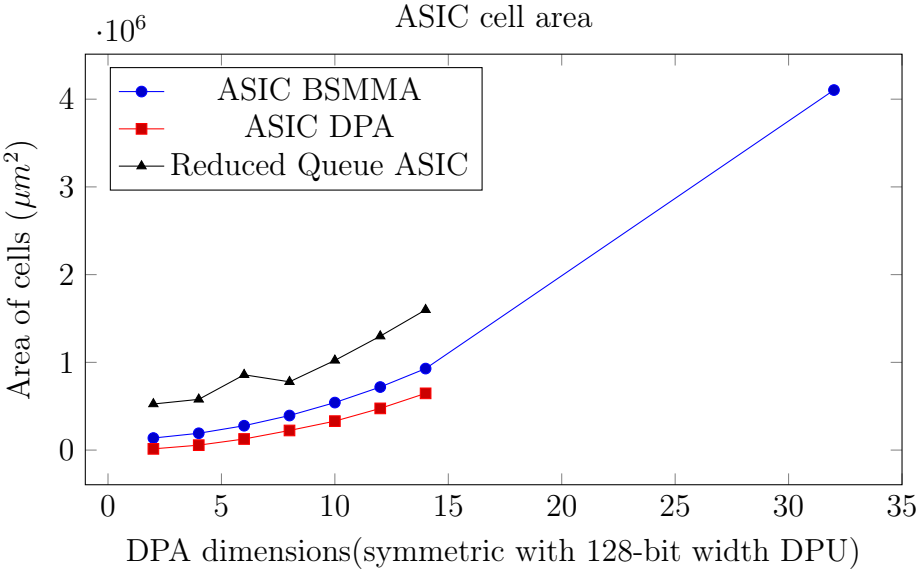


Figure 5.7: Cell area of ASIC designs with increasing dimensions of DPA.

The synthesis was also performed with less constraint on clock frequency, this could impact the designs cell count/area and power consumption. When given a clock constraint of 2 ns (500MHz), all of the configurations gave a critical path of 0.9819 ns (1018MHz). They also had a slightly lower cell count/area, shown in figure 5.8 and 5.9. The full results from the BSMMA synthesis is included in appendix A.I.

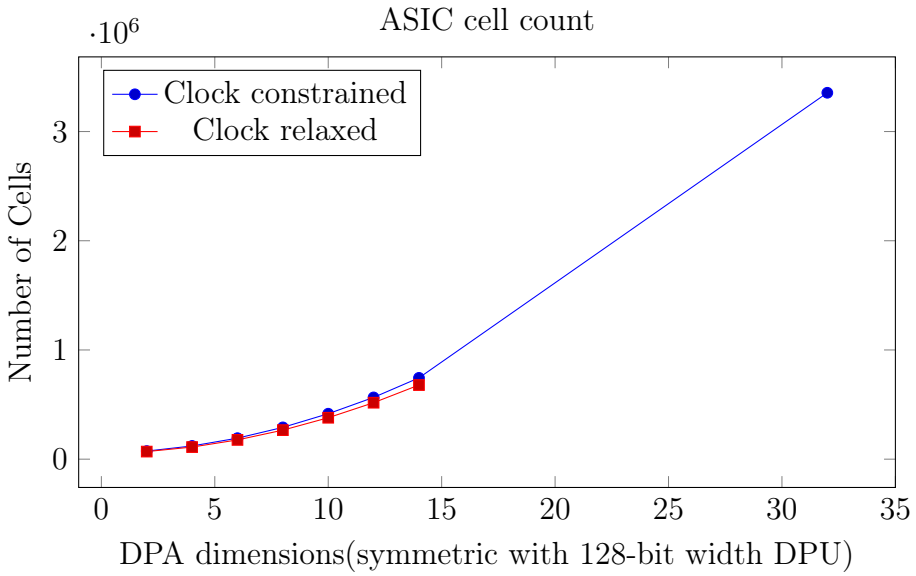


Figure 5.8: Cell count of ASIC designs compiled using different clock constraints with increasing dimensions of DPA.

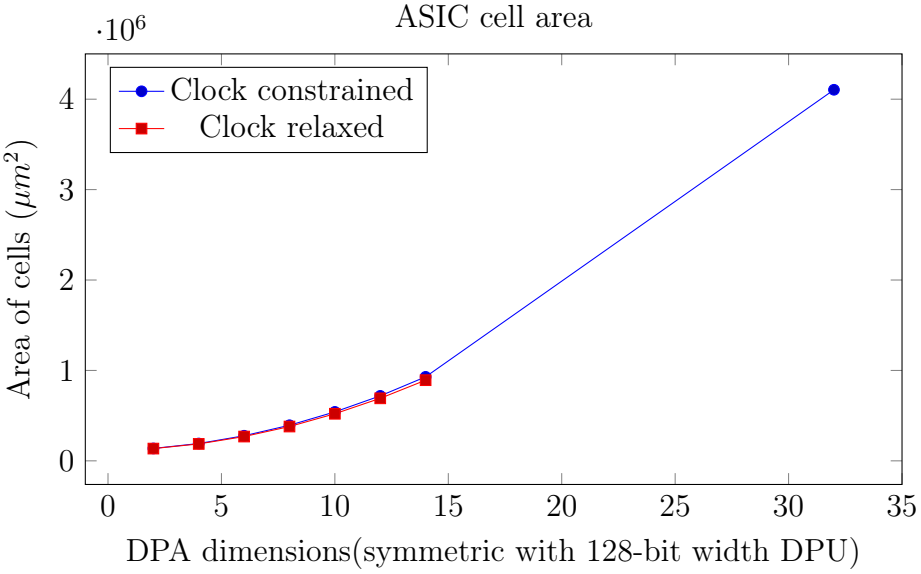


Figure 5.9: Cell area of ASIC designs compiled using different clock constraints with increasing dimensions of DPA.

6 | Discussion

The results from exploring the BSMMA design in Vivado gave insight into the structure for utilizing the FPGAs resources. Due to the finite amount of LUTs and BRAM, the architecture designers had to make certain compromises when increasing the dimensions of the DPA, and bit-width of the DPUs. When taking the maximum number of LUTs on a PYNQZ1 FPGA device, the largest symmetrical DPA array that may be implemented on a PYNQZ1 FPGA device for 64-, 128- and 256-bit DPUs are 14x14, 10x10 and 8x8, respectively. At the time the project was carried out, the standard BISMO project did not allow the implementation of any DPA in other sizes than the power of 2, on the PYNQZ1. The reason for this is the BRAM utilization of the design.

When the array's dimensions are increased, the number of matrix buffers also increase, but the depth of each buffer stays the same. As the BRAM utilization chart in figure 5.4 shows, the increase from 4x4 to 6x6 array resulted in about 50% more BRAM-tiles required, and increase from 10x10 to 12x12 increased by approximately 20%. As the 4x4 design utilizes 129 BRAM tiles, the 50% increase to the 6x6 design leads to 192 BRAM tiles required. The PYNQZ1 board has exactly 140 BRAM tiles, and as a result, the implementation fails. An 8x8 design is, however, synthesized just fine. This is due to the amount of memory per matrix buffer being dimidiated, in practice by reducing the address space of the attached memory unit by one bit. In figure 6.1, a simplified scheme for a 4-bit word-size, with a 4-bit address space (A0-A3) is shown. Increasing the address space with one bit allows for a total memory size of $2^4 * 4 = 64$ bits. By adding an extra address space bit (A0-A4), the number of accessible memory cells doubles in size. ($2^5 * 4 = 128$ bits)

The same step is taken when doubling the DPU width, as it would double the amount of memory required to buffer the same amount of items. In order to

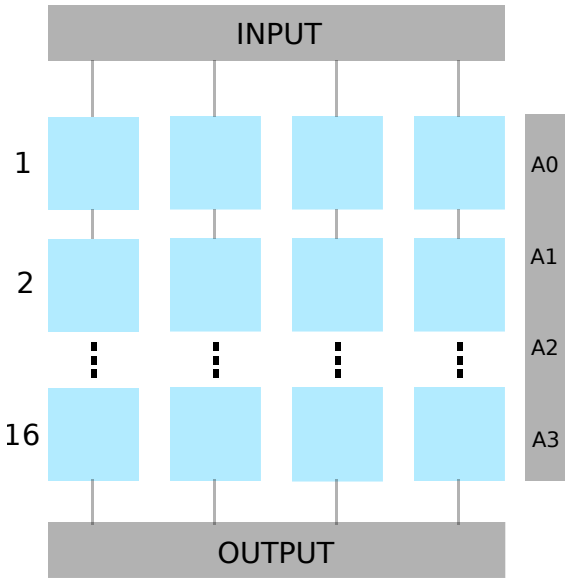


Figure 6.1: Simplified model of a memory address space

properly implement the design on a PYNQZ1 FPGA, the buffer depth would have to be reduced. This particular property is interesting to consider when porting the design to an ASIC. In an ASIC, the amount of memory available is not a problem as the amount is not prefabricated as in an FPGA. There will still have to be made a decision of what buffer depth to support.

In order to continue the development in an ASIC, the simple black box memory modules will need to be replaced with macros provided from a manufacturer with a process technology that is compatible with the cell library utilized. The manufacturers would then need information about the organization of the memory. Two attributes that need to be defined are word size and number of address bits. This describes the dimensions of the memory. From the inspection of the design, the result stage queues by default 256 entries in Blocked RAM. The number of entries in the queues that store values between the fetch-stage and execution stage are determined, assuming the DPA is symmetric, by the following formula;

$$NumberOfEntriesPerBuffer = \frac{64 * 32 * 1024}{2 * DPADimension * DPUBitwidth}$$

The rationalization for this is that each BRAM tile has 36*1024 bits, and the buffers use 32 of the native 36-bit width due to constraints from the fetch stage since DRAM buses are typically power of-two-wide and the design require BRAM read/write widths to be an integer multiple of each other [1]. Each entry is 64 bits. The total number of bits are divided between the product of the DPU-Bitwidth and the number of matrix buffers ($2 * DPADimension$). The take away from this analysis is that the SRAM macros should be of 64 bits word sizes, with a 2^k address size, where k is the number of address bits desirable for the buffer size. It is possible to reduce the address space by not allowing access to the full k bits of address space and reduce the number of memory cell rows but would require additional logic if to be implemented.

The buffers are implemented as asymmetrical First In First Outs (FIFOs) since the read/write speed of the DDR memory and BRAM differ. Conventionally, a dual-port RAM is used when implementing a FIFO memory, which allows read and write operations to be performed independently within the RAM. However, one disadvantage of the dual-port RAM is that it is almost twice the size of a single port RAM having the same capacity. In contrast, a single port RAM can perform only one operation (read or write) at a time. It is possible to combine single port SRAMs to create a FIFO with simultaneous read/write capabilities [26]. If a dual-port SRAM is not available for the process technology used to implement the BSMMA, it is still possible to implement a FIFO.

The results from the synthesis of ASIC BSMMA shows promising potential for an improved throughput for the architecture when implemented in ASIC Technology. The maximum frequency at which a circuit can operate is in relation to the length of the longest critical path. Since the synthesis is performed in "zero wire-load" mode, the resistance, capacitance, and length of the wiring are assumed ideal. Therefore the reported critical paths are only theoretic and will assumably not function in real applications. The purpose of the synthesis is not to get an accurate estimate for the ASIC's specifications, but rather serve as confirmation that the design is able to be represented by a certain set of cells and motivate for further research and development.

Overlay Configuration	Max Frequency(MHz)
2x128x2	2574.00
4x128x4	2532.29
6x128x6	2503.13
8x128x8	2402.11
10x128x10	2374.17
12x128x12	2341.37
14x128x14	2351.83
32x128x32	2277.90

Table 6.1: Max Frequency of BSMMA Black Box Overlay Configurations

$$MAXFREQUENCY(Hz) = \frac{1}{LongestCriticalPath(s)}$$

When observing the maximum length of the critical paths as the dimensions of the DPA was incremented, all the three kinds of synthesis increased at scale. There were points where the critical path was shorter than with a smaller configuration. Since the synthesis process is not deterministic, the synthesis tool might have found some configurations of cells that allowed for a shorter critical path that was not discovered during the synthesis of a smaller overlay. The maximum frequencies obtained for the ASIC BSMMA with black box memory modules are displayed in table 6.1. It is tempting to compare the results to the previous characterization results of the FPGA, where the maximum frequency ranges between 400-800MHz, but as previously mentioned, the metrics are not quite comparable. The process technology used for the programmable logic in the PYNQ-Z1 device is 28nm [27], the same generation as the cell library used for synthesis in this project. Even if the critical paths increase dramatically due to wiring, the zero wire load synthesis results reveal the possibly great potential for increasing the throughput using ASIC technology.

The cell count and cell area both scale the same way in both the DPA, the Black Box BSMMA, and the Reduced Queue ASIC. The difference in cell count and area of the three kinds of synthesizes also seem to scale linearly with the dimensions of the DPA, indicating that some additional logic is required in the stages other than the execution stage when increasing the dimensions of the DPA. There is a small hike in both cell count and cell area in the 6x128x6 Reduced Queue ASIC overlay. The probable cause that the hike occurs in only that particular kind of synthesis and not the other two

may be explained that it actually synthesizes memory modules in place of the BRAM. The drop from the 6x128x6 to the 8x128x8 overlay configuration may be seen in correlation with the drop in BRAM usage seen in figure 5.4

The relaxation of the clock constraint during synthesis showed a hike in maximum critical path length, with only minor decreases in cell count and area. It is possible to run synthesis with constraints with respect to power and area. The takeaway from this is that if resources in the form of spatial area or number of cells are scarce, it seems to yield greater reductions to reduce the dimensions of the DPA array than reducing the max clock.

7 | Conclusion

During the course of the project, the design flow of both FPGAs and ASICs have been investigated and compared. Both of the design flows include many of the same steps, but due to the rapid testing environment of the FPGA, there is more room for error and revising in an FPGA design flow. The ASIC design flow also includes steps that require human intervention to yield satisfactory results, while the FPGA's EDA tools are able to perform these tasks independent of human intervention due to the prefabricated nature of FPGAs. The specialized functionality of ASICs should yield improvements in terms of energy efficiency, speed, and volume, but are expected to require more time and resources to develop. The project has been aimed at exposing the possible detriments and modifications required to map the BISMO architecture to an ASIC.

When implementing the BISMO architecture in an ASIC, making use of peripherals and memory might be more complex in ASIC design due to peripherals specific for FPGA platforms provide out-of-the-box processing functionality. Replacing dual-port BRAM with optimized SRAM will require the generation of foundry-specific memory macros. A synthesis and mapping of the BISMO architecture to a standard cell library was performed, using both the black box method for the memory modules and replacing the RAM with registers. There is a theoretical potential for speedup in the form of about four times higher maximum frequency of the BSMMA compared to the implementation on an FPGA. This is comparable to previous findings when comparing FPGAs and ASICs [16].

Further work

Through analysis of the synthesized hardware, there has been identified a demand for memory modules with 64-bit word size and an address depth that ensures a buffer size that does not cause congestion in the form of a full buffer delaying the read from main memory. When increasing the bit width of the DPU, the matrix buffer sizes would also have to be increased to store the same number of entries. If dual-port RAM is not available, other types of memory supporting a FIFO architecture should suffice. When a cell library containing wire information and foundry-compatible memory macros have been acquired, the project can be continued by using the same framework and proceed with functional verification of the synthesis. When the functionality of the synthesized design is verified, the "Place & Route" phase may commence for further analysis and development.

References

[1] Y. Umuroglu, L. Rasnayake, and M. Sjalander, "BISMO: A Scalable Bit-Serial Matrix Multiplication Overlay for Reconfigurable Computing," *ArXiv e-prints*, Jun. 2018.

[2] C. M. Maxfield, "Chapter 2 - fpga architectures," in *FPGAs: Instant Access*, ser. Instant Access, C. M. Maxfield, Ed. Burlington: Newnes, 2008, pp. 13 – 48. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780750689748000028

[3] "Processing system 7 product guide," https://www.xilinx.com/support/documentation/ip_documentation/processing_system7/v5_5/pg082-processing-system7.pdf, accessed: May 23, 2019.

[4] "Huawei mate 20 kirin 980 soc," https://consumer.huawei.com/en/phones/mate20/, accessed: March 13, 2019.

[5] "A12-bionic," https://www.apple.com/iphone-xs/a12-bionic/, accessed: March 13, 2019.

[6] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22Nd ACM International Conference on Multimedia*, ser. MM '14. New York, NY, USA: ACM, 2014, pp. 675–678. [Online]. Available: http://doi.acm.org/10.1145/2647868.2654889

[7] V. Sze, Y. Chen, J. S. Emer, A. Suleiman, and Z. Zhang, "Hardware for machine learning: Challenges and opportunities," *CoRR*, vol. abs/1612.07625, 2016. [Online]. Available: http://arxiv.org/abs/1612.07625

[8] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. New York, NY, USA: ACM, 2017, pp. 65–74. [Online]. Available: http://doi.acm.org/10.1145/3020078.3021744

[9] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2016, pp. 1–12.

[10] T. K. Tan, A. K. Raghunathan, G. Lakishminarayana, and N. K. Jha, "High-level software energy macro-modeling," in *Proceedings of the 38th Annual Design Automation Conference*, ser. DAC '01. New York, NY, USA: ACM, 2001, pp. 605–610. [Online]. Available: http://doi.acm.org/10.1145/378239.379033

[11] P. R. Panda, P. R. Panda, N. D. Dutt, and A. Nicolau, "On-chip vs. off-chip memory: The data partitioning problem in embedded processor-based systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 5, no. 3, pp. 682–704, Jul. 2000. [Online]. Available: http://doi.acm.org/10.1145/348019.348570

[12] R. C. Pang, S. P. Young, and T. J. Bauer, "Block ram with configurable data width and parity for use in a field programmable gate array," U.S. Patent US6 346 825B1, Feb. 12, 2002.

[13] F. C. Furtek, M. T. Mason, and R. B. Luking, "Field programmable gate array with distributed ram and increased cell utilization," U.S. Patent US5 894 565A, Apr. 13, 1999.

[14] J. Bhasker and R. Chadha, *Static Timing Analysis for Nanometer Designs: A Practical Approach*, 1st ed. Springer Publishing Company, Incorporated, 2009.

[15] "Xilinx fpga design flow," https://www.xilinx.com/support/documentation/sw_manuals/xilinx10/isehelp/ise_c_advanced_flow.htm, accessed: December 1, 2018.

[16] I. Kuon and J. Rose, "Measuring the gap between fpgas and asics," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, Feb 2007.

[17] D. Markovic, C. Chang, B. Richards, H. So, B. Nikolic, and R. W. Brodersen, "Asic design and verification in an fpga environment," in *2007 IEEE Custom Integrated Circuits Conference*, Sept 2007, pp. 737–740.

[18] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: Constructing hardware in a scala embedded language," in *DAC Design Automation Conference 2012*, June 2012, pp. 1212–1221.

[19] Y. Guo, "A Survey on Methods and Theories of Quantized Neural Networks," *ArXiv e-prints*, Aug. 2018.

[20] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1," *CoRR*, vol. abs/1602.02830, 2016. [Online]. Available: http://arxiv.org/abs/1602.02830

[21] Y. Umuroglu and M. Jahre, "Streamlined deployment for quantized neural networks," *CoRR*, vol. abs/1709.04060, 2017. [Online]. Available: http://arxiv.org/abs/1709.04060

[22] "Edge tpu," https://cloud.google.com/edge-tpu/, accessed: March 13, 2019.

[23] "An in-depth look at google's first tensor processing unit (tpu)," https://cloud.google.com/blog/products/gcp/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu, accessed: March 13, 2019.

[24] "Synopsys investor relations," https://www.synopsys.com/content/dam/synopsys/company/investor-relations/corporate-overview-investor-q1-2019.pdf, accessed: March 13, 2019.

[25] Synopsys, *Design Compiler® User Guide O-2018.06, June 2018*. Synopsys, 2018.

[26] A. Andreev, A. Bolotov, and R. Scepanovic, "Fifo memory with single port memory modules for allowing simultaneous read and write operations," U.S. Patent US7 181 563B2, Feb. 20, 2007.

[27] "Zynq-7000 soc first generation architecture," https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf, accessed: May 23, 2019.

REFERENCES

A | Complete Synthesis Results

A.I ASIC BSMMA Results

The Configuration, symmetric DPA size, critical path(ns), maximum frequency(MHz), number of cells, and the designs cell area(μm^2) are respectively shown in the table columns.

overlay configuration	nr	crit path	max freq	cell count	cell area
2x128x2	2	0.3885	2574.002574	74062	137197.5072
4x128x4	4	0.3922	2532.286655	121033	191694.0683
6x128x6	6	0.3945	2503.128911	192358	277616.0956
8x128x8	8	0.4027	2402.11386	291071	393834.082
10x128x10	10	0.4053	2374.169041	416135	541445.3841
12x128x12	12	0.4085	2341.372044	565462	718594.0917
14x128x14	14	0.4073	2351.834431	744523	928986.8031
32x128x32	32	0.439	2277.9043	3355057	4104072.41

Table A.1: Clock Constrained ASIC BSMMA Black Box Results

overlay configuration	nr	crit path	max freq	cell count	cell area
2x128x2	2	0.9819	1025.115325	69483	135616.2624
4x128x4	4	0.9819	1025.115325	110847	187159.8827
6x128x6	6	0.9819	1025.115325	176454	268744.2173
8x128x8	8	0.9819	1025.115325	265742	379599.2885
10x128x10	10	0.9819	1025.115325	379426	520785.7332
12x128x12	12	0.9819	1025.115325	517014	691552.3418
14x128x14	14	0.9819	1025.115325	678784	892191.4054

Table A.2: Clock Relaxed ASIC BSMMA Black Box Results

overlay configuration	nr	crit path	max freq	cell count	cell area
2x128x2	2	0.3895	2567.3940	397854	525236.0264
4x128x4	4	0.3895	2567.3940	441303	578233.9219
6x128x6	6	0.3971	2518.2573	674303	858446.6921
8x128x8	8	0.4037	2477.0869	598362	778189.9925
10x128x10	10	0.4045	2472.1878	800500	1022458.1267
12x128x12	12	0.4033	2479.5437	1030527	1298672.1719
14x128x14	14	0.4142	2414.2926	1278722	1598911.2162

Table A.3: ASIC BSMMA Register Memory Results

A.II ASIC DPA Results

conf	nr	critpath with slack	max frequency	cellcount	area
2x128x2	2	0.3895	2567.394095	12457	14006.8035
4x128x4	4	0.3926	2547.121752	49439	56075.8475
6x128x6	6	0.3935	2541.296061	111272	126400.0346
8x128x8	8	0.3917	2552.974215	196124	223559.8509
10x128x10	10	0.3952	2530.364372	270800	329957.7672
12x128x12	12	0.4072	2455.795678	389952	475139.1847
14x128x14	14	0.4026	2483.854943	530768	646717.2236

Table A.4: ASIC DPA Results

A.III BISMO FPGA Resource Utilzation

The symmetric DPA dimesions, design overlay configuration, number of utilized LUTs, maximum number of LUTs on the PYNQ-Z1 device, device LUT utilization(%), number of utilized BRAM tiles, maximum number of BRAM tiles on the PYNQ-Z1 device and device BRAM utilization(%) are respectively shown in the table columns.

nr	conf	lut	lut max	lut util	bram	bram max	bram util
2	2x64x2	3902	53200	7.33	129	140	92.14
4	4x64x4	6676	53200	12.55	129	140	92.14
6	6x64x6	13812	53200	25.96	192	140	137.14
8	8x64x8	19170	53200	36.03	121	140	86.43
10	10x64x10	26075	53200	49.01	150	140	107.14
12	12x64x12	37056	53200	69.65	180	140	128.57
14	14x64x14	46277	53200	86.99	210	140	150
16	16x64x16	59853	53200	112.51	129	140	92.14

Table A.5: BISMO FPGA Resource Utilization 64-bit width

nr	conf	lut	lut max	lut util	bram	bram max	bram util
2	2x128x2	4444	53200	8.35	129	140	92.14
4	4x128x4	9593	53200	18.03	121	140	86.43
6	6x128x6	18577	53200	34.92	180	140	128.57
8	8x128x8	27792	53200	52.24	129	140	92.14
10	10x128x10	41059	53200	77.18	160	140	114.29
12	12x128x12	55568	53200	104.45	192	140	137.14
14	14x128x14	75474	53200	141.87	224	140	160

Table A.6: BISMO FPGA Resource Utilization 128-bit width

nr	conf	lut	lut max	lut util	bram	bram max	bram util
2	2x256x2	5458	53200	10.26	121	140	86.43
4	4x256x4	13843	53200	26.02	129	140	92.14
6	6x256x6	28121	53200	52.86	192	140	137.14
8	8x256x8	47135	53200	88.6	129	140	92.14
10	10x256x10	68519	53200	128.8	160	140	114.29

Table A.7: BISMO FPGA Resource Utilization 256-bit width

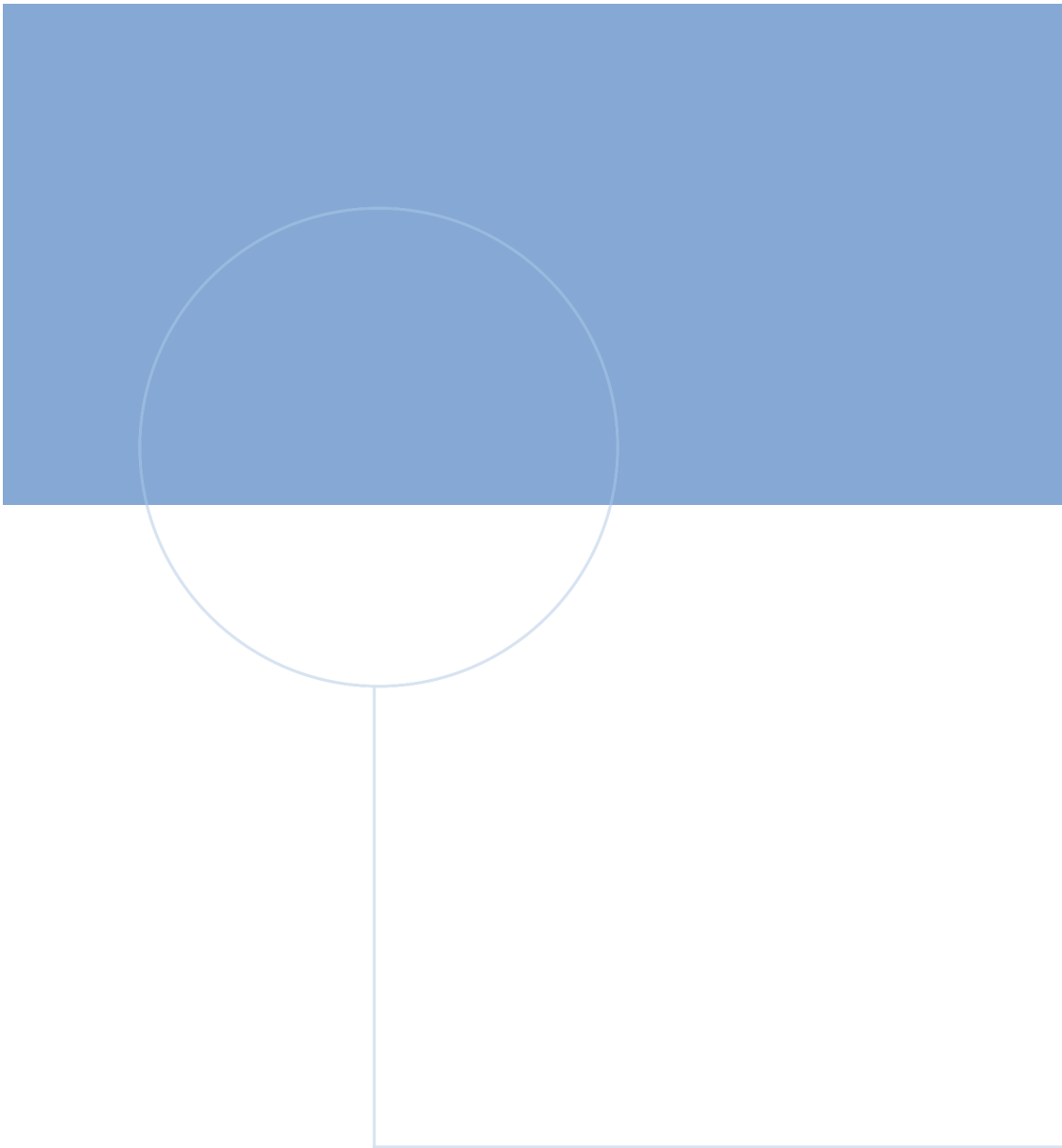
B | TCL Scripts

```
#TCL that compiles, stores and reports the design
saif_map -start
set synscriptdir $::env(PROJECTDIR)
set scripts $synscriptdir/scripts
set constraints $scripts/constraints
source ${synscriptdir}/scripts/read_hdl.tcl
#For a top-down approach only compile BSMMA
#source $scripts/compile_pcu.tcl
#source $scripts/compile_dpu.tcl
#source $scripts/compile_dpa.tcl
source $scripts/compile_BitSerialMatMulAccel.tcl
change_names -rules verilog -hierarchy
write -format verilog -hierarchy -output
   BitSerialMatMulAccel.vg
write_sdf BitSerialMatMulAccel.sdf
write_sdc BitSerialMatMulAccel.sdc
write_parasitics -o BitSerialMatMulAccel.spef
saif_map -type ptpx -write_map BitSerialMatMulAccel.map
report_timing -significant_digits 4 > ./report_timing.txt
report_qor -significant_digits 4 > ./report_qor.txt
report_power > ./report_power.txt
exit
```

Listing B.1: Complete compilation

```
#Module compiliation TCL
#compile_BitSerialMatMulAccel.tcl
elaborate BitSerialMatMulAccel
source $constraints/BitSerialMatMulAccel.tcl
compile
write -format ddc -hierarchy -output BitSerialMatMulAccel.ddc
```

Listing B.2: BitSerialMatMulAccel compilation



NTNU
Norwegian University of
Science and Technology