Reconfigurable Architectures and Systems for IoT Applications

by

Naveen Suda

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved December 2015 by the
Graduate Supervisory Committee:

Yu Cao, Chair
Bertan Bakkaloglu
Sule Ozev
Shimeng Yu
Jae-sun Seo

ARIZONA STATE UNIVERSITY

May 2016

ABSTRACT

Internet of Things (IoT) has become a popular topic in industry over the recent years, which describes an ecosystem of internet-connected devices or things that enrich the everyday life by improving our productivity and efficiency. The primary components of the IoT ecosystem are hardware, software and services. While the software and services of IoT system focus on data collection and processing to make decisions, the underlying hardware is responsible for sensing the information, preprocess and transmit it to the servers. Since the IoT ecosystem is still in infancy, there is a great need for rapid prototyping platforms that would help accelerate the hardware design process. However, depending on the target IoT application, different sensors are required to sense the signals such as heart-rate, temperature, pressure, acceleration, etc., and there is a great need for reconfigurable platforms that can prototype different sensor interfacing circuits.

This thesis primarily focuses on two important hardware aspects of an IoT system: (a) an FPAA based reconfigurable sensing front-end system and (b) an FPGA based reconfigurable processing system. To enable reconfiguration capability for any sensor type, Programmable ANalog Device Array (PANDA), a transistor-level analog reconfigurable platform is proposed. CAD tools required for implementation of front-end circuits on the platform are also developed. To demonstrate the capability of the platform on silicon, a small-scale array of 24×25 PANDA cells is fabricated in 65nm technology. Several analog circuit building blocks including amplifiers, bias circuits and filters are prototyped on the platform, which demonstrates the effectiveness of the platform for rapid prototyping IoT sensor interfaces.

IoT systems typically use machine learning algorithms that run on the servers to process the data in order to make decisions. Recently, embedded processors are being used to preprocess the data at the energy-constrained sensor node or at IoT gateway, which saves considerable energy for transmission and bandwidth. Using conventional CPU based systems for implementing the machine learning algorithms is not energy-efficient. Hence an FPGA based hardware accelerator is proposed and an optimization methodology is developed to maximize throughput of any convolutional neural network (CNN) based machine learning algorithm on a resource-constrained FPGA.

DEDICATION

To my parents

ACKNOWLEDGMENTS

The years of research at Nanoscale Integration and Modeling (NIMO) group have given me opportunities to meet and collaborate with many inspiring people. It is a pleasure to convey my gratitude to all of them in my humble acknowledgement. First and foremost, I would like to thank my advisor Dr. Yu Cao for his supervision, encouragement and motivation on my research. I appreciate all his contributions of time, ideas and tremendous intellectual and moral support throughout this pursuit.

I would like to extend my gratitude to my co-advisor Dr. Bertan Bakkaloglu, for his support and guidance through the countless hours of inspiring discussions and constructive suggestions, all of which were required for the continued progress towards the end goal. I would also like to show my appreciation to Dr. Jae-sun Seo for his constant encouragement and insightful suggestions on the reconfigurable hardware accelerators, which were crucial for successful completion of this research.

I am also grateful to my other committee members Dr. Sule Ozev and Dr. Shimeng Yu for their precious time and effort in reviewing this work and providing invaluable inputs. I would also like to convey thanks to my mentors Dr. Nagib Hakim, Intel Corp., Santa Clara and Dr. Vikas Chandra, ARM Research, San Jose for their constructive discussions and suggestions on this research work, during my summer internships in 2014 and 2015.

The members of our Research group have contributed immensely for successful completion of this research work. I would like to thank Jounghyuk Suh, Cheng Xu, Ketul Sutaria, Zihan Xu, Raveesh Magod, Rongjun Zhu, Venkatesa Ravi, Abinash Mohanty, Atul Ramkumar, Pei An and Jyothi Bhaskarr Velamala (Amar) for their immense support

providing valuable feedback on my research. The group has been a great source of friendship and motivation. I would also like to thank James Laux for all his support with PDKs and help in solving technical issues during the tape-outs.

I am deeply indebted to my family, especially to my wife, Gayathri, for the unconditional love and support throughout my PhD. Finally, I would like to thank all my friends who were also important in the successful realization of this thesis.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

CHAPTER 1

INTRODUCTION

Internet of Things (IoT) is a popular topic in industry that has emerged over the recent years. The term “IoT” is used to describe an ecosystem of smart devices or things connected through internet that improve the quality of life by increasing our productivity and efficiency. It encompasses a wide range of applications such as wearables, smart appliances, home and industrial automation, automobiles and security. IoTs have already been experimentally and commercially deployed in some applications [1], such as health monitoring wearable devices, smart water and energy meters, asset tracking and manufacturing quality control in industrial applications, etc. However, there still are many critical challenges for large scale deployment of IoT devices in the real world, the most important of which is the energy efficiency.

The primary components of the IoT ecosystem are hardware, software and services. Software and services play a crucial role in functioning of the IoT ecosystem by collecting the sensor data, uploading it to cloud servers, processing or analyzing the data and presenting in an efficient and easily interpretable form in order to make decisions from the sensed information in real-time. The hardware sensor node is responsible for sensing the information and transmitting it to the cloud servers via the IoT gateway as shown in a typical IoT application scenario depicted in Figure 1.1. Actuator nodes, on the other hand, provide a means to act based on the decision made from the sensed information. For example, based on the detected motion and temperature, light and AC or heat can be automatically controlled to reduce overall energy usage.

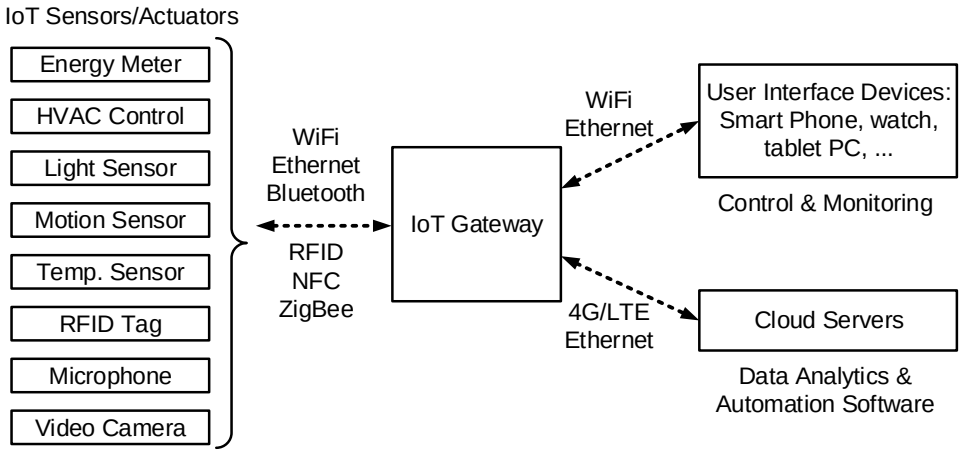


Figure 1.1. A typical IoT application scenario.

All the sensors and actuators are connected to the cloud via a local IoT gateway to enable remote monitoring and control and to perform data analytics or to run automation software on the sensed data. The sensors and actuators connect to IoT gateway through WiFi, Bluetooth, NFC, RFID, ZigBee or Ethernet cable, depending on multiple factors such as data volume to be transferred, input power supply of the sensor/actuator node and proximity of the node to the gateway. IoT gateway also allows node-to-node low-latency communication without having to connect to the cloud for small automation tasks.

The block diagram of a typical IoT sensor node is shown in Figure 1.2. It consists of a sensor, an Analog Front-End (AFE) circuit, an optional processing unit and a transceiver. Depending on the target IoT application, different sensors are required to sense the physical phenomena such as heart-rate, sound, temperature, pressure, acceleration, proximity, light intensity, etc. Sensors convert these physical phenomena to electrical property such as voltage, current, charge, impedance (resistance or capacitance), which is detected by subsequent analog front-end (AFE) signal conditioning circuits. Depending on the type of input sensor, AFE circuit consists of

trans-impedance amplifier, capacitive sensing charge amplifier or variable gain amplifier, filters and ADC to convert the conditioned analog signal to digital format. It is followed by pre-processing unit and transceiver to send the data to the cloud server for further processing or analytics.

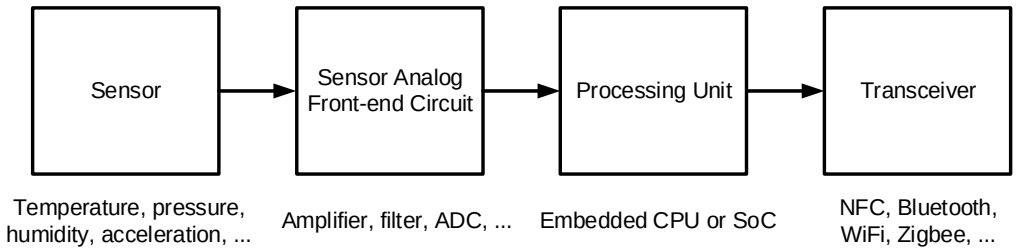


Figure 1.2. Block diagram of an IoT smart sensor node.

Since the IoT ecosystem is still in a developing phase, rapid prototyping platforms would help accelerate the hardware design process. Furthermore, rapid prototyping platforms are required for validation of IoT sensor AFE circuits before an expensive tape-out. However, designing a generic sensor front-end circuit to handle all the different signal types and magnitudes is non-trivial. Field-Programmable Analog Arrays (FPAAs) provide a good solution for rapid prototyping of such sensor AFE circuits.

IoT systems typically employ machine learning algorithms, especially artificial neural networks that run on the servers to process the data in order to make decisions. If the sensed data is directly transmitted to the cloud, it could consume large amount of network bandwidth and also lead to high latency. However, if the sensed data is processed at sensor node or at the IoT gateway, it reduces the consumption of the expensive network bandwidth and also minimizes the overall latency of the IoT application. Moreover, if pre-processing or analytics of the data is performed locally,

then only required information is transmitted to the next tier, i.e. the cloud, thus offloading the datacenters to perform other critical tasks.

Sensor nodes in most IoT applications would be powered from battery or energy harvesters and hence high energy-efficiency is of paramount importance. It is not energy-efficient to perform compute and memory intensive processing required for neural networks algorithms on such energy-constrained sensor nodes. Hence, it is more suitable to pre-process the data at an IoT gateway, which is powered from wall outlet. Furthermore, processing capability of a single IoT gateway can be time-multiplexed and shared among multiple IoT applications. Using conventional embedded CPU based systems for implementing the neural network based processing systems is not energy-efficient and may not meet the real-time performance requirements in some critical applications. FPGA based hardware accelerators for neural networks have become popular among researchers mainly because of their high reconfigurability, short turnaround time, good performance and high energy-efficiency.

Convolutional Neural Networks (CNNs), inspired by visual cortex of the brain, are a category of feed-forward artificial neural networks. CNNs, which are primarily employed in computer vision applications such as character recognition [2], image classification [3]-[6], video classification [7], face detection [8], gesture recognition [9], etc., are also being used in a wide range of fields including speech recognition [10], natural language processing [11] and text classification [12]. Over the past decade, the accuracy and performance of CNN-based algorithms improved significantly, mainly due to the enhanced network structures enabled by massive training datasets and increased raw computational power aided by CMOS scaling to train the models in a reasonable

amount of time. Because of their ability to achieve accuracy close to or even better than human-level perception in a wide range of applications, CNNs are apt candidates that can be implemented in processing units of the IoT gateways.

1.1. Previous Research on FPAAs

Previous academic and industrial efforts focused on Field Programmable Analog Arrays (FPAA) with a wide variety of continuous time and discrete time based Configurable Analog Blocks (CAB) consisting of coarse grained macros like operational amplifiers [13], operational transconductance amplifiers [14]-[17], ADCs [18], DACs [19], variable gain amplifiers and reconfigurable filters [20] along with common analog primitives like capacitors and MOS resistors [21]. Some researchers focused on CABs with medium grained primitives like differential pairs, matched loads and transconductors [22], [23]. FPAAs with transistor-level CABs known as Field Programmable Transistors Arrays (FPTAs) are explored in [24], [25] for evolvable hardware applications. Some other researchers developed FPAAs with CABs consisting of a combination of coarse-grained macros along with programmable transistors using floating-gate transistor as reconfiguration switches [26]-[30]. However, due to the wide variety of analog circuits like bias circuits, amplifiers, filters, switching circuits, oscillators etc., it is not possible to map an arbitrary analog function of an IoT sensor AFE circuit to a generic set of CABs.

Hence a transistor-level reconfigurable analog platform, named Programmable ANalog Device Array (PANDA), is proposed in [31], which enables rapid prototyping and validation of AMS circuits across different technology nodes. Previous work on PANDA [32] focuses on the cell design based on the device physics and proposes a

mapping methodology to match device characteristics like current (I_D), transconductance (G_m) and output resistance (R_{out}) of each transistor in the target circuit to a PANDA cell. It further demonstrates the potential of the methodology by emulating a set of AMS circuits designed in 22nm and 90nm nodes using a common 45nm PANDA platform, which is termed as forward emulation (FE) and backward emulation (BE). However, it is primarily based on simulations using Predictive Technology Models (PTM) [33] and does not consider circuit performance degradation because of the reconfigurable interconnect.

1.2. Previous Research on Hardware Accelerators

A typical CNN architecture has multiple convolutional layers which extract features from the input data, followed by classification layers. The operations in CNNs are computationally intensive with over billion operations per input image [4]-[6], thus requiring high performance server CPUs and GPUs to train the models. However, they are not energy efficient and hence various hardware accelerators have been proposed based on FPGA [34]-[38], SoC (CPU + FPGA) [39] and ASIC [40]. FPGA based hardware accelerators have gained momentum owing to their reconfigurability and fast development time, especially with the availability of high-level synthesis (HLS) tools from FPGA vendors. Moreover, FPGAs provide flexibility to implement the CNNs with limited data precision which reduces the memory footprint and bandwidth requirements, resulting in a better energy efficiency (e.g. GOPS/Watt).

Previous FPGA-based CNN accelerator designs primarily focused on optimizing the computational resources without considering the impact of the external memory transfers [34]- [36] or optimizing the external memory transfers through data reuse [37],

[38]. Majority of the previous work implemented generic CNN accelerators that are independent of the model configuration (i.e., the number of layers and convolution kernel size) [34]-[37], [39] and hence they do not fully utilize the hardware resources to maximize the throughput. The authors of [38] proposed a design space exploration methodology for CNN accelerator by optimizing both computation resources and external memory accesses, but implemented only convolution layers.

1.3. Contribution of this work

This thesis mainly focuses on two important hardware aspects of an IoT system: (a) an FPAA based reconfigurable sensing front-end and (b) an FPGA based hardware accelerator for reconfigurable processing system. To enable reconfiguration capability for any sensor type, Programmable ANalog Device Array (PANDA), a transistor-level analog reconfigurable platform is proposed in [31]. This work extends the previous simulation framework on PANDA [32] to silicon implementation of the full system in 65nm CMOS technology, including an array of 24×25 PANDA cells, reconfigurable interconnect, configuration memory and interface for programming.

To map AMS circuits onto the PANDA platform, a new set of computer-aided design (CAD) tools are required for design partitioning, technology mapping, placement, routing and configuration bit-stream generation. Despite the mature FPGA CAD methodologies, they do not quite suit well for PANDA because of the intrinsic differences in nature of analog circuit behavior with routing switches, heterogeneous cell types and special requirements like circuit topology, matching and sensitive nodes etc. Hence a new CAD tool, PANDA-PRO is developed for implementation of AMS circuits on the designed platform.

Several analog circuit building blocks including amplifiers, voltage and current references and active filters are designed in 65nm technology and mapped to the platform using PANDA-PRO. Measured performance metrics of these circuits show a good match with those of target designs, demonstrating the effectiveness of the platform for rapid prototyping IoT sensor front-end circuits. Dynamic reconfiguration capability of the platform is demonstrated through input offset cancellation of an amplifier implemented on the platform along with an FPGA in a closed loop. This capability, which enables on-the-fly reconfiguration of the PANDA cell size and connectivity, provides new opportunities for validation of self-calibrating and adaptive circuits which cost considerable amount of simulation time to validate.

An FPGA based hardware accelerator for Convolutional Neural Networks (CNN) is proposed for implementing in the processing unit in an IoT gateway. CNN model consisting of all the layers: convolution, normalization, pooling and classification layers with fixed-point operations are implemented on FPGA using OpenCL based high-level synthesis (HLS) framework. Critical design variables that impact the throughput are identified for optimization. Execution time of each CNN layer is analytically modeled as a function of the design variables and validated on FPGA. Logic utilization is empirically modeled using FPGA synthesis data for each CNN layer as a function of the design variables. A systematic methodology is proposed to minimize total execution time of a given CNN algorithm, subject to the FPGA hardware constraints of logic utilization, computational resources, on-chip memory and external memory bandwidth.

The new methodology is demonstrated by implementing and maximizing the throughput of two state-of-the-art large-scale CNNs: AlexNet [4] and VGG [5] (which

achieved top accuracies in ImageNet challenges 2012 and 2014, respectively), on two Altera FPGA platforms with different hardware resources.

1.4. Thesis Organization

The organization of the thesis report is as follows: chapter 2 presents the overview of developed 65nm PANDA platform for rapid-prototyping IoT sensor front-end circuits. It also details the PANDA cell structure, architecture of reconfigurable interconnect and presents the cell-level measured I-V characteristics to validate the approach at device-level. Chapter 3 presents computer aided design (CAD) tool methodologies developed for implementing any analog circuits on PANDA platform, which includes automatic mapping, placement, routing and bit-stream generation. Chapter 4 presents the measurement results of benchmark circuits implemented on 65nm PANDA platform and outlines the limitations and other applications of the platform. Chapter 5 presents the FPGA based hardware accelerator for a class of neural networks known as convolutional neural networks that can be used in IoT pre-processing unit. Chapter 6 summarizes the key contributions of this work along with recommendations for future work.

CHAPTER 2

OVERVIEW OF 65NM PANDA PLATFORM

Figure 2.1 shows the block diagram of PANDA system, which consists of an array of reconfigurable cells along with island-style interconnect and segmented routing channels similar to digital FPGAs [41]. Since each cell can be connected to any other cell through the reconfigurable connection block (CB) and switch blocks (SB) as shown in Figure 2.1(b), this platform is versatile for implementation of any analog circuit. To demonstrate the potential of PANDA for hardware validation on silicon, a full system consisting of 24×25 array of programmable cells, reconfigurable interconnect, configuration memory and computer interface for configuration is designed and fabricated in a 65nm CMOS technology.

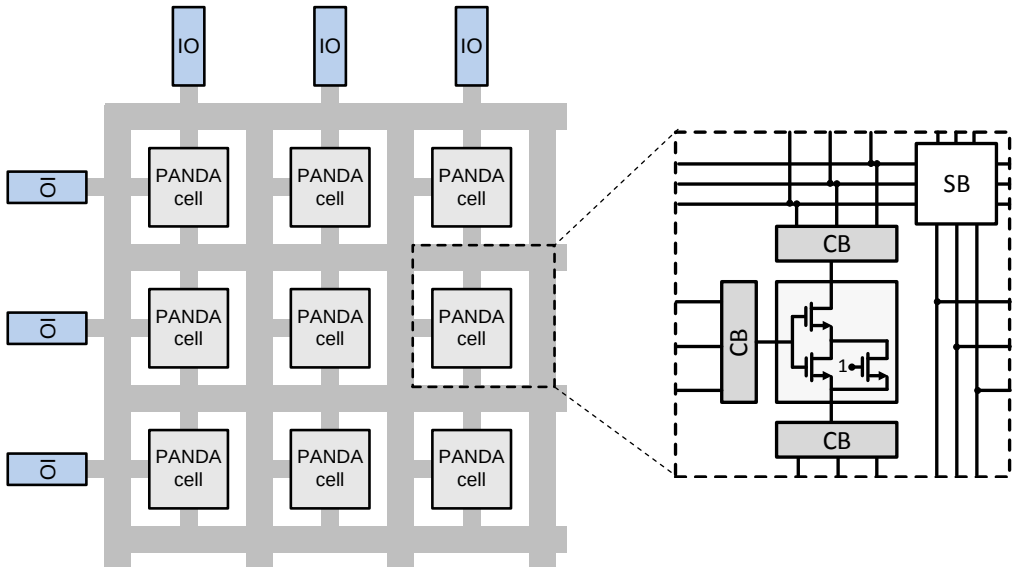


Figure 2.1. Architecture of PANDA. (a) Island-style reconfigurable interconnect similar to FPGAs. (b) A typical tile consisting of a PANDA cell connected to the reconfigurable interconnect via connection blocks (CB) and switch blocks (SB). A PANDA cell with 3-transistor stack is shown as an example.

2.1. PANDA Cell Structure

Implementation of an AMS circuit on PANDA platform requires mapping of each transistor to a PANDA cell to match the drain current (I_D), output resistance (R_{out}) and transconductance (G_m). To implement any generic AMS circuit, PANDA cells comprise of transistors with different widths and lengths. For practical implementation, transistor widths are discretized using binary weighted transistors (1x, 2x and 4x) in parallel to a fixed transistor (Fx) as shown in Figure 2.2(a), such that the effective transistor width can be configured from Fx to Fx+7x using the digital controls (b_0, b_1, b_2). 6 different types of PANDA cells each with different fixed and variable programmable width transistors are designed to achieve an effective width coverage of 80nm to 10μm with a maximum discretization error of 10%. The transistor sizes in these 6 cell types are shown in Table 2.1. If a circuit demands for more accuracy, PANDA cells with coarse width transistors can be used in parallel to cells with fine width transistors.

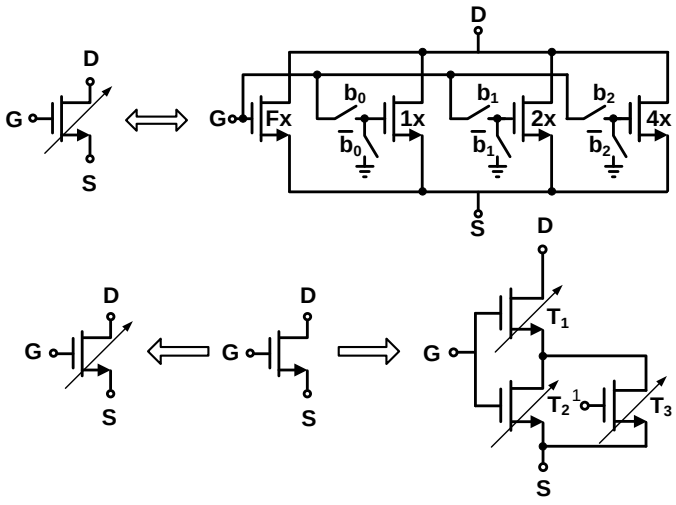


Figure 2.2. (a) Programmable NMOS transistor using digital controls (b_0, b_1, b_2) (b) Target analog transistor mapped to either 1-transistor PANDA cell (left) or 3-transistor PANDA cell (right) based on I_D, R_{out} and G_m requirements.

Table 2.1. Transistor Size Discretization

Type	Fixed Width	Programmable width
1	-	80nm, 120nm, 160nm
2	-	90nm, 100nm, 140nm
3	320nm	80nm, 160nm, 320nm
4	850nm	170nm, 340nm, 680nm
5	2.0μm	400nm, 800nm, 1.6μm
6	5.76μm	960nm, 1.92μm, 3.84μm

To map long-channel transistors that have higher output resistance, transistors T1 is stacked to T2 to form a cascode pair as shown in Figure 2.2 (b). Transistor T3 with its gate connected to VDD is added to the stack for flexibility in matching I_D, R_{out} and G_m. Two variants of transistor length L=60nm and L=120nm are provided in the platform for flexibility to map a wide range of target transistors. To add versatility to the platform, programmable resistors, capacitors and parasitic BJTs are also incorporated into the array. The different types of PANDA cells, their sub-types and number of cells of each type present in the array are summarized in Table 2.2.

To demonstrate the transistor-level I-V characteristic matching between a target transistor and PANDA cell, an NMOS transistor of W/L=560nm/60nm is mapped to a 1-transistor PANDA cell at bias conditions of V_G=0.4V, V_D=0.5V and V_S=0V. Measured I-V characteristics of the mapped PANDA cell compared to the simulated characteristics of the target transistor is shown in Figure 2.3. I_D-V_D characteristics show a close match at the nominal bias of V_G=0.4V. I_D-V_G characteristics match till V_G=0.5 V, but after that voltage drop across the routing switches increases which reduces the effective V_D at the transistor drain terminal.

Table 2.2. Summary of PANDA Cell Types

PANDA cell type	No. of sub-types	No. of cells
1-Transistor cell with L=60nm	NMOS: 6, PMOS: 6	132
1-Transistor cell with L=120nm	NMOS: 6, PMOS: 6	132
3-Transistor cell with L=60nm	NMOS: 6, PMOS: 6	120
3-Transistor cell with L=120nm	NMOS: 6, PMOS: 6	120
Programmable resistor (4kΩ to 252kΩ in steps of 8kΩ)	1	32
Programmable capacitor (0.4pF to 1.6pF in steps of 0.4pF)	1	32
Parasitic BJTs	NPN: 1, PNP: 1	32

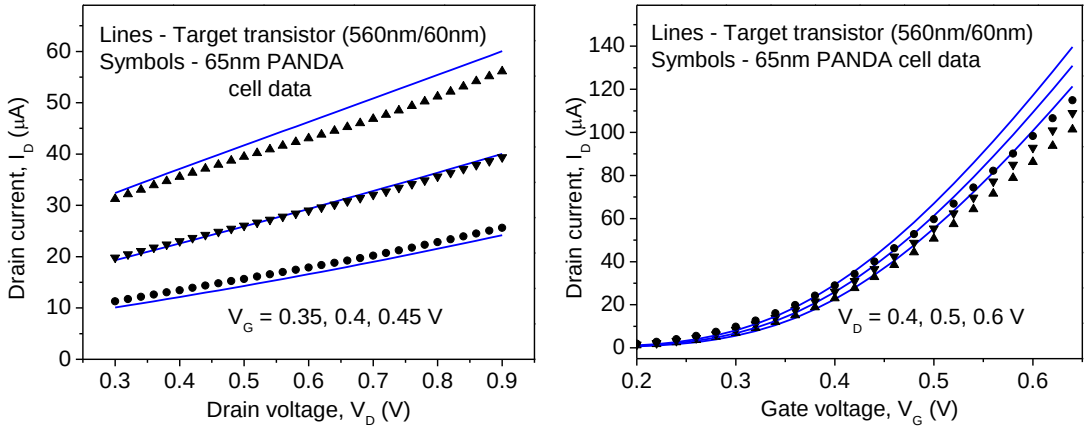


Figure 2.3. Measured I-V characteristics of PANDA cell with effective W/L = 8×80nm/60nm emulating a target transistor of W/L=560nm/60nm mapped at nominal V_G=0.4V, V_D=0.5V, V_S=0.

Figure 2.4 shows the measured I-V characteristics of a 3-transistor PANDA cell which maps to a long-channel transistor of W/L=560nm/260nm at bias conditions of V_G=0.6V, V_D=0.6, V_S=0V. In Figure 2.4 I_D-V_G characteristics, the change in I_D with V_D is minimal due to the high output resistance of the long-channel transistor.

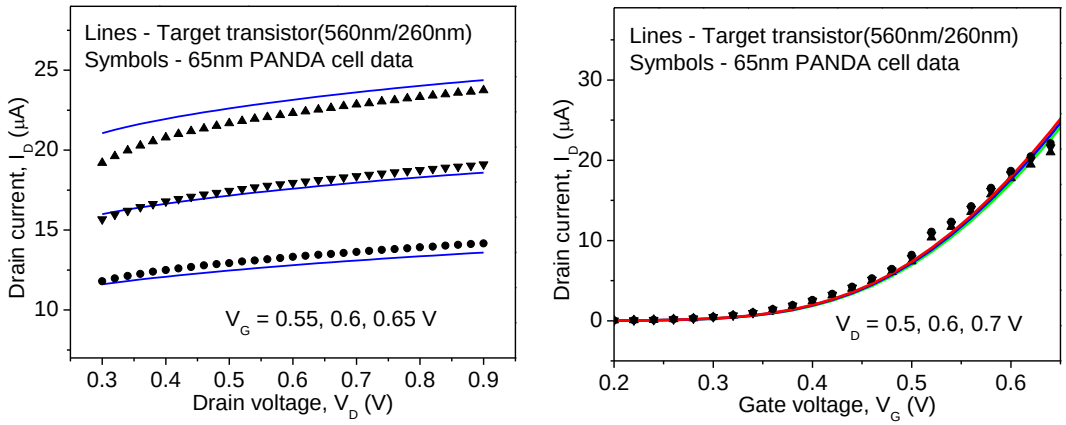


Figure 2.4. Measured I-V characteristics of a 3-transistor PANDA cell with T1 and T2 as 230nm/120nm and T3 is OFF emulating an analog transistor of W/L=560nm/260nm at nominal V_G=0.6V, V_D=0.6V, V_S=0.

2.2. Reconfigurable Interconnect

Previous FPAAs comprised of opamp based Configurable Analog Blocks (CABs) used crossbar interconnects, where each cell I/O has a dedicated routing channel that runs over the entire length of the array. This routing segment has switches to each of the perpendicular routing channels. Since the array has less number of such coarse-grained CABs, the crossbar interconnect architecture is feasible in those FPAAs. On the contrary, in PANDA where there are 100's of PANDA cells distributed across the platform, crossbar interconnects are not feasible as the routing switch cost increases in the order of $O(N^2)$ [42]. PANDA utilizes the island-style routing architecture in conjunction with segmented routing architecture [41] from FPGAs as shown in Figure 2.5, where each cell connects to any other cell in the array through the reconfigurable interconnect. Each tile consists of a PANDA cell, connection blocks (CB), switch block (SB) and configuration memory. Connection block is required for each terminal of the PANDA cell, to connect a cell to the routing tracks. Switch blocks are located at intersection of horizontal and

vertical tracks to connect a source track to destination track(s). Each tile is carefully designed so that the configuration memory required for configuring the transistor widths, CB and SB connectivity, is present in the same tile.

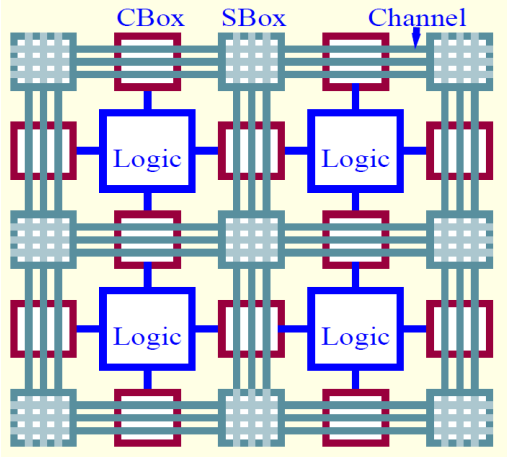


Figure 2.5. Mesh Interconnect architecture used in FPGAs [42]

The main drawback of island-style architecture with routing segments that extend only one block length is that a signal from a cell to other cell which is ‘n’ blocks away has to pass through ‘n’ SB switches and 2 CB switches. This degrades/destroys the analog circuit functionality for large ‘n’, because of the resistance of the each switch along the path. Hence segmented routing similar to FPGAs is used, where each routing track extends to more than one block length before ending at a SB. This reduces the number of passing SB switches when connecting two distant cells. In the current implementation, routing tracks with segments which extend up to 16 blocks are provided for parasitic reduction while routing. Routing tracks with smaller segment lengths are also provided in the platform for flexibility in routing when connecting nearby cells.

Switch block (SB) is situated at the intersection of horizontal and vertical routing tracks and connects an incoming track to some specific outgoing tracks based on the

architecture of the switch block. The number of tracks to which each incoming track can connect is known as switch block flexibility (Fs), where Fs=3 in typical FPGAs. There are different switch block architectures available in FPGA literature such as disjoint [43], universal [44], Wilton [45] and Imran switch block [46] each having its own advantages/disadvantages. Imran switch block is used in PANDA, as it is area efficient and gives best routable designs for segmented routing architecture [46]. The number of tracks to which connection block connects, is called connection block flexibility (Fc), which should be half the total number of tracks for best routability and area efficiency [41]. Hence the connection block in PANDA is designed such that it connects to 7 routing tracks among the total of 13 tracks.

The extracted parasitic resistance and capacitance of a routing line that connects two adjacent cells are 7Ω and 1fF, respectively, which are considerably less than those of the switches. The total capacitance associated with a routing line can be evaluated as sum of parasitic capacitance of all the switches connected to that line and the capacitance of the routing segment itself. For example, the total estimated capacitance on a long routing line that extends to 16 blocks is 216 fF, whereas the total capacitance on a short line that connects adjacent cells is 21 fF.

2.3. Reconfiguration switches

Similar to FPGAs, transmission gate (TG) based switches are used in the reconfigurable routing, as they can easily be configured by changing the memory locations that drive the gates of the TG. However, different from the switches in FPGAs, switches in the PANDA platform carry DC current, which may induce voltage drop affecting the DC bias conditions and may also destroy the circuit functionality. Increasing

the TG size reduces the resistance, but it also increases the parasitic capacitance, which in-turn decreases the maximum operating frequency of the platform. Hence TG sizing is critical for the overall functionality and performance of the platform. NMOS transistor is sized to 3μm/60nm and PMOS transistor to 6μm/60nm to reduce the maximum ON resistance of TG switch to 400Ω, whereas the approximate capacitance of the switch is 10fF. External I/Os are also connected to the internal routing tracks through the same reconfiguration switches.

2.4. Configuration Interface and System Integration

Memory required to store the PANDA cell sizing, connection block and switch block connectivities is placed in each tile. Byte-wise addressable memory is distributed into two columns in each tile for ease in routing of the memory outputs to the reconfiguration switches. Although the cells have different dimensions, in order to maintain the uniform array structure, all the tiles are designed to have uniform width while two tile heights are chosen based on the number of reconfiguration switches. This gives different ratios of area of actual transistor to that of the entire tile as 0.05% for the smallest cell and 17% for the largest cell. This shows that over 80% of the total chip area is occupied by the reconfigurable routing fabric.

To transfer configuration bit-stream from computer to the PANDA platform a customized Serial Peripheral Interface (SPI) protocol is used. Using 8 row address bits and 6 column address bits, 16KB of configuration memory can be accessed using this customized SPI protocol. A SPI slave along with row/column address decoding logic, data/address buffers is incorporated into the system to help in reconfiguration. To simplify the circuit, only write access is provided to the configuration memory whereas

both read and write access is given to the test registers distributed across the system, to help debug the SPI interface. The micrograph of the PANDA die fabricated in a standard 65nm digital CMOS technology with 1-poly and 8 metal layers is shown in Figure 2.6. It occupies a total area of 3.46 mm^2 including the ESD protection diodes and I/O bonding pads.

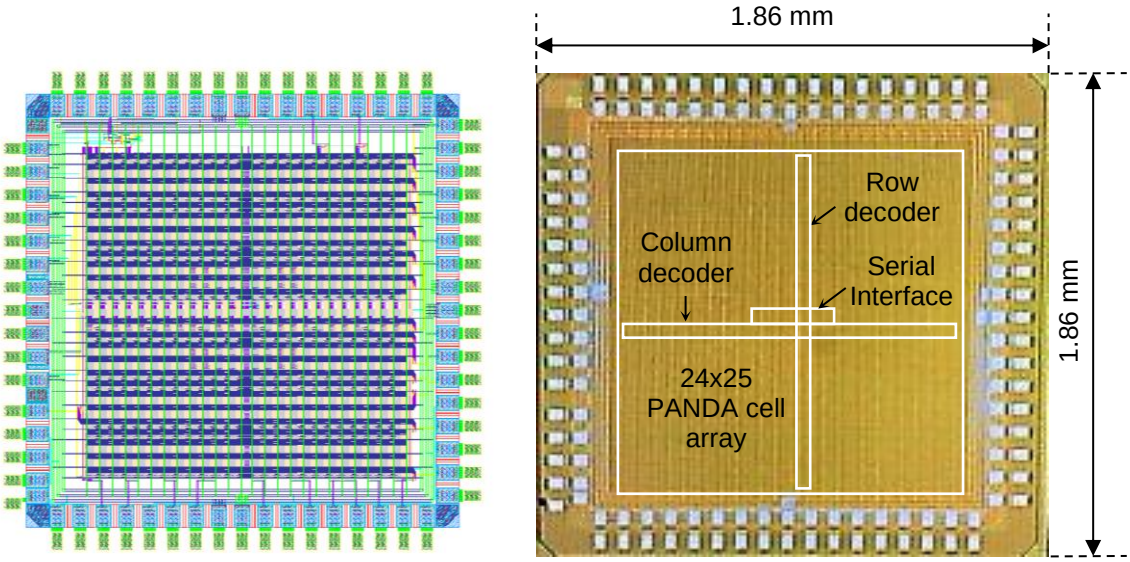


Figure 2.6. Layout (left) and die micrograph (right) of the 65nm PANDA chip.

The parameters of the designed 65nm PANDA platform are summarized in Table 2.3. The CAD tool flow and the methodology to implement any AMS circuit on the 65nm PANDA platform are detailed in the next chapter.

Table 2.3. Summary of Parameters in PANDA Platform

Process node	65 nm
Die size	1.86mm × 1.86mm
Power supply	1.2 V
PANDA cells	600
Programmable I/Os	24
Width programming control bits	3,368
Total routing switches	32,040

CHAPTER 3

COMPUTER AIDED DESIGN (CAD) METHODOLOGY

The digital FPGA platforms gained popularity in the design community since they facilitate rapid prototyping and design validation of digital circuits. Computer Aided Design (CAD) tools play a major role in the development of FPGAs as they efficiently map the design to the logic cells in FPGAs. In previous FPAAs which had only few CABs, manual placement and routing is feasible. On the other hand, the number of cells and the complexity of interconnect in PANDA platform makes it impractical to manually place and route the cells. The main tasks of an effective CAD tool in the PANDA methodology are partitioning of the analog circuit into discrete transistors, mapping of each transistor to PANDA cells, placement, routing and configuration bit-stream generation. Despite the mature FPGA CAD methodologies, they do not quite suit well for PANDA because of the following intrinsic differences.

- **Heterogeneity:** All the logic blocks in typical digital FPGAs are identical, whereas PANDA has heterogeneous cells with different transistor sizes, resistors, capacitors etc. Recent FPGAs have heterogeneous blocks such as memory, arithmetic units along with conventional logic blocks and there are also tools available for such architectures [47]. However, these tools are not customized for mapping analog circuits to PANDA.
- **Routing Parasitics:** Parasitics from routing not only degrade circuit performance as that happens in digital FPGAs, but can also completely destroy the functionality of analog circuits because of the DC and AC voltage drops in switches. Hence

performance degradation because of interconnect fabric must be fully addressed at each step of placement and routing.

- **Special Requirements for Analog Circuits:** The new analog synthesis tool should be aware of different constraints such as circuit topology, matching and sensitive nodes etc., which are specific for analog circuits, but not required in FPGAs.
- **Design Scale:** Modern FPGAs typically have 0.1-4 million reconfigurable logic cells [48]; hence the CAD tools need to trade-off the quality of final solution for configuration speed. In a typical analog circuit which only has 100-1000 transistors, the tool can afford multiple placement and routing iterations to achieve the target performance and related accuracy.

Hence a new CAD tool, PANDA-PRO is developed overcoming the above shortcomings of FPGA CAD tools, to implement AMS circuits on PANDA platform. The steps involved in PANDA-PRO starting from mapping to bit-stream generation are summarized in Figure 3.1.

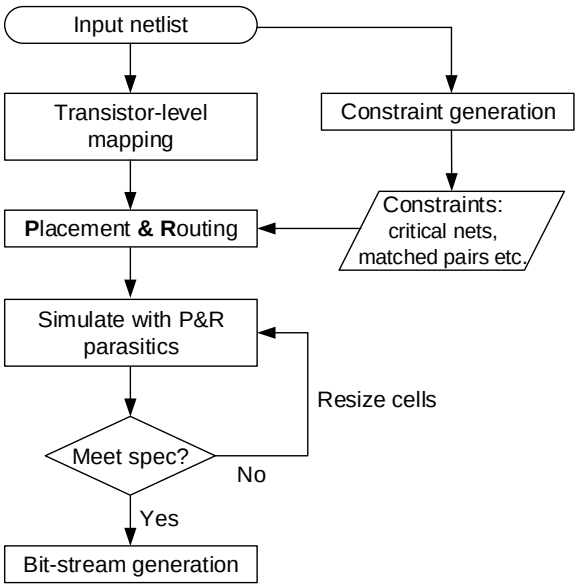


Figure 3.1. PANDA-PRO CAD tool flow.

3.1. Automatic Mapping

Mapping of the input design to PANDA cells is the fundamental and critical step in the implementation. First, the input design SPICE netlist is hierarchically partitioned and operating bias conditions of each transistor are extracted. From the circuit connectivity generic constraints such as input differential transistors and matched pairs are extracted. These constraints will be used in placement and routing phase. Then, each transistor is mapped to the PANDA cells for the extracted voltage bias conditions. Mapping involves sizing the transistors of the PANDA cell to match its I-V characteristics to that of the target transistor, thereby matching I_D, G_m and R_{out}. First, a coarse search on the discrete-sized 1-transistor and 3-transistor PANDA cell types in Table-2.1 is performed and the cell types which yield smaller error in I_D, G_m and R_{out} with respect to target transistor are selected for a detailed-search. In the detailed search, the transistor sizes in the selected PANDA cells are iteratively changed in the direction to reduce the error in I_D, G_m and R_{out}. The PANDA cell that achieves least error is selected as the final solution for that target transistor. Transmission gate switches are included into the cell in mapping stage, so that their impact is compensated during cell sizing.

3.2. Placement

PANDA-PRO placer is based on simulated-annealing [49], similar to traditional FPGA placers [50]; however the main differences being the heterogeneous cell types and additional design constraints of analog circuits. Simulated-annealing is a probabilistic global optimization method, emulating a physical process called annealing, where a material at high temperature (T) is gradually cooled to achieve a minimum energy state.

PANDA-cell mapped netlist is given to the placement tool which minimizes the placement cost given in the following equation.

$$\text{Placement cost} = \sum_{k}^{Nets} wt_k \left(\left| x_i - x_j \right| + \left| y_i - y_j \right| \right) \quad (1)$$

where wt_k is the weight assigned to each net, (x_i,y_i) and (x_j,y_j) are the coordinates of source and every destination cells for that net. Critical nets can be assigned higher weights so that its net length is optimized.

Placement process using simulated annealing is summarized as pseudo-code shown in Figure 3.2. The placement starts with an initial random placement at a high initial temperature (T), which is determined based on the circuit size [51]. If N_{inst} is the number of instances, then the initial temperature is set to 20 times the standard deviation of placement costs of N_{inst} random placements. Since different types of cells are present in the array, a lookup table is utilized to aid the placement process with information about the cell type, its locations in the array and whether the location is already occupied. First, a randomly chosen instance (I_1) is moved to a new location (X_1) of the same cell type. If this new cell location is already occupied by another instance (I_2), then the instances I_1 and I_2 are swapped. If it leads to a better placement, the move is accepted. If the new placement has higher cost, the move is accepted with certain probability and the acceptance probability reduces as the process continues. Temperature is slowly decreased while performing N moves at each temperature, where $N=10 \cdot N_{inst}^{4/3}$ [52]. The process is terminated when the placement cost converges.

```
1. Start with an initial random placement.
2. While (placement cost decreasing)
3.         Do N times at each temperature (T)
4.                 Select a random instance-I1.
5.                 Select a random location-X1 based on type of I1.
6.                 If X1 is already occupied with I2.
7.                         Swap I1 and I2.
8.                 Else
9.                         Place I1 at X1.
10.                Compute new cost and Δcost = (new cost-old cost).
11.                If (Δcost<0) or (random[0,1] < exp(-Δcost/T))
12.                        Accept this move.
13.                Else
14.                        Reject this move.
15.        Decrease T=0.95*T
```

Figure 3.2. Pseudo-code for placement algorithm based on simulated-annealing.

3.3. Routing

PANDA-PRO router is based on Dijkstra's algorithm [53] to find the shortest path from a source to destination utilizing fixed routing resources. For each net, the router starts expanding all possible connectivities from the source connection block, which is termed as *wavefront expansion*. Then the routing cost at each expanded node is computed from the following equation.

$$\text{cost}_i = \text{cost}_{i-1} + \text{cost}_{SB} + L_{seg}\text{cost}_{seg} \quad (2)$$

where $cost_{i-1}$ is the cost at the previous node, $cost_{SB}$ is the cost of switch block, L_{seg} is the length of the segment, $cost_{seg}$ is the cost of segment of length 1. The last term in the cost function ($L_{seg}cost_{seg}$) is added to penalize longer routing segments. When a wavefront reaches the destination, the routing cost is noted and wavefront expansion is continued till all the other nodes either exceed this routing cost or reach the destination. This makes sure that short segments are given higher priority than long segments for the same number of passing switches. The $cost_{SB}$ is set a large number compared to $cost_{seg}$, so that the router selects a route with a longer segment than the route that passes through multiple switch blocks, thereby reducing switch parasitics. An example of routing for reduced parasitics is shown in Figure 3.3. Using $cost_{SB}$=50 and $cost_{seg}$=1, the costs of route-1, 2 and 3 to route a net from source cell 'S' to destination cell 'D' are 154, 54 and 61, respectively. Route-1 passes through 3 switches and thus, has a high routing cost. Both route-2 and 3 pass through only 1 switch, but route-2 uses shorter length segments than route-3 and hence it has lower cost.

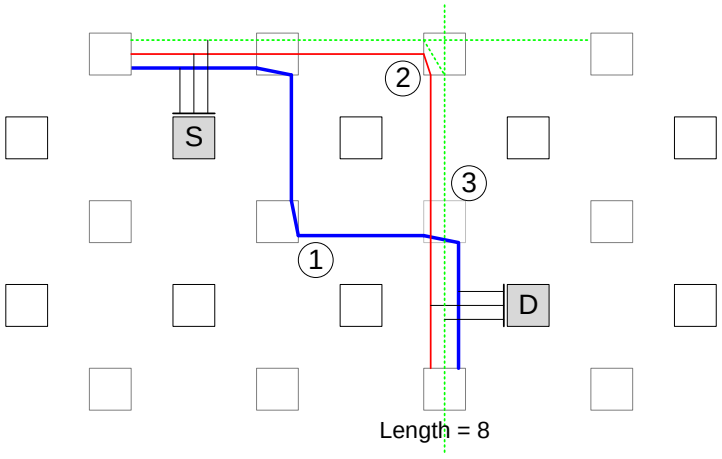


Figure 3.3. Illustration of routing for parasitic reduction: Among multiple routing solutions, route-2 is the final solution since it passes through only 1 switch block and uses routing segments of smaller length.

After placement and routing, parasitics from routing including are stitched back to PANDA-mapped netlist and simulated to verify if target circuit specifications are met. If the target specifications are not met, then the cells which are connected through large number of switches are resized to compensate for the DC drop across the switches. This process is repeated till the target specifications are met and then configuration bit-stream that controls the cell sizing is generated. To generate configuration bit-stream corresponding to the routing, all the nets in the circuit are traversed again from source to destination cell(s) using the final routing solution obtained in previous step. The CB and SB switches through which the net is routed are noted while traversing and the corresponding memory locations from look-up tables are set. Then, all the non-zero memory locations are written into a file in a {row address, column address and data} format, which will be serially transferred to the PANDA platform to configure the array.

The overall tool flow including the netlist parsing, constraint generation, transistor-level mapping using HSPICE simulations, place and route and post-route netlist generation is coded in Perl; while the placement, routing and bit-stream generation from the PANDA-mapped netlist are implemented in C. For graphical visualization of the placement and routing for debugging purposes, VPR tool [50] is customized for PANDA architecture.

Measurement results of the several analog circuit building blocks such as amplifiers, biasing circuits and active filters that are mapped to the 65nm PANDA platform are detailed in the next chapter.

CHAPTER 4

CIRCUIT MEASUREMENTS

Several fundamental AMS building blocks, such as amplifiers, filters, voltage and current reference circuits, are designed in 65nm technology and mapped to the PANDA platform to demonstrate the potential of this methodology for rapid prototyping and validation. Each of these designs including the biasing circuits are mapped to PANDA cells, placed, routed and configuration bit-stream is generated using the developed PANDA-PRO tool. This bit-stream is transferred from PC to the platform via an USB to SPI converter IC (MCP2210) [54]. SPI receiver in the PANDA platform receives the configuration bit-stream, decodes the addresses and sends the data to corresponding memory locations that configure the transistor sizing and connectivity of the routing network. Once programmed, the platform performs the functionality of the designed target circuit till power down, reset or reconfiguration. The typical measurement setup used for measurements of circuits implemented on PANDA platform is shown in Figure 4.1. PANDA chip is integrated on to a test board in a socket along with the peripherals such as voltage regulators, USB to SPI protocol converter IC (MCP2210) etc., as shown in the Figure 4.2 to help in circuit measurements in different configurations.

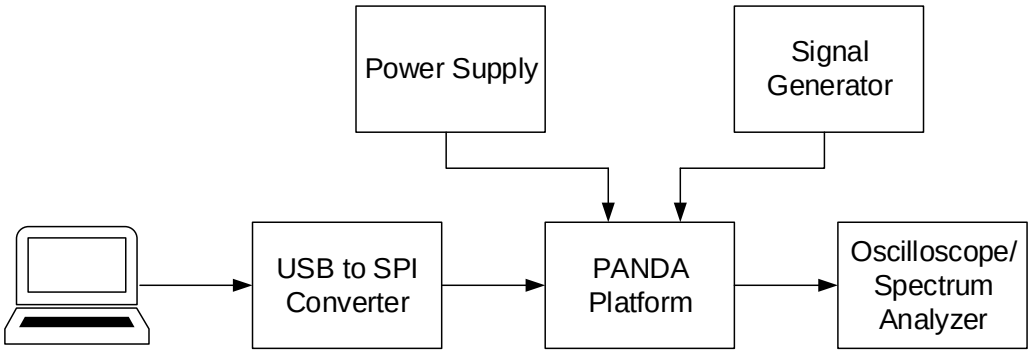


Figure 4.1. Typical measurement setup for configuration and circuit measurements.

Figure 4.2. Test board consisting of PANDA chip in a socket, an add-on board with MCP2210 for reconfiguration and other components for flexibility in measurements.

4.1. Benchmark Circuit Measurements

Amplifiers are the fundamental building blocks in analog IC design ranging from biasing circuits to precision amplification stages and filters. Basic amplifier topologies such as 5-transistor operational transconductance amplifier (OTA) and 2-stage Miller-compensated OTA are designed in 65nm technology and implemented on PANDA platform using the developed tool. The measured performance metrics such as DC gain, unity gain frequency, common mode rejection ratio (CMRR), power supply rejection ratio (PSRR), total harmonic distortion (THD) at 0.2 V output swing and current consumption of these circuits compared to the target design simulation results are shown in Table 4.1. The target circuits are simulated with a load capacitance of 15pF to account for pad and probe capacitance. The measurement results show a good match in DC characteristics including gain, CMRR, PSRR, but show degradation in AC characteristics especially at frequencies >10MHz. Though AC performance can be improved to some extent by increasing bias current [17], it increases voltage drop in the routing switches and may further degrade the circuit performance. Output distortion, on the other hand,

shows significant degradation compared to that target circuits because of the non-linearity in the transmission gate based switches.

Table 4.1. Measured Performance of 65nm PANDA-mapped OTAs

	Target design[1]	**65nm PANDA**
5-transistor OTA		
DC Gain	19.2 dB	19.4 dB
Unity gain frequency	13.6 MHz	8.6 MHz
CMRR	36.2 dB	35.0 dB
PSRR	20.9 dB	20.5 dB
THD	49.1 dB	41.1 dB
Current consumption	280 μA	320 μA
2-stage Miller OTA		
DC Gain	41.9 dB	41.7 dB
Unity gain frequency	4.1 MHz	3.3 MHz
CMRR	50.2 dB	47.1 dB
PSRR	49.7 dB	45.4 dB
THD	38.3 dB	25.2 dB
Current consumption	25.2 μA	31 μA

[1]Simulated with load capacitance of 15pF.

Using these OTAs as building blocks, biasing circuits like bandgap voltage reference and current reference circuits are implemented on PANDA platform. Voltage reference circuit is one of the most widely used circuits in AMS designs such as ADCs, DACs, and power supply regulators. One of the most commonly used voltage reference circuit is the bandgap circuit, which uses a signal (current or voltage) that varies in proportional to absolute temperature (PTAT) in conjunction with another signal that varies complementary to absolute temperature (CTAT) to generate a final output voltage which is independent of temperature. Figure 4.3 shows the low voltage band gap

reference circuit that is commonly used in latest technologies with scaled power supplies. An NMOS input 2-stage opamp is used in the bandgap reference to achieve higher power supply rejection. The opamp is self-biased with the tail NMOS gate and load PMOS gate tied together. The bandgap reference along with its startup circuit is designed for a reference voltage of 950mV with 50ppm temperature variation across 27^{o}C to 80^{o}C. It is designed with capability to digitally trim the resistances LR and NR so as to center the temperature curvature for the required operating temperature in the presence of process variations. Bandgap reference voltage of PANDA implementation measured from room temperature to 80^{o}C compared to that of custom designed bandgap reference circuit after trimming is shown in Figure 4.4.

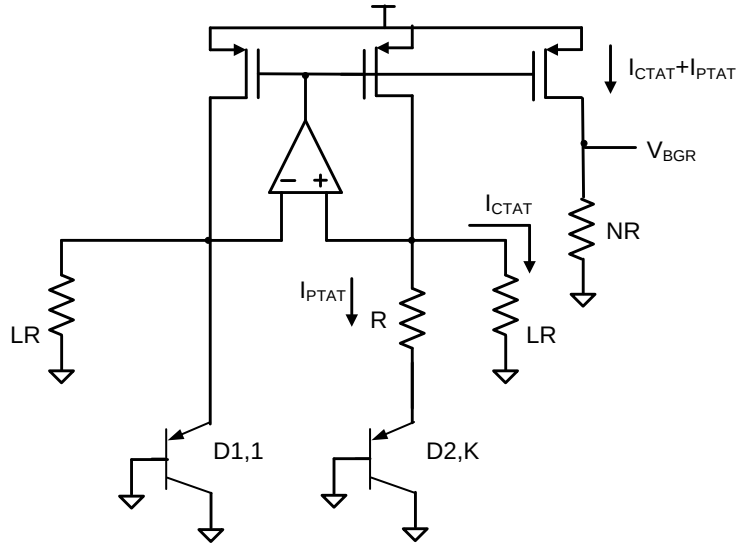


Figure 4.3. Schematics of bandgap voltage reference circuit (start-up circuit not shown).

Current reference circuit as shown in Figure 4.5 is designed and implemented on the platform, where a reference voltage (V_{REF}) is forced on a resistor (R) using negative feedback which defines the current through the resistor as V_{REF}/R. Measured reference

current of PANDA implementation shows a good match to the target design as shown in Figure 4.6.

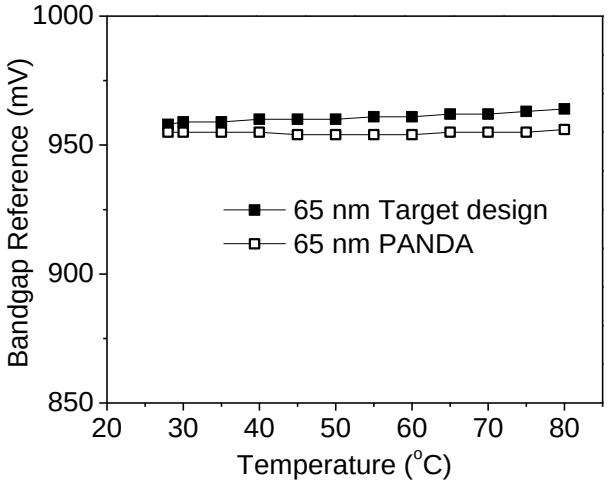


Figure 4.4. Measured bandgap reference voltage of the 65nm PANDA implementation compared to target bandgap reference circuit on the same die.

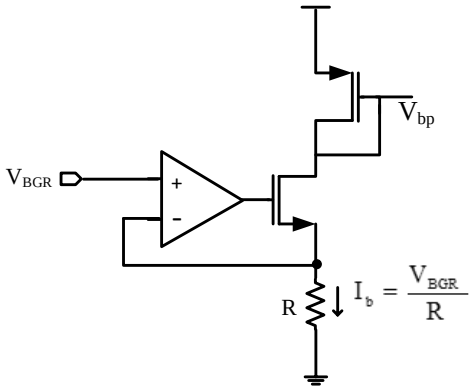


Figure 4.5. Current reference generator using bandgap voltage reference

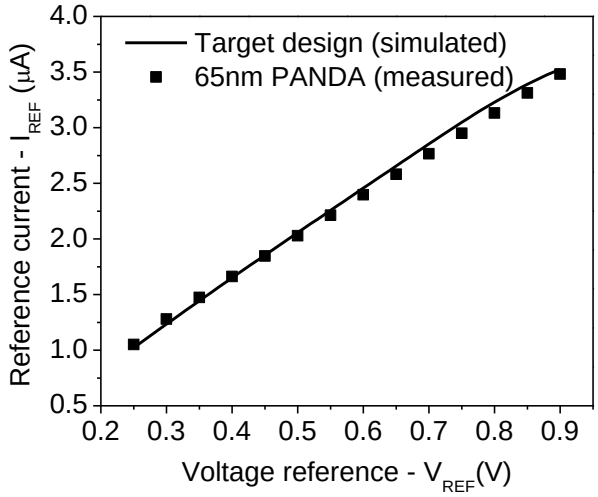


Figure 4.6. Measured current reference from PANDA-implemented circuit.

An amplifier with resistive feedback, a first order G_m-C filter, active RC low pass and high pass filters are also implemented on the platform. The measured performance metrics of these circuits match with the simulation results of the target designs as shown in Table 4.2. Since I_D, R_{out} and G_m of each transistor of the target analog circuit is reproduced by PANDA cells successfully, circuits implemented on PANDA capture the essential DC and AC characteristics of the target circuits.

The number of programmable cells along with the number of reconfiguration switches and the time taken for place and route of some circuits implemented on the platform are summarized in Table 4.3. Applications of the proposed PANDA methodology for rapid prototyping sensor front-end circuits for IoT applications as well as limitations of the methodology are presented in next chapter.

Table 4.2. Measured Performance of 65nm PANDA-mapped Circuits

	Target design[1]	**65 nm PANDA**
Amplifier with feedback		
DC Gain	20.5 dB	19.5 dB
Unity gain frequency	606 KHz	560 KHz
G_m-C low pass filter		
Pass band gain	-1.3 dB	-1.1 dB
3-dB corner frequency	57 KHz	41 KHz
Active RC low pass filter		
Pass band gain	-0.6 dB	-0.8 dB
3-dB corner frequency	204 KHz	195 KHz
Active RC high pass filter		
Pass band gain	-1.1 dB	-1.2 dB
3-dB corner frequency	120 KHz	108 KHz

[1]Simulated with load capacitance of 15pF.

Table 4.3. Design Scale, Routing switches and Time to Place & Route

Circuit	**No. of cells**	**No. of switches**	**Time (s)**
5-transistor OTA	11	51	0.84
2-stage OTA	14	70	1.74
Active RC filter	19	92	3.47
Bandgap reference	26	109	4.31

4.2. Dynamic Reconfiguration Capability

Since configuration bit-stream is sent to the byte-wise addressable memory that stores cell sizing and connectivity information, any memory location can be manually rewritten after configuration, thus enabling dynamic reconfiguration capability of PANDA. This capability to reconfigure only a part of the circuit facilitates multiple applications like calibration/trimming, automatic gain control, offset cancellation etc.

Dynamic reconfiguration capability of PANDA is demonstrated via offset cancellation of a 2-stage OTA by using FPGA to detect the offset error and correct it by sending the bit-stream to resize its input transistors. The block diagram of offset calibration is shown in Figure 4.7 (a) and Figure 4.7 (b) shows the circuit used to measure offset. To cancel the offset of the OTA, a comparator is configured in PANDA and its output is fed to FPGA. The status of the comparator output is read from the FPGA using SPI interface. Based on the output of the comparator, new bit-stream that resize the input transistors to cancel the offset is generated and sent to the platform. For demonstration purposes, transistor sizing update based on comparator output is implemented in software, which could easily be implemented in FPGA to make it a standalone platform. The offset cancellation loop continues till the system converges i.e. when the residual offset is limited by the LSB of the input transistor size. Using this technique the offset of the 2-stage OTA is reduced from 452μV to 29μV. If the application demands for higher accuracy, smaller cells can be used in parallel to original input transistors to have finer control over the sizing.

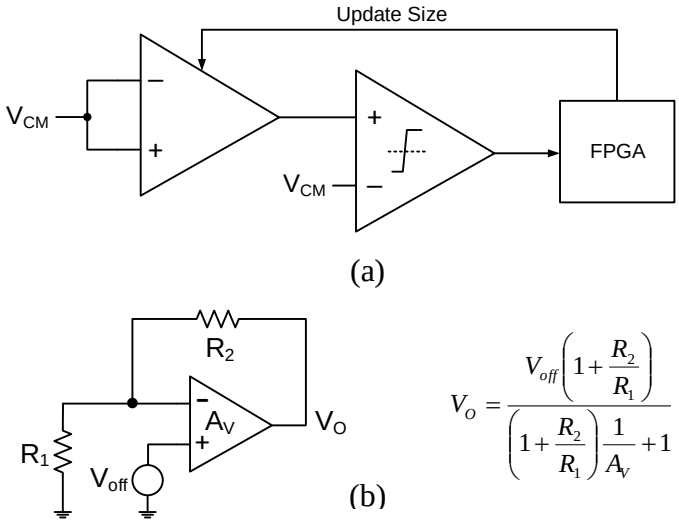


Figure 4.7. (a) Block diagram to demonstrate offset cancellation circuit using a comparator and FPGA in a closed loop. (b) Offset measurement circuit.

4.3. Other Applications

FPGAs are successful as both rapid prototyping tools and application platforms for digital circuits. On the other hand, due to the variety and complexity of analog circuits, FPAAs are very specific to a limited set of applications depending on the type of building blocks and routing architecture used in them. FPAAs are deployed in a wide range of applications like bio-medical applications [55], sensor readouts [56] [57], audio processing [58], emulation of smart power grids [59], etc. Depending on scale of the design, PANDA platform is also capable of implementing the above-mentioned applications. Furthermore, owing to its transistor-level fine granularity, PANDA provides a unique opportunity for analog designers to validate new circuit topologies by rapid prototyping before an expensive tape-out. PANDA also provides opportunity for tuning the circuit components to match the target performance.

There is a growing demand for integrated sensors in mobile platforms, where low power consumption is critical. These sensors need diverse front-end circuits for signal conditioning of the different signal types to be sensed like current, voltage, charge, resistance etc. A generic multi-sensor front-end circuit design to meet the requirements of all the diverse sensors is non-trivial [60] and might lead to over-design and/or high power consumption. Moreover, with the increasing demand for low cost consumer electronics and shrinking time-to-market, PANDA presents a viable solution for multi-sensor front-end circuit implementation with its ability to reconfigure on-the-fly to any sensor front-end circuit.

4.4. Limitations

Measurement results of circuits implemented on PANDA demonstrate a good match in primary performance metrics with those of the targets circuits like gain, CMRR, PSRR, bandwidth, etc., which depend on the transistor's intrinsic properties – I_D, R_{out} and G_m. Second-order performance metrics like noise, distortion, matching etc., which depend more on technology parameters and physical layout than transistor properties, cannot be matched exactly by PANDA-implemented circuits.

PANDA cells are interconnected by transmission gate based switches that have a finite resistance (<400Ω), which places an upper limit on the current per circuit branch from VDD to GND. As the DC current increases, voltage drop across the transmission gate switches increase which may destroy the circuit functionality by changing the bias conditions. Depending on the number of transistors in each branch, required voltage headroom for biasing each transistor and number of switches that connect them after placement and routing, the current per branch is limited to 100 μA. For example, if a

circuit branch consists of 2 transistors connected by a total of 5 switches from the supply (1.2 V) to the ground and if it can tolerate a maximum voltage drop of 0.2 V across the switches, then the maximum current through that branch is 100 μA. This current limitation in turn limits the maximum operating frequency of the PANDA-implemented circuits driving external components to 10's of MHz, unless driven by custom on-chip analog buffers. However, in this work the maximum measured frequency of the implemented OTA shown in Table 4.1 is only 8.6 MHz, because of the passive probes used in measurements, which add a load of 12pF to the circuit.

Implementation of large-scale AMS circuits with 1000's of transistors is not feasible in the current implementation of the platform because of the limited number of PANDA cells. For such large-scale AMS circuit implementation, it is desirable to have a platform with a combination of coarse-grained macros like opamps, ADCs, DACs along with fine-grained transistor-level cells such that it will have the ease of mapping with coarse-grained macros and versatility and flexibility of fine-grained configurable blocks. Although digital circuits can be implemented by mapping each transistor to PANDA cells, it is quite cumbersome and area inefficient. It is preferable to have field programmable mixed-signal array [61] with a combination of configurable logic blocks (CLB) with look-up tables, SRAMs and registers like in FPGAs in combination with PANDA cells and coarse-grained macros so that any digital circuit in feedback loop with the primary analog circuit can also be implemented and validated using the same platform. Nevertheless, PANDA provides a promising solution for rapid prototyping and validation of analog circuits with versatility and flexibility of transistor-level granularity.

CHAPTER 5

RECONFIGURABLE PROCESSING SYSTEM

5.1. Overview of CNN Operations

A typical CNN is comprised of multiple convolutional layers, interspersed by normalization, pooling and non-linear activation function. These convolution layers decompose the input image to different features maps varying from low-level features such as edges, lines, curves, etc., in the initial layers to high-level/abstract features in the deeper layers. These extracted features are classified to output classes by fully-connected classification layers that are similar to multi-layer perceptrons. For example, Figure 5.1 shows the architecture of AlexNet CNN [4], which won the ImageNet challenge in 2012. It consists of 5 convolutional layers each with a Rectified Linear Unit (ReLU) based activation function, interspersed by 2 normalization layers, 3 pooling layers and concluded by 3 fully connected layers which classify the input 224×224 color images to 1,000 output classes. The ImageNet database-based models are characterized by top-1 and top-5 accuracies, which represent that the input image label matches with top-1 and top-5 predictions respectively.

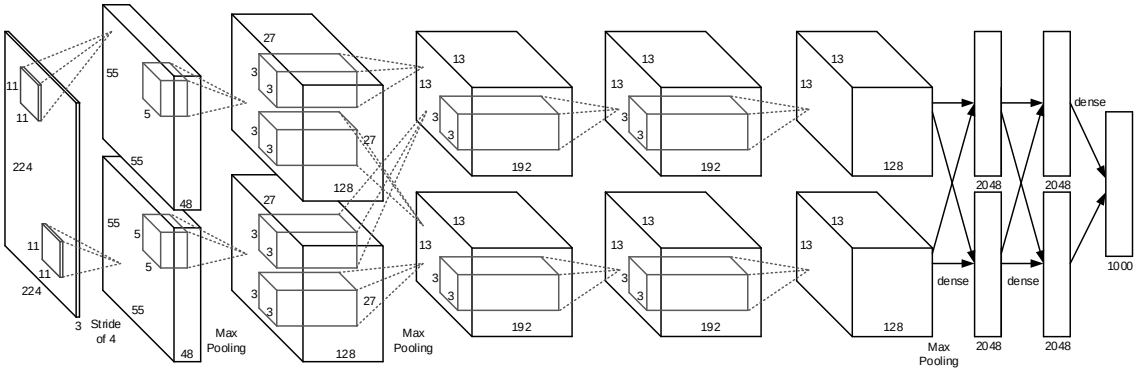


Figure 5.1. Architecture of AlexNet CNN [4].

5.1.1. Convolution

Convolution is the most critical operation of CNNs and it constitutes over 90% of the total operations in AlexNet model [38]. It involves 3-dimensional multiply and accumulate operation of N_{if} input features with $K \times K$ convolution filters to get an output feature neuron value as shown in Equation (1).

$$out(f_o, x, y) = \sum_{f_i=0}^{N_{if}} \sum_{k_x=0}^{K} \sum_{k_y=0}^{K} wt(f_o, f_i, k_x, k_y) \times in(f_i, x+k_x, y+k_y) \quad (1)$$

where $out(f_o,x,y)$ represents the output neuron at (x,y) position in the feature map f_o, $in(f_i,x,y)$ denotes the input neuron in the feature map f_i and $wt(f_o,f_i,k_x,k_y)$ is the kernel weights at position (k_x,k_y) that gets convolved with input feature map f_i to get the output feature map f_o. The kernels and convolution filter weights are learned during the training phase.

5.1.2. Normalization

Local Response Normalization (LRN) or simply normalization implements a form of lateral inhibition [4] by normalizing each neuron value by a factor that depends on the neighboring neurons. Mathematically LRN across neighboring features and within the same feature are computed as shown in Equations (2) and (3), respectively.

$$out(f_o, x, y) = \frac{in(f_o, x, y)}{\left(1 + \frac{\alpha}{K} \sum_{f_i=f_o-K/2}^{f_o+K/2} in^2(f_i, x, y)\right)^{\beta}} \quad (2)$$

$$out(f_o, x, y) = \frac{in(f_o, x, y)}{\left(1 + \frac{\alpha}{K^2} \sum_{k_x=x-K/2}^{x+K/2} \sum_{k_y=y-K/2}^{y+K/2} in^2(f_o, x+k_x, y+k_y)\right)^{\beta}} \quad (3)$$

where K in Equation (2) is the number of neighboring features considered for LRN computation, K in Equation (3) is the number of neurons in x, y directions in the same feature, while α and β are constants.

5.1.3. Pooling

Spatial pooling or subsampling is utilized to reduce the feature dimensions as we traverse deeper into the network. As shown in Equation (4), pooling computes the maximum or average of neighboring $K \times K$ neurons in the same feature map, which also provides a form of translational invariance [62]. Although max-pooling is popularly used, average pooling is also used in some CNN models [62]. Reducing the dimensionality of lower-level features while preserving the important information, the pooling layer helps abstracting higher-level features without redundancy.

$$out(f_o, x, y) = \max_{0 \le (k_x, k_y) < K} / \text{averag e} \left(in(f_o, x + k_x, y + k_y) \right) \tag{4}$$

5.1.3. Fully Connected Layer

Fully-connected layer or inner product layer is the classification layer where all the input features (N_{if}) are connected to all of the output features (N_{of}) through synaptic weights (wt). Each output neuron is the weighted summation of all the input neurons as shown in Equation (5).

$$out(f_o) = \sum_{f_i=0}^{N_{if}} wt(f_o, f_i) \times in(f_i) \tag{5}$$

5.1.4. Activation Function

The commonly used activation functions in traditional neural networks are non-linear functions such as tanh and sigmoid, which require a longer training time in CNNs [4]. Hence, Rectified Linear Unit (ReLU) defined as $y = max(x,0)$ has become the popular

activation function among CNN models as it converges faster in training. Moreover, ReLU has less computational complexity compared to exponent functions in tanh and sigmoid, also aiding hardware design.

The outputs of the inner-product layer traverse through ReLU based activation function to the next inner-product layer or directly to a Softmax function that converts them to probability in the range (0, 1). The final accuracy layer compares the labels of the top probabilities from softmax layer with the actual label and gives the accuracy of the CNN model.

5.2. Hardware Implementation Challenges

While CNNs are proven indispensable in many computer vision applications, they consume significant amount of storage, external memory bandwidth, and computational resources, which makes it difficult to implement on an embedded platform. The challenges in implementation of a large-scale CNN on FPGAs are illustrated using AlexNet model as an example. It has over 60 million model parameters, which needs ~250MB of memory to store the weights using 32-bit floating point representation and hence they cannot be stored in on-chip memory of commercially available FPGAs. They need to be stored in an external memory and transferred to the FPGA accelerator at the time of computation, which could become a performance bottleneck. The AlexNet model consists of 5 convolution layers with ReLU, 2 LRN layers, 3 pooling layers and 3 fully connected layers, where each layer has different number of features, input and output dimensions. If they are implemented independently without resource sharing, it would be either hardware-inefficient or may not fit on the FPGA due to the limited computational and memory resources. The problem gets exacerbated in the state of the art models such

as VGG [5] and GoogLeNet [6], which have a larger number of repeated CNN layers. To efficiently share hardware resources, repeated computation (e.g. convolution) should be implemented with a scalable hardware [38], such that the same hardware is reused by iterating the data through them in software. The performance limitation due to the external memory bandwidth can be alleviated by using reduced precision model weights. Hence, precision study is performed by sweeping model weights and chose the precision values that have minimal impact on the classification accuracy.

5.3. Precision Study for Hardware Accelerator Modules

Traditionally CNN models are trained in CPU/GPU environments using 32-bit floating point data. Such high precision is not necessarily required in the testing or classification phase, owing to the redundancy in the over-parameterized CNN models [63]. Reducing data precision of the weights/data without any impact on the accuracy directly reduces the storage requirement as well as the energy for memory transfers.

Precision requirements of convolution and fully connected layer weights are explored using AlexNet and VGG models from Caffe framework [64]. First, the pre-trained models from Caffe are obtained, convolution weights and inner product weights are rounded off separately, and the models are tested on the ImageNet-2012 validation dataset of 50K images. Although data precision is reduced, Caffe tool still performs CNN operations in 32-bit floating point precision using the truncated weights. Figure 5.2 shows the top-1 and top-5 accuracies of the model for a precision sweep of the weights. It shows that the accuracy steeply drops if the weight precision reduces below 8 bits. Since the same hardware block will be reused for all the convolution layer iterations, a common precision is used for the weights in all convolution layers. 8-bit precision is chosen for the

convolution weights and 10-bit precision is chosen for inner product weights, which degrades the accuracy by only <1% compared to full precision weights. Similarly, by performing the precision study, 16-bit precision is chosen for the intermediate layer data.

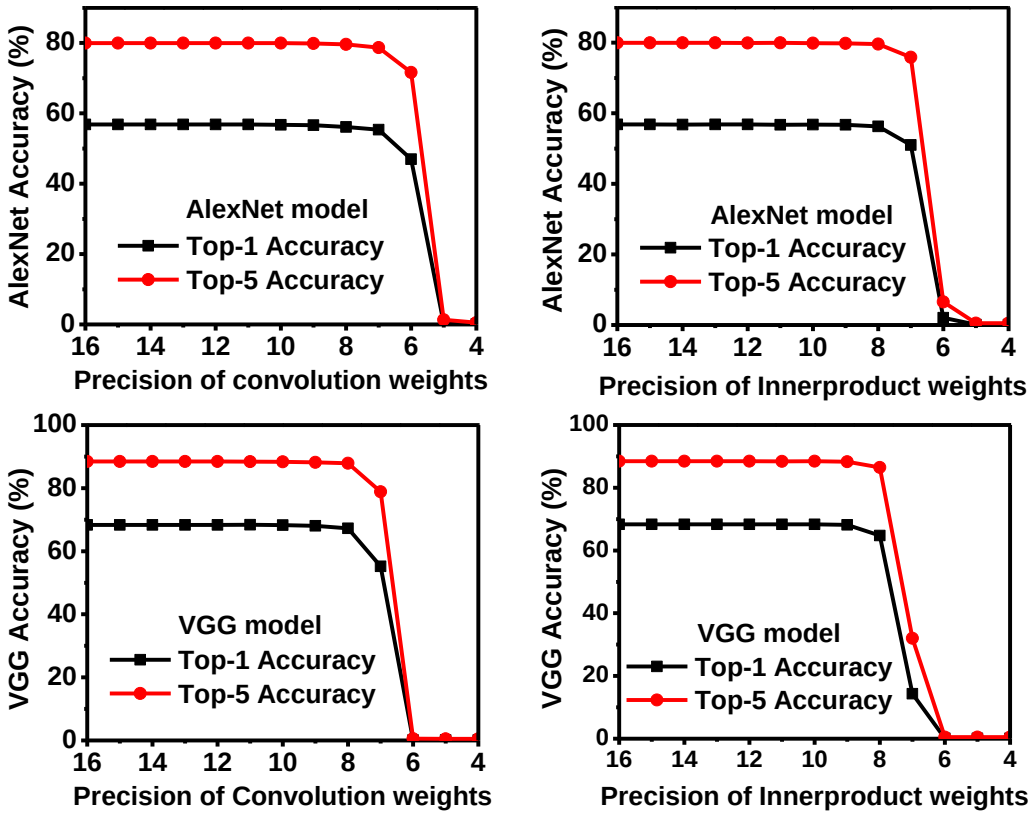


Figure 5.2. AlexNet and VGG model classification accuracies are shown for different weight precisions of convolution and inner-product layers.

5.4. OpenCL Implementation of CNN Layers

High Level Synthesis (HLS) tools are gaining popularity in the FPGA community, as they enable faster hardware development by automatically synthesizing an algorithm in high-level language (e.g. C) to RTL/hardware. There is a recent interest in using OpenCL, a C-based programming language, for FPGAs because of its parallel programming model [65] which matches with the parallel computation capabilities of FPGAs. Moreover, the same OpenCL codes can easily be ported to different platforms:

CPUs, GPUs, DSPs or heterogeneous systems consisting of a combination of them. OpenCL compilers not only compile an OpenCL code to RTL, but also integrate it with the interfacing IPs for external memory and for communication between host CPU and FPGA accelerator board. They abstract the designer/user from the intricacies of traditional FPGA design flow such as RTL coding, integration with interfacing IPs and timing closure, which considerably reduces the design time, while achieving performance comparable to the traditional flow, but possibly at the expense of higher on-chip memory utilization [66].

The design flow of the OpenCL based FPGA accelerator for CNN used in this work is shown in Figure 5.3. It consists of an FPGA accelerator board that is integrated into the PCIe slot of a desktop CPU that acts as the OpenCL host. In general, OpenCL framework consists of two components (a) an OpenCL code that is compiled and synthesized to run on the FPGA accelerator and (b) a C/C++ based host code with vendor-specific application program interface (API) to communicate with the FPGA accelerator.

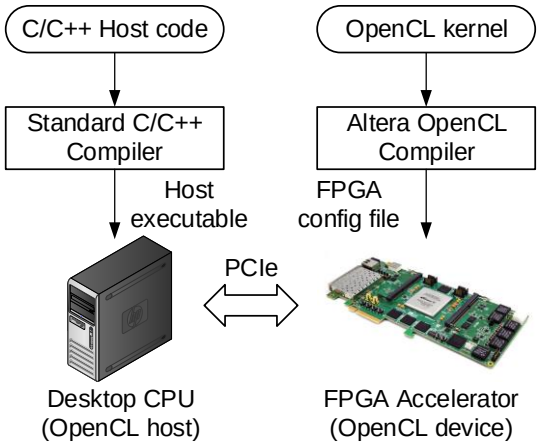


Figure 5.3. OpenCL based FPGA accelerator for CNN.

In this work, Altera OpenCL software development kit (SDK) is used for compilation of OpenCL code to RTL, which takes a few minutes for initial compilation, followed by full synthesis which could take hours depending on the size of the design. The tool-kit provides support for emulation, which runs the OpenCL code on host CPU, thus allowing for quick functional verification before going to the full FPGA implementation. The Altera SDK for OpenCL provides different synthesis constructs to enable acceleration of OpenCL kernels such as loop unroll factor and Single-Instruction-Multiple-Data (SIMD) vectorization factor. The details about how these factors improve the performance of the OpenCL kernels and impact the logic utilization are discussed in the following sections.

5.4.1. 3-D Convolution

Convolutions are the most performance-critical operations in CNNs, constituting up to 91% of the total operations in AlexNet model [4]. It involves computationally intensive 3-D multiply and accumulate (MAC) operations of the input features with the convolution weights as given in Equation (1). To maximize the overall throughput of the accelerator and also make the design portable to any other CNN model, a scalable convolution block is needed such that the data can be iterated through it in software.

A scalable convolution block is implemented by mapping the 3-D convolutions as matrix multiplication operations similar to that in [67] by flattening and rearranging the input features. As an example, Figure 5.4 illustrates how Convolution-1 layer in AlexNet is mapped from 3 input features with dimensions 224×224 to a rearranged matrix with dimensions of (3×11×11) × (55×55). The input features from the first convolution window of 11×11 are flattened and arranged vertically as shown in Figure 5.4. Similarly,

the entire rearranged matrix can be generated by sliding the 11×11 convolution filter across the input features. After input features are rearranged, the convolution operation transforms to a generic matrix multiplication operation. Input feature rearrangement is performed on-the-fly by storing them in the FPGA on-chip memory before performing the matrix multiplication, which reduces the external memory requirement by eliminating the need to store the entire rearranged input feature matrix.

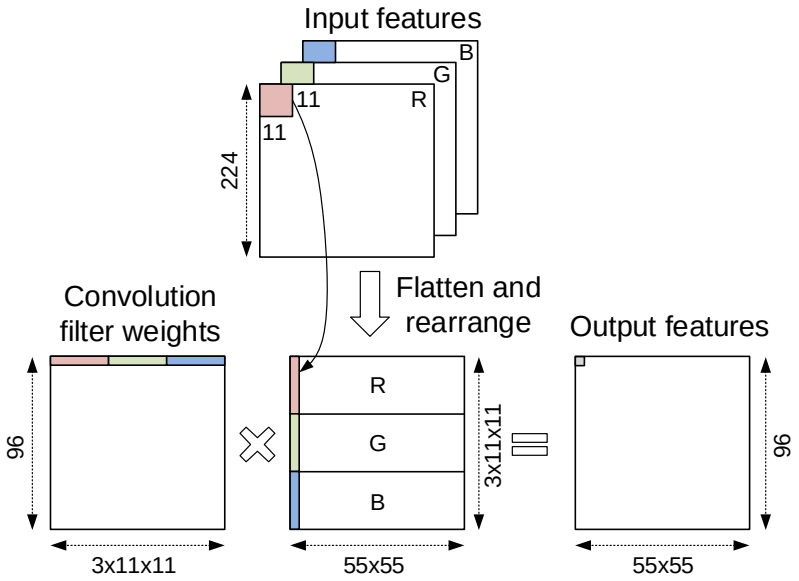


Figure 5.4. Mapping 3D convolutions to matrix multiplication operations.

The pseudo-code for matrix multiplication based convolution implementation in OpenCL is shown in Figure 5.5. It can be summarized as the following three basic operations which are repeated over each row of the weight matrix.

a) Fetch the convolution weights to the local memory which is implemented using FPGA on-chip memory.

b) Compute the input feature actual address locations before flattening and fetch them to local memory.

c) Compute N_{CONV} multiply and accumulate operations in parallel on the weights and inputs from local memory.

---

1. Get current work-item/thread identifiers (x, y).
2. For each N_{CONV} elements width-wise in weight matrix:
3. Compute address locations for input features and weights.
4. Fetch input features to inputs[x][y] in local memory.
5. Fetch convolution weights to weights[y][x] in local memory.
6. Wait till $N_{CONV} \times N_{CONV}$ inputs and weights are loaded.
7. Do the following N_{CONV} MAC operations in parallel:
8. convolution output += weight[x][k]*input[y][k].
9. Wait till all work-items complete computation on fetched data.
10. Save convolution output to output buffer.

Figure 5.5: Pseudo code of convolution implementation based on matrix multiplication.

Input feature rearranging operation is appended with matrix multiplication OpenCL code from [68]. Understanding the matrix multiplication OpenCL implementation is critical for acceleration of the convolution operation. The implementation of matrix multiplication operation in OpenCL is illustrated in Figure 5.6, which consists of convolution weight matrix A (M×N), multiplied by the rearranged input feature matrix B (N×P) to compute the output feature matrix C (M×P). It consists of $N_{CONV} \times N_{CONV}$ threads or OpenCL work-items, which fetch the first $N_{CONV} \times N_{CONV}$ inputs to the local memory where $N_{CONV}=4$ in this example. Each thread performs N_{CONV} parallel multiply and accumulate (MAC) operations on the local memory data, which is accomplished by loop unrolling that replicates the hardware resources for acceleration.

Each thread waits till all the other threads complete the MAC operations, which is achieved by the OpenCL synchronization construct 'barrier'. This process is repeated by sliding the $N_{CONV} \times N_{CONV}$ window column-wise in matrix A and row-wise in matrix B and performing the MAC operations to get $N_{CONV} \times N_{CONV}$ output elements in the product matrix C.

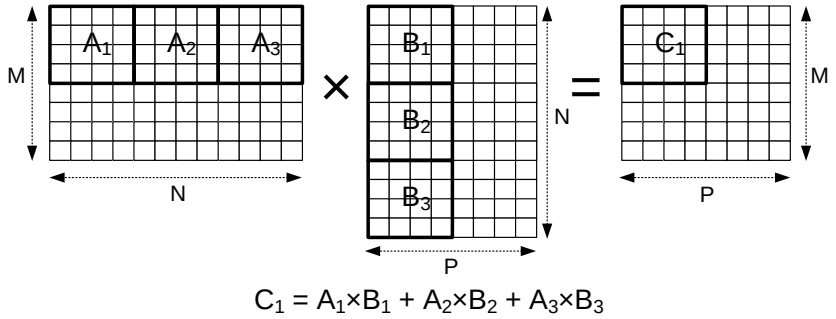


Figure 5.6. Accelerating matrix multiplications in OpenCL.

From Figure 5.6, we see that the input and output matrix dimensions must be a multiple of N_{CONV}, which might not always be possible because of different number of input and output features and different feature dimensions in different convolution layers. Hence zero padding is applied the input matrices to make their dimensions a multiple of N_{CONV}. Increasing N_{CONV} boosts the throughput as it fetches larger number of inputs to the local memory and performs computations on them without having to wait for external data. On the other hand, it increases the logic utilization and overhead in case zero-padding is excessive in some layers.

SIMD vectorization factor (S_{CONV}) is used as the design variable to accelerate the convolution operation, which represents the factor by which computational resources are vectorized to execute in a Single-Instruction-Multiple-Data (SIMD) fashion. This factor improves the throughput by a factor of S_{CONV}. The allowed values for S_{CONV} are 1, 2, 4, 8,

and 16. OpenCL standard imposes a restriction on work-group size (i.e. N_{CONV}) such that it has to be a multiple of S_{CONV}. Depending on the model configuration parameters such as feature dimensions, number of features, and number of convolution layers, choosing an appropriate combination of (N_{CONV}, S_{CONV}) maximizes the overall throughput of CNN.

5.4.2. Normalization

Local response normalization (LRN) implementation requires an exponent operation as shown in Equation (2), which is expensive to precisely implement in hardware. Hence the exponent function $f_1(x_o)$ shown in Equation (6) is implemented using a piece-wise linear approximation function $pwlf(x_o)$.

$$out(f_o, x, y) = in(f_o, x, y).f_1(x_o) \tag{6}$$

$$f_1(x_o) = (1 + x_o)^{-\beta};\ x_o = \frac{\alpha}{K} \sum_{f_i = f_o - K/2}^{f_o + K/2} in^2(f_i, x, y) \tag{7}$$

Here K represents the number of features used for normalization. Using the AlexNet model data as an example, the exponent function $f_1(x_o)$ is approximated using a piece-wise linear function using 20 points with a maximum error of 1%. Because of the wide dynamic range of values involved in x_i computation, normalization is implemented in 32-bit floating point representation. Normalization is implemented as a single-threaded code using loop unroll factor (N_{NORM}), which represents the number of normalization operations it performs in a single cycle. The Altera OpenCL compiler automatically infers pipelining whenever there are no data dependencies between multiple iterations. The pseudo code for normalization is shown in Figure 5.7. It uses local memory to store the sum of squares of a sliding window of K input features, while performing the

normalization operation on the computed sum of squares using the piece-wise linear approximation function, *pwlf*(x_o).

---

1. Compute *sum_of_squares* of first *K*/2 features.
2. For each *input_feature i*:
3. For each neuron *j* in feature *i*:
4. Do the following for N_{NORM} neurons in parallel:
5. Compute *sum_of_squares*[*j*] += *input_feature*[*i*+*K*/2][*j*]
6. Compute *output_feature*[*i*][*j*] = *input_feature*[*i*][*j*]
7. **pwlf*(*α*/*K***sum_of_squares*[*j*])
8. Update *sum_of_squares*[*j*] –= *input_feature*[*i*–*K*/2][*j*]

Figure 5.7. Pseudo-code for normalization implementation.

5.4.3. Other Layers

Pooling is implemented using a single work-item kernel where acceleration is achieved by unrolling the loop to generate N_{POOL} parallel outputs in a single cycle. Fully-connected layer or inner-product layer is also implemented as single work-item kernel, where acceleration is achieved by performing N_{FC} parallel multiply and accumulate operations, which accelerates the performance by a factor of N_{FC}. Nonlinear activation function ReLU, which performs the function $y=max(x,0)$ is incorporated at the output of convolution and inner product implementations with a flag to enable or disable it.

5.5. Design Space Exploration

Choosing the best combination of the design variables (N_{CONV}, S_{CONV}, N_{NORM}, N_{POOL}, N_{FC}) that maximizes the performance of the CNN accelerator, while still being able to fit in the limited FPGA resources is a non-trivial task, which emphasizes the need

for a systematic design space exploration methodology. Optimization framework that relies on full FPGA synthesis at each design point may not be feasible especially because of the long run time, which could take hours, or potential synthesis failures that occur due to utilization of hardware resources. Hence performance and resource utilization are modeled and used for fast design space exploration.

In this section, optimization problem formulation is presented and then analytical and empirical modeling of the performance and FPGA resource utilization as a function of the design variables for each CNN layer are presented using AlexNet as an example.

5.5.1. Problem Formulation

The resource-constrained throughput optimization problem can be formulated as follows.

$$\text{Minimize} \sum_{i=0}^{T_L} runtime_i(N_{CONV}, S_{CONV}, N_{NORM}, N_{POOL}, N_{FC}) \tag{8}$$

$$\text{Subject to} \sum_{j=0}^{L} DSP_j \leq DSP_{MAX} \tag{9}$$

$$\sum_{j=0}^{L} Memory_j \leq Memory_{MAX} \tag{10}$$

$$\sum_{j=0}^{L} Logic_j \leq Logic_{MAX} \tag{11}$$

where T_L represents the total number of CNN layers in the model, L denotes the total number of CNN layer types and $runtime_i$ is the execution time of the layer-i. DSP_{MAX}, $Memory_{MAX}$, and $Logic_{MAX}$ represent the total DSP, on-chip memory and FPGA logic resources, respectively, available in a given FPGA.

5.5.2. Performance Modeling

The execution time of each CNN layer is analytically modeled as a function of the respective design variables and validated on the actual execution time obtained by

performing full synthesis at selective design points and running them on the FPGA accelerator.

The execution time of convolution layer-*i* is modeled as follows.

$$Convolution\ Runtime_i = \frac{No.\ of\ Convolution\ Ops_i}{N_{CONV} \times S_{CONV} \times Frequency} \tag{12}$$

$$\begin{aligned} &No.\ of\ Convolution\ Ops_i \\ &= PAD_{NCONV}(\text{Conv filter dimensions} \times \text{No. of input features}) \\ &+ PAD_{NCONV}(\text{No. of output features}) \\ &+ PAD_{NCONV}(\text{output feature dimensions}) \end{aligned} \tag{13}$$

where PAD_{NCONV} ceils its inputs to the multiple of N_{CONV}. Maximum frequency of the kernel, which is also a function of N_{CONV} and S_{CONV}, is modeled empirically from the synthesis data with different random seeds, as shown in Figure 5.8. The run time model and the actual measured run times of convolution layers 1-4 of AlexNet implementation for a sweep of N_{CONV} at different S_{CONV} values are compared in Figure 5.9.

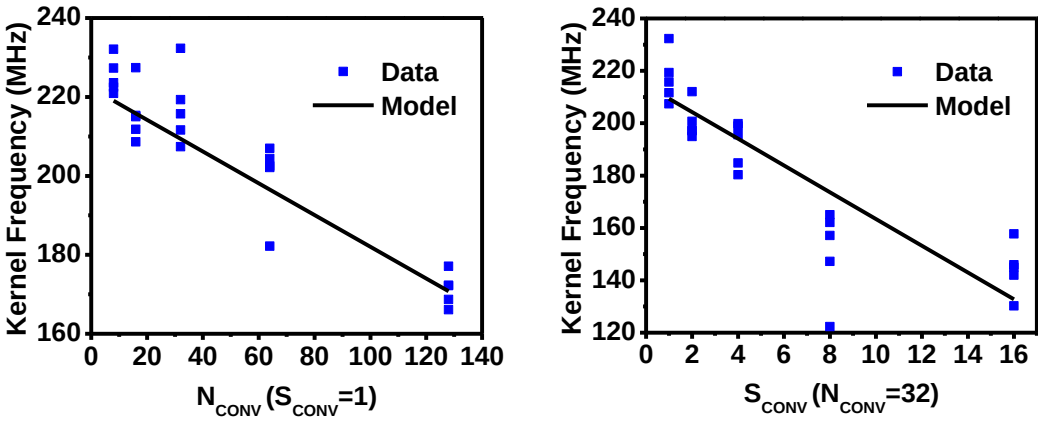


Figure 5.8. Kernel frequency modeling from full synthesis data at 5 random seeds. (RMS error of the fit: 12.57 MHz).

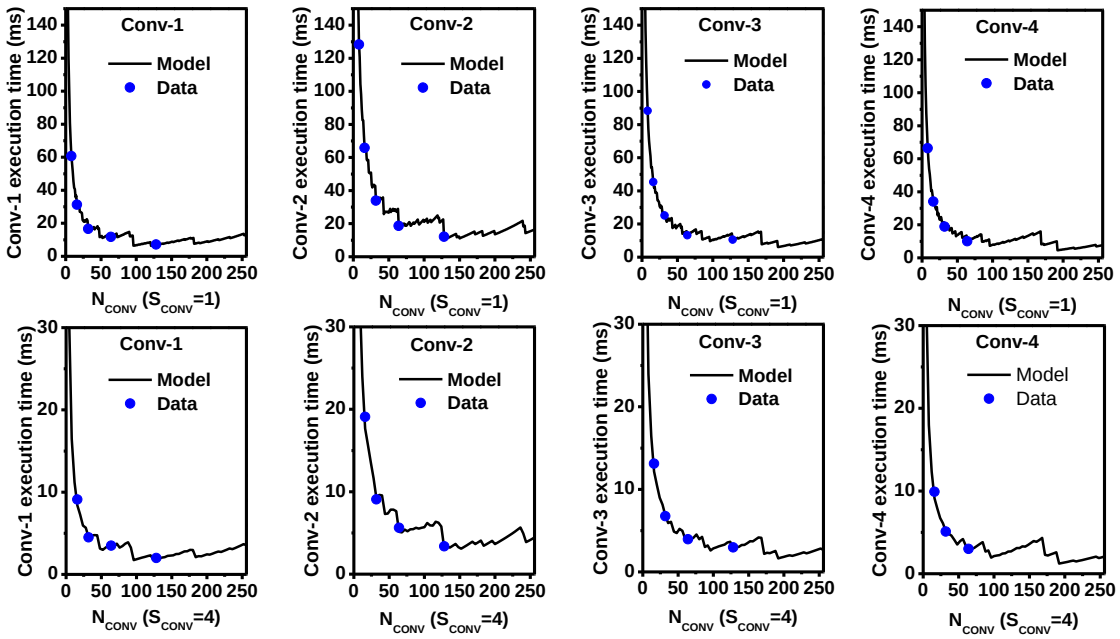


Figure 5.9. Run time model vs. measured time of convolution layers 1-4 of AlexNet CNN model for a sweep of N_{CONV} with S_{CONV} = 1 and 4.

Similarly, the execution time of normalization, pooling and fully connected layers are modeled as functions of their respective loop unroll factors used for acceleration as follows.

$$Runtime_i = \frac{\#Operations_i}{Unroll\ \ factor \times Frequency} \tag{14}$$

The execution time model vs. measured run time of normalization and fully connected classification layers are shown in Figure 5.10.

Input data, weights, intermediate data and output data are stored on the external memory that is present on the FPGA accelerator board. During computation of each layer, the inputs are loaded from the external memory to the FPGA and the computation results are stored back in the external memory. For this purpose, Altera OpenCL compiler generates complex load/store units similar to those in GPUs, which combine multiple

external memory accesses into a single burst access, known as coalescing. This ensures the efficient use of available external memory bandwidth with less contention for memory accesses between multiple computational blocks. On the other hand, this makes it difficult to model the external memory bandwidth usage with respect to the design variables used for acceleration. This problem is aggravated by the reuse of the scalable hardware blocks in multiple iterations of CNN layers with different input dimensions, which will have different access patterns. For example, the execution time of fully connected layers 6 and 7 of AlexNet model shown in Figure 5.10, shows that the model matches well with the measured time till N_{FC}=100. For N_{FC}>100, the measured time increases slightly, but the model still shows a reduction in execution time. This discrepancy is caused by the bandwidth limitation of the FPGA board used for the model validation. Hence, bandwidth limitation of the FPGA board is used to define the upper limits for the design variables in our optimization framework.

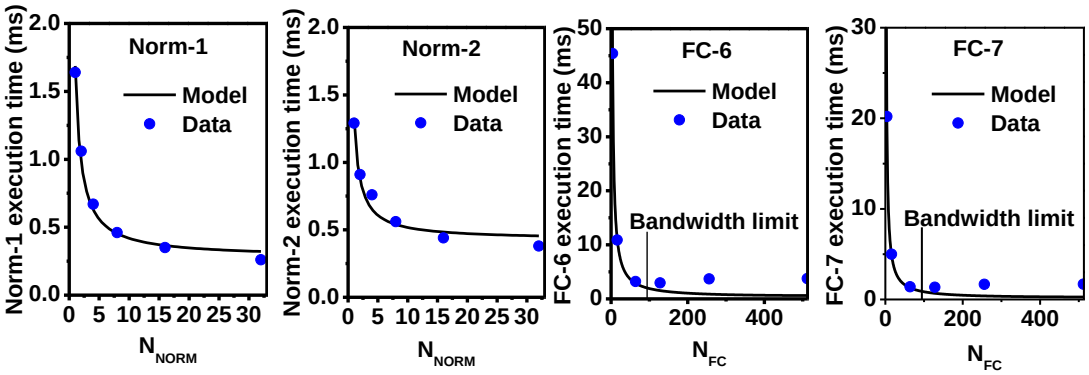


Figure 5.10. The execution time model vs. measured data of normalization and fully connected layers in AlexNet for sweep of loop unroll factors N_{NORM} and N_{FC}.

5.5.3. Resource Utilization Modeling

Analytically modeling the FPGA resource utilization of an algorithm in high-level language such as OpenCL may not be feasible because of the optimizations performed in

the HLS tools. Hence, synthesis results are used to empirically model the FPGA resource utilization. The post-synthesis resource utilization results are not precisely accurate due to the optimizations performed after flattening the design hierarchy during the place and route stage. However, resource utilization data acquired from synthesis is still a good representative of the final resource utilization after place and route. DSP block usage, on-chip memory and logic utilization from synthesis results of each CNN layer are fitted to linear regression models as a function of the respective design variables.

Resource utilization models of normalization block are shown in Figure 5.11. Logic element and DSP utilization from the synthesis data in Figure 5.11 show a linear increase with the swept design variable N_{NORM}. On the other hand, on-chip memory utilization model shows small discrepancy with the synthesis data at intermediate points because of the implementation of coalescing type load/store units, where the memory resource utilization depends on whether the external memory data width is an integer multiple of the design variables i.e. N_{NORM}.

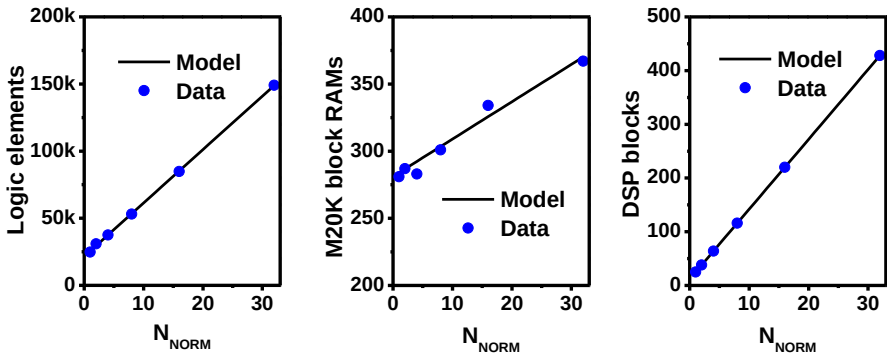


Figure 5.11. Resource utilization empirical models for normalization block.

5.5.4. Optimization Framework

From the convolution run time model in Figure 5.9, we see that it is non-monotonic, because of the differences in dimensions of the CNN layers. Although

exhaustive search of all the design variables could be done using the performance and resource utilization models, it may not be feasible if the number of design variables and/or the FPGA resources increase substantially. This calls for global optimization methodologies such as simulated annealing, genetic algorithm or particle swarm optimization with integer variables and multiple inequality constraints. In this work, genetic algorithm with integer constraints from the global optimization toolbox in Matlab is used for the design space exploration.

Genetic algorithm is a stochastic optimization technique that mimics the biological evolution process and is popularly used to find the global minimum of an objective function subject to a set of constraints. It can also handle mixed integer programming problems, where some design variables are integers. It iteratively improves the quality of the solution by generating a set of candidate solutions at each iteration or generation from a combination of the best solutions from the previous generation based on a set of genetic rules – selection, crossover and mutation. The solutions that violate the constraints (i.e. Equations (9)-(11)) are penalized with a higher objective function value to ensure convergence of the feasible solutions to a global minimum.

The design space of the OpenCL-based FPGA accelerator design is illustrated in Equation (15).

$$\begin{aligned} S_{CONV} &= 1, 2, 4, 8 \; or \; 16 \\ N_{CONV} &= N \times S_{CONV}, 0 < N < N_{MAX} \\ 0 &< N_{NORM} < N_{NORM(MAX)} \\ 0 &< N_{POOL} < N_{POOL(MAX)} \\ 0 &< N_{FC} < N_{FC(MAX)} \end{aligned} \tag{15}$$

where all the design variables are integers, and upper limits of the design space exploration such as N_{MAX}, $N_{NORM(MAX)}$, $N_{POOL(MAX)}$, and $N_{FC(MAX)}$ are determined by the external memory bandwidth of the FPGA board. For example, in a fully connected layer implementation where k bytes are required for each MAC operation, N_{FC} of an accelerator board with external memory bandwidth of M_{BW} is computed as shown in Equation (16).

$$N_{FC(MAX)} = \frac{Memory\ bandwidth\ (M_{BW})}{k \times Frequency} \tag{16}$$

For an FPGA system with 6 GB/s external memory bandwidth, requiring 2 bytes per MAC operation in a fully connected layer with 100MHz kernel frequency, the upper limit for N_{FC} can be computed from Equation (16) as 30. Similarly, the upper limits of other blocks can be computed based on the number of external memory transfers required for each operation.

5.6. Experimental Results

The proposed optimization framework is validated by implementing and accelerating two large-scale CNN models: AlexNet and VGG (16-layer) models on two FPGA boards with different hardware resources. The hardware specifications of the two Altera Stratix-V based boards are summarized in Table 5.1.

Both networks are implemented in OpenCL with fixed-point operations using 8-bit weights for convolution and fully connected layers as obtained from the precision study in Section 5.3. Although 10 bit precision is chosen for inner product weights, they are still represented in 8-bits because of their limited range. Using the performance and resource utilization models and the maximum hardware resources available in the two

boards, optimization framework is run on both AlexNet and VGG models to find the optimal combination of design variables (N_{CONV}, S_{CONV}, N_{NORM}, N_{POOL}, N_{FC}) that maximizes the throughput. The final values of the design variables for both networks optimized for the two FPGA boards are shown in Table 5.2. VGG model does not include normalization layers; hence the corresponding kernel is removed for the FPGA implementation.

Table 5.1. Comparison of FPGA Accelerator Boards.

Specification	P395-D8 [69]	DE5-Net [70]
FPGA	Stratix-V D8	Stratix-V A7
Logic elements	695K	622K
DSP blocks	1,963	256
M20K RAMs	2,567	2,560
External memory	4× 8GB DDR3	2× 2GB DDR3

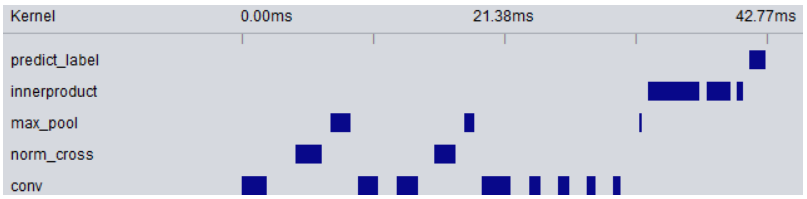
Table 5.2. Optimized Parameters.

Parameter	P395-D8 board		DE5-Net board	
	AlexNet	VGG	AlexNet	VGG
N_{CONV}	64	64	32	64
S_{CONV}	8	8	4	2
N_{NORM}	2	-	2	-
N_{POOL}	1	1	1	1
N_{FC}	71	64	32	30

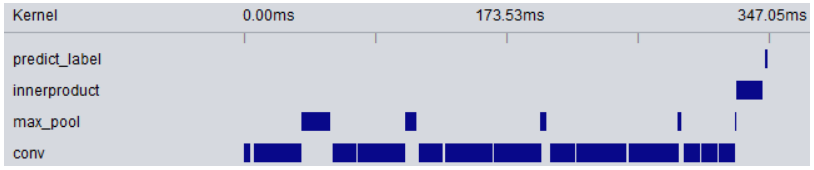
Using Altera OpenCL SDK, the OpenCL kernel codes for AlexNet and VGG models are compiled for the two boards using the corresponding optimized parameters from Table 5.2. Using the host code APIs, FPGA is programmed and the CNN model is run by queueing the OpenCL implemented CNN kernels with appropriate arguments that

consist of input/output buffer address locations and the layer dimensions. The execution time of each kernel and the entire model are measured and throughput is computed as (total number of operations)/(execution time).

The execution time of the CNN layers in AlexNet and VGG models implemented on P395-D8 board with kernel profiling support) is shown in Figure 5.12. The final classification time without kernel profiling will be significantly lower than that shown in Figure 5.12 because of the delay involved with kernel profiling itself. The execution of fully-connected layers can be overlapped with the initial convolution layers of the next image, which increases the overall throughput of the accelerator (by 27% in AlexNet implementation on P395-D8). The next input image transfer from the OpenCL host to the off-chip memory on the FPGA board is overlapped with current CNN operations, thus not hampering the throughput. The initial model weight transfer from the host to the board, which only occurs once in the beginning, is not included for throughput computation.



(a)



(b)

Figure 5.12. The execution time of CNN layers in (a) AlexNet and (b) VGG models on P395-D8 FPGA accelerator.

The total classification time per image and overall throughput of AlexNet and VGG models on P395-D8 and DE5-Net boards are compared with Caffe tool [20] running on Intel core i5-4590 CPU (3.3 GHz) in Table 5.3. Although both FPGAs have similar number of logic elements and on-chip memory blocks, the smaller number of DSP blocks in DE5-Net accounts for its lower throughput compared t that of P395-D8. The software implementation in Caffe tool uses libraries optimized for basic vector and matrix operations (i.e., ATLAS [71]) for performing CNN operations. Our OpenCL based FPGA implementations on P395-D8 achieve 9.5x and 5.5x speedups for AlexNet and VGG models, respectively, compared to the CPU implementation in Caffe tool.

Table 5.3: Classification Time/Image and Overall Throughput.

	FPGA	Classification time/image (ms)	Throughput (GOPS)
AlexNet	P395-D8	20.1	72.4
	DE5-Net	45.7	31.8
	CPU	191.9	7.6
VGG	P395-D8	262.9	117.8
	DE5-Net	651.2	47.5
	CPU	1437.2	21.5

The execution time, throughput and the resource utilization of each kernel type of the AlexNet on P395-D8 and DE5-Net FPGA accelerator boards are shown in Figure 5.13. VGG implementation on P395-D8 achieves a peak throughput of 136.5 GOPS for convolution layers, and 117.8 GOPS including all layers and operations while performing image classification. From the implementation results, we see that throughput of the accelerator is largely proportional to the number of DSP blocks used in the

implementation. AlexNet implementation on P395-D8 board is limited by the number of available M20K block RAMs, while only 727 out of 1963 available DSP blocks are utilized. On the other hand, the implementation on DE5-Net FPGA board is limited by the lower number of available DSP blocks, although the on-chip memory resources are not fully utilized.

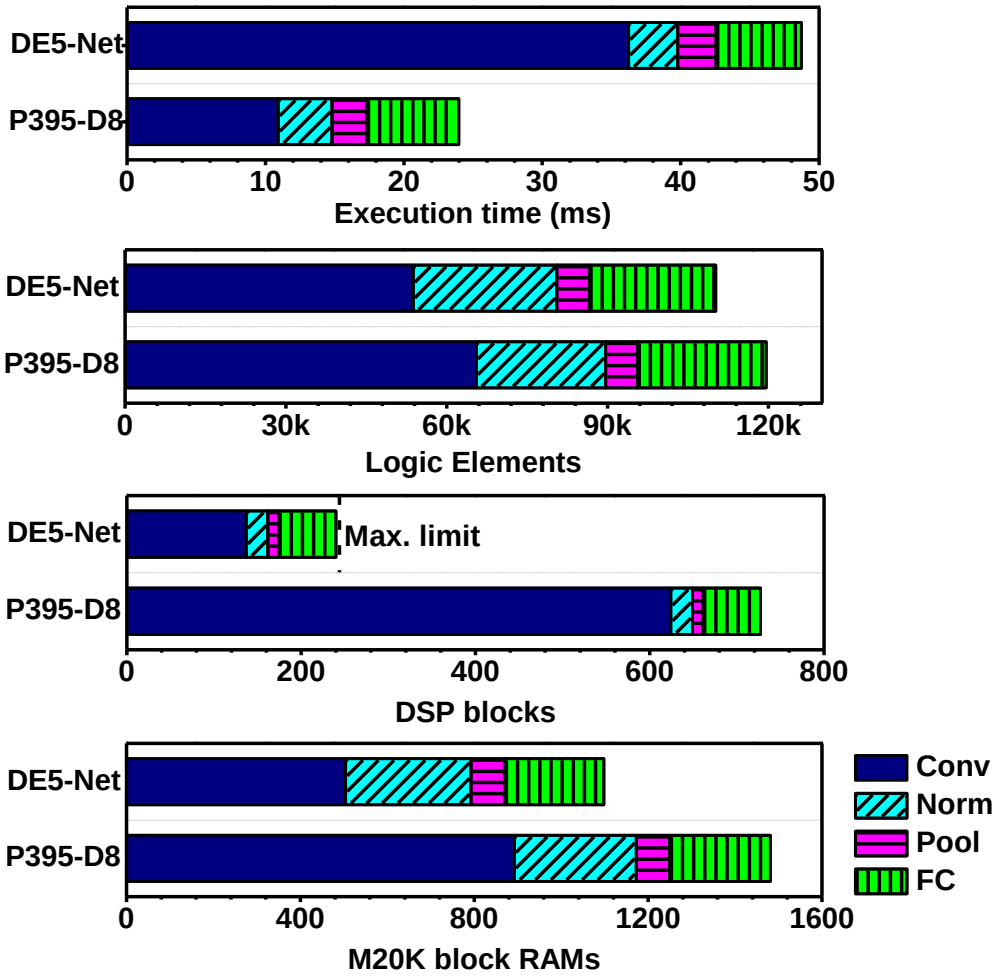


Figure 5.13. Execution time and resource utilization of each CNN layer type for AlexNet implementation on P395-D8 and DE5-Net FPGA boards.

The optimization framework reports the hardware resource that causes the performance bottleneck, such that the user can choose another FPGA hardware, which

has larger number of the specific hardware resources (e.g. DSP blocks). This methodology can also be used to find the ideal specifications of an FPGA suited for CNN, by performing optimization with relaxed constraints for the bottleneck hardware resource. For example, increasing the on-chip memory resources on P395-D8 FPGA by 10% directly increases the throughput of AlexNet implementation by ~10%.

The top-1 and top-5 accuracies of FPGA implementation of AlexNet and VGG models compared to those of the full-precision Caffe models are summarized in Table 5.4. The accuracy degradation due to fixed-point operations in FPGA implementation is <2% for top-1 accuracy and <1% for top-5 accuracy for both AlexNet and VGG models.

Table 5.4. Model Accuracy Comparison.

Accuracy	Full precision in Caffe tool		Fixed-point FPGA implementation	
	Top-1	Top-5	Top-1	Top-5
AlexNet	56.82%	79.95%	55.41%	78.98%
VGG	68.35%	88.44%	66.58%	87.48%

Both DE5-Net and P395-D8 boards are connected to a PCIe slot of a desktop computer whose CPU operates as the OpenCL host. Since the FPGA board receives power from external power port as well as PCIe slot, the power measurement of the FPGA board itself is not straightforward. We attempted to block the power connection through PCIe and have the FPGA board powered only through the external power port. This way, the average power consumption of DE5-Net board was measured as 24.2W after programming AlexNet configuration, and as 25.8W while performing classification. On the other hand, the same measurement method was not feasible on P395-D8 board as

it was designed to use both power supplies. Nonetheless, its power consumption was measured as 19.1W after programming with AlexNet configuration file, using a utility function provided by board manufacturer that measures the steady state power of the board. The power consumption difference between the desktop computer without FPGA and with FPGA running AlexNet is measured as 26W for DE5-Net and 35W for P395-D8 boards. This difference includes the power consumption of CPU running the OpenCL host code, which could be much smaller with embedded processors in FPGA chips.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1. Thesis Conclusions

The application fields for Internet of Things are expanding at a rapid pace and the expected time for an idea to reach the market is shrinking. In order to accelerate the IoT sensor node hardware design process, this work presents an FPAA based solution for rapid-prototyping sensor analog front-end circuit and an FPGA based processing unit. To enable reconfiguration capability for any sensor analog front-end circuit, a Programmable ANalog Device Array (PANDA) is developed with transistor-level fine granular configurable analog blocks. A full system consisting of an array of 24×25 PANDA cells, mesh-style reconfigurable interconnect and configuration memory is developed, which is presented in Chapter 2.

Although FPGA CAD tools are quite mature, they are not well suited for PANDA platform because of the intrinsic differences between analog and digital circuits. Hence, a new CAD tool, PANDA-PRO, is developed to implement analog circuits on the implemented platform, which is presented in chapter 3. Methodologies for transistor-level mapping of target circuits to PANDA cells, automatic placement and routing of the mapped cells are also presented in Chapter 3.

In Chapter 4, measurement results of several benchmark circuits including amplifiers, voltage and current references and active filters are presented. The measurements match the simulation results of the target designs, demonstrating the efficacy of the platform to rapid-prototype and validate IoT sensor front-end circuits. Dynamic reconfiguration capability is demonstrated through offset cancellation of an

amplifier using an FPGA in a closed loop. This capability which enables on-the-fly reconfiguration of transistor sizes and connectivity opens up new realms such as self-calibrating circuits, adaptive circuits, reconfigurable multi-sensor readout circuits, evolvable hardware etc.

In Chapter 5, an FPGA based hardware accelerator for Convolutional Neural Networks (CNN) is proposed for implementing in the processing unit of IoT gateways. A systematic methodology is proposed to maximize the throughput of a given CNN algorithm, subject to the FPGA hardware constraints of logic utilization, computational resources, on-chip memory and external memory bandwidth. The new methodology is demonstrated by implementing and maximizing the throughput of two state-of-the-art CNNs: AlexNet [4] and VGG models [5], on two Altera FPGA platforms with different hardware resources, achieving a peak performance of 136.5 GOPS.

6.2. Future Work

This work could be extended in several research directions, some of which are described as follows. While transistor-level granularity of PANDA cells is ideal for implementing an arbitrary analog function, implementing large-scale circuits would be cumbersome at that granularity. Further investigation could be required to explore the design space to find an optimal combination of transistor-level PANDA cells and coarse-grained macros (e.g., Opamps, OTAs, ADCs, etc.) suitable for a set of target applications.

The developed design space exploration methodology to maximize the performance of the hardware accelerator for CNNs can be extended to any other feed-forward class of neural networks. Furthermore, it could be extended to different classes of deep learning, for instance, recurrent neural networks (RNN), where the connections

between different units form a directed cycle. RNNs, analogous to state machines in digital circuits, are different from CNNs, which are analogous to combinational circuits in digital circuits, in the network structure as they have feedback connections. This creates an internal state of the network that makes them suitable for a wider class of applications including language modeling, speech recognition and data analytics.

REFERENCES

[1] Global IoT Deployments. Available at www.iotwf.com/iotwf2015/deployment-map.

[2] Y. LeCun, B. Boser, J. S. Denker, R. E. Howard, W. Habbard, L. D. Jackel, D. Henderson, "Handwritten digit recognition with a back-propagation network," *Advances in Neural Information Processing Systems*, 396-404, 1990.

[3] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, L. Fei-Fei, "ImageNet large-scale visual recognition challenge," In *Int. J. Computer Vision*, 2015.

[4] A. Krizhevsky, I. Sutskever, G. E. Hinton, "ImageNet classification with deep convolutional neural networks," In *Advances in Neural Information Processing (NIPS)*, 1097-1105, 2012.

[5] K. Simonyan, A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv:1409.1556.

[6] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, "Going deeper with convolutions," In *CVPR*, 1-9, 2015.

[7] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, L. Fei-Fei, "Large-scale video classification with convolutional neural networks," *CVPR*, 1725-1732, 2014.

[8] H. Li, Z. Lin, X. Shen, J. Brandt, G. Hua, "A convolutional neural network cascade for face detection," *CVPR*, 5325-5334, 2015.

[9] P. Barros, S. Magg, C. Weber, S. Wermter, "A multichannel convolutional neural network for hand posture recognition," *Int. Conf. on Artificial Neural Networks (ICANN)*, 403-410, 2014.

[10] O. Abdel-Hamid, A.R. Mohamed, H. Jiang, G. Penn, "Convolutional neural networks for speech recognition," In *IEEE Trans. on Audio, Speech and Language Processing*, 1533-1545, Oct 2014.

[11] R. Collobert, J. Weston, "A unified architecture for natural language processing: deep neural networks with multitask learning," In *Int. Conf. on Machine Learning*, 160-167, 2008.

[12] S. Lai, L. Xu, K. Liu, J. Zhao, "Recurrent convolutional neural networks for text classification," In *AAAI Conf. on Artificial Intelligence*, 2267-2273, 2015.

[13] C.A. Looby, C. Lyden, "Op-amp based CMOS field-programmable analogue array," *IEE Proc. Circuits, Devices and Systems*, vol.147, no.2, pp.93-99, Apr 2000.

[14] B. Pankiewicz, M. Wojcikowski, S. Szczepanski, Y. Sun, "A field programmable analog array for CMOS continuous-time OTA-C filter applications," *IEEE J. Solid-State Circuits*, vol. 37, no. 2, pp. 125–136, Feb. 2002.

[15] T. Hall, C. Twigg, J. Gray, P. Hasler, D. Anderson, "Large-scale field-programmable analog arrays for analog signal processing," *IEEE Trans. Circuits and Syst. I: Reg. Papers*, vol. 52, no. 11, pp. 2298-2307, Nov. 2005.

[16] J. Becker, F. Henrici, S. Trendelenburg, M. Ortmanns, Y. Manoli, "A continuous-time hexagonal field-programmable analog array in 0.13µm CMOS with 186MHz GBW," *IEEE Int. Solid-State Circuits Conf., ISSCC 2008. Dig. Tech. Papers.*, pp.70,596, 3-7 Feb. 2008.

[17] S. Mahmoud, K. Ali, M. Rabea, A. Amgad, A. Adel, A. Nasser, H. Mohamed, Y. Ismail, "Low power FPAA design based on OTA using 90nm CMOS technology," *Int. Conf. on Energy Aware Computing (ICEAC), 2011* , pp.1-4, Nov. 30 - Dec. 2 2011.

[18] Dynamically Reconfigurable dpASP: Anadigm AN231E04 datasheet. Available online: http://www.anadigm.com/_doc/DS231000-U001.pdf.

[19] C.R. Schlottmann, S. Shapero, S. Nease, P. Hasler, "A digitally enhanced dynamically reconfigurable analog platform for low-power signal processing," *IEEE J. Solid-State Circuits*, vol.47, no.9, pp. 2174-2184, Sept. 2012.

[20] W.H. Fu, J. Jiang, X. Qin, T. Yi, Z.L. Hong, "A reconfigurable analog processor based on FPAA with coarse-grained, heterogeneous configurable analog blocks," *Int. Conf. on Field Programmable Logic and Applications*, pp.211-216, 2010.

[21] J. Yasunari, T. Inoue, A. Tsuneda, "A CMOS continuous-time FPAA analog core using automatically-tuned MOS resistors," *47th Midwest Symp. on Circuits and Syst., 2004,* , vol.1, pp.153-156, July 2004.

[22] E.K.F. Lee, P.G. Gulak, "A CMOS field-programmable analog array," *IEEE J. Solid-State Circuits*, vol.26, no.12, pp. 1860–1867, Dec 1991.

[23] E.K.F. Lee, P.G. Gulak, "A transconductor-based field programmable analog array," *IEEE Int. Solid-State Circuits Conf.*, pp. 198–199, Feb. 1995.

[24] J. Langeheine, J. Becker, S. Folling, K. Meier, J. Schemmel, "A CMOS FPTA chip for intrinsic hardware evolution of analog electronic circuits," *Proc of NASA/DoD Workshop on Evolvable Hardware*, pp.172-175, 2001.

[25] A. Stoica, R. Zebulum, D. Keymeulen, R. Tawel, T. Daud, A. Thakoor, "Reconfigurable VLSI architectures for evolvable hardware: from experimental field programmable transistor arrays to evolution-oriented chips," *IEEE Trans. VLSI Systems,* vol.9, no.1, pp.227-232, Feb. 2001.

[26] A. Basu, S. Brink, C. Schlottmann, S. Ramakrishnan, C. Petre, S. Koziol, F. Baskaya, C.M. Twigg, P. Hasler, "A floating-gate based field programmable analog array," *IEEE J. Solid State Circuits,* vol. 45, no. 9, pp. 1781-1794, Sept. 2010.

[27] A. Basu, C. Twigg, S. Brink, P. Hasler, C. Petre, S. Ramakrishnan, S. Koziol, C. Schlottmann, "RASP 2.8: A new generation of floating-gate based field programmable analog array," *Proc. IEEE Custom Integrated Circuits Conference (CICC),* pp. 213-216, Sept. 2008.

[28] C.R. Schlottmann, D. Abramson, P.E. Hasler, "A MITE-based translinear FPAA," *IEEE Trans. VLSI Systems,* vol. 20, no.1, pp.1-9, Jan. 2012.

[29] A. Basu, S. Ramakrishnan, C. Petre, S. Koziol, S. Brink, P.E. Hasler, "Neural dynamics in reconfigurable silicon," *IEEE Trans. Biomedical Circuits and Systems,* vol. 4, no. 5, pp. 311-319, Oct. 2010.

[30] R.B. Wunderlich, F. Adil, P. Hasler, "Floating gate based field programmable mixed-signal array," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol.21, no.8, pp.1496-1505, Aug. 2013.

[31] R. Zheng, J. Suh, C. Xu, N. Hakim, B. Bakkaloglu, Y. Cao, "Programmable analog device array: a platform for transistor-level analog reconfigurability," *Design Automation Conference,* pp.322-327, June 2011.

[32] J. Suh, N. Suda, C. Xu, N. Hakim, Y. Cao, B. Bakkaloglu, "Programmable ANalog Device Array (PANDA): A methodology for transistor-level analog emulation," *IEEE Trans. Circuits and Syst. I: Reg. Papers*, vol.60, no.6, pp.1369-1380, June 2013.

[33] W. Zhao, Y.Cao, "New generation of predictive technology model for sub-45nm early design exploration," *IEEE Trans. on Electron Devices,* vol. 53, no. 11, pp.2816-2823, Nov. 2006 (available at http://ptm.asu.edu).

[34] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, E. Culurciello, "Hardware accelerated convolutional neural networks for synthetic vision systems," In *Int. Symp. Circuits and Systems (ISCAS),* 257-260, 2010.

[35] C. Farabet, B. Martini, B. Corda, P. Akselrdo, E. Culurciello, Y. Lecun, "NeuFlow: A runtime reconfigurable dataflow processor for vision," In *Computer Vision and Pattern Recognition workshops,* 109-116, 2011.

[36] S. Chakradhar, M. Sankaradas, V. Jakkula, S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," In *Int. Symp. on Computer Architecture,* 247-257, 2010.

[37] M. Peemen, A. Setio, B. Mesman, H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," In *Int. Conf. on Computer Design (ICCD),* 13-19, 2013.

[38] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," In *ACM Int. Symp. On Field-Programmable Gate Arrays,* 161-170, 2015.

[39] V. Gokhale, J. Jin, A. Dundar, B. Martini, E. Culurciello, "A 240 G-ops/s mobile coprocessor for deep neural networks," In *IEEE Conf. on Computer Vision and Pattern Recognition Workshops,* 696-701, 2014.

[40] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, O. Temam, "DaDianNao: A machine-learning supercomputer," In *IEEE/ACM Int. Symp. on Microarchitecture*, 602-622, 2014.

[41] V. Betz and J. Rose, "FPGA routing architecture: segmentation and buffering to optimize speed and density", *ACM/SIGDA Int. Symp. on Field Programmable Gate Arrays*, p.p. 59-68, 1999.

[42] H. Zhang, M. Wan, V. George, and J. Rabaey, "Interconnect architecture exploration for low-energy reconfigurable single-chip DSPs," in Proceedings IEEE Computer Society Workshop VLSI '99, pp. 2–8, 1999.

[43] G.G. Lemieux and S.D. Brown, "A detailed router for allocating wire segments in field programmable gate arrays," *Proc. of ACM Physical Design Workshop,* April 1993.

[44] Y.W. Chang, D. Wong, C. Wong, "Universal switch modules for FPGA design," *ACM Trans. on Design Automation of Electronic Syst.*, vol. 1, pp. 80-101, January, 1996.

[45] S.J.E. Wilton, "Architectures and algorithms for field programmable gate arrays with embedded memory", PhD thesis, Univ. of Toronto, 1997.

[46] M.I. Masud, S.J.E. Wilton, "A new switch block for segmented FPGAs", *Proc. of 9th Int. Workshop on Field Programmable Logic*, pp. 274-281, 1999.

[47] J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, W. Fang, and J. Rose, "VPR 5.0: FPGA CAD and architecture exploration tools with single-driver routing, heterogeneity and process scaling," *ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays*, pp. 133–142, 2009.

[48] Xilinx 7 series FPGAs Overview. Available online: http://www.xilinx.com/support/ documentation/data_sheets/ds180_7Series_Overview.pdf.

[49] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, "Optimization by Simulated Annealing," *Science*, pp. 671 – 680, May 13, 1983.

[50] V. Betz, J. Rose, "VPR: A new packing, placement and routing tool for FPGA research", *Proc. of 7th Int. Workshop on Field Programmable Logic*, pp. 213-222, 1997.

[51] M. Huang, F. Romeo, A. Sangiovanni-Vincentelli, "An efficient general cooling schedule for simulated annealing," *ICCAD*, pp. 381 – 384, 1986.

[52] W. Swartz, C. Sechen, "New algorithms for the placement and routing of macro cells," *Int. Conf. on Computer-Aided Design*, pp. 336 – 339, 11-15 Nov. 1990.

[53] C.Y. Lee, "An algorithm for path connections and its applications," *IRE Trans. on Electronic Computers*, vol. EC-10, pp. 346 – 365, 1961.

[54] Microchip MCP2210, USB-to-SPI Protocol Converter with GPIO (Master Mode). Available online: http://ww1.microchip.com/downloads/en/DeviceDoc/22288A.pdf.

[55] U.F. Chan, W.W. Chan, S.H. Pun, M.I. Vai, P.U. Mak, "Flexible implementation of front-end bioelectric signal amplifier using FPAA for telemedicine system," *IEEE Int. Conf. Engineering in Medicine and Biology Society*, pp.3721-3724, Aug. 2007.

[56] A. Baccigalupi, A. Liccardo, "Field programmable analog arrays for conditioning ultrasonic sensors," *IEEE J. Sensors*, vol.7, no.8, pp.1176-1182, Aug. 2007.

[57] D.P. Morales, A. Garcia, A.J. Palma, A. Martinez-Olmos, "Merging FPGA and FPAA Reconfiguration Capabilities for IEEE 1451.4 Compliant Smart Sensor Applications," *3rd Southern Conf. on Programmable Logic,* pp.217-220, 28-26 Feb. 2007.

[58] P. Falkowski, A. Malcher, "Audio signal processing based on dynamically programmable analog arrays," *Int. Conf. on Signals and Electronic Syst.*, pp.29-32, 7-10 Sept. 2010.

[59] A.S. Deese, C.O. Nwankpa, "Design and testing of custom FPAA hardware with improved scalability for emulation of smart grids," *IEEE Trans. Smart Grid*, vol.5, no.3, pp.1369-1378, May 2014.

[60] J. Zhang, J. Zhou, P. Balasundaram, A. Mason, "A highly programmable sensor network interface with multiple sensor readout circuits," *Proc. of IEEE Sensors*, Vol.2, pp.748-752, 2003.

[61] R.B. Wunderlich, F. Adil, P. Hasler, "Floating gate based field programmable mixed-signal array," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol.21, no.8, pp.1496-1505, Aug. 2013.

[62] Y.L. Boureau, J. Ponce, Y. Lecun, "A Theoretical Analysis of Feature Pooling in Visual Recognition," In *Int. Conf. on Machine Learning*, 2010.

[63] M. Denil, B. Shakibi, L. Dinh and N. D. Freitas, "Predicting parameters in deep learning," In *Advances in Neural Information Processing Systems*, 2148–2156, 2013.

[64] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, "Caffe: Convolutional architecture for fast feature embedding." arXiv:1408.5093.

[65] Khronos OpenCL Working Group. The OpenCL Specification, version 1.1.44, 2011.

[66] M. S. Abdelfattah, A. Hagiescu, D. Singh, "Gzip on a chip: high performance lossless data compression on FPGAs using OpenCL," In *Int. Workshop on OpenCL 2014.*

[67] K. Chellapilla, S. Puri, P. Simard, "High performance convolutional neural networks for document processing," In *Int. Workshop on Frontiers in Handwriting Recognition,* 2006.

[68] Altera OpenCL design examples. Available online at https://www.altera.com/support/support-resources/design-examples/design-software/opencl.html

[69] Nallatech P395-D8 OpenCL FPGA accelerator cards. Available online at http://www.nallatech.com/wp-content/uploads/openclcardspb_v1_51.pdf

[70] DE5-Net FPGA kit user manual. Available online at ftp://ftp.altera.com/up/pub/Altera_Material/Boards/DE5/DE5_User_Manual.pdf

[71] R.C. Whaley and J.J. Dongarra, "Automatically tuned linear algebra software," In *Proc. SuperComputing 1998: High Performance Networking and Computing*, 2001.