POWER-EFFICIENT MACHINE LEARNING-BASED HARDWARE ARCHITECTURES FOR BIOMEDICAL APPLICATIONS

A Dissertation

presented to

the Faculty of the Graduate School

at the University of Missouri-Columbia

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

---

by

OMIYA HASSAN

Dr. Syed Kamrul Islam, Dissertation Supervisor

MAY 2023

The undersigned, appointed by the dean of the Graduate School, have examined the thesis entitled

POWER-EFFICIENT MACHINE LEARNING-BASED
HARDWARE ARCHITECTURES FOR BIOMEDICAL APPLICATIONS

presented by Omiya Hassan,

a candidate for the degree of Doctor of Philosophy,

and hereby certify that, in their opinion, it is worthy of acceptance.

Professor Syed Kamrul Islam

Professor Jianlin (Jack) Cheng

Professor Dominic K. C. Ho

Dr. Abu Saleh Mohammad Mosa

Dedicated to my sister, Orchi, for always guiding and supporting me;
to my parents, Neely and Rabi, for their unbounded love and care;
and to my grandmother, Nanu whose prayers were heard.

ACKNOWLEDGMENTS

This work would not have been possible without the constant support, and guidance from my Supervisor, Prof. Syed Kamrul Islam. As a teacher and mentor, he has shown me, by his example, what a good researcher, supervisor, and academician should be. I would also like to thank my dissertation committee members, Prof. Jianlin Cheng, Prof. Dominic K. C. Ho, and Dr. Abu Saleh Mohammad Mosa, who have encouraged and supported me in every step, both research and career-wise.The tremendous help and endless support from my labmates and colleagues from the Analog/Mixed-Signal, VLSI, Devices Laboratory: Mow, Samira, Shuvo, Nazmul, Twisha, and Maruf have made this dissertation a reality. I am truly blessed to have an amazing supervisor and a supportive group.

The two-year graduate fellowship provided by IEEE Instrumentation and Measurement Society (IMS) and the tape-out fund provided by IEEE Solid-State Circuits Society (SSCS) have played a significant role in successfully finishing my Ph.D. dissertation with qualitative work. The Electrical Engineering and Computer Science Department (EECS) of the University of Missouri for allowing me to teach as an instructor, helped me prepare myself for academia.

I am fortunate to have an amazing group of people I call family at the University of Missouri. Especially Afia, Tanmoy, Mow, Tonmoy, Samira and Sajid have been the greatest support throughout my Ph.D. journey, and I can always count on them. A special heartfelt gratitude to Somudro, Rafi, Tamim, and Galib, who has supported and cheered me thousands of miles away during my crucial times. Lastly, Lamiya, Rommo, and my undergraduate university friends were always there to support me, especially during the pandemic.

Nobody has been more important to me in the pursuit of my Ph.D. than the members of my family. I want to thank my sister Orchi, whom I was inspired to pursue my Ph.D., and Eresh, who ensured I enjoyed my graduate life; my Uncle and Aunty: Smritee, and Shaheen, whom I am privileged to call my USA-parents, have always been supporting and cheering me on. And last but not least, the two most important people in my life, my parents, Neely and Rabi, for always believing in me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The future of critical health diagnosis will involve intelligent and smart devices that are low-cost, wearable, and lightweight, requiring low-power, energy-efficient hardware platforms. Various machine learning models, such as deep learning architectures, have been employed to design intelligent health-care systems. However, deploying these sophisticated and intelligent devices in real-time embedded systems with limited hardware resources and power budget is complex due to the requirement of high computational power in achieving a high accuracy rate. As a result, this creates a significant gap between the advancement of computing technology and the associated device technologies for healthcare applications. Power-efficient machine learning-based digital hardware design techniques have been introduced in this work for the realization of a compact design solution while maintaining optimal prediction accuracy. Two hardware design approaches, DeepSAC and SABiNN have been proposed and analyzed in this work. DeepSAC is a shift-accumulator-based technique, whereas SABiNN is a 2's complement-based binarized digital hardware technique. Neural network models, such as feedforward, convolutional neural nets, residual networks, and other popular machine learning and deep neural networks, are selected to benchmark the proposed model architecture. Various deep compression learning techniques, such as pruning, n-bit (n = 8,16) integer quantization, and binarization on hyper-parameters, are also employed. These models significantly reduced the power consumption rate by 5x, size by 13x, and improved the model latency. For efficient use of these models, especially in biomedical applications, a sleep apnea (SA) detection device for adults is developed to detect SA events in real-time. The input to the system consists of two physiological sensor data, such as ECG signal from the chest movement and SpO_2 measurement from the pulse oximeter to predict the occurrence of SA episodes. In the training phase, actual patient data is used, and the network model is converted into the proposed hardware models to achieve medically backed accuracy. After achieving acceptable results of 88% accuracy, all the parameters are extracted for inference on edge. In the inference phase, reconfigurable hardware validated the extracted parameter for model precision and power consumption rate before being translated onto the silicon. This research implements the final model in CMOS platforms using 130 nm and 180 nm commercial CMOS processes.

Chapter 1

INTRODUCTION

The evolution of Artificial Intelligence (AI) has opened up a whole new approach to biomedical sensing [1]. AI technology has been extensively incorporated as a powerful tool in image analysis, speech recognition, translation, signal processing, and other tasks and is rapidly proliferating in our everyday lives. These AI systems incorporate machine learning, artificial neural networks (ANN), and deep-learning neural networks (DNN) for building sophisticated systems that closely resemble the performance of human-like behavior. Neural networks, particularly modern DNNs, are becoming quite popular for their ability to predict and make decisions by being trained using different types of data sets. Due to their human-like detection and prediction capabilities, these models have become suitable tools for biomedical research and applications. The learning technique of the DNN model through analyzing complex features makes it possible to obtain highly accurate information from biophysical signals such as electrocardiograms (ECGs), electroencephalograms (EEGs), glucose levels etc. Generally, such neural network models can automatically extract essential features from heavily dense data sets with a high degree of accuracy [2]. For patients with no visible symptoms at the onset of the disease, real-time monitoring is necessary for the timely intervention of possible health complexities. The system must have the capabilities of low power consumption rate, high accuracy, and a large amount of data delivery [3]. Most existing works have focused on dealing with heavy-dense data by extracting the results on the cloud and using different deep learning techniques. In contrast, fewer research works have developed energy-efficient and real-time detection methodologies. These approaches often consume a significant amount of energy due to the handling of large data sets and an increasing amount of memory power [4]. In 2019, a study conducted by a group of researchers from Stanford University found that training an off-the-shelf Natural Language Processing (NLP) model can emit nearly 14,000 pounds of carbon emissions. This amount of CO_2 emission is approaching the amount of carbon emitted by a single passenger flying a round trip from New York to San

Francisco [5].

Running highly computational AI models requires exhaustive hardware resources and memory devices. Research conducted by the MLPerf group [6] states that, as Moore's Law [7] is stagnant due to the limited number of transistor densities on a single CPU, AI is outperforming Moore's Law by 11x times as shown in Fig 1.1. Thus, there are fewer hardware memory resources to train AI models than in cloud computing. This creates a significant gap between the advancement of high-powered AI-model and existing computational hardware resources.

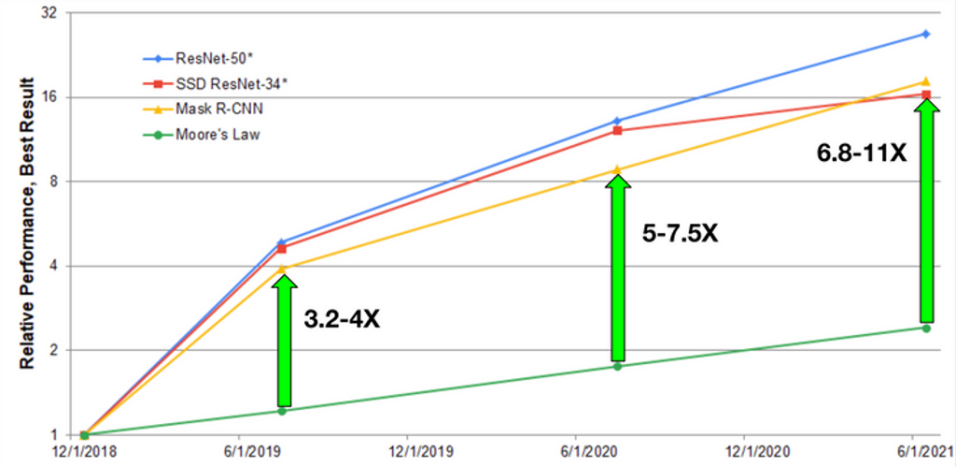


Figure 1.1: Statistical analysis of AI models such as ResNet, and CNN outperforming Moore's Law through the years [6].

On the contrary, processing the data on the cloud does not guarantee any desired results in real-time. As a result, it needs to provide a faster-efficient solution for critical diagnosis. Therefore, researchers focus on utilizing and improving these AI models into a more compact design structure while maintaining its highly accurate learning rate. Various deep learning (DL)-tailored processing units such as graphics processing units (GPUs), tensor processing units (TPUs), application-specific integrated circuits (ASIC), and field-programmable gate arrays (FPGA) [8–12] are being used and designed to reduce computational resources and minimize data storage. Efficient hardware architectures and software-hardware co-simulation processes are employed to reduce the complexities of DL models while simultaneously meeting the system requirements [13]. Software-hardware co-simulation process means training the models in software and then inferring those models on hardware platform. Many of these co-simulation approaches have been developed to reduce the computing complexity of DLs, such as in multi-core platforms [3], bit-width reduction [10], and low-power accelerator design. Conversely, algorithm developers focus on developing various techniques contributing to energy efficiency and model compressing, such as network compression, quantization,

and pruning [11].

This work leverages the compression technique and introduces power-efficient neural network (NN) model-based digital hardware design schemes called DeepSAC: Shift Accumulate Deep Neural Network and SABiNN: Shift Accumulate Binarized Neural Network. The proposed techniques enable energy-efficient AI accelerator design for various edge computing applications for real-time classification, detection, and prediction. These two proposed hardware architectures are used in the development of biomedical devices capable of monitoring real-time health conditions. Design of a dual sensor-based sleep apnea detection system is proposed by following a software-hardware co-simulation process and a four-step design scheme [14–17]. This design scheme aims at improving the processing limitations between the NN models and the edge devices as described below:

1. Instead of exclusively training the models on-chip or in a computer, users can create a co-simulation platform by training the model on the cloud or in the computer and then infer the application-specific trained model on edge devices. This makes it possible for real-time monitoring.

2. n-bit (n = 8,16) integer quantization, pruning, and binarization approaches are applied for model compression and memory utilization.

3. Low power techniques such as the shift accumulate (SAC) method and binarized compression technique are adopted and improved to meet the energy efficiency and low memory requirements.

1.1 Motivation

In recent years, healthcare is becoming increasingly important in our daily lives as mortality rates are increasing due to various complicated diseases due to the lack of timely interventions. Thus, proper diagnosis and monitoring of health are getting more indispensable. Apnea is one of the leading causes of death not only in the USA but also in numerous developing countries in the world. According to the American Association of Sleep Medicine (AASM), an apnea event occurs by the absence of breathing that persists for at least 10-15 seconds, accompanied by 3% or more oxygen de-saturation levels from the pre-event baseline. On the other hand, hypopnea occurs by decreasing the nasal airflow signal by 50% or greater and lasts for at least 10 seconds, associated with a 3–4% drop in blood oxygen saturation level. The apnea-hypopnea index (AHI) represents the severity of sleep apnea, which is the total number of apnea and hypopnea events per sleep hour [18]. According to the survey conducted by AASM, sleep apnea is in the top 10% of high-risk factor diseases leading to

sudden death. An 18-year follow-up study by the University of Wisconsin-Madison in collaboration with the National Institute of Health (NIH) states that people with sleep apneic disorder have 3x times the higher chance of dying due to sudden stroke, high pressure, and heart attack than an average patient illustrated in Fig 1.2 [19].

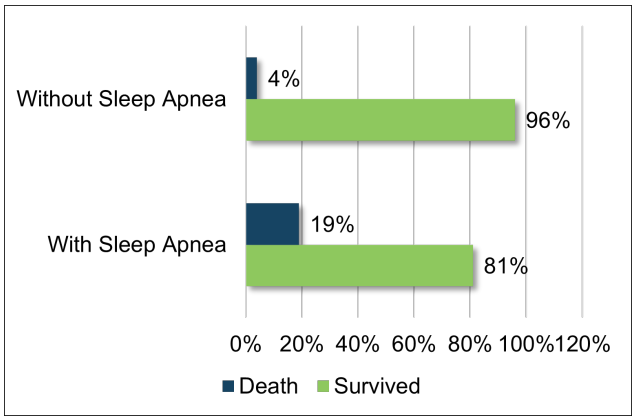


Figure 1.2: Study of sleep apnea (SA) mortality rate by the University of Wisconsin-Madison.

However, although the apneic disorder is prone to leave patients at high risk of sudden deaths, the USA alone has nearly 80% undiagnosed adults aged over 30 years. According to the survey, [20], in every 100 people, nearly nine suffer from sleep disorders. The reason for such a low diagnosis rate is that the present diagnosis methods are expensive, inadequate, and limited in number. The apnea diagnosis requires the patients to undergo extensive overnight sleep studies, such as polysomnography (PSG) or pneumocardiogram, for over 12 to 24 hours, which is expensive and time-consuming. Apneic events are diagnosed by analyzing the PSG recording data based on the nasal-oral airflow amplitude and the blood oxygen saturation (SpO_2) level. During this procedure, the body and the face of the subjects get attached to numerous sensors, causing them a great deal of inconvenience. The manual scoring of apneic events from PSG data is time-consuming and demands specially trained sleep experts. Thus, designing and developing a wearable detection device that alleviates these problems by utilizing AI/Machine-Learning (ML) algorithms is becoming necessary. AI/Machine-Learning (ML) algorithms can automatically detect apneic events from the data collected by the sensors providing automation capabilities. Several machine learning algorithms and neural network schemes are being developed in diagnosing and screening apneic events. They can accurately predict and process data with high sensitivity rates as reported in literature [21–25]. Besides, research in medical imaging and data diagnosis with AI/ML models has resulted in a breakthrough in medical science. In addition, low-power and high-speed data acquisition instruments with wireless and wearable features have gained attention in healthcare applications. Designing miniaturized AI-based integrated circuits

for critical and sensitive medical data processing can become a promising method for developing efficient medical hardware systems. Table 1.1 showcases the research trends and years of developing SA detection devices. However, most of these systems utilized AI/ML models, but none offered both wearable and real-time detection capabilities.

Table 1.1: Research Review on Real-time Detection and Wearability

Work	**Diagnosis Method**	**Classifier**	**Sensor**	**Accuracy (%)**	**Real-Time Detection**	**Wearable**
Selvaraj et. al. [26]	SA Prediction	SVM	ECG, Respiratory	82-85	X	X
Kopaczka et. al. [27]	SA Detection	Variance and Spectral Analysis, Wavelet transform	Thermal Infrared Thermography	N/A	Yes	X
Wang et. al. [28]	SA Prediction	TW-MLP	ECG and SpO_2	87	X	X
Yong et. al. [29]	SA Screening	UWM Radar	PSG	97-98	X	Yes
Sina et. al. [30]	SA Diagnosis	3D-CNN	Sleep Position	73	X	X

This dissertation incorporates AI-enabled power-efficient architectures in CMOS integrated circuits (IC) to realize smart diagnostic decisions enabling both real-time detection and wearable capabilities. Such a unique feature is unavailable in other devices currently used for sleep apnea detection. Overall, the successful implementation of the system using transfer learning-based software-hardware co-simulation design schemes has far-reaching consequences in healthcare applications, leading to the development of innovative wearable sensors.

1.2 Contribution and Dissertation Outline

The dissertation proposes two design architectures for a power-efficient decision-making block in an automatic sleep apnea detection system. Two neural network designs on edge called DeepSAC and SABiNN were designed using the software-hardware co-simulation process and deep compression techniques. These designs replace the MAC operation's multipliers with digital logic components such as shifters and 2's complements. The contribution of this dissertation are:

1. Development of a software-hardware co-simulation technique that utilizes both cloud and edge

devices for accurate transfer learning of neural net models on hardware architectures.

2. A hardware model compression technique called DeepSAC uses shifters as weights, replaces multipliers, and removes on-chip memory for weight storage.

3. A binarized hardware network SABiNN removes the MAC component and introduces 2's complement whenever a negative weight value is associated with the neuron.

4. Redesigning the XNOR-based binarized neural network model into NAND-based XNOR gated digital neural network model for higher precision rate.

5. Design of DeepSAC on reconfigurable hardware and SABiNN on reconfigurable hardware and CMOS platform.

6. Benchmark of SABiNN hardware design architecture on popularly used deep learning models such as VGG19, ResNet50, and MobileNetV2 using cifar10 image dataset and MedMNIST biomedical image dataset.

Based on the contributions, the dissertation outline is organized as follows:

Chapter 2 provides the background for neural network architectures used for biomedical applications and their importance by doing an intensive literature review. The proposed sleep apnea (SA) detection system is introduced after thorough market and research analysis.

Chapter 3 describes the design methodology of the newly proposed software-hardware co-simulation process. This process is used in designing the decision-making block of the SA detection system, where each design step is explained briefly. This content is primarily based on the two works by Hassan et al. [14, 15].

Chapter 4 provides existing and modified model optimization and compression techniques such as pruning, binarization, and integer quantization for hardware that are used in optimizing the proposed models for biomedical signal processing and applications. Experiments are performed with open-source data sets that extensively use neural network models for classification, detection, and prediction.

Chapter 5 introduces the shifter-based DeepSAC method, which is experimented on biomedical applications such as sleep apnea detection and diabetes prediction. The experimental analysis illustrates the low power consumption rate by embedding the model onto the re-programmable hardware. The content of this chapter is based primarily on the work by Hassan et al. [16] and from Hossain et al. [17].

Chapter 6 introduces the NAND-based XNOR gated 2's complement-based SABiNN method, and the module is embedded onto both re-programmable hardware and the CMOS platform using 130 nm and 180 nm PDK processes. The algorithm, test-bench simulations, physical design space exploration, and measurement results are included in this chapter. The content and results of this chapter are based on the two works by Hassan et al. [31, 32]

Chapter 7 showcases the benchmarked results of SABiNN on popular neural networks such as VGG19, ResNet50, and mobilenetV2. The models are trained and tested using image datasets such as cifar10 and biomedical image datasets such as MedMNIST.

The dissertation is summarized in **Chapter 8**, and the future of power-efficient machine learning-based hardware architecture for biomedical and other applications are discussed.

Chapter 2

BACKGROUND

This chapter introduces the usage and research incorporating neural network models in biomedical applications such as detecting sleep apneic events through an extensive literature study. Based on the studies, a novel detection system tackles the current limitations of existing devices by utilizing neural networks such as feedforward neural networks (FNN).

Due to the advent of complex feature extraction capabilities, automated classification, and decision-making functionalities of neural networks, they have been extensively used in the medical domain and have revolutionized some medical applications. FNN is a widely used model in feature extraction, classification, and prediction-based applications [28, 33–38]. The FNN model automatically learns data by accumulating the learned parameter from the previous layer to the next in a feed-forward manner. The vectorized implementation of the forward propagation for a single layer of FNN is presented in [39] as 2.1 and 2.2.

$$Z_{([l])} = W_{([l])} A_{([l])} + b_{([l])} \tag{2.1}$$

$$A_{([l])} = g_{([l])} Z_{([l])} \tag{2.2}$$

Where Z is calculated by adding the bias, b, of each layer, l, to the sum of the product between the trained weights, w, and the output data from the previous layer, x, the resultant value of Z is then normalized through an activation function and later passed onto another layer.

In this research, trained FNN structures are embedded in digital hardware to detect sleep apnea in real-time. Open-source clinical data collected from Philips University Medical Center [40] and St. Vincent Hospital [41] are used to train and validate the model for adult screening. A typical FNN consists of millions of connections, making it computationally expensive and memory intensive.

Therefore, deploying FNN on edge is challenging, especially when limited hardware resources are available, which constrain extensive and computationally intensive models from being embedded into the hardware. As a result, these networks must be compact and optimized for embedded deployment. Significant computational power and memory size reduction can be achieved by transforming a typical FNN model into the proposed models called DeepSAC and SABiNN, making these suitable for inference on the digital hardware system.

2.1 Neural Network Architecture for Biomedical Applications

Using neural networks in biomedical applications such as predicting respiratory diseases leading to apneic events in real-time can provide faster and more accurate results [42–44]. Murphy et al. [45] executed and compared two control algorithms: a linear adaptive filter and a nonlinear neural network. A comparative study concluded that a nonlinear neural network filter is preferable in detecting breathing patterns since linear filters have been proven inadequate for accurately observing irregular breathing patterns. In addition, incorporating the neural networks presents no additional computational burden in the control loop when executing heuristic predictive algorithms. Liu et al. implemented a "long-short-term-memory" prediction algorithm (LSTM), a recurrent neural network. This LSTM model can predict neonatal amplitude-integrated electroencephalogram (aEEG) signals so that physicians can forecast the possible abnormality of the brain functions of neonates in advance and can provide early intervention [46]. This method has been tested with 276 neonatal EEG data, and the predicted EEG data was remarkably similar to the actual EEG signals. Banner et al. used an artificial neural network (ANN) model to determine the work of breathing per minute or power of breathing (POB) non-invasively without the need for an esophageal catheter in patients with respiratory abnormalities [47]. The method was tested in 45 incubated adults (age 51$\pm$ 11 years, 28 males and 17 females) receiving pressure support ventilation (PSV). The training data was from an esophageal catheter and airway pressure/flow sensors that measured the power of breathing. The trained ANN provided an automated calculation of non-invasive POB. Bataille et al. reported an improvement in the accuracy of diagnostic classifiers between lung ultrasonography (LUS) and thoracic ultrasonography (TUS) using machine learning models [48]. ML models, such as neural networks, show significant promise in medical applications and have tremendous potential for paving the way for the development of emerging medical diagnostic systems.

2.2 Sleep Apnea Detection System

Most machine-learning-based apnea detection devices are connectivity dependent (due to the separate data acquisition and processing unit) and lack privacy and security. Besides, using a high computational system requires proper management and resources and, as a result, becomes an expensive diagnostic system. Due to the high cost of such diagnosis, nearly 80% of the adult US population remains undiagnosed. Based on the limited accessibility and resource of the current sleep apnea diagnostic devices, a detection device that is accessible, affordable, wearable, and power-efficient with automatic detection and screening capabilities is long overdue.

This dissertation proposes an apnea detection system for adults that takes in two types of input signal: single channel bio-potential ECG data from the chest strap and blood oxygen saturation (SpO_2) signal from a finger-tip pulse oximeter. The chest movement data comes from an adhesive single-lead ECG patch currently used in wearable and mobile application-based medical devices. These are replacing the Holter monitoring devices requiring multiple ECG leads. The oxygen saturation data (SpO_2) is derived from a commercially available pulse oximeter device attached to the patient's finger. Fig. 2.1 illustrates the block diagram of the system where the neural network inference block takes in the chest movement and the (SpO_2) data. Both signals are subsequently processed into digital form and then fed as the input to the decision-making block. The complete biomedical system will be portable and compact due to its minimal usage of sensors. This proposed design can significantly reduce the complexity of bulk, expensive devices, and discomfort due to multiple sensors attached to the patient's body.

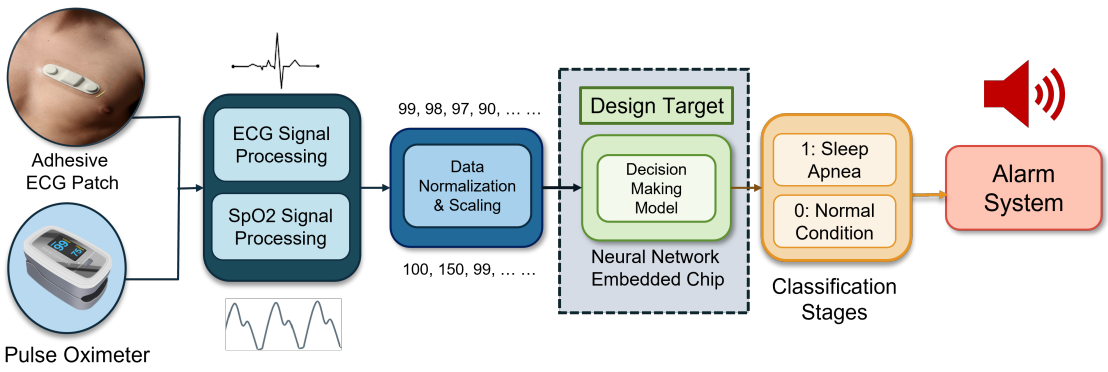


Figure 2.1: Block diagram of the proposed neural network (NN) embedded sleep apnea detection system. An adhesive ECG patch and fingertip pulse oximeter are front-end sensors, and NN is used as the decision-making block [16].

The decision-making block is designed based on the FNN model, which processes data feed-

forwardly. The proposed pre-trained FNN inference module takes input data from these front-end sensors and predicts apneic occurrences. The binary output (1: sleep apnea, 0: absence of apnea/normal condition) will initiate an alarm that will facilitate the patients to resume breathing by waking them up or alerting caregivers in the healthcare facilities to respond upon detection of an apneic event. Two primary sleep apnea detection system functions involve collecting data from biosensors and predicting apneic occurrences with high accuracy (over 70%).

2.2.1 Current Devices in the Market

The gold standard PSG method is commonly used for screening and diagnosing sleep apnea. But recently, various screening devices such as Home Sleep Apnea Test (HSAT) devices have proliferated the market which can potentially be used for pre-screening of apnea before ensuring expensive treatment through PSG method [49, 50]. Table 2.1 lists currently available HSAT devices, program providers, and their screening methods.

According to Table 2.1, most devices are incapable of accurate auto-scoring with AI/ML, and no precise accuracy rates were documented or claimed. In contrast, the proposed device provides high accuracy (over 70%) auto-scoring with minimal sensor features. Registered Polysomnographic Technologist (RPSGT) scoring in this system will be optional unless further diagnostic is necessary.

2.2.2 Emerging Solutions for Sleep Apnea Detection Systems

Various methods available for the detection of apnea presented in recent literature include a pyroelectric sensor, chest belt with a piezoelectric sensor, single-lead Electrocardiogram (ECG), blood oxygen saturation (SpO_2), microphone for tracheal sound monitoring, and snoring signals [49–52]. These devices are the first-line diagnostic methods for establishing the point-of-care diagnosis of sleep-disordered breathing. Mahbub et al. works [49] present the electronic circuitry for sleep apnea detection focusing on the respiratory monitoring of neonatal infants. The device integrated a pyroelectric transducer that converts the heat generated due to breathing into an equivalent electrical charge. If no breathing signal arrives within 10 seconds, the device considers this as an apneic event and, via wireless communication, transmits an impulse to a central coordinator, which alerts the caregivers. Significant efforts to develop non-invasive, low-power respiration monitoring devices with wireless telemetry capability for adults and neonatal infants are underway [49, 52]. An apnea detection and monitoring system to restart the breathing function of patients presents [53] a MEMS-based 3-axis accelerometer and a wristband. The acceleration sensor placed on the chest continuously

Table 2.1: Market Available at Home Sleep Apnea Testing Devices (HSAT)

Company	**Device**	**Method**	**Provider**
DreamClear	REMware DreamClear	Autoscoring via Ensodata or manual RPSGT	All (public, physicians, trucking companies, health plan, etc.)
ApneaMed	Philip Alice NightOne, ResMed ApneaLink Air, Itamar WatchPATONE	Manual Scoring and edited by RPSGT	All (public, physicians, trucking companies, health plan, etc.)
BetterNight	Philips Alice OneNight, ResMed ApneaLink Air, Itamar WatchPAT 300/WatchPAT ONE	Autoscoring assisted by AI, manual review and edited by RPSGT	Physicians, sleep specialists, health plans and ACOS, clinics, dentists, transportation organizations, enterprise/employers
Bioserenity	AccuSom	RPSGT Scoring	All (public, physicians, trucking companies, health plan, etc.)
Blackstone Medical Service	ResMed ApneaLink Air, Philips Alince NightOne	RPSGT Scoring	All (public, physicians, trucking companies, health plan etc.)
Sleepview Direct	CleveMed Sleepview	Autoscoring (not assisted by AI, Manual scoring by RPSGT	Sleep specialists, sleep centers, ENT, employers
Itamar	WatchPAT200/300, WatchPATONE	Autoscoring with Manual score (AI not included)	All (Any medical specialty, practice, hospital system. Or payor)
Sleep Care Online	Ectosense NightOwl	Autoscoring and sleep expert scoring	All (Any medical specialty, practice, hospital system. Or payor)
Sleeptest	ResMed ApneaLink Air	Autoscoring followed by manual scoring	Dentist, physicians
MedBridge Healthcare	ResMed ApneaLink Air, Philips Alice NightOne, Itamar WatchPAT/WatchPAT ONE, Ectosence NightOwl	Autoscoring (AI not mentioned and reviewed by RPSGT	Sleep physicians, general practitioners, hospital systems, DOT patients and employers, occupational health clients

monitors the movements of the diaphragm to detect apneic events during sleep. If an apneic event occurs, a closed-loop control system sends a signal to the wristband to trigger a vibration motor to disturb the patient until breathing is resumed. One underlying problem with these devices is their subject dependence.

Different subjects have different breathing patterns; therefore, calibration is needed to ensure accurate detection of the apneic event or proper monitoring of the breathing function of each patient. A comparative study in [54] identifies practical machine learning (ML) algorithms and appropriate physiological signals for sleep apnea detection based on a trade-off between computational resources and performance. Based on a comparison of deep learning (RNN and CNN) with conventional machine learning algorithms (Random Forests, Decision Tree, and Multi-Layer Perceptron), researchers have identified deep learning as the best performing algorithm, and oxygen saturation level (SpO_2) as the most important physiological parameter for SA detection [54]. Similarly, in [23], it is stated that a deep learning model outperforms classical machine learning methods for Central Sleep Apnea (CSA) detection for data obtained through pressure-sensitive mat (PSM). In most existing deep learning-based solutions, the data acquired from complex PSG experiments are often computationally intensive. Kristiansen et al.'s work [55] presents (SpO_2) based apnea monitoring devices with separate sensing and processing units with a wireless interface. This device requires pre-processing, complex hand-crafted feature extraction and a classical threshold-based classification model typically performed in a microcomputer.

Despite significant improvement in recent years, these devices still suffer from computational needs, power budget, additional circuitry requirements for wireless transmission, and a separate processing unit. Therefore, a real-time machine learning-based portable hardware model is necessary for point-of-care diagnosis of sleep apneic (SA) events.

2.3 Conclusion

Based on the limitations and strengths of the currently existing devices and recent research work in sleep apnea detection, a neural network-based intelligent electronic device for adults that can automatically detect apneic events with a high precision rate is proposed. During training, the NN model fed on a large amount of data from subjects of different ages, gender, body pattern, and different physiological conditions. This smart apnea detection device offers significant advantages over traditional instruments, such as on-chip decision-making capability, diagnosis based on multiple parameters, and less or no calibration due to training from a large volume of data. Studies

demonstrated that, among all the physiological signals, oxygen saturation level SpO_2 [56] and ECG [57] are the most relevant signals for sleep apnea detection among adults. The proposed system for adults presented in this paper takes (SpO_2) and ECG signals as the input and provides the presence or absence of apnea as a binary output.

Chapter 3

DESIGN METHODOLOGY OF MACHINE-LEARNING BASED HARDWARE

3.1 Introduction

Obtaining a high accuracy rate of Machine Learning (ML) based deep models often requires extensive data sets, which enables the model to run complex data structures with multiple features. As a result, the training phase of the models becomes computationally intensive and requires significant computational resources for executing numerous iterative weight updates. In many sensor-based applications, the system requires these models to infer into the hardware for real-time processing, such as image recognition and complex biomedical-related signal processing applications. Thus, extracting meaningful diagnostic results and reports in real-time is necessary instead of employing them in the cloud, which can reduce communication costs and enable on-spot emergency aid.

A four-step software-hardware co-simulation design process is introduced in this research as illustrated in Fig. 3.1 [14–16], which enables the designers to utilize both the software-based computer simulation and edge computing benefits in designing application-specific ML models embedded onto hardware. This process significantly reduces the overhead error in designing smart biomedical devices as each step undergoes multiple levels of verification and validation before moving on to the next step to ensure accurate design architecture.

3.2 Software-Hardware Co-Simulation Process

As shown in Fig. 3.1, the hardware-software co-simulation process is divided into four categories, each containing multiple sub-categories.

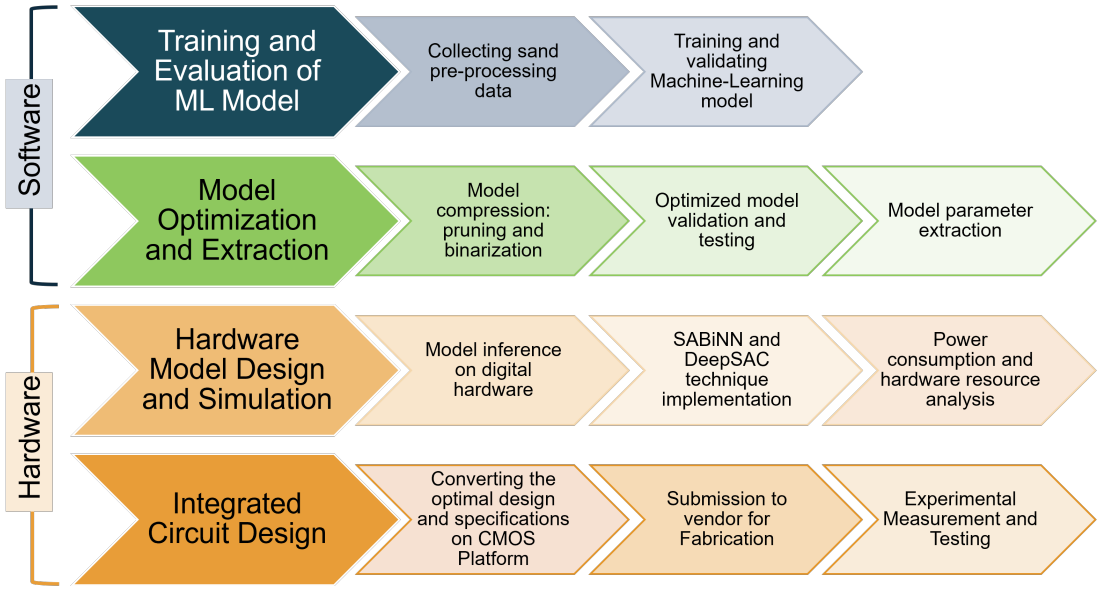


Figure 3.1: Software-Hardware Co-Simulation Design Process: From Software Simulation to Machine-Learning Model Inference on Hardware [15–17].

- **First category** is the training and validation of the chosen ML model, which is performed on software using high-processing machine learning libraries such as TensorFlow, Numpy, Keras, sci-kit-learn, etc. This step enables the designers to feed large data sets onto the model for achieving low error, high precision trained parameters, i.e., weights and biases.

- **Second category** of the proposed co-simulation process involves optimizing and reducing the sparsity of the trained ML model using various energy-efficient, compact design techniques such as integer quantization, pruning, binarization, data normalization, etc. These techniques are beneficial for the hardware inference of the model.

- **Third category** involves extracting all the necessary parameters and creating a design architecture of the trained ML model. This trained ML model will be the blueprint for designing the digital hardware system. In this step, the two proposed low-power design techniques - DeepSAC: Shift Accumulate Deep Neural Network and SABiNN: Shift Accumulate Binarized Neural Network will be used to enable a more compact and high-precision design architecture for the hardware model. Based on the accuracy rate, the selected deep learning model will be translated and implemented into hardware description language (Verilog/VHDL), which will be programmed into FPGAs. Then it will be tested for the consistency of the results

between the software's accuracy and the hardware. Next, the hardware testing will focus on speed, power, and processing

- **Fourth category** converts the digital design elements programmed onto FPGA into ASIC design and will be sent for fabrication. After the desired ML inference chip is fabricated, further testing and measurement will be done for simulation and experimental result consistency.

The benefit of using this co-simulation design process is evaluating and verifying the ML model in each step before proceeding to the next. This way, the error rate of the final model will be significantly reduced, resulting in a more confident, high-precision ML hardware architecture. Inference of the ML model onto CMOS integrated circuits will also aid in commercializing smart wearable and portable biomedical devices.

3.2.1 Discussion

Using the software-hardware co-simulation design scheme and implementing machine learning-based deep models which process medical data and signals represents a novel approach to designing smart biomedical instrumentation. This scheme will open doors for developing smart wearable sensors and lightweight medical instruments by having a high-speed diagnosis rate with a low power consumption rate. When targeting real-time applications for medical devices requiring data processing such as blood glucose level, respiratory signals, EEG, ECG, and other types, it is essential to address power consumption, sustainability, portability, and the overall system size. The traditional mode of implementing deep learning models, such as neural networks (NN) on commercial hardware such as GPUs and FPGAs, might successfully make the application work. However, the lack of portability and size will make it inadequate for wearable device applications. Thus, the device's size, power, and portability play significant roles in wearable biomedical system design. Integrating ML functions and algorithms into a single CMOS integrated circuit and targeting a power consumption rate of 50 µW will be a potential new approach to developing future wearable medical devices and technologies.

3.2.2 Experimental Benchmark

The DeepSAC and SABiNN hardware models are implemented onto the decision-making block of the proposed SA detection system by following the proposed software-hardware co-simulation method. In designing an SA detection device for adults, the ML training employs two commonly used medical data sets collected from Phillips-University, Marburg, Germany, and St. Vincent University Hospital

Sleep Disorder Clinic, Dublin, Ireland [19, 41]. Nexys Artix-7 general purpose FPGA is used in the first step of hardware inference, and for CMOS integrated circuit design, 130 nm and 180 nm design processes are used.

Chapter 4

MODEL OPTIMIZATION AND COMPRESSION TECHNIQUES

4.1 Introduction

A typical FNN model consists of millions of connections, making it computationally expensive and memory intensive. Therefore, deploying an FNN model in an embedded system is challenging, especially with limited hardware resources. However, few compression and acceleration techniques can be applied to make the NN model suitable for an embedded system. Two such techniques are pruning and quantization. This chapter discusses the sparse pruning technique introduced by Han et al. [1] and a newly improved hardware-based n-bit (n = 8,16) integer quantization technique to compress further and optimize the proposed FNN model. Two data sets are used in experimentation to validate and test the techniques.

4.2 Pruning

Pruning is a popular compression technique that removes redundant and inefficient parameters from the network without significantly reducing accuracy. One of the initial pruning techniques introduced in model compression was biased weight decay [58]. In contrast with human brain neuron connectivity concerning neural network training, LeCun et al. [59], and Hassibi et al. [60] state that the accuracy can be improved if the connections are removed based on the Hessian of the loss function, which is an indirect method of pruning. However, these two methods introduce an added computational complexity in the form of a second-order derivative. Even though other research works conducted in [61–63] prove that the pruning method removes the redundancy of weight values and keeps valuable neural network connections for decreasing computational and storage requirements for performing

inference. The major challenge is the preservation of the original prediction accuracy after pruning.

The pruning technique used and validated in this study is introduced by Han et. al., which is a magnitude-based technique that recursively discards the weights below a specified threshold [1]. Unlike other magnitude-based approaches, it recovers the model's accuracy through an iterative retraining process.

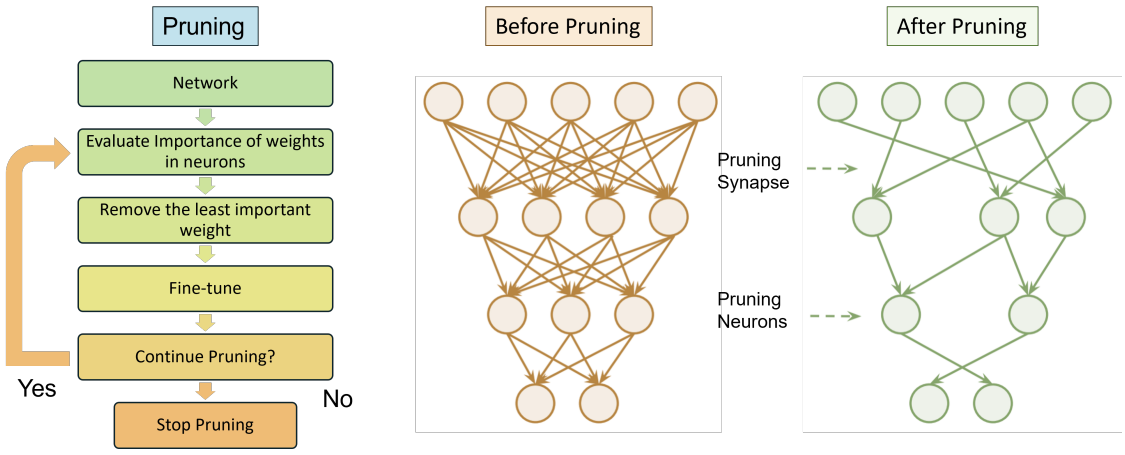


Figure 4.1: Iterative pruning process and network structure before and after pruning.

Fig. 4.2 illustrates the three steps involved in this technique. In the first step, conventional network training learns the connectivity, where the importance of each connection is learned instead of learning the final values of the weights. Then, the network connections with weights less than a specified threshold get discarded. The remaining sparse network retains the final learned weights in the final step. L2 (Ridge Regression) regularization was incorporated into the retraining process since it gives better accuracy after pruning. The regularization and gradient descent ensures the automatic removal of any neuron with zero input or output connections during the retraining process [64].

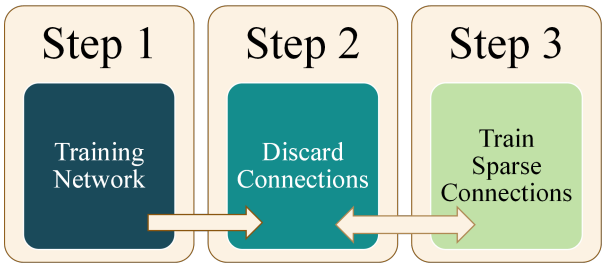


Figure 4.2: Three-step pruning process used for model compression and optimization.

The neurons with zero input or output connections are removed from the pruned connections. In the retraining step, the resultant value of the dead neurons is zero. These dead neurons occur due to gradient descent and L2 regularization. A neuron with zero input or output connections will not

affect the final loss calculation.

4.2.1 Pruning Experiments

Two types of FNN models are trained on three datasets where Han et. al.'s pruning technique [64]. For Image classification, MNIST [65], and Fashion MNIST [66] data sets are used. For biomedical screening and classification, Pima Native American data set is used for diabetes prediction [67], and sleep data collected from Phillips-University [40] and St. Vincent University Hospital [41] are used for sleep apnea detection among adults.

The hardware experiments of pruned models are tested on re-programmable hardware such as on Nexys Artix-7 FPGA boards. The network parameters and accuracy before and after pruning are described in the **Model Evaluation** section.

Pruning for Image Classification

In this section, pruning techniques got performed on the FNN shown in Fig 4.3. The first layer is called the input layer, the last layer is called the output layer, and based on the optimization and model design technique, the middle layers are called hidden layers. Each node in Fig 4.3 of the network is called a neuron unit. This type of neural network model works with 2D image classification, where the algorithm takes input images, then assigns importance through learnable weights and biases to various features in the image. MNIST and Fashion-MNIST data sets are used for training the FNN dense model and enabling it to classify image data.

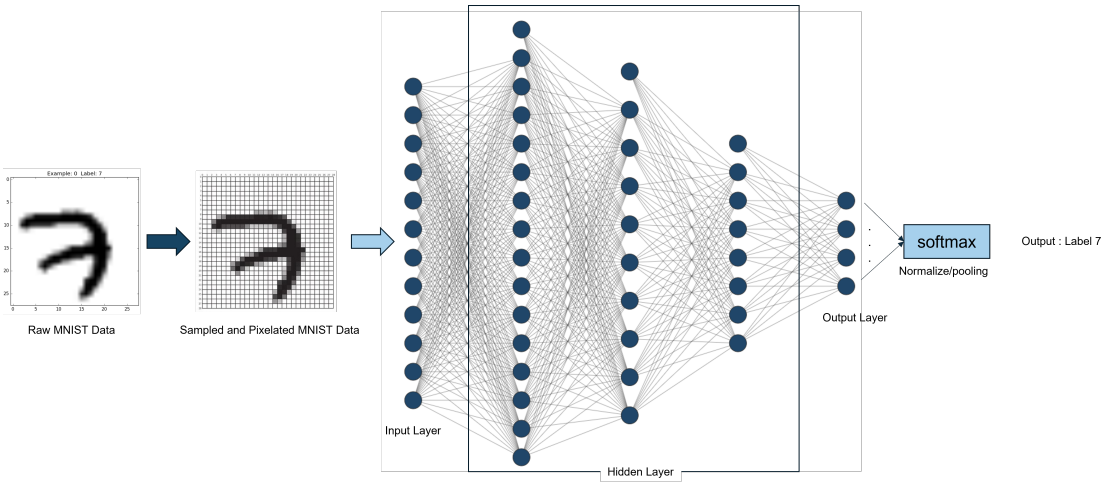


Figure 4.3: Sample model of a feedforward neural network (FNN) classifying MNIST data.

MNIST and Fashion MNIST Dataset

The first step in experimenting and learning how to train a model is by training it with a clean, extensive data set to understand how it works and can be further optimized. Created by Yaun LeCunn and the team, [65] MNIST is a handwritten digits data set in black and white color containing a training set of 60,000 examples and a test set of 10,000 examples. It is a commonly used data set for pattern recognition and image classification [65]. The fashion-MNIST dataset classifies clothing varieties for computer vision and deep learning. The data set comprises of 60,000 square images with 28x28 greyscale images. The data set includes ten types of clothing, and the mapping of all the clothing is labeled as integers from 0-9 [66]. Fig 4.4 showcases a sample of the MNIST numeric data set and some fashion-MNIST data sets with their labels.

Pruning for Biomedical Signal Processing

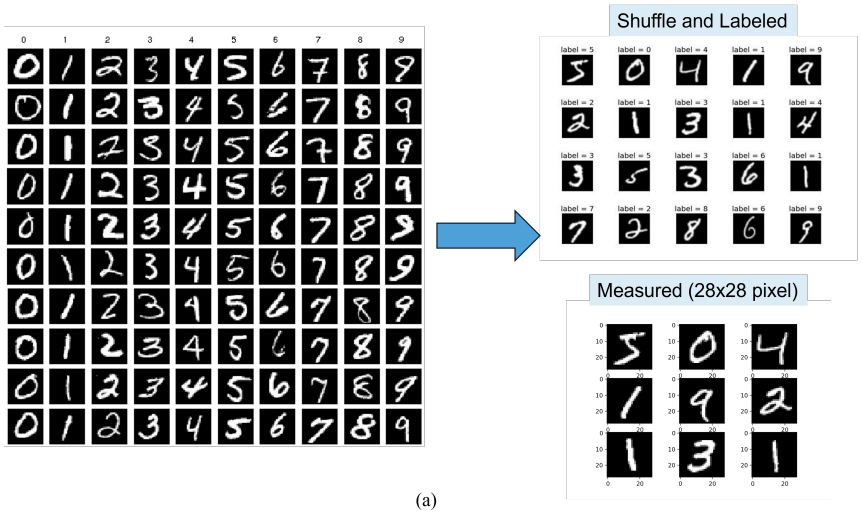
There are publicly available data sets that aid in benchmarking these methods to evaluate the accuracy and validate the concept of power-efficient techniques of a given Deep Neural Network (DNN) used in biomedical applications. Public data sets are essential for comparing the accuracy of different approaches for establishing any proposed concept in research communities, especially in biomedical applications. For predicting and detecting several diseases and complexities wide variety of open-sourced data sets are collected from various existing biosensors. In developing and designing the apnea detection system for adults, ECG, and SpO_2, signals from two sleep apnea databases are collected from PhysioNET Bank [41]. For developing a diabetes prediction device, eight attributes of diabetes measurement from the Pima Native Americans data set repository are used [67].

Sleep Apnea Data set

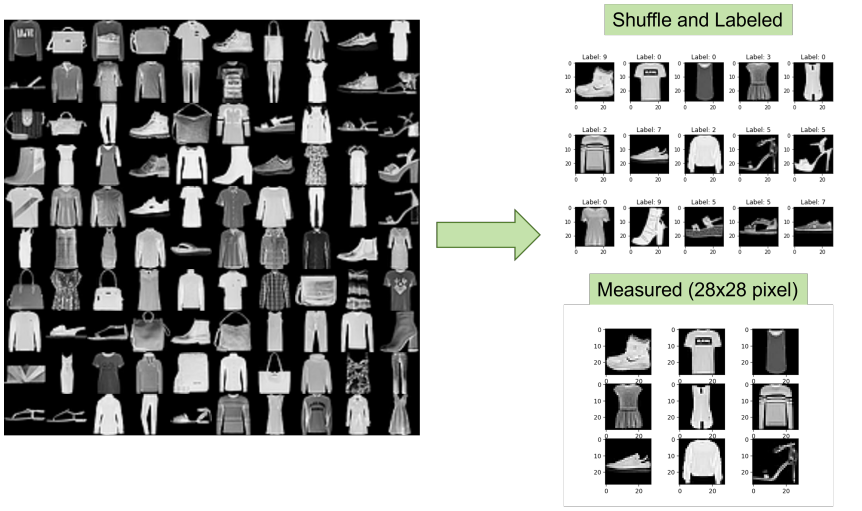
Two datasets are collected from Physionet Bank [41] as follows:

1. **Apnea-ECG** database records 8 overnight ECG recordings and fingertip SpO_2 patient data. For every minute of the data stream, an annotation indicates whether any apneic event has occurred at the start of the particular minute. Due to a mismatch of annotations, the entire dataset was cleaned and pre-processed before the training.

2. **St.Vincent's University Hospital** contains 25 full overnight PSG records with 3-channel Holter ECG and fingertip SpO_2 data from patients who have sleep disorders over six months. This dataset contains subjects over 18 years of age, and the total set contains 21 males and 4



(a)



Label	
0	T-shirt/Top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle Boot

(b)

Figure 4.4: (a) MNIST and (b) Fashion-MNIST data set samples which are labeled, shuffled, and measured at 28x28 pixels for classification.

females (age: 50 ± 10 years, range 28-68 years; BMI: 31.6 ± 4.0 kg/m^2, range 25.1-42.5 kg/m^2; AHI: 24.1 ± 20.3, range 1.7-90.9). Sleep technologists scored and annotated the sleep stages according to standard Rechtschaffen and Kales rules.

Fig 4.5 showcases a section of the raw ECG and SpO_2 signals that are later pre-processed and normalized for designing the FNN model. In the data processing scheme, both signals are divided into segments of 30 seconds intervals. R-R interval was extracted from the raw ECG signal by applying the R-peaks detector provided in the database. Finally, the R-R interval signal and the SpO_2 data were processed according to the previously reported techniques [28, 68].

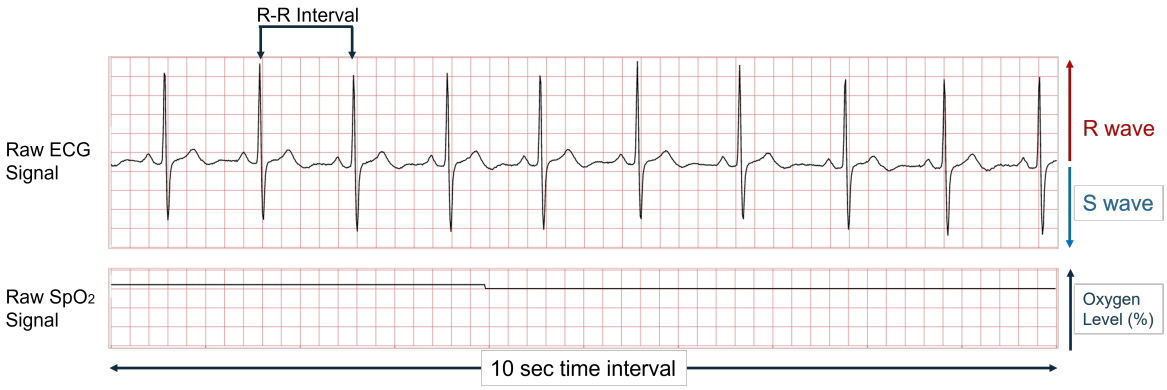


Figure 4.5: Raw ECG data and SpO_2 data within 10 seconds of time-frame collected from sleep apnea dataset.

After each minute, sleep experts annotate the Apnea-ECG (Phillips University Hospital) dataset. The ECG and the SpO_2 data of Apnea-ECG had a sampling rate of 100 Hz. This work divides the data stream into 30-second segments instead of 1 minute to achieve a higher precision rate. On the other hand, the St. Vincent University Hospital database recorded the ECG and the SpO_2 data at a sampling rate of 128 and 8 Hz, respectively. Both datasets were further processed, and existing artifacts were removed for higher accuracy. Any SpO_2 value less than 50% and any sudden change of saturation level greater than 4% within a 1-second interval are marked as artifacts since such values are physiologically impossible [55]. Once the artifacts were rejected, the signal was re-sampled at 1 Hz using a simple moving average filter. In the case of ECG signals, the dataset provided machine-generated QRS annotation. From the QRS, an R-R interval series is created by taking the time interval between two successive R-peaks. This is done using an R-peak feature extraction tool provided by PhysioNET. A sliding window technique is implemented to remove the ectopic sample points from the R-R interval series. The window length is 5 seconds, and any R-R interval larger than 20% of the average value within the window is marked as an ectopic beat and is removed. The

generalized data pre-processing scheme can be illustrated in Fig. 4.6.

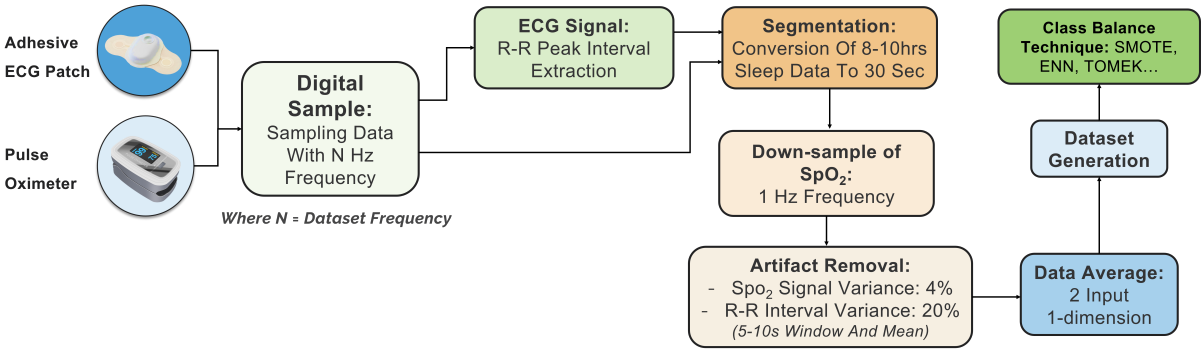
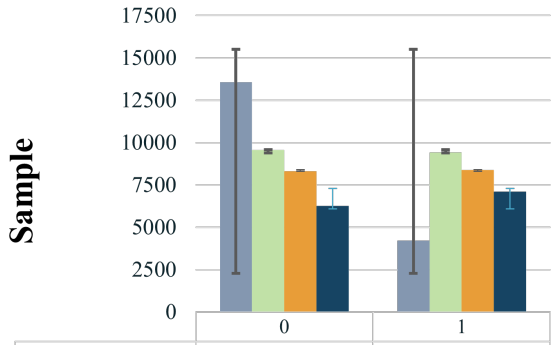


Figure 4.6: Data pre-processing, generation and class balance scheme [31].

In the original dataset, a significant amount of class imbalance was present. Due to the miss-classification and unbalanced dataset, the trained FNN model showcased a high recall rate (around 100%) and low accuracy. As a result, the FNN model is over-trained and is biased in identifying only the True Positives (TP). Researchers utilize various class imbalance techniques that avoid such training biases. These techniques include either generating minor class labels or removing the major class labels of the datasets. In this research, a combination of three techniques, SMOTE (Synthetic Minority Oversampling Technique), Tomek, and ENN (Edited Nearest Neighbor), were implemented and observed. SMOTE technique generates synthetic data from the minority class, while the Tomek technique removes major class data by locating all cross-class nearest neighbors. The ENN technique finds each observation's k-nearest (k = 3) neighbors and detects the difference between the major and minor classes. If different, then the observation and its k-nearest neighbor are removed. Generally, a combination of SMOTE-Tomek and SMOTE-ENN generates a balanced dataset. Fig. 4.7 illustrates the original imbalanced and balanced datasets generated using the three techniques. A reviser operating characteristic curve (ROC) of the proposed FNN was executed to understand its performance over each technique.

Fig. 4.8 (a)- (d) graphically showcases the ROC curves where the SMOTE+ENN dataset performed the best with a true positive rate of 83%. With a balanced dataset, the proposed FNN model showed optimal accuracy (80%) for medical screening and diagnosis. Further evaluation was performed based on the evaluation metrics such as precision, sensitivity, and F1-score, as shown in Table 4.1.

Table 4.1 concludes that the proposed model has a higher accuracy rate with a balanced percentage rate of precision, sensitivity, and F1 score.



	0	1
Original	13560	4207
SMOTE	9557	9425
SMOTE+TOMEK	8325	8379
SMOTE+ENN	6265	7118

Figure 4.7: Number of balanced samples versus data labels obtained by executing various class imbalance techniques.

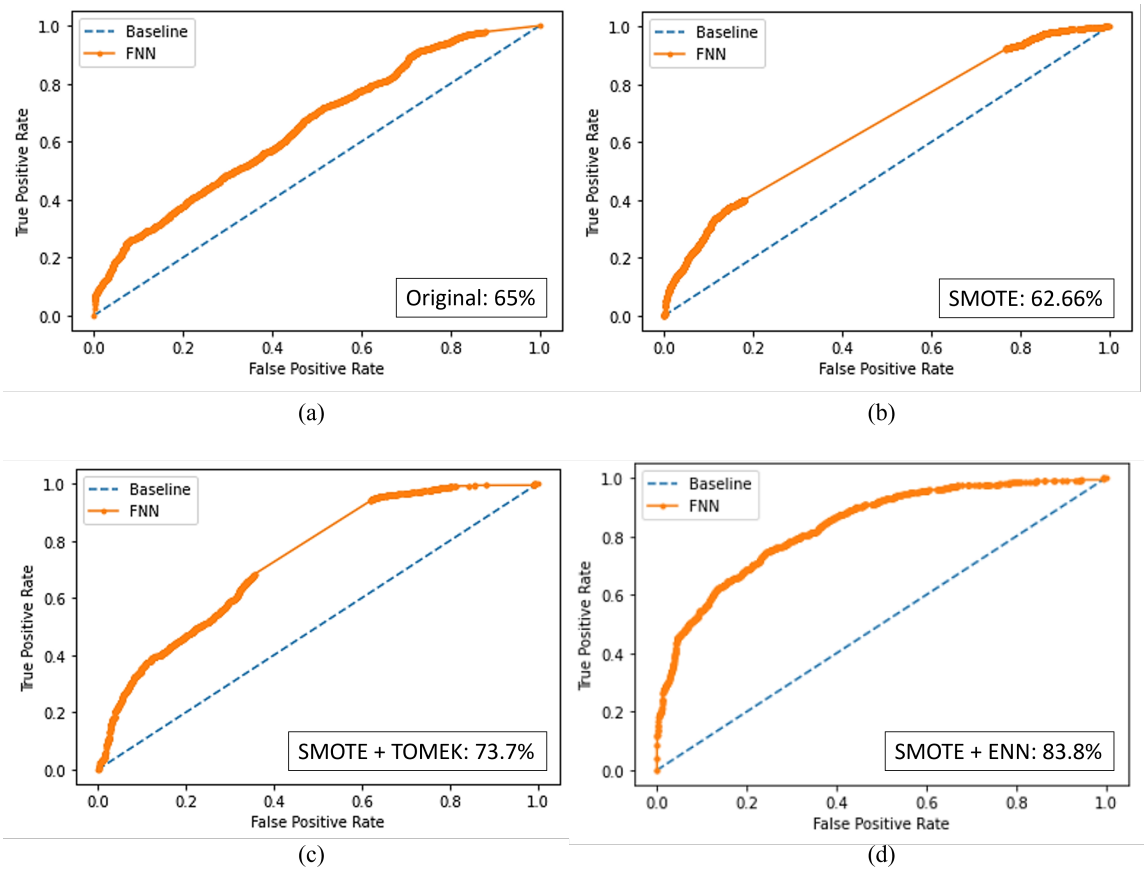


Figure 4.8: Reviser Operating Characteristics Curve (ROC Curve) between (a) original (65%), (b) SMOTE (62.66%), (c) SMOTE+Tomek (73%), and (d) SMOTE+ENN (83%).

Table 4.1: Comparative Study between proposed FNN and other ML models

Parameter	FNN	TW-MLP [28]	LS-SVM [69]	HMM-SVM [68]
Accuracy	87%	87%	83%	80%
Precision	88%	88%	84%	85%
Recall	85%	85%	84%	85%
F1-Score	86%	85%	79%	72%

Pima Native Americans Data set

Pima Native American is a data set that consists of eight input attributes related to diabetes, and each outcome is binary classified (1: positive case-risk of diabetes and 0: safe case-no risk diabetes). These eight attributes are pregnancies, plasma glucose concentration for 2 hours, diastolic blood pressure (mmHg), skin thickness, 2-hour serum insulin (μ U/ml), body mass index (kg/m^2), diabetes pedigree function and age. Among 768 female samples in the dataset, 268 are healthy, and 500 are unhealthy. A synthetic monitor oversampling technique (SMOTE) overcame the class imbalance of the data set for a more efficient NN training in achieving a high accuracy rate. The data set was normalized between 0-1 using min-max scaling [17]. Table 4.2 shows an example and structure of the Pima Native American dataset.

Table 4.2: Pima Native American Diabetes Dataset

Pregnancy	Glucose	BP	Skin thickness	Insulin	BMI	Diabetes Pedigree	Age
6	148	72	35	0	33.6	0.627	50
1	85	66	29	0	26.6	0.351	31
8	183	64	0	0	23.3	0.672	32
1	89	66	23	94	28.1	0.167	21
0	137	40	35	168	43.1	2.288	33
5	116	74	0	0	25.6	0.201	30
3	78	50	32	88	31.0	0.248	26
10	115	0	0	0	35.3	0.134	29
2	197	70	45	543	30.5	0.158	58

Pruning Neural Networks

For both image classification and biomedical signal processing FNN model was chosen. Fig 4.6 shows the basic structure of FNN, and Table 4.3 shows the characteristics and dimensions of each network for each type of data set. Adaptive Moment Estimation also know as ADAM optimizer was used to optimize the FNN model. This optimizer is a combination of two gradient descent algorithms.

One being the momentum algorithm and the other is the Root Mean Square Propagation (RMSP). The momentum algorithm accelerates the gradient descent technique by calculating the exponential calculation of the weighted average [70]. Using such an average makes the algorithm converge toward the minima faster. Equations 4.1 and 4.2 showcase the momentum calculation:

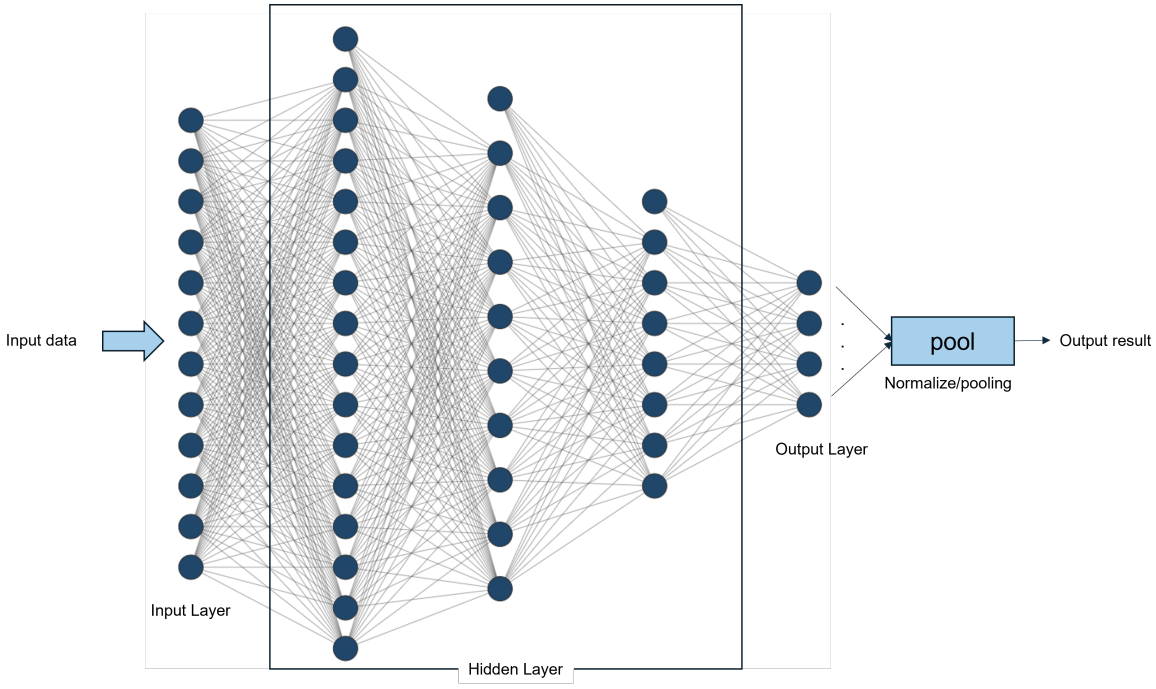


Figure 4.9: Sample model of a feedforward neural network (FNN) with input layers, hidden layers and output layers.

Table 4.3: FNN Training and Model Features

Data set	**Model Architecture**	**Loss function**	**Activation (output layer)**	**Activation (input layer)**
MNIST	784-(64-64)-10	categorical cross entropy	softmax	Rectified Linear Unit
Fashion-MNIST	784-(1000-1000-500-200)-10	categorical cross entropy	softmax	Rectified Linear Unit
Sleep Apnea (combined)	2-(8-12-6-4)-1	mean squared error	sigmoid	Rectified Linear Unit
Pima Diabetes	8-(12-8-4)-1	mean squared error	sigmoid	Rectified Linear Unit

$$w_{t+1} = w_t - \alpha m_t \tag{4.1}$$

$$m_t = \beta m_{t-1} + (1-\beta)[\frac{\delta L}{\delta w_t} \tag{4.2}$$

Where w_t is weighted at time t, w_{t+1} is weighted at time t+1, m_t is aggregate of gradients at time t,m_{t-1} is the aggregate of gradients at the time (t-1), β is moving average parameter (constant = 0.9),δL= derivative of the loss function and δw_t = derivative of weights at time t.

The RMSprop or RMSP is an improved form of AdaGrad [71], an adaptive learning algorithm. It considers the exponential moving average instead of the sum of squared gradients such as AdaGrad. The equations 4.3 and 4.4 show the algorithm for RMSP calculation [71].

$$w_{t+1} = w_t - \frac{\alpha_t}{((v_t + \epsilon)^{1/2}} * [\frac{\delta L}{\delta w_t} \tag{4.3}$$

$$v_t = \beta v_{t-1} - (1-\beta) * [\frac{\delta L}{\delta w_t}]^2 \tag{4.4}$$

Where w_t is weighted at time t, w_{t+1} is weighted at time t+1, v_t is the sum of the square of past gradients, β is the moving average parameter (constant = 0.9),δL= derivative of the loss function and δw_t = derivative of weights at time t, ϵ is a small positive constant (10^{-8}).

By combining the formulas, the mathematical expression of ADAM is developed, which is used in training the model [70] shown in equation (4.5).

$$m_t = \beta m_{t-1} + (1-\beta_1)[\frac{\delta L}{\delta w_t}] v_t = \beta_2 v_{t-1} | (1 - \beta_2 [\frac{\delta L}{\delta w_t}]^2) \tag{4.5}$$

Where β_1 (= 0.9) and β_2 (=0.999) are decay rates of an average of gradients in the above two methods and the learning rate α is 0.001 as default.

Two functions are used for loss calculation based on the class type and labels. For Image classification, categorical cross-entropy is used [72] as the data sets consisted of multi-class features. This function quantifies the difference between two probable distributions by calculating the loss of an example. It can mathematically be derived as shown in equation (4.6):

$$Loss = -\sum_{i=1}^{output \top size} y_i \cdot .log\ \hat{y}_i \tag{4.6}$$

Where $\hat{y}_i$ is the i-th scalar value in the model output, y_i is the corresponding target value and the output size is the number of scalar values in the model output.

For biomedical signal processing, since the dataset is 1-dimensional, the binary class (between 0

and 1) can be categorized in a linear-regression fashion. Mean squared error (MSE) [73] is used to calculate the loss for weight update purposes. MSE is a regression analysis that identifies how close a sample point is near the regression line by taking the distances from the point to the regression line and then squaring them. Using squaring removes the negative points. The MSE formula is given in equation (4.7):

$$MSE = \frac{1}{n}\sum (y_{\text{actual}} - y_{\text{forecast}})^2 \tag{4.7}$$

Where n = number of items, y_{actual} or observed y value and y_{forecast} is y-value from regression.

k-fold cross validation

The biomedical data set contained a limited number of data points; thus k-fold (where k = 10) cross-validation technique shuffled and folded the data set into ten folds. This technique avoids the model being over or under-fitted. It is a re-sampling technique that splits the training data into two parts: 1. training the model with training data and 2. testing the model with testing data set for validation. As data splitting happens randomly, one can also predict if the data set is balanced by iterative training. "k" is the number of re-sampled sets; ten sets are used in our training. On the validation side, the first set is treated as the test set, and the model is trained on the remaining k-1 sets. The error rate is calculated after the model is fitted on the test data. Fig 4.10 gives a graphical illustration of how the sampling works.

Figure 4.10: Re-sampling per iteration in K-fold cross-validation technique.

The mean of errors of all the iterations is calculated by the cross-validation (CV) test error estimate shown in equation 4.8.

$$CV(\mathrm{k}) = \frac{1}{k}\sum_{i=1}^{k} MSE_{\mathrm{i}} \tag{4.8}$$

In this calculation, the outputs of k-fitted models are averaged and somewhat less correlated since the overlap between the training sets is smaller.

Model Evaluation

Table 4.4 showcases the accuracy study and the model size of FNN by using MNIST and Fashion-MNIST data and Table 4.4 demonstrates the study of FNN by using Sleep Apnea ECG and SpO_2, and Pima Native American Diabetes data. These studies compare baseline and pruned models, illustrating the effectiveness and benefit of such techniques while maintaining a high accuracy rate.

Table 4.4: Pruned Test Results on Image Classification Dataset

MNIST-Data set			
Parameter	**Accuracy (%)**	**Model Size (Byte)**	**Model Speed**
Baseline	98	78230	18.4
Pruned	97.4	25765	10.3
Fashion MNIST-Data set			
Parameter	**Accuracy (%)**	**Model Size (Byte)**	**Model Speed**
Baseline	88.3	78300	18.5
Pruned	85.6	26035	9.3

Table 4.5: Pruned Test Results on Biomedical Data set

Parameter	**Accuracy (%)**	**Model Size (Byte)**	**Model Speed**
Sleep Apnea - Data set			
Baseline	79.35	53776	9.06
Pruned	79.36	23896	8.82
Diabetes - Data set			
Baseline	91.15	51728	9.54
Pruned	90	23396	6.91

According to Table 4.4, the model becomes 3x times smaller, and in Table 4.5, the model becomes nearly 2x times smaller after pruning, which is a significant reduction when embedding onto edge computing. All the models maintained their accuracy rate.

4.3 n-bit Integer Quantization

In hardware inference, integer quantization has become popular in optimizing Deep Neural Network (DNN) models that are computationally intensive and require large memory sizes. This technique uses n-bit integer values where n = 8,16 instead of floating-point numbers and integer math instead of floating-point math. As a result, it reduces the memory and computing requirements of the digital hardware. As stated in [74], using an n-bit integer point arithmetic operation instead of a 32-bit floating point reduces the energy by 18x times per operation by inference of trained parameters, i.e., weights and biases in the design of the neurons.

4.3.1 Methodology

In this research, for embedding FNN onto general-purpose hardware such as Field-Programmable Gate Arrays (FPGAs), each parametric real value was multiplied by 2^{10} while discarding the fractional parts during quantization, and the first three digits of that value were taken into consideration when designing each neuron unit. Appropriate scaling of weight and bias values of neural network models is possible due to their limited range. This is due to the weights being closer to zero and feature maps not exceeding upper bounds [75]. The dimension of weights and biases of the quantization technique in this research consists of 8-bit and 16-bit, where the most significant bit (MSB) represents its sign value [16].

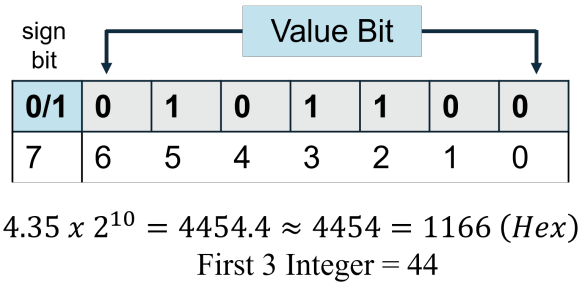


Figure 4.11: 8-bit integer quantization technique for hardware inference.

Fig 4.11 shows the calculation and transformation of floating point bit into integer bit quantization without sacrificing data information.

4.4 Conclusion

This chapter presents two feasible compression and optimization methods that can be used during hardware inference. By implementing the pruning method of Han et al. [1], the number of connections on FNNs is significantly reduced. An n-bit (n=8,16) integer quantization technique is also implemented to design the system using n-bit values for easy inference. The effectiveness of these two methods is executed by experimenting with two crucial classes of data sets and applications. One is MNIST data set and the other is the Fashion-MNIST data set that can be used in image classification, sleep apnea, and diabetes data sets in multiple biomedical screening and diagnosing applications. The technique compressed the models by 2x to 3x while maintaining the accuracy with a confidence interval (CI) of +/- 0.1%.

Chapter 5

DEEPSAC: SHIFT ACCUMULATE BASED DEEP LEARNING MODEL

5.1 Introduction

Deployment of the FNN model on edge is the key to designing real-time automated devices and systems capable of prediction, analysis, and classification. A significant cost reduction in communication with the cloud is seen in network bandwidth, latency, and power consumption in deploying FNN models on edge. On the contrary, edge devices have limited memory, power, and computing resources, constraining extensive and computationally intensive models from embedding into hardware. As a result, these networks must be compact and optimized for embedded deployment. As discussed earlier, two compact design techniques in this research are proposed that enable users to quickly deploy neural network (NN) models onto any digital hardware system. This chapter focuses on the Shift Accumulate Based Deep Learning Model Inference on Hardware (DeepSAC) and successfully builds a hardware-friendly FNN model to detect apneic events.

Neural Networks

Neural networks (NN) are composed of multiple consecutive layers and can be of different types, such as convolutional, pooling, and fully connected (FC) layers. There are dense, shallow, and deep neural networks based on their connection and layer numbers. Based on their learning style, they can be called convolutional neural networks (CNN), feedforward neural networks (FNN), shared weight networks, and others. Despite the variation and the learning diversity of NNs, the most fundamental component are their neuron and synapse unit [2].

Based on Fig. 5.1, the synapse unit takes the output value from the last neuron of the layer

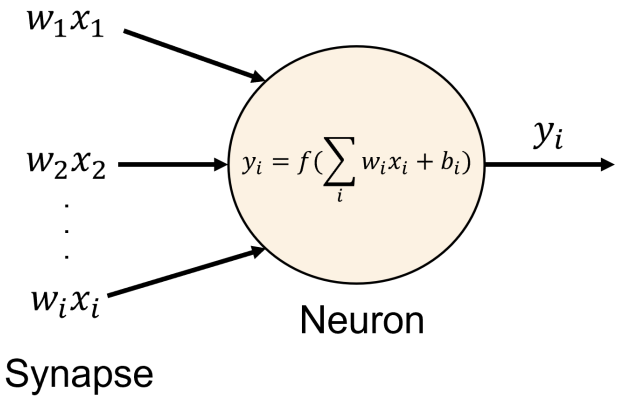


Figure 5.1: Synapse-neuron connection of NN model where w_i are the weights, b_i are the biases, x_i are the output values from the previous neuron or sensors, f is the activation function, and y_i is the output of the neuron for the next layer.

multiplied by the trained weights. It sends it to the activation function in the current neuron unit, which classifies the resultant value based on the selected activation function and sends it to the neuron of the next layer. For digital hardware, the multiply-accumulate (MAC) operation is the synapse-neuron unit of NNs. The major drawback of using the MAC operation is the multiplier component being a high power-consuming digital logic element. Therefore, the inference of dense NNs will require many MAC units with multiple usages of multipliers resulting in a high power consumption rate. The proposed shift accumulate (SAC) method completely removes the multipliers with shifters based on the trained weight parameters [16, 17]. Table 5.1 below illustrates the power consumption rate of the major hardware components that make up a hardware accelerator designed explicitly for machine-learning training and testing. The study of power consumption rate is acquired when the digital components are embedded into the CMOS platform.

Table 5.1: Power Consumption Study

Digital Logic Block	**Power**	**Ref**
16 bit Multiplier	10mW	[76]
16 bit Adder	0.78mW	[77]
16 bit Shifter	0.5nW	[78]

The algorithm for shift-accumulate-based (SAC) neuron-synapse design on digital hardware is given in equation 5.1.

$$SAC = \sum_{i=1}^{k} 2^{n} * input + b \tag{5.1}$$

$$n\epsilon R \tag{5.2}$$

SAC = the shift-accumulate operation, input = output from the previous layer into the current layer's input, b = node bias, and k = weight number on each layer.

Fig 5.2 shows a comparative study between multiplier and shifter units when they were designed on general-purpose FPGAs. According to Fig 5.2, a 16-bit multiplier consumes nearly 13x times more power than a 16-bit shifter.

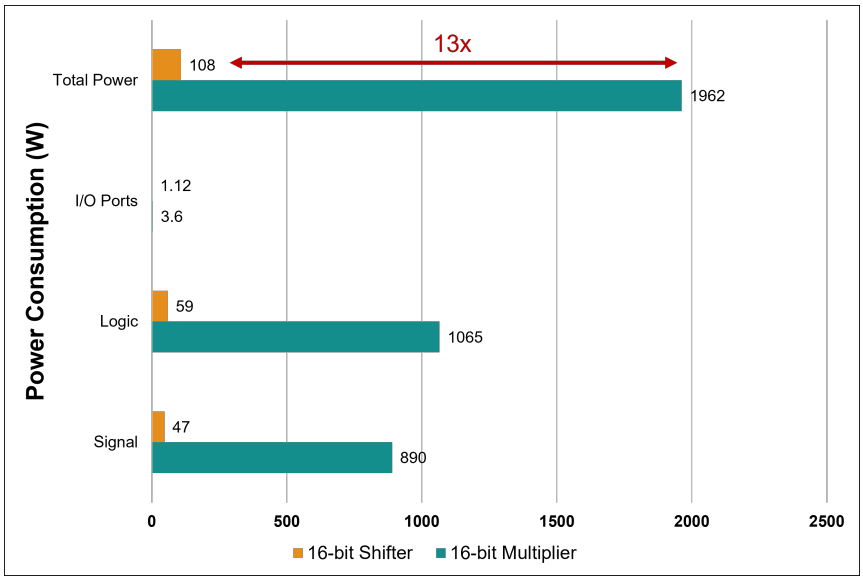


Figure 5.2: Comparative power consumption analysis between the 16-bit multiplier and 16-bit shifter simulated and measured on general-purpose Nexys Artix-7 FPGA board.

To develop the DeepSAC module, first, a targeted NN model is trained, in this case, a feedforward model, and all its weights are extracted. At first, compression techniques such as n-bit quantization and pruning are implemented to reduce the sparsity of the FNN model. After compression, the next task is to convert the hyperparameters into multiples of 2s to replace multipliers with shifters. After a successful conversion, the weights are plugged into the original NN model, and a forward pass is executed to check if the accuracy is maintained. Fig. 5.3 showcases the full algorithmic flow of the DeepSAC model architecture.

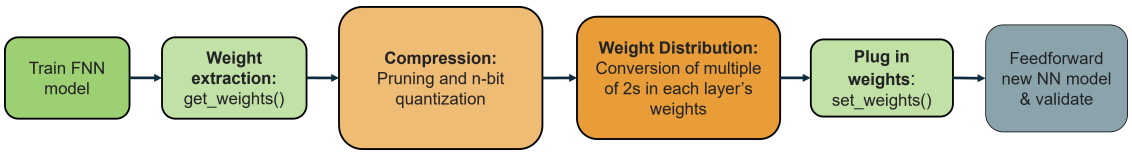


Figure 5.3: DeepSAC compression and conversion process in a feedforward neural network.

5.2 DeepSAC for Biomedical Applications

The DeepSAC method is implemented in the decision-making block of the sleep apnea detection system shown in Fig. 2.1 in Chapter 2 section **Sleep Apnea Detection System**. It is a power-efficient deep learning model-based digital hardware design scheme that enables the development of biomedical devices capable of monitoring real-time health conditions [16]. This design scheme aims at improving the processing limitations between the DL models and the edge devices as described below:

1. During the pre-training and optimization of the model, the user can utilize the machine-learning software platforms through the cloud instead of training it exclusively on the hardware accelerators. This significantly reduces the excessive hardware resource utilization and on-chip memory usage.

2. For model compression and on-chip memory utilization, before inference, the model goes through pruning and integer quantization for compression and avoiding floating point values.

3. Low-power hardware design techniques such as shifters are used instead of the matrix-multiplication (MAC) operation in the neural networks synapse-neuron connection.

4. All the activation functions are designed using the piece-wise linear algorithm and leveraging Look Up Tables (LUTs)

5.2.1 Experiments on Re-programmable Hardware

Two major components make up a neural network are:

1. **Neuron-Synapse Unit:** This unit takes input data from the previous layer, multiplies it with the trained weights, and sends it to the next unit.

2. **Classification Block:** Before sending the data from one neuron to another, there is a classification block called the activation function, which classifies the output value of the neuron and, produces a result based on the condition of the function and sends it to the next neuron. In both biomedical system designs, a 1-dimensional neuron unit is designed using shifters. In the model, Rectified Linear Unit (ReLU) in the hidden layer and Sigmoid as the output layer are used as activation functions. The following section describes the mathematical calculation of the activation functions and DeepSAC's shifter-based neuron design.

Activation Function Design

The activation function plays a crucial role in data classification. In neural networks (NNs), the product of the weights and the input data gets classified through these functions. Therefore, the proper choice of such functions for different layers in the network is essential, especially for gaining a high accuracy rate [16]. The activation function enables NNs to learn complex data patterns in training the model and their usage is high as each neuron unit uses this block for classification. Therefore, it is essential to design the NNs in a way that consumes low power while also utilizing limited resources from hardware. Fig. 5.4 showcases a comparative power consumption study report between three widely used activation functions. These studies were performed in Vivado HLx software using Artix-7 FPGA.

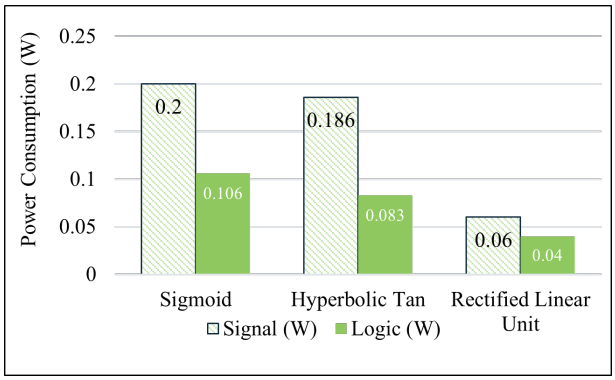


Figure 5.4: Power consumption study between three commonly used activation functions sigmoid, hyperbolic tangent and rectified linear unit in NNs.

Two activation functions called ReLU and Sigmoid are used in the trained models. ReLU is a piece-wise linear function that sends the input data as the output if it results in a positive value and zeroes if the value is negative, as shown in the equation below for the ReLU function. The function designed used a simple if-else logic unit on the hardware. The sigmoid function is used in the output layer for both models, which successfully categorizes the input values from the previous layer between 0 and 1. According to Fig 5.4, although the hyperbolic tan function consumes less power than the sigmoid when training the proposed NN model, the accuracy rate when using this function dropped from 80% to nearly 60%. As for using the sigmoid function, the model accuracy was maintained at 80%. In digital hardware, the n-bit (n = 8, 16) sigmoid function is designed using a piece-wise linear function as shown in the equation below:

$$ReLU = \begin{cases} 0, & \text{if } x \leq 0 \\ x, & \text{if } x \geq 0 \end{cases} \tag{5.3}$$

$$Sigmoid = \begin{cases} 128, & \text{if } x \geq 512 \\ 2^{-5}|x| + 107, & \text{if } 256 \leq |x| < 512 \\ 2^{-3}|x| + 80, & \text{if } 128 \leq |x| < 256 \\ 2^{-2}|x| + 64, & \text{if } |x| \geq 0 \end{cases} \tag{5.4}$$

The ReLU and sigmoid activation functions are designed on 180 nm CMOS process as 8-bit digital logic units based on equations 5.3 and 5.4 and 16-bit on 130 nm CMOS process. The ReLU activation function uses stack multiplexers, and the sigmoid activation function uses OR gates and stacked multiplexers. The layout of the ReLU and sigmoid functions are shown in Fig 5.5. The transient analysis of ReLU is shown in Fig 5.6 where $a_{1\text{-}8}$ are output channels, and $b_{1\text{-}8}$ are input channels. According to the Fig. 5.6, whenever the most significant bit MSB is negative, the output shows "00000000," and whenever positive 8-bit values are passed, the output forwards the exact value. The transient analysis of the sigmoid activation function is shown in Fig. 5.7, where $a_{1\text{-}8}$ are input channels, and output is the 1-bit output channel. According to Fig. 5.7, whenever the value is below "00000000" or negative, then sigmoid results in "0," and on positive values, it classifies as "1". Table 5.2 presents the CMOS characteristics of ReLU and sigmoid.

Table 5.2: Measurements of ReLU and Sigmoid on 180 nm CMOS Process

Parameter	**ReLU**	**Sigmoid**
Area	0.004 μm^2	0.008 μm^2
Supply Voltage	5V	5V
Period	10ns	10ns
Power	0.0995μW	1.47μW
Energy	9.95fJ	1.477pJ

Synapse-Neuron Design

According to Fig. 5.8, which is a digital synapse-neuron connection of a DeepSAC model, the input value is shifted based on the n-bit shifter where n is the assigned bit number and gets shifted back to an 8, 16-bit integer for data consistency and then added to either the bias or directly passes onto the assigned activation function. Each weight extracted from the model gets conditioned into the power

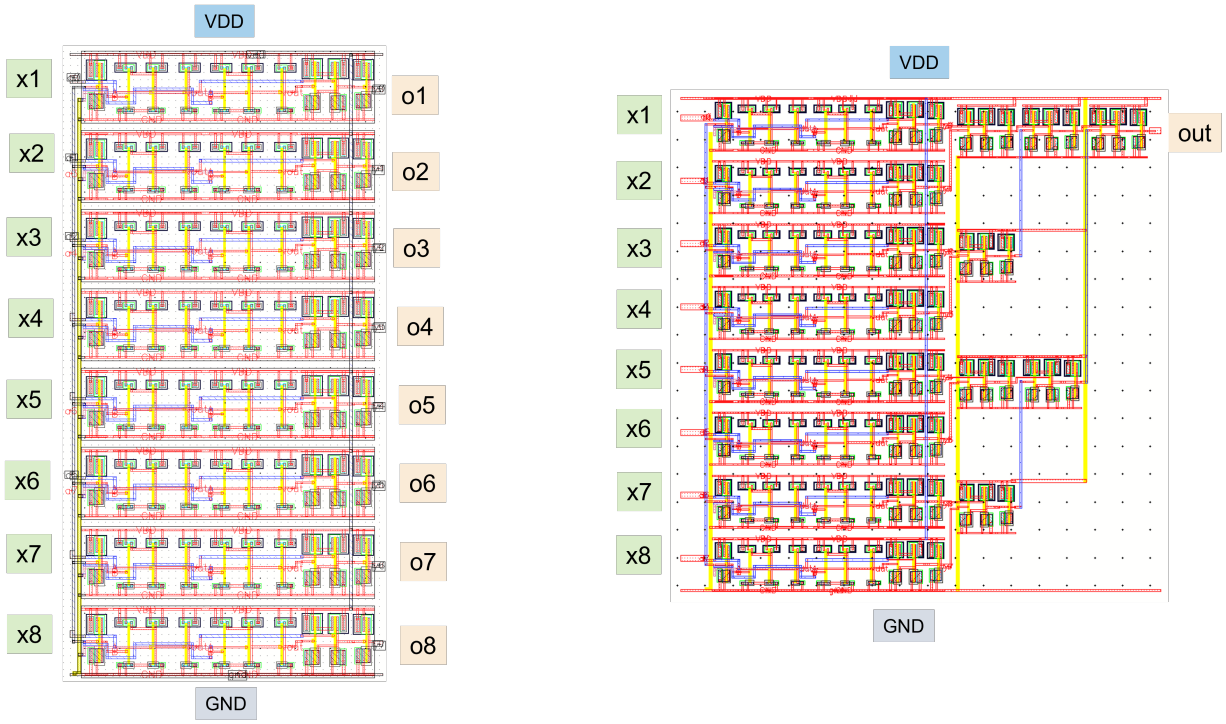


Figure 5.5: Layout image of activation functions on 180nm PDK (a) Rectified Linear Unit (ReLU) with 8bit input/output and (b) Sigmoid function with 8-bit input and 1-bit output.

of 2. As a result, the shifter acts as the multiplier. It gets its bits based on the weights designated to their corresponding synapse, which is a significant improvement as reported in [79]. The weight values within a specific range get replaced with their corresponding shifter in the SAC operation. For example, if a weight has a value of 34 in an 8-bit integer, then instead of multiplying the input data by the value 34, the input data gets shifted to 5 bits, as shown in Table 5.2.

Table 5.3: Shifter Bit Based on Weight Range

Weight Value Range	**Shifter Bit**	**Weight Value Range**	**Shifter Bit**	**Weight Value Range**	**Shifter Bit**
0	0	6-9	3	59-80	6
1-2	1	10-28	4	81-180	7
3-5	2	29-58	5	181-280	8

Validation of the Proposed FNNs

Two 3-hidden layer FNNs are designed and trained for classifying the two selected data sets. One dataset is for classifying and predicting diabetes among pregnant women, and another is screening

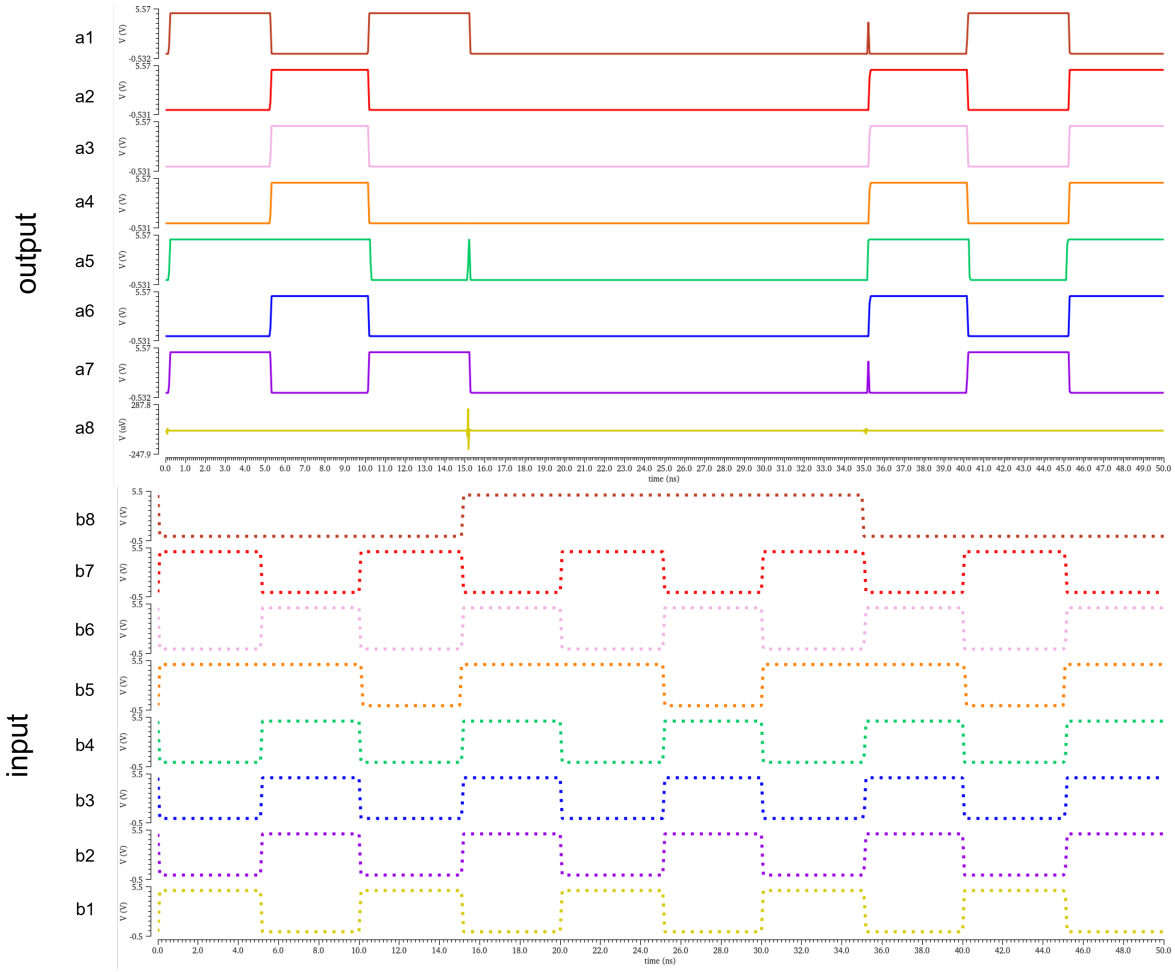


Figure 5.6: Transient analysis of ReLU activation function with 50ns period and 5V supply voltage. $a_{1\text{-}8}$ are output channels, and $b_{1\text{-}8}$ are input channels.

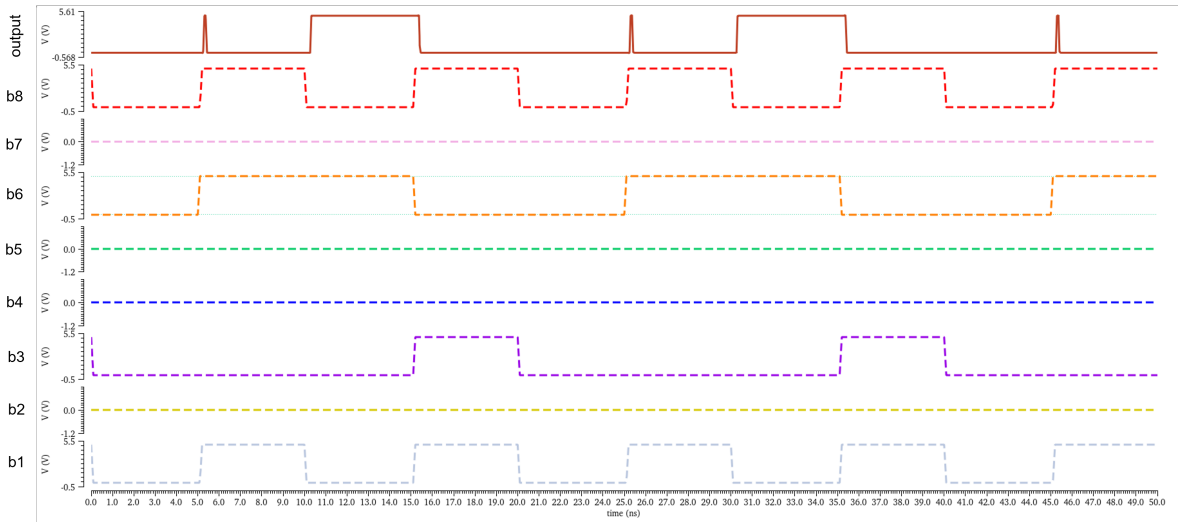


Figure 5.7: Transient analysis of Sigmoid activation function with 50ns period and 5V supply voltage. $b_{1\text{-}8}$ are input channels, and output is a 1-bit output channel.

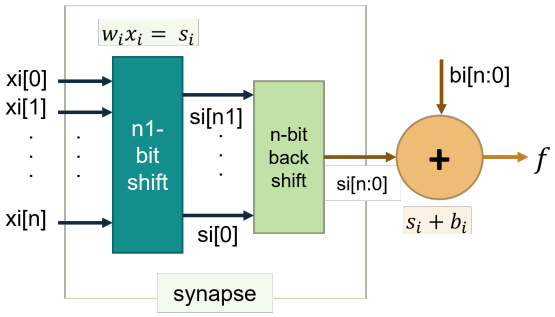


Figure 5.8: Shifter base synapse-neuron connection of NN model where w_i are the weights, b_i are the biases, x_i are the output values from the previous neuron or sensors, s_i are the shifted value f is the activation function and y_i is the output of the neuron for the next layer.

sleep apneic events among adults. Tables 5.4 and 5.5 summarizes the validation results of the two models before and after they were converted into the DeepSAC model. The validation was carried out based on their performance evaluation metrics and the accuracy of the tested data. The performance evaluation metrics are calculated using the four widely accepted quality metric schemes shown in the equations (5.5) - (5.8) below:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.5}$$

$$Precision = \frac{TP}{TP + FN} \tag{5.6}$$

$$Recall = \frac{TP}{TN + FP} \tag{5.7}$$

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{5.8}$$

Where, true positive (TP) is the number of correctly classified labels, which results in the binary value “1”; true negative (TN) is the correctly classified label which results in the binary value “0”; false negative (FN) is the incorrect classification label resulting in “0” and false positive (FP) is the incorrect classification label resulting in “1”. The FNN models were trained and validated using the selected data sets from [80].

Table 5.4: Performance Evaluation Between FNN and DeepSAC: Apnea Detection

Parameters	**FNN**	**DeepSAC**
Accuracy %	79	77
Precision %	77	77
Recall %	77	77
F1-Score %	87	87

Table 5.5: Performance Evaluation Between FNN and DeepSAC: Diabetes Prediction

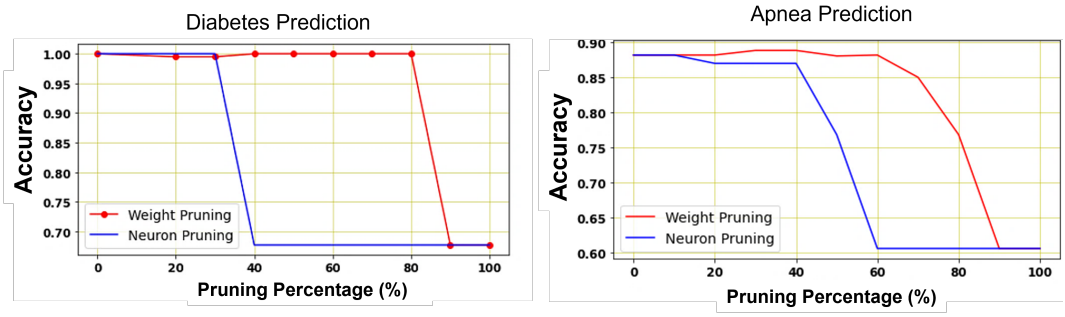
Parameters	**FNN**	**DeepSAC**
Accuracy %	91	90
Precision %	84	85
Recall %	88	88
F1-Score %	86	86

5.3 Significant Improvement

After successful training and evaluation of the FNN models, each component of the NN model is translated into digital logic and embedded into digital hardware. Widely available general-purpose Nexys Artix-7 FPGA is selected for embedding the trained FNN model. This general-purpose FPGA from the Nexys family tests how the SAC-based FNN model can utilize limited hardware resources due to its basic logic block design architecture. The hardware performance analysis validates the proposed technique, such as hardware resource utilization, power consumption rate, and output signal generation. After pruning the FNN models to 40% and 30%, respectively, as shown in Fig 5.9 (a) and (b), and applying the integer quantization technique in converting real numbers into integer bits, the power consumption rate is significantly reduced in both the models as shown in Table 5.6 and Table 5.7.

Table 5.6: Power Consumption Report for DeepSAC model: Apnea Model

	Without Pruning		**With Pruning**	
Parameter	**Vivado HLx Software (W)**	**Artix-7 Nexys Hardware (W)**	**Vivado HLx Software (W)**	**Artix-7 Nexys Hardware (W)**
Signal	14	18.156	3.288	2.364
Logic	18.5	22.79	3.830	2.781
I/O Ports	0.82	10.2	0.709	8.733
Total Power	**34.37** Dynamic: 33.3 Static: 0.446	**51.932** Dynamic: 51.135 Static: 0.797	**7.976** Dynamic:7.827 Static: 0.150	**14.186** Dynamic: 13.879 Static: 0.308



(a) (b)

Figure 5.9: Accuracy vs. pruning percentage for (a) diabetics prediction model and (b) apnea prediction model using magnitude based pruning.

Table 5.7: Power Consumption Report for DeepSAC model: Diabetes Model

	Without Pruning		**With Pruning**	
Parameter	**Vivado HLx Software (W)**	**Artix-7 Nexys Hardware (W)**	**Vivado HLx Software (W)**	**Artix-7 Nexys Hardware (W)**
Signal	47	61.127	19.974	15.530
Logic	159	69.941	17.985	17.553
I/O Ports	1.12	9.579	1.047	10.844
Total Power	**109.171** Dynamic: 108.380 Static: 0.791	**141.443** Dynamic: 140.647 Static: 0.797	**39.797** Dynamic:39 Static: 0.797	**44.724** Dynamic: 43.927 Static: 0.797

The logic block and the I/O ports are the most power-consuming elements due to the dense design structure of the FNN model, which consists of multiple neuron units and connections.Such dense amount of digital components result in high input-output data streaming blocks. The model pruning reduces the network connections resulting in a low number of I/O ports. The conversion from the float type to the integer type projects a low power consumption rate due to low memory allocation and computing operation.

5.3.1 Simulation Results

For FNNs, the main bottleneck for analyzing data is the memory access. Each MAC operation requires three memory reads (weight, activation function, and bias) and one memory writes. For the calculated value (partial sum), [81] in the worst-case scenarios where the FNN structures are deep and consist of more than two hidden layers and using many MAC blocks must utilize off-chip DRAMs resulting in a high power consumption rate. In [82], a trained AlexNet embedded in digital

hardware utilizes 724 MAC units accessing over 3000 MB DRAM memory. However, using the proposed DeepSAC operation where each shifter behaves as a weight requires zero memory access as no weight values get stored in the hardware. In such a scheme, an input data stream is shifted based on its weighted value and is passed through the output into an activation function. Each activation function also does not have to access memory allocation due to the usage of shifters instead of multipliers contributing to low power consumption. During the pruning of both networks, parameters such as biases in the model are discarded. Even when biases are discarded, the accuracy rate (over 77%) was maintained as described in Chapter 4, section 4.1. Thus, there is no memory allocation for the sum of the biases in neuron calculation. Such a shifting method significantly reduces the model size for both models (using PIMA dataset: without pruning 2409 Bytes, with pruning 2202 Bytes, and using the sleep apnea dataset: without pruning 1994, with pruning 1921 bytes). Fig. 5.10 showcases the comparative study of resource utilization between two testing setup scenarios in pre- and post-pruning. In Fig 5.11, zero RAMs were used in both models, and LUTs and slice registers were mostly utilized in the design structure. Fig. 5.12 presents the total model size before and after pruning for both (a) SA detection and (b) diabetes prediction.

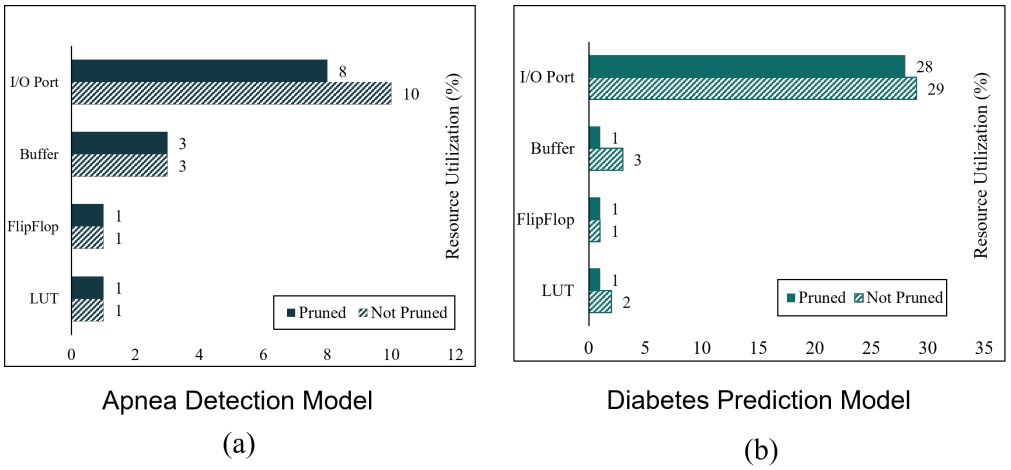
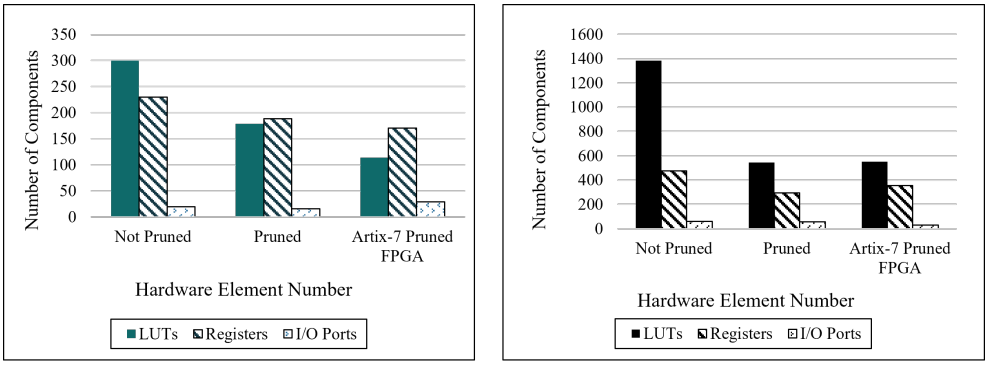


Figure 5.10: Resource utilization analysis on re-programmable hardware before and after pruning with integer quantization (a) apnea detection model (b) diabetes prediction model.

5.3.2 Test Bench Results

According to Fig. 5.13 and 5.14, the testbench simulation of the FNN models on Artix-7 FPGA hardware shows accurate results. The embedded models are tested using the test sets from the collected datasets and matched with the prediction results of the software and the provided labeled dataset. According to Fig. 5.13, a set of input values represented as x1 and x2 from the two sensors



Apnea Detection Model

(a)

Diabetes Prediction Model

(b)

Figure 5.11: Number of digital logic units on re-programmable hardware before and after pruning and integer quantization embedding on FPGA board (a) apnea detection model (b) diabetes prediction model.

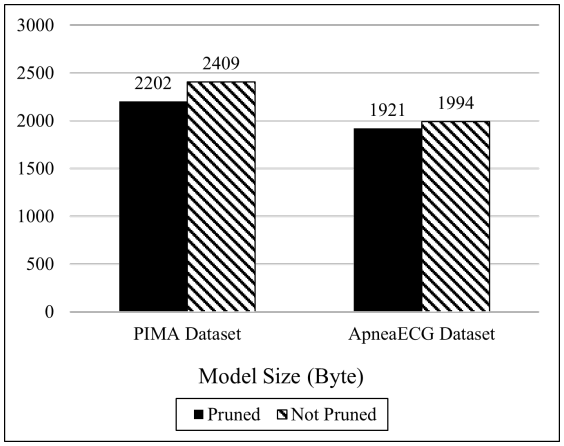


Figure 5.12: Final embedded model size on re-programmable hardware before and after pruning on FPGA board (a) apnea detection model (b) diabetes prediction model.

(single channel ECG and SpO_2 respectively) resulted in "0" for normal condition and "1" for detection of apnea. In Fig. 5.14, eight input values x1, x2, x3, x4, x5, x6, x7, and x8 representing the diabetic parameters resulted in "0" for no risk of diabetes and "1" for risk of diabetes based on a set of input values. The output of both models matched the labels generated by the prediction results of the FNNs. The models are simulated in Vivado HLx software. According to the sleep care foundation, if R-R intervals are greater than 99 and SpO_2 is smaller than 93%, then the patient has a possible sleep apnea attack. In Fig 5.13, when the value of x1 is lower than 99, and SpO_2 is higher than 93%, then the output y results in "0," and when x1 value is greater than 99, and SpO_2 is lower than 93%, then the output y is "1" detecting possible sleep apnea. Each classification was done in a 10 ms time frame.

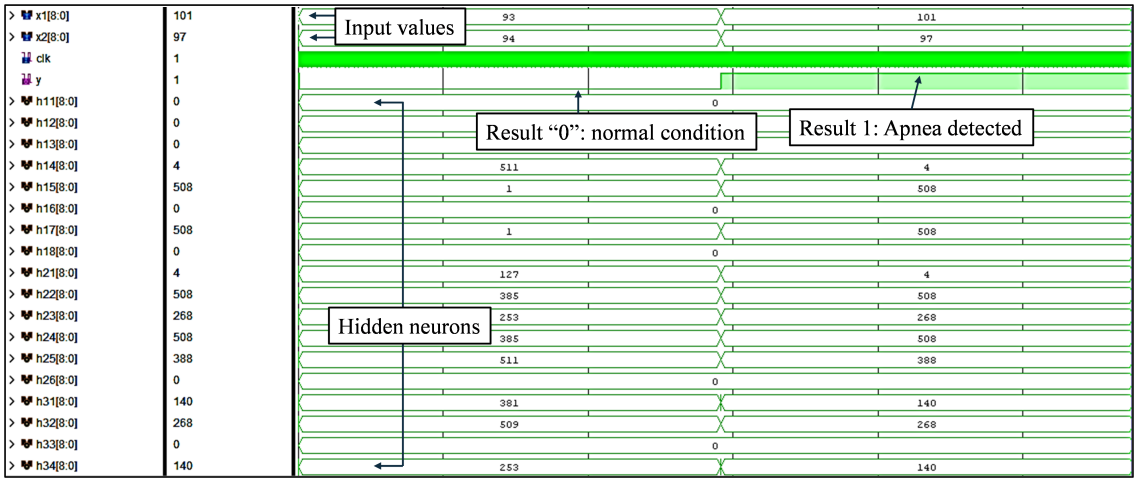


Figure 5.13: Simulation test result demonstrating the hardware prediction of SA detection (1: when an apneic event occurs and 0: normal condition) using an unseen test data set.

In Fig. 5.14, according to the eight attributes, if the glucose level is around 140, blood pressure is near 74.5, the insulin level is over 169.5, and BMI rate is near 34.3, then the patient aged over 35 is considered to be diabetic [17].

5.4 Conclusion

Applying deep learning models (DL) such as DNNs in medical diagnosis is becoming increasingly popular and is showing promising results. However, as DNN delivers highly accurate prediction results and diagnosis, it still requires high-end computational processors, which require cloud services that do not provide a cost-effective solution. In contrast, techniques enabling power-efficient, cost-effective solutions must sacrifice performance accuracy, which is not beneficial in critical diagnosis,

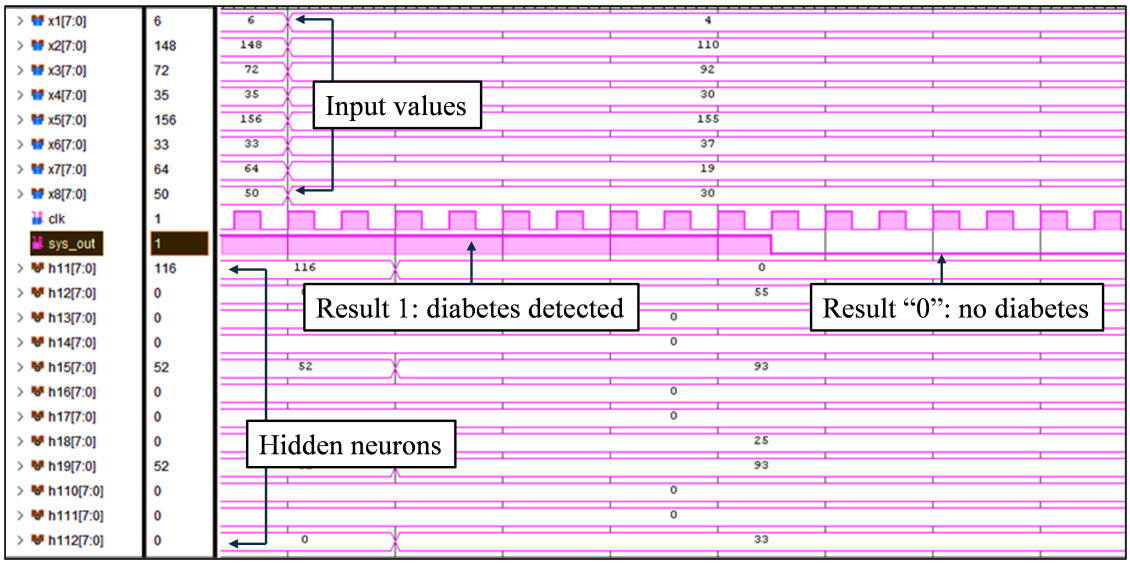


Figure 5.14: Simulation test result demonstrating the hardware prediction of diabetes prediction (1: diabetes predicted and 0: normal condition) by using unseen test data set.

especially in medical applications. In overcoming power consumption issues, the DeepSAC method introduces power-efficient deep neural network models for digital hardware. However, this technique is applicable in post-trained neural networks as the weights are irreplaceable after the inference on hardware. The significant advantages of using the DeepSAC method are reducing memory access due to minimal parametric (weights and biases) storage and the absence of multipliers in the neuron units providing a low power consumption rate. This method shows promising results and accurate testbench simulations when validated using two widely used medical data sets. Alongside this, many digital logic parameters are implemented and is bench-marked using 3-hidden layer FNN models. In the future, this design technique will be employed in developing and designing integrated circuits on the CMOS platform, creating opportunities for fabricating smart wearable biomedical devices.

Chapter 6

SABINN: SHIFT ACCUMULATE BASED BINARIZED NEURAL NETWORK

6.1 Introduction

A power-efficient hardware model called SABiNN: Shift Accumulate Based Binarized Neural Network has been developed in this work that converts all the hyper-parameters of the network, such as weights and their biases, into binary values (between +1 and -1). The SABiNN model is inspired by Matthieu Courbariaux, Yoshua Bengio, and his team's [74] newly developed Binarized Neural Network (BiNN) model. SABiNN significantly reduces the power consumption rate of the model and decreases the percentage of resource utilization when embedded onto re-programmable hardware. On-chip memories, such as SRAMs and DRAMs, were not used during inference as each weights and biases of the network are fixed binary value.

This chapter focuses on implementing and experimenting with the SABiNN model in a proposed sleep apnea detection system. Firstly, the model is embedded onto re-programmable general-purpose hardware such as Nexys Artix-7 FPGA and later integrated onto 130 nm and 180 nm CMOS platforms for low power and fast processing.

6.2 Design Scheme

Currently, there are a variety of accelerators and processors available which are used in training the AI/ML models. The matrix multiplication function is the core component and the basic computation of the machine learning-based hardware accelerators. Due to their parallel computational scheme GPUs are widely used for their processing and calculation efficiency over traditional CPUs. ASIC-based AI-accelerators, designed solely for AI/ML training, are being developed to increase efficiency

in model training, validation, and embedding. An AI/ML-based hardware accelerator such as Google TPU's single-core architecture is shown in Fig 6.1 that illustrates the training and computation mechanism [83].

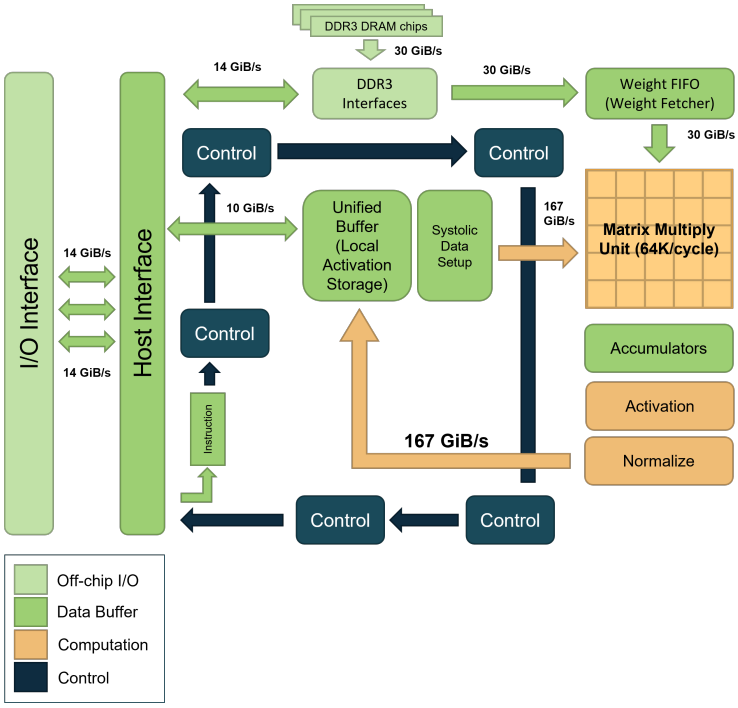


Figure 6.1: Google TPU block Diagram: System Architecture.

According to Fig 6.1, the matrix multiplication unit takes data from the input and multiplies it in a first in, first out (FIFO) manner with its associated weights stored in DRAM chips. After multiplication, the multiplied resultant is accumulated and classified according to its activation function. Later, it is normalized (according to the model specificity), and is sent via the buffer for back-propagation and weight update purposes.

One of the fastest ASIC-based hardware accelerators in the market is Google TPUs, which runs on 70 MHz frequency with a clock period of 700ns. The specifications of a single google TPU core are listed in Table 6.1. But due to its high-speed computational power and maintenance, it is one of the most expensive accelerators in the market. Table 6.2 shows the current price range (2022) of a google TPU [83].

It can be concluded from Table 6.2 that the accelerator chip fabricated on an ASIC platform is highly expensive. As a result, most system developers and scientists still use CPUs and GPUs to run machine-learning models. Thus, size reduction and accessibility remain to be important issues as CPUs and GPUs take up much space, time, and power. Therefore it is more than necessary to

Table 6.1: Google TPU Signal Core Specification

Parameter	Unit
Diameter	331mm^2
Frequency	70mHz
Clock Period	700ns
Supply Voltage	1.8V

Table 6.2: Google TPU Pricing as of 2022

Cloud TPU POD	Price/hr	1-yr commitment	3-yr commitment
32-core Pod slice	$24 USD	$132,451 USD	$283,824 USD
128-core Pod slice	$96 USD	$529,805 USD	$1,135,296 USD
256-core Pod slice	$192 USD	$1,059,610 USD	$2,270,592 USD
512-core Pod slice	$384 USD	$2119,219 USD	$4541,184 USD

develop power-efficient ways to reduce the pricing and size and introduce alternative ways to create on-chip acceleration and classification.

It is evident from the study that due to multipliers consuming the highest power and the most extensive area, the target was to replace the multiplier block with a shifter. The DeepSAC module eliminated the issue and completely removed the matrix multiplication section of the hardware by introducing shifter-based multipliers. The target for SABiNN is converting all the weight values into 1-bit binarized values. An illustration of the extraction and conversion process of the weight values into binarized values of +1 and -1 from a trained FNN model is shown in Fig 6.2 below.

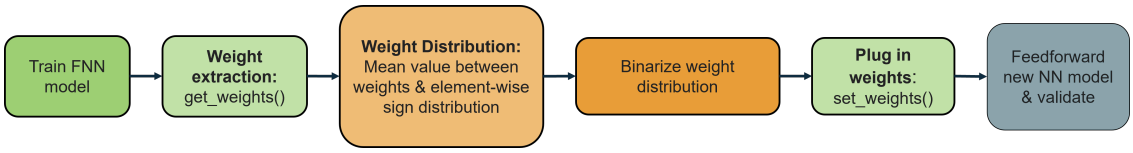


Figure 6.2: Extraction and conversion process of weights transformed into binarized weights.

To create the SABiNN module, first, a targeted NN model was trained, in this case, a feedforward model, and all its weights and biases were extracted. To convert them into binary values, element-wise weight distribution was executed by calculating the mean value of the weights from each layer. By taking the mean value of each hidden layer, binarization of +1 and -1 and distribution was done among the weights. After a successful conversion, it was plugged into the original NN model, and a forward pass was executed to check if the accuracy was maintained. Upon gaining acceptable accuracy, the model parameters and architecture are extracted from the software and used for hardware implementation. Otherwise, the FNN model is re-designed, discarding the binarized value

to avoid vanishing gradients when executing backpropagation.

6.3 SABiNN for Sleep Apnea Detection

The proposed sleep apnea detection system illustrated in Fig 2.1 from **Chapter 2** takes in pre-processed and normalized R-R intervals from the ECG signal and oxygen saturation level data from SpO_2 signal. Then the input data is classified between apneic and normal conditions. Machine-learning-based FNN module executes the classification of the input data and results output of "0" and "1". The FNN structure requires a high memory size and consumes significant energy in its MAC operation. Due to this limitation, most diagnostic systems avoid using a machine-learning-based model in real-time detection and only use it in computer simulation and cloud-based applications, especially during data processing and analysis. This dissertation aims to design and infer the FNN model on hardware that requires near-zero memory storage for weights and biases and consumes low power by replacing the multiplier component with an alternative logic block in the MAC operation. Thus, the proposed energy-efficient SABiNN module is used in the decision-making section of the system. Fig. 6.3 showcases the newly developed sleep apnea detection system.

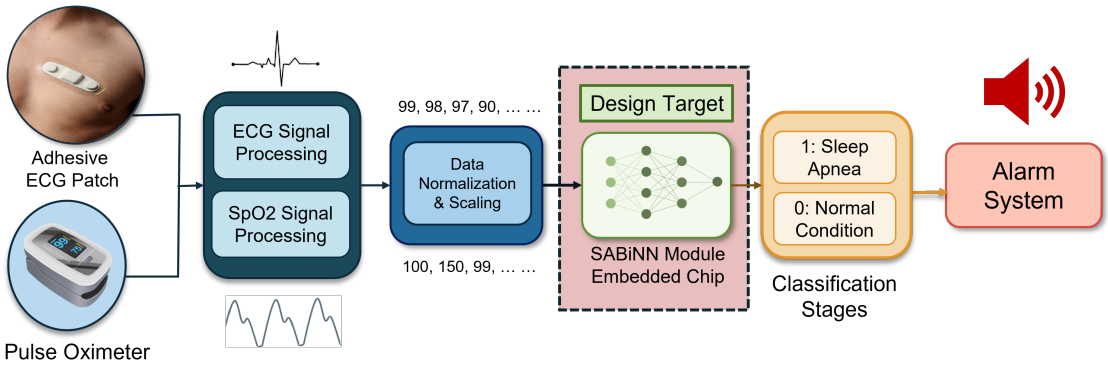


Figure 6.3: SABiNN module embedded proposed sleep apnea detection block diagram.

According to [74], a significant reduction is observed in the memory size when binarizing the weights in +1 or -1. This is due to the arithmetic operations being replaced with bit-wise operations. Besides, it reduces power consumption rate, contributing to compact design architecture. In this work [74], an NN with binarized weights and activations was trained and benchmarked on MNIST, CIFAR-10, and SVHN datasets which produced near state-of-the-art results. A recent work [84] introduced XNOR gate in a parallel computation scheme where both the weights and the input values were binarized. This drastically reduced the memory size and showcased high energy efficiency. However, these models were benchmarked with image datasets. Images can be easily transformed

into their histogram version where the pixel values are in binary "-1/0" and "+1" forms. The binary operation applied in convolution is shown in Fig. 6.4.

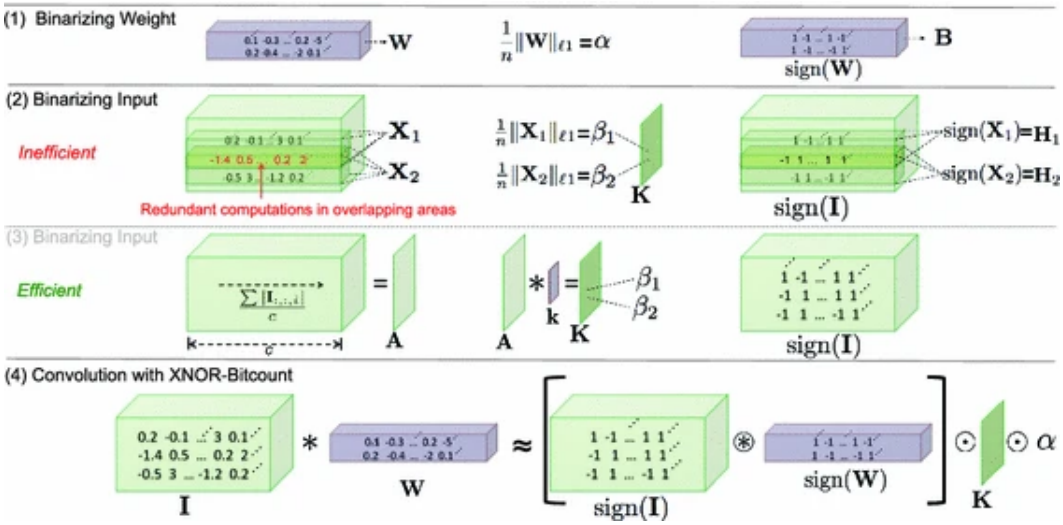


Figure 6.4: Approximation of a convolution using binary operations. The input data are converted into binary and multiplied with weights of +1 and 1. W = weights, X1 and X2 are binarized input values, sign() = sign activation function, K = filter [84].

But physiological signals generated from biosensors cannot be translated into the binary format as a significant amount of data will be lost due to their 1-dimensional nature. Thus, this binarization method [74, 84] results in a low accuracy rate even if it is power-efficient. Considering this, the post-training SABiNN model is developed where the binarization technique is used only on the hyperparameters instead of the input values [31], and the input values are quantized into n-bit integers instead of 1-bit binary values. A significant reduction in memory and power is observed when the data is forward passed through the proposed network. Instead of using traditional sigmoid function at the output node as its activation function [74], the sign function was used post-training for balanced distribution shown in Fig. 6.5

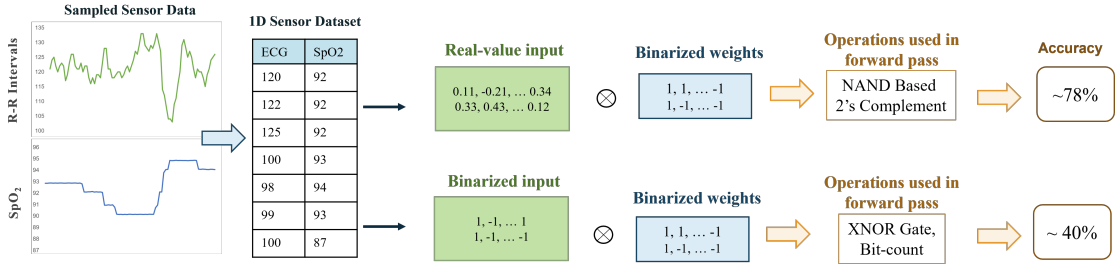


Figure 6.5: Comparative study between quantized input and binary input values. The input data are multiplied with weights of +1 and 1. By using NAND-based 2s compliment, the accuracy rate is achieved around 70% whereas using binary input and hyperparameters then, the accuracy degrades to 40%.

6.3.1 Software Simulation Results

Table 6.3 shows that the accuracy level between FNN, DeepSAC, and SABiNN are similar based on the evaluation metrics. Thus, no data loss during weight binarization was observed upon verifying the model accuracy. The SABiNN module is a 3-hidden layer-based neural network with ReLU as its hidden layer activation function and hard sigmoid (sign) function as its output layer activation function. Fig 6.6 showcases a graphical representation of the 3-hidden layer neural network model embedded in re-configurable hardware (Artix Nexys-7 FPGA).

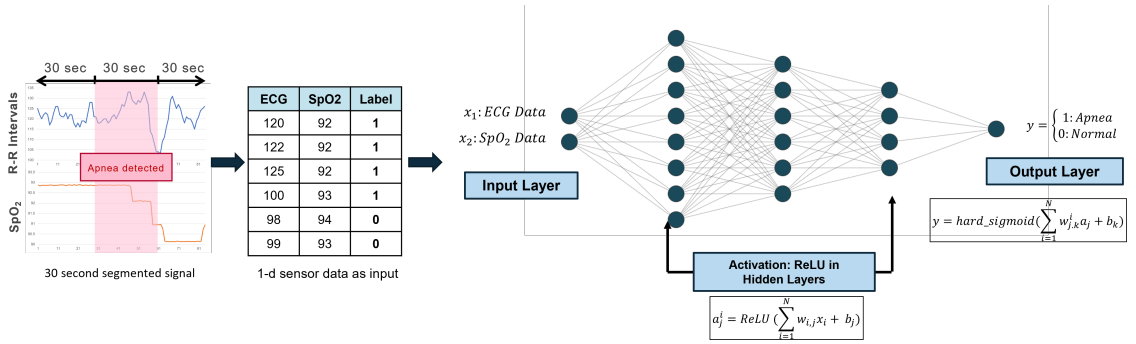


Figure 6.6: Construction of a 3-layer 2-(8-6-4)-1 SABiNN model with two 30-second segmented 1-dimensional sensor inputs (x_1: R-R interval and x_2: SpO_2) generated from ECG patch and pulse oximeter. The hidden layer consists of ReLU activation function, and the hard-sigmoid is the output layer activation function.

Table 6.3: Performance Evaluation Metrics Between FNN, DeepSAC, and SABiNN

Parameter	FNN	DeepSAC	SABiNN
Accuracy %	80	78	77
Precision %	77	77	71
Recall %	78	77	73
F1-Score %	84	81	81

6.3.2 Experiments on Re-programmable Hardware

A re-programmable hardware prototype is developed to test and validate the proposed model. The reconfigurable hardware setup calculates the power consumption analysis and the resource utilization percentage. For training and validating the hardware model, sleep apnea data was collected from the PhysioNET bank, and for added versatility, two data sets were combined. One was collected from the Philips University Medical Center, and the other from St. Vincent's Hospital [40, 41]. All patients were aged over 18 years and each patient had sleep data for over 8-10 hours.

SABiNN Synapse-Neuron Design

According to Fig 6.7, in the synapse part of the design, for -1 valued weights, the input values x[n] are complemented using 2's complement, and for +1 valued weights, the x[n] values are directly forward passed. All the on-passed or completed values s[n] accumulate with an adder and then are sent to its associated activation function f for classification. If bias is included in the model, the bias bi[0:n] gets added to the accumulated value si[0:n].

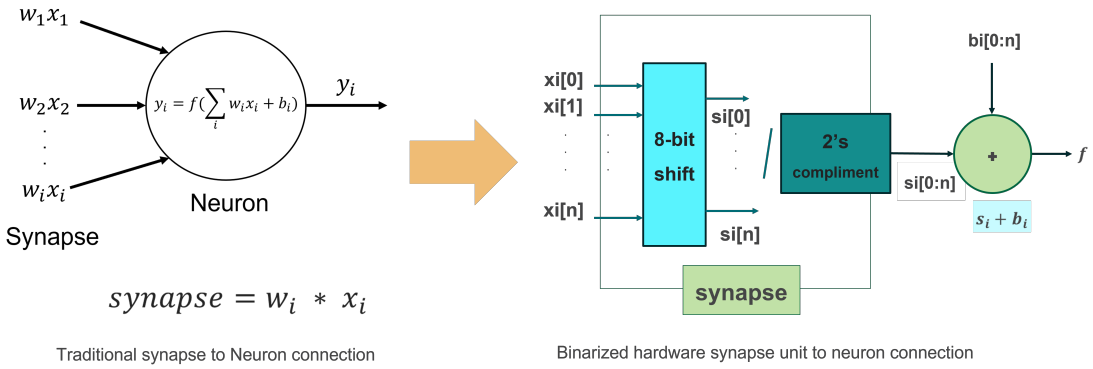


Figure 6.7: Binarized synapse-neuron connection converted from the traditional synapse-neuron connection.

Results

Two types of power consumption analysis are performed to properly evaluate both DeepSAC and SABiNN models. One is a software-based power consumption analysis shown in Table 6.4, and the other is when the proposed design was physically embedded in the selected FPGA hardware as shown in Table 6.5. Both the models are optimized and quantized before embedding them onto digital hardware. According to Tables 6.4 and 6.5, between the DeepSAC and SABiNN modules, the power consumption rate of the SABiNN module is significantly reduced by 9x times.

Table 6.4: Power Consumption Analysis: Vivado HLx Software Simulation

Parameter	DeepSAC	SABiNN
Signal	3.288	1.446
I/O Ports	3.380	0.117
Logic Block	0.709	1.792
Dynamic	7.668	4.015
Static	0.308	0.116
Total	7.976	4.131
Thermal Margin	47	12.1

Table 6.5: Power Consumption Analysis: Nexys Artix-7 FPGA Embedded Simulation

Parameters	DeepSAC	SABiNN
Signals	2.364	1.736
I/O Ports	2.781	2.459
Logic Block	8.733	0.068
Dynamic	13.879	4.263
Static	0.308	0.105
Total	14.186	4.368
Thermal Margin	47	11.4

Table 6.6 is a comparative study of resource utilization between DeepSAC and SABiNN models. Based on the results of Table 6.6, it could be identified that the SABiNN module also significantly reduced the component number by 4x times, and the maximum temperature it dissipates is 44°C which eliminates the need for extra cooling sink during processing.

Table 6.6: Resource Utilization on Nexys Artix-7 FPGA

Parameters	DeepSAC	SABiNN
LUTs	114	108
I/O Ports	29	18
Buffer	1	1
Registers	171	7
Flipflops	60	33
Highest Operating Temperature	89.7°C	44.3°C

As described, the multiply-accumulate (MAC) unit of the neural network plays a significant role in consuming the maximum percentage of power. It also uses a larger part of the available hardware resources. Fig. 6.8 illustrates the hardware resource utilization percentage between multiply, shifter, and binarized accumulation. According to the bar chart MAC and SAC introduce a digital signal processing unit (DSP) for the multiplication unit whereas BAC has no DSP unit.

According to Table 6.7, the total power of the MAC unit is 3x times higher than the binarized accumulator (BAC) and 4x times higher than the shift accumulator (SAC).

Based on the data type, system design, and accuracy requirement, the design of SAC and BAC goes hand in hand. Some applications, such as [17] and [85], utilize shift-accumulate-based neuron design, whereas in [31] binarized neuron design is more beneficial.

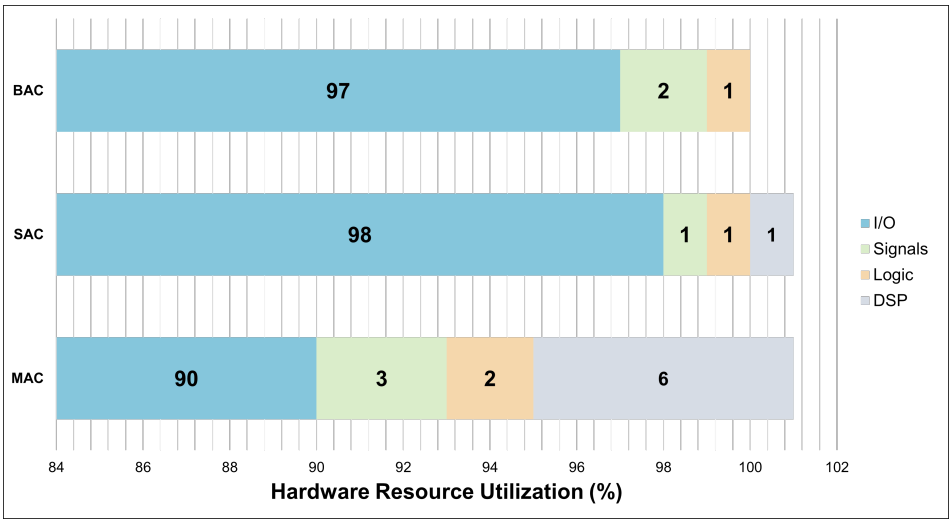


Figure 6.8: Hardware resource utilization percentage between Multiply-Accumulate (MAC), Shift-Accumulate (SAC), and Binarized Accumulate (BAC).

Table 6.7: Power Consumption Rate Between MAC, SAC, and BAC

Parameters	**MAC 8-bit**	**MAC 16-bit**	**SAC 8-bit**	**SAC 16-bit**	**BAC 8-bit**	**BAC 16-bit**
Signals (W)	0.434	1.047	0.56	0.117	0.098	0.198
I/O Ports (W)	13.885	30.402	3.804	7.604	5.942	12.399
Logic (W)	0.139	0.475	0.035	0.064	0.059	0.120
DSP (W)	0.933	1.490	0	0.053	0	0
Dynamic (W)	15.391	33.414	3.895	7.838	6.009	12.717
Total Power (W)	15.66	33.946	3.985	7.953	6.201	12.903

Performance

Fig 6.9 shows the physical instrumentation of the FPGA hardware setup with its associated connections. Since actual sensors are not integrated, such as an ECG patch and pulse oximeter, to measure the heart rate and blood oxygen saturation level, the computer acted as a dummy sensor module connected to the FPGA board via USB/UART port. For validating and testing the whole system, a test bench is generated from the data set collected from PhysioNET [19, 40], and a 7-segment display is used to showcase the results. Whenever apnea is detected, it shows "1" in its LSB (Least-significant bit) unit, and in normal conditions, it shows "0".

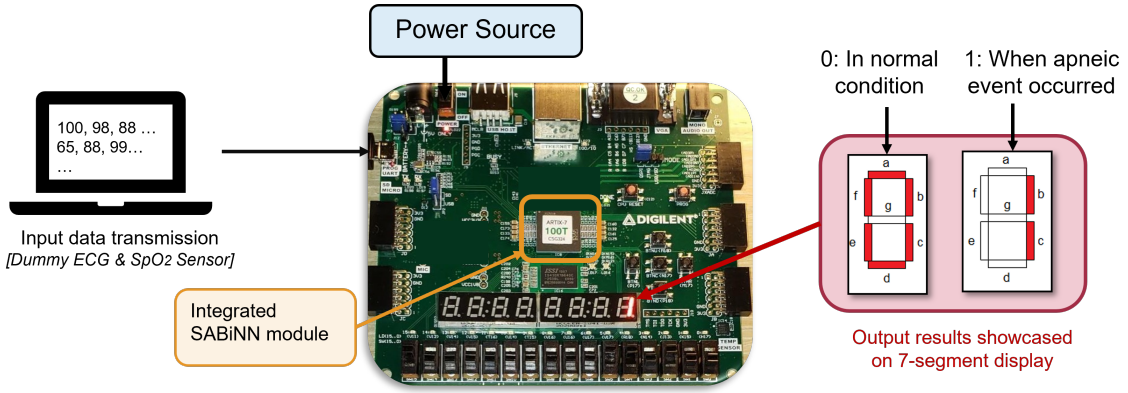


Figure 6.9: Physical implementation of the SABiNN module integrated onto a general purpose Nexys Artix-7 series FPGA with the computer acting as dummy sensors of the ECG patch and SpO_2 Sensor (pulse oximeter) and a 7-segment display showcasing the output result.

The processing speed of an 8-bit SABiNN module in FPGA hardware is around 10 ms. The test bench generation wave is executed on Vivado HLx software, shown in Fig 6.10. According to Fig. 6.10, in a 10 ms time interval, the module detected apneic and non-apneic occurrences. x1[7:0] and x2[7:0] are the input ports for ECG and SpO_2 signals, respectively. "clk" is the clock period embedded onto the design, and y is the 1-bit output port. From the test dataset, according to the Sleepcare foundation, if the RR interval from the ECG signal is over 99% and SpO_2 values are less than 93%, then that patient underwent apneic event. In the testbench results, the model detected possible apneic cases when the ECG signal is over 99%, and SpO_2 value is less than 95%. Furthermore, the model showed normal conditions when the RR interval was below 100%, and SpO_2 was over 90%. In realistic scenarios human biophysical data varies from person to person. Events when the R-R interval is high even though the blood oxygen saturation is normal (in the range over 93%) is also considered an apneic case. According to the testbench simulation shown in Fig. 6.10, the system successfully detected apneic events when the R-R intervals were over 99 even though the

blood saturation level was normal. From the measurement results, it can be concluded that SABiNN can be successfully deployed on CMOS with minimal area requirements.

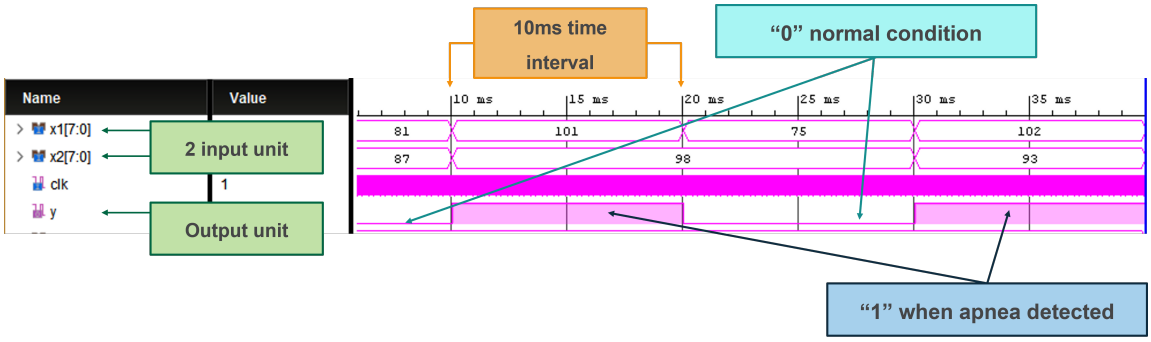


Figure 6.10: Test-bench simulation test demonstrating the hardware prediction of SA detection (1: when an apneic event occurs and 0: normal condition) using the unseen test dataset.

After properly validating and testing the SABiNN model, the proposed hardware model is translated onto the ASIC platform for low-power calculation. For on-chip processing and classification, the digital hardware module is designed onto both the 130 nm CMOS process provided by Google-Skywater [86] and a 180 nm commercial CMOS process.

6.3.3 Experiments on CMOS Platform

The goal of developing the SABiNN model is to design a low-cost, low-power, and high-precision wearable healthcare device that can run for 8-10 hours straight while enabling real-time automatic detection. Thus, customization and miniaturization are highly important to design the SA detection device successfully. This section documents two implementations of the SABiNN model on two different CMOS process design kits (130 nm and 180 nm). A basic multi-layer perceptron (MLP) is designed and fabricated on 180 nm.

Experimental Results: Multi-Layer-Perception (MLP)

A multilayer perception (MLP) is a fundamental component that makes up a neural network. It has three layers: input, hidden, and output. It is a fully connected class of feedforward artificial neural networks. Fig. 6.11 (a) showcases a 3-layer MLP with two inputs and one output, and (b) showcases an ASIC implementation of a binarized MLP with the same design architecture.

According to the SABiNN architecture design, MAC units are replaced with BAC. When weights are "-1", a 2s complement is introduced, and the input value gets flipped. When weights are "+1," the input values are forward passed without any 2s complements. Lastly, the weights are accumulated

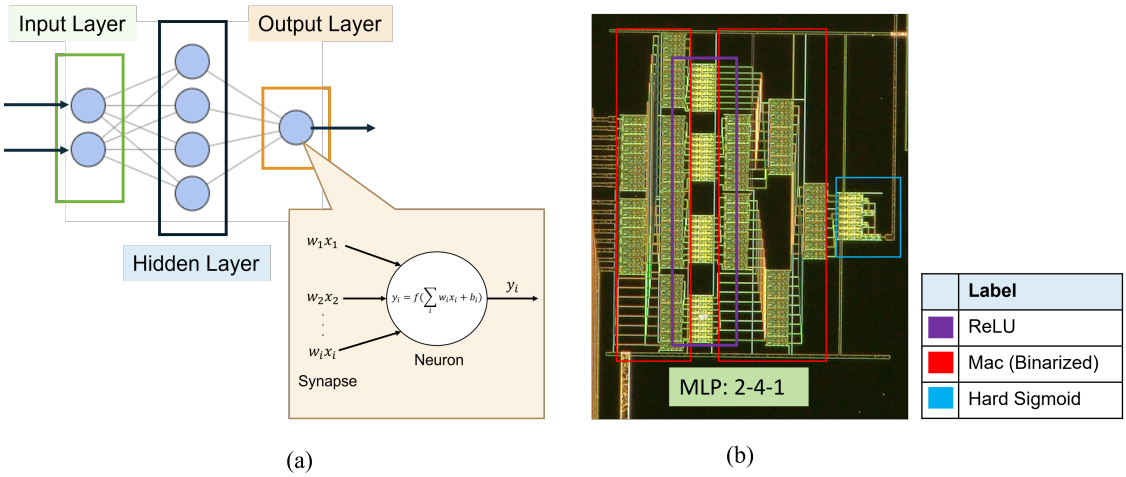


Figure 6.11: Illustration of a baseline MLP model (a) graphical view of a typical fully connected 3-layer 2-(4)-1 MLP (b) CMOS layout of 3-layer 2-(4)-1 binarized MLP with the rectified linear unit as its hidden layer activation function and hard sigmoid (sign) as its output activation function. The red label indicates the binarized MAC unit (Adder and 2s compliment), the purple indicates the ReLU function, and the blue indicates the hard sigmoid function.

using adders and fed into their associated activation functions. The indirect multiplication of weights "1" and "-1" can be represented as XNOR gates in hardware implementation. However, in CMOS ASIC design, the XNOR gate results in lower precision and higher noise-induced signal, contributing to significant accuracy degradation when utilizing a dense number of XNOR gates in neuron-synapse connection. When an XNOR gate is implemented in CMOS, a significant 279 mV to 456 mV spike is observed on the XNOR gate. This spike repeatedly occurred when input **A** flipped the bit from "1" to "0" and input **B** flipped the bit from "0" to "1". For 1-bit XNOR gate this can be overlooked however when designing 8-bit, 16-bit neuron-synapse unit on digital hardware platforms, these voltage spikes will significantly contribute to the resultant value. As a result, this will reduce the precision rate of the overall model classification. Therefore, a NAND-based XNOR gate is designed on the BAC unit to ensure clean voltage reading and reduce voltage spikes, achieving a higher precision rate and clean signal generation. Transient analysis of XNOR and NAND-based XNOR gate is presented in Fig. 6.12 (a)-(b), indicating the voltage spikes in the XNOR gate.

The SABiNN-based MLP layout with its integrated 5mm^2 pad frame is shown in Fig 6.13. The total area of the MLP model resulted in around 0.327 um^2. The SABiNN-MLP chip has an input voltage of 5 V. From the layout image, VDD is the input DC voltage source, GND is the ground connection, x_1 and x_2 are 8-bit input signals, and output is a 1-bit output signal.

Transient analysis is done to calculate the speed of the proposed MLP; according to Fig. 6.14, the rise-fall time of each edge was around 1 ns with an input pulse voltage of 5 V with a 10 ns

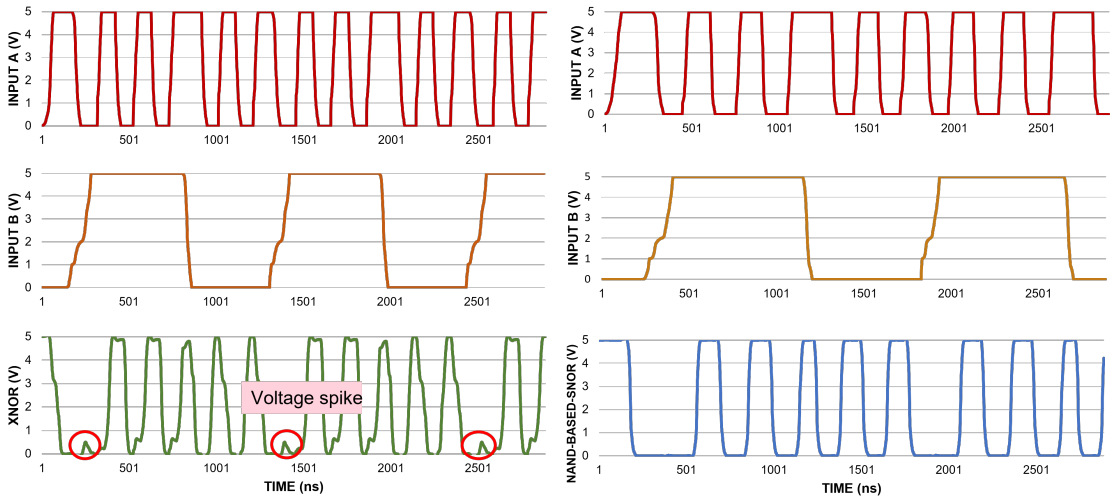


Figure 6.12: Comparative study of the output values during transient analysis between (a) XNOR gate and (n) NAND-based XNOR gate. In the XNOR gate, (-279 mV to 456 mV) voltage spike is observed during bit flipping in transient analysis indicated at (b), where the NAND-based XNOR gate resulted in a much cleaner signal.

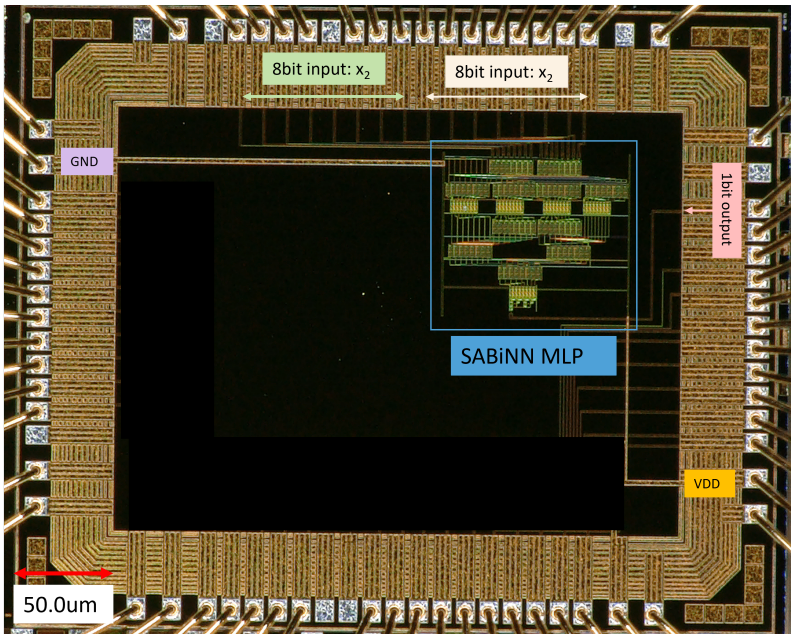


Figure 6.13: Layout image of the SABiNN MLP chip inside a 5 mm^2 pad frame where VDD = DC voltage source (1.8 V), x_1 and x_2 are 8-bit inputs, the output is a 1-bit output. The entire chip was designed on a 180 nm CMOS process.

period. The total power consumption rate of the MLP model is estimated to be around 44 mW. The SABiNN-MLP characteristics are shown in Table 6.7

Table 6.8: SABiNN-MLP Chip Characteristics

Parameters	**Unit**
MLP	2-4-1
Supply Voltage (Vdc)	5V
Period	10ns
Total Area	0.327 μ m**2**
Power Consumption	44mW
Total Energy	44.61uJ

Table 6.8 showcases a comparative study between the simulated and experimental results of the SABiNN-MLP chip and the software baseline MLP model, which uses MAC units between synapse-neuron connections.

Design Space Exploration: SABiNN Layout

Two SA detection prototypes are created that successfully detect apneic events. In the 180 nm process, a 3-hidden layer 8-bit quantized NN is developed, shown in Fig. 6.15; in the 130 nm process, a 4-hidden layer 16-bit quantized NN is developed in Fig 6.16. Both NN models follow the SABiNN design architecture.

Fig. 6.17 showcases a schematic view of the 3-hidden layer SABiNN model where a) is both the input and layer one with an eight-node connection, b) is layer 2 with six nodes, c) is layer 3 with four nodes, and d) are the output layer with one node and a sigmoid block at the end for classification.

Fig. 6.18 illustrates the layout model designed on 180 nm commercial CMOS process. The total design area took 9.5um^2, and each component showcases gate-level design.

Fig 6.19 (a) showcases a schematic view of the 4-hidden layer SABiNN model, (b) illustrates the physical layout of the model designed on 130 nm CMOS process and (c) showcases the caravel digital pad frame where the SABiNN model was integrated. The total area is around 0.16 mm^2, which utilized only 32% of the design area. Fig. 6.18 showcases the physical view of the SABiNN chip integrated into the 20mm^2 Google+Skywater pad frame with 10mm^2 design area. The entire model is implemented using digital synthesis on Vivado. The full GitHub repository can be found [87].

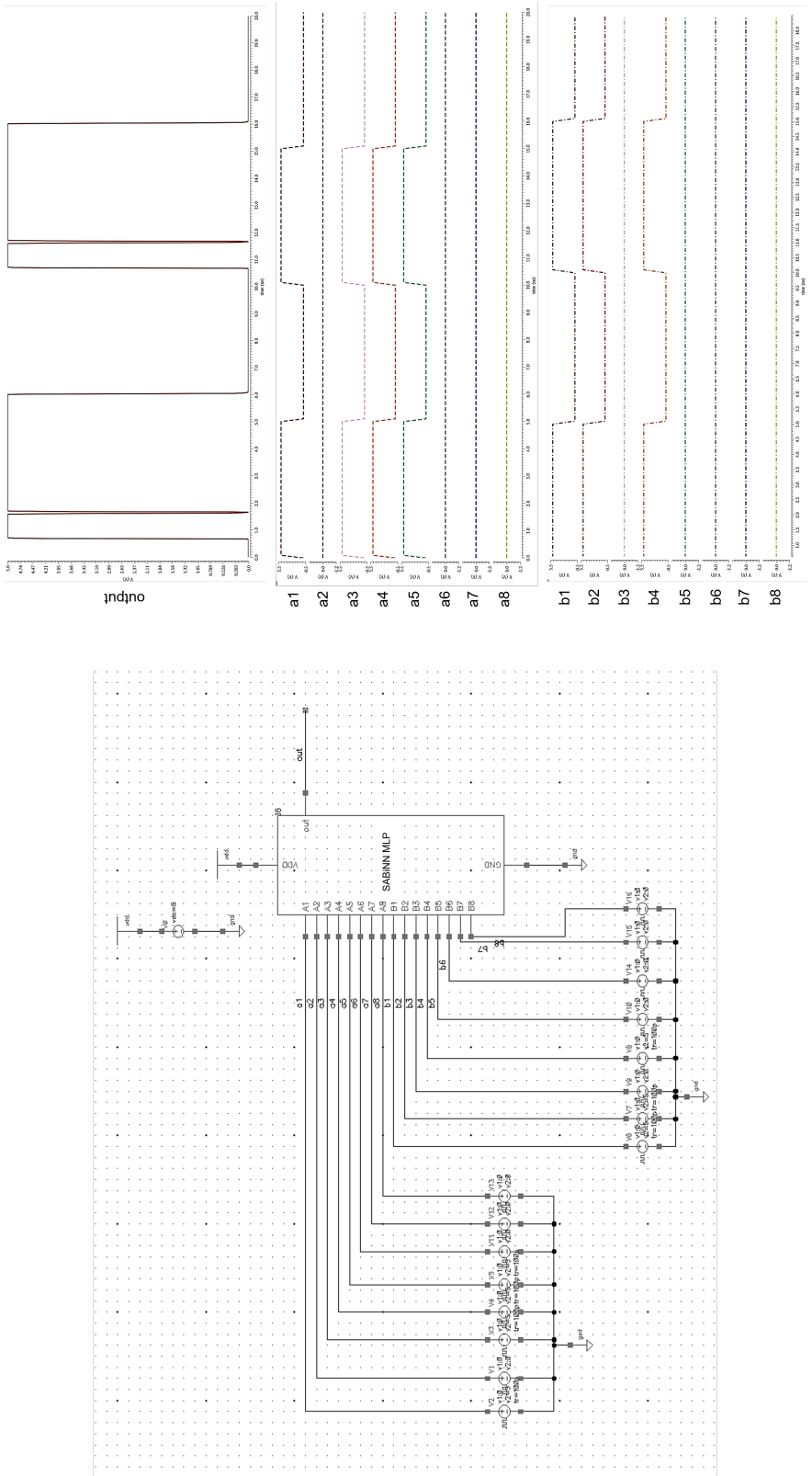


Figure 6.14: SABiNN-MLP test circuit and simulation results. $a_{(1\text{-}8)}$ and $b_{(1\text{-}8)}$ are input pulse voltage signals, and out is the output signal generated by the MLP circuit.

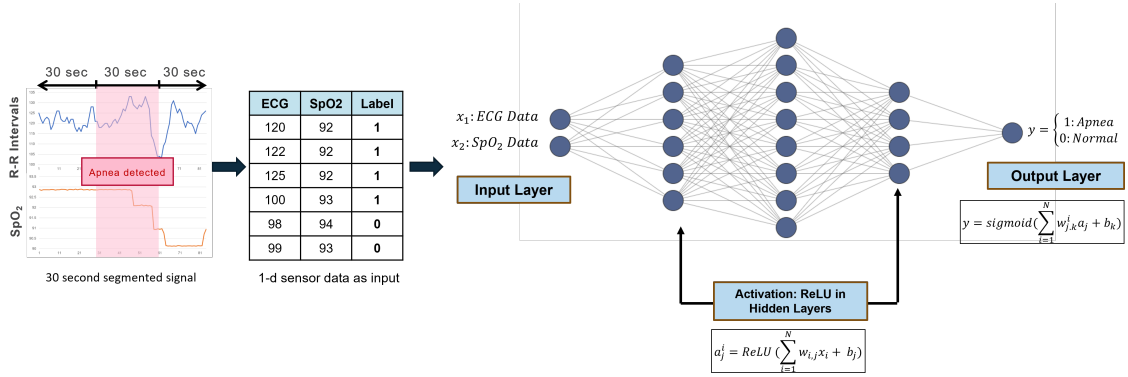


Figure 6.15: Construction of a 3-layer 2-(-6-8-4)-1 FNN model for 180nm PDK process with two 30-second segmented 1-dimensional sensor inputs (x1: R-R interval and x2: SpO_2) generated from ECG patch and pulse oximeter. The hidden layer consists of ReLU activation function and the sigmoid as the output layer activation function.

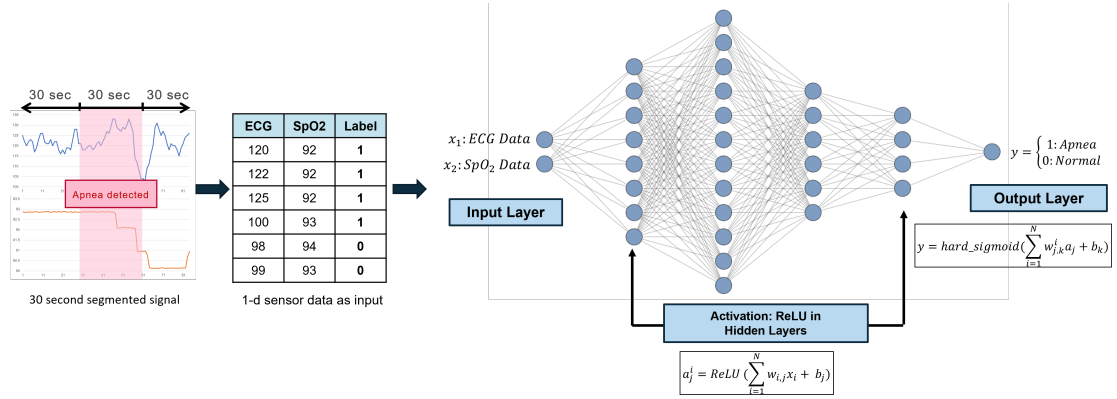


Figure 6.16: Design of a 4-layer 2-(8-12-6-4)-1 SABiNN model for 130nm PDK process with two 30-second segmented 1-dimensional sensor inputs (x1: R-R interval and x2: SpO_2) generated from ECG patch and pulse oximeter. The hidden layer consists of the ReLU activation function and sigmoid as the output layer activation function.

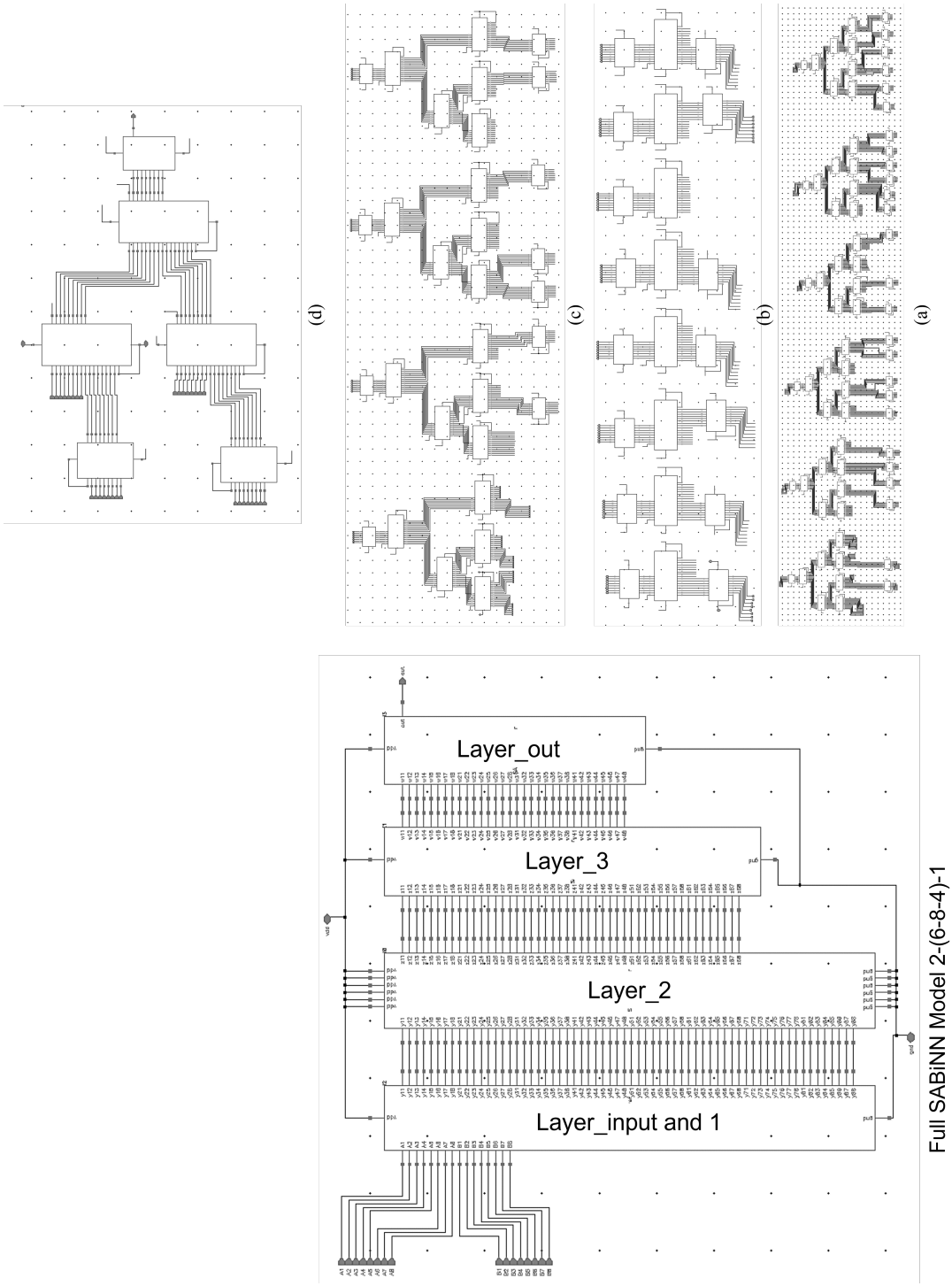


Figure 6.17: Schematic view of a full three hidden layer SABiNN model on 180nm. (a) is both the input and layer one with an eight-node connection, (b) is layer 2 with six nodes, (c) is layer 3 with four nodes, and (d) is the output layer with one node and a sigmoid block at the end for classification.

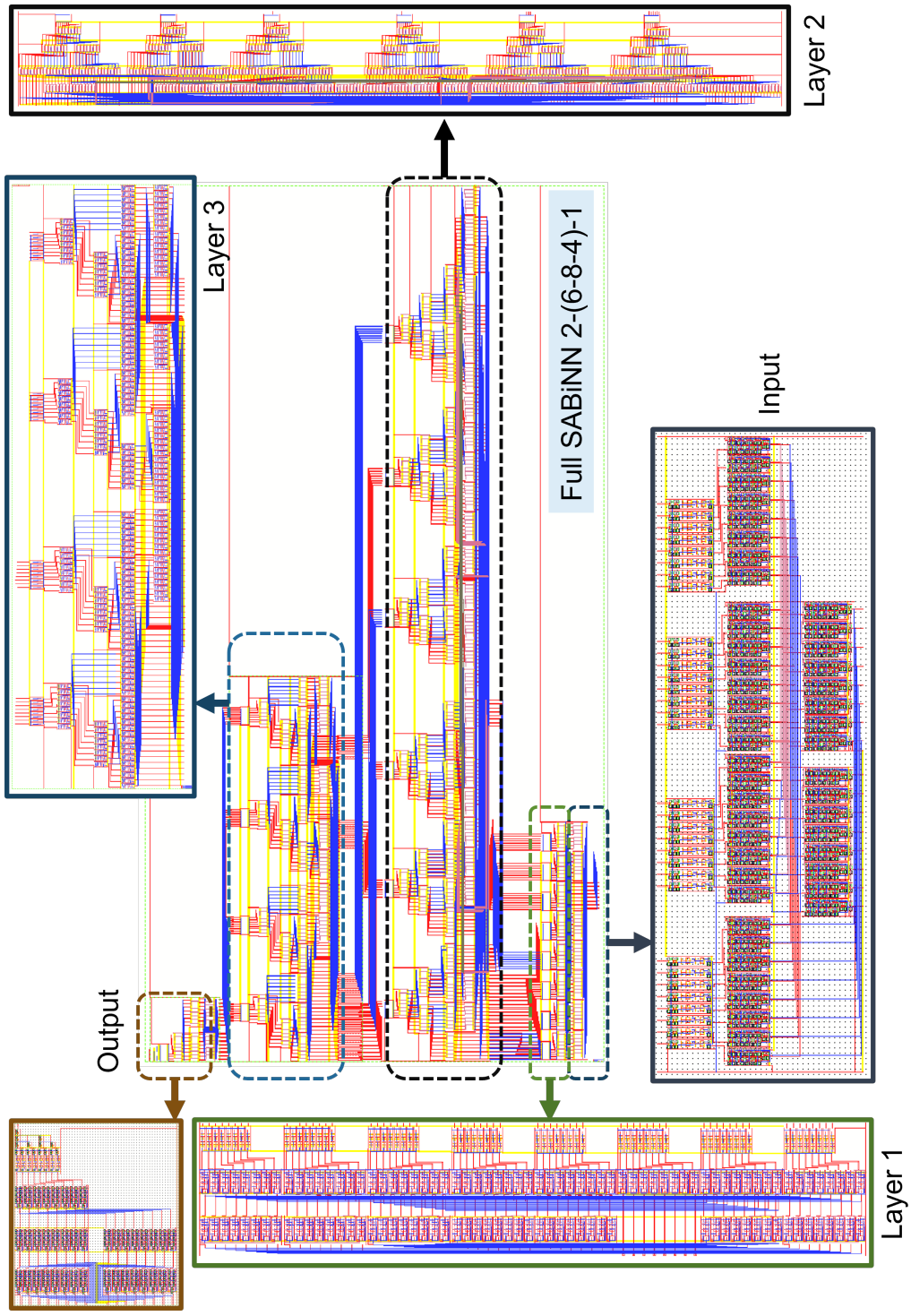


Figure 6.18: Full layout image of a 3-hidden layer SABiNN model.

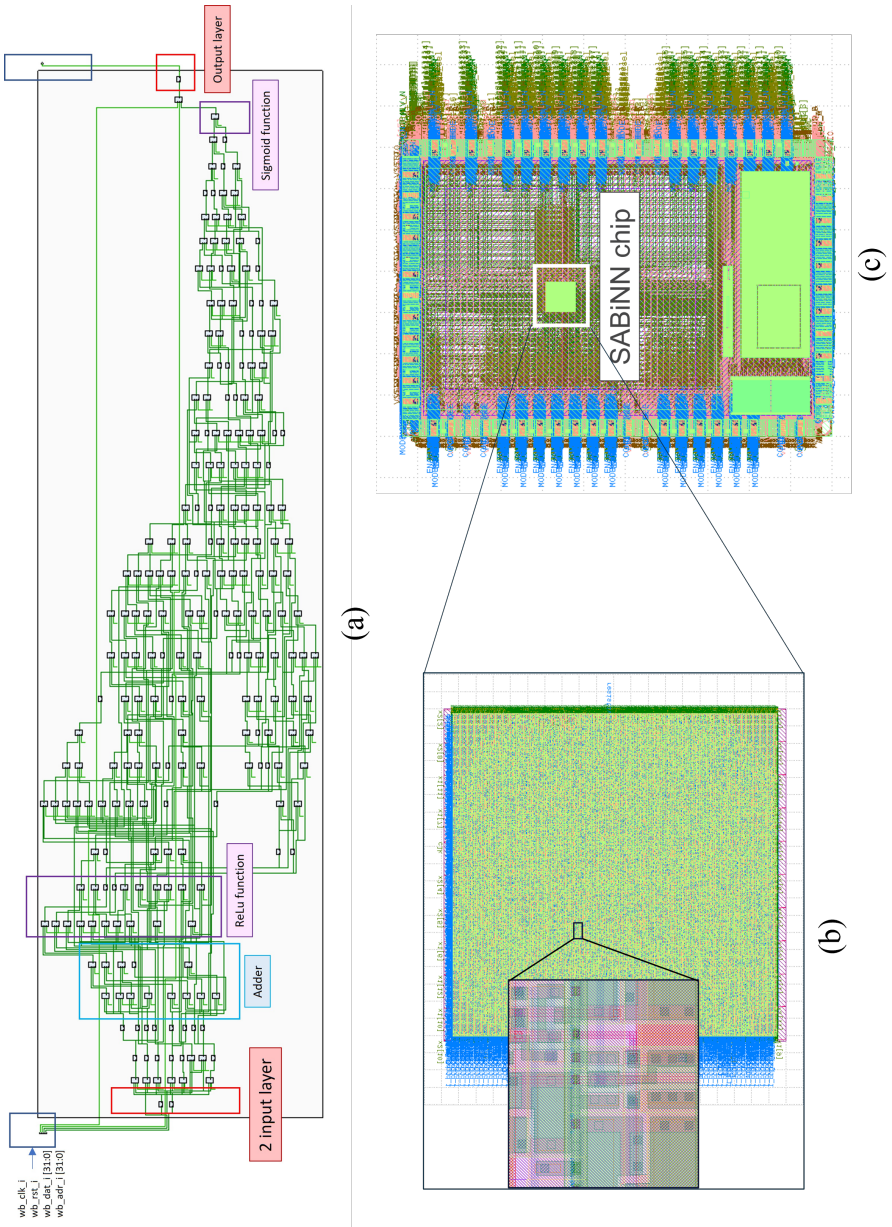


Figure 6.19: Full layout image of a 4-hidden layer 16-bit input SABiNN model with its schematic. (a) schematic view of the digital 4-hidden layer SABiNN model designed on Vivado HLx. (b) the digital layout of the synthesized SABiNN model and (c) 16-bit 4-hidden layer SABiNN model integrated onto Google+Skywater's caravel digital padframe.

SABiNN Result on CMOS Platform

Testbench simulation is executed on both designs to ensure the proper precision rate and speed. Fig. 6.20 showcases the transient analysis result of the 8-bit two-input signal and a 1-bit output signal. The entire design process was performed on the Cadence Virtuoso platform. According to the simulation results the rise time of each detection had a delay of 1.5 ns and the fall-time was 2 ns an input pulse voltage of 5 V with a 10 ns period was supplied to the input ports.

Fig. 6.21 showcases the digital simulation result of the 4-hidden layer SABiNN model with two 16-bit input ports, 1 "clk" port, 1 "rst" port, and 1-bit output port. [31].The characteristics of both chips are described in Table 6.8.

Table 6.9: SABiNN Chip Characteristics between 180nm and 130nm PDK

Parameters	**180 nm CMOS**	**130 nm CMOS**
FNN	2-(6-8-4)-1	2-(8-12-6-4)-1
Supply Voltage (Vdc)	5V	1.8V
Period	10ns	100ns
Total Area	9.5 μm^2	0.16 mm^2
Power Consumption	0.4W	10 μ W
Total Energy	3.98nJ	1nJ

6.4 Discussion

Inference of trained SABiNN model with a power consumption rate below 10 uW opens up the opportunities to infer deeper and denser neural network (NN) models with higher complexity and sensitivity rate. Currently, most NN models for classification are built on cloud platforms limiting on-chip prediction and analysis even though ensuring high accuracy but with no assurance of power efficiency [82,84,88–93]. In contrast with the FPGA schemes, neither ML nor NN models were utilized, and no power consumption rate was documented [94–96]. The proposed SABiNN architecture can optimize and infer deep neural networks (DNN) more straightforwardly while cutting the overhead cost. The CMOS implementation introduces future intelligent wearable devices with on-chip classification without the help of the cloud and servers, enabling higher security and lower latency of around 10 μs per prediction. SABiNN design architecture is the first to introduce CMOS implementation of an optimized binarized neural network for sleep apnea (SA) detection. Table 6.9 compares the proposed state-of-the-art methods of detecting sleep apnea with the 130 nm SABiNN model, demonstrating promising results. However, this technique is currently applicable in post-trained NNs as the weights

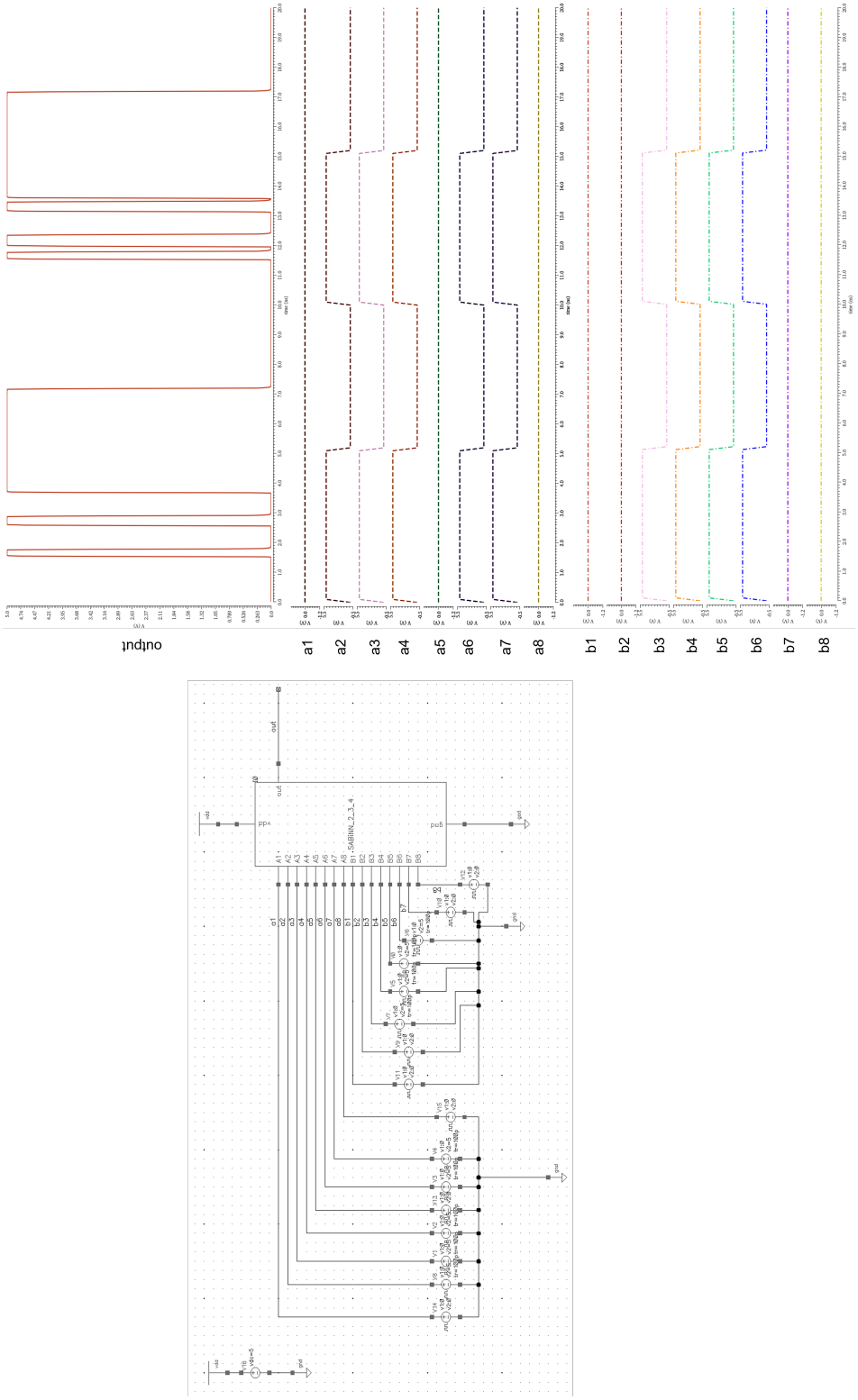


Figure 6.20: SABiNN on 180nm CMOS test circuit and simulation results. $a_{(1\text{-}8)}$ and $b_{(1\text{-}8)}$ are input pulse voltage signals, and out is the output signal generated by the MLP circuit.

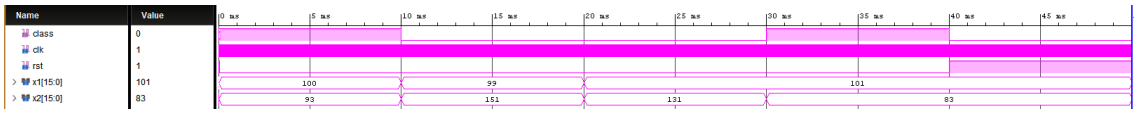


Figure 6.21: Test-bench simulation test demonstrating prediction of SA detection (1: when an apneic event occurs and 0: normal condition) using the unseen test dataset.

are fixed after the inference on hardware. But the significant advantage of using the proposed SABiNN method is the reduction of memory access due to the need for storing hyperparametric values and the absence of multipliers in neuron units. The proposed method showed promising results and accurate testbench simulations when validated using an open-source, widely used dataset.

Table 6.10: State-of-the-art Method in SA Detection

Year/Work	**2022** [97]	**2021** [98]	**2022** [99]	**2021** [100]	**2022** [101]	**This Work**
Platform	Real-Time	Wearable	Software	IoT	ASIC = 180nm	ASIC =130nm
Feature	Audio: Snoring	PSG: ECG	Nasal airflow: respiratory cycle, respiratory rate (RR), tidal volume (TV)	ECg and SpO_2	RR Interval, R-S Amplitude	SpO_2, RR Interval from ECG
Classifier	TCN	SVM	XGBoost	1D-CNN	SVM	SABiNN
Sample Window	1.04 sec	60 sec	5 sec	1 sec	10 sec	30 sec
Sensitivity	-	71%	83.82%	97.44%	83.85%	91%
Specificity	96%	88%	85.97%	-	85.58%	86%
Accuracy	96%	83%	83.76%	99.62%	84.60%	88%
Area	N/A	N/A	N/A	N/A	$0.429mm^2$	$0.16mm^2$
Power	N/A	N/A	N/A	N/A	0.46 μ W	10 μ W
Energy	N/A	N/A	N/A	N/A	0.46nJ	1pJ

6.5 Conclusion

The application of DL models, such as neural networks (NN), in medical diagnosis and monitoring, is becoming increasingly popular. However, as NN delivers highly accurate prediction diagnosis, it still requires high-end computational processors which leverage expensive cloud services. In contrast, techniques enabling energy-efficient, cost-effective solutions must sacrifice performance accuracy. To overcome such issues, the proposed SABiNN method is energy-efficient, resulting in 5mJ on general-purpose FPGAs and 1pJ on the CMOS platform. However, this technique is applicable in

post-trained NNs as the weights are fixed after the inference on hardware. But the major advantage of using the proposed SABiNN method is the reduction of memory access due to the lack of storing hyperparametric values and the absence of multipliers in neuron units. The proposed method showed promising results and accurate testbench simulations when validated using an open-source, widely used dataset. The success of this design technique can be further employed in developing a fully System-on-Chip (SoC) integrated biomedical system that could accurately detect and screen SA events with designated front-end sensors.

Chapter 7

BENCHMARK OF PROPOSED MODEL ARCHITECTURE

7.1 Introduction

Benchmarking a proposed model's architecture and it's algorithm is necessary to ensure versatility when developing a specific design architecture for various platforms. The term benchmarking is used for evaluating and comparing a proposed algorithm or method on widely used "benchmarked" models and datasets. This way, the specific architecture or method can learn patterns on various DL models using various datasets. In this research, the SABiNN architecture is benchmarked on popular DL models with widely used image datasets.

7.2 Model and Dataset Selection

Three popularly used DL models are selected for benchmarking the design architecture of SABiNN, where SABiNN is implemented on each model's "Dense" (Fully Connected) layers. These models are: Karen Simonyan and Andrew Zisserman's VGG19 [102], which is a very deep convolutional neural network, Microsoft's ResNET50 [103], which is a residual neural network, and Google's MobileNetV2 [104] which is used in developing embedded system and IoT applications requiring low area, power, and restricted resources. The two selected datasets for training and validation are: Cifar10 [105], which consists of various RGB images of animals, objects, plants, etc., and MedMNIST [106], a databank consisting of MNIST-like collection of standardized biomedical images that are 2-dimensional. This chapter briefly introduces each model architecture and the datasets and documents the evaluation results before and after implementing the SABiNN model architecture.

7.2.1 VGG19

Introduced by Sominyan and Zisserman [102], VGG19 is a deep convolutional network for large-scale image recognition. It is an improved version of Ciresan et al. (2011) and Krizhevsky et al. (2021) [102] ConvNet architecture by increasing the depth of the model architecture as shown in Fig. 7.1.

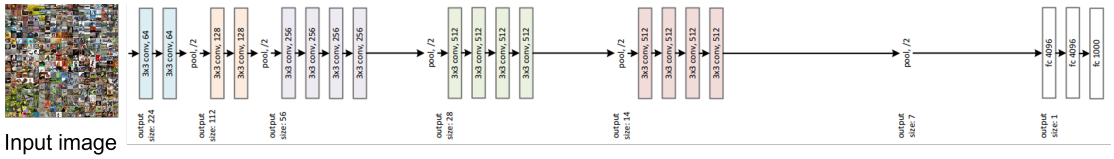


Figure 7.1: Example of a VGG19 model with 19.6 billion FLOPs [102].

During training, the input to the proposed ConvNet is a fixed-size 224x224 RGB image, where pre-processing is done only by subtracting the mean RGB. The in-depth structure and parameters of the full VGG19 model are shown in Table 7.1

VGG19 takes 32x32 or 224x224 RGB image sizes. The first convolutional layer is a 3x3 filter size with 64 kernels; then, through max-pooling, the next layer is a 128-kernel convolutional layer with another max-pool layer. Each convolution layer of the VGG19 model ends with a max pool layer, and the final layers end with a fully connected neural net. Lastly, classification is executed through an activation function of either softmax or sigmoid.

7.2.2 ResNet50

Developed by Kaiming He and his team at Microsoft Research [103], they introduced a residual learning framework that is easy to train and substantially deeper than previous DL models. It is a 50-layer residual network. The first 48 are the convolutional neural networks (CNN); the rest are dense or fully connected neural nets (FNN) with a max pool layer shown in Fig. 7.2. Activation functions like softmax and sigmoid are used as the classification layer.

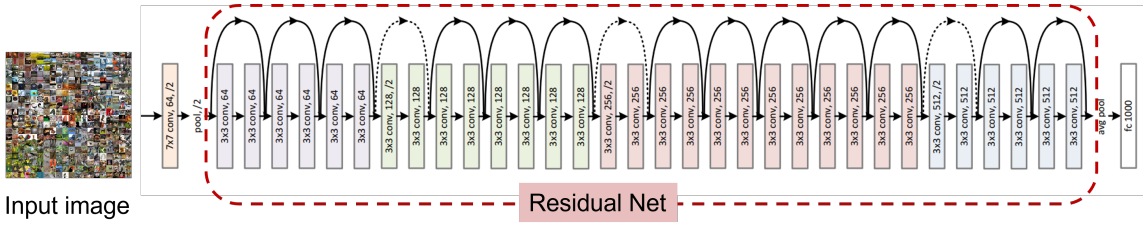


Figure 7.2: A residual network with 34 parameter layers (3.6 Billion FLOPS). The dotted shortcuts of the residual network are increased dimensions. The last layer is the feedforward layer, and the model is fed in with image dataset [103].

Table 7.1: Full stack VGG Model

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224x224) RGB Images					
conv3-64	conv3-64 LRN	conv3-64, conv3-64	conv3-64, conv3-64	conv3-64, conv3-64	conv3-64, conv3-64
maxpool					
conv3-128	conv3-128	conv3-128, conv3-128	conv3-128, conv3-128	conv3-128, conv3-128	conv3-128, conv3-128
maxpool					
conv3-256, conv3-256	conv3-256, conv3-256	conv3-256, conv3-256	conv3-256, conv3-256, conv1-256	conv3-256, conv3-256, conv3-256	conv3-256, conv3-256, conv3-256, conv3-256
maxpool					
conv3-512, conv3-512	conv3-512, conv3-512	conv3-512, conv3-512	conv3-512, conv3-512, conv1-512	conv3-512, conv3-512, conv3-512	conv3-512, conv3-512, conv3-512, conv3-512
maxpool					
conv3-512, conv3-512	conv3-512, conv3-512	conv3-512, conv3-512	conv3-512, conv3-512, conv1-512	conv3-512, conv3-512, conv3-512	conv3-512, conv3-512, conv3-512, conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max/sigmoid					

Traditionally in CNN if deeper the network the higher the chances of vanishing gradients. During back-propagation, the model's hyperparameters are multiplied by the error gradient. However, in the long chain of such multiplication, if multiple hyperparameters are multiplied by less than one, the resulting gradient will be very small, resulting in zero gradients unable to update the early layers for higher performance. This is why skip-connection or residual layers are used, which provides an alternative path for the gradient. A residual network is made of off-residual connections, a type of skip connection shown in Fig. 7.3.

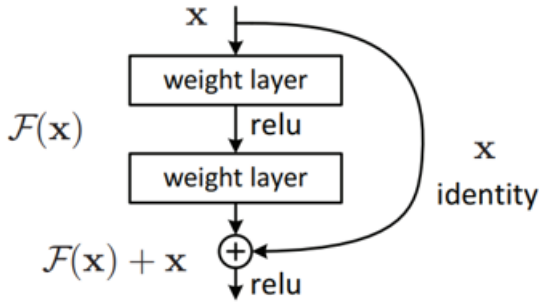


Figure 7.3: Residual learning. A skip connection block.

As the name suggests, during feedforward and backpropagation, it skips some layers in the network and feeds the output of one layer as the input to the subsequent layers. In residual architectures, skip connections are used as vector addition through the identity function. Thus, in this case, the gradient will be multiplied by one, and its value will be maintained in earlier layers. Equation 7.1 shows the mathematical representation of a residual block calculation:

$$y = F(x, W_{\mathrm{i}}) + x \tag{7.1}$$

where y is the output of a layer, x is the input of that layer and F(x, W_{i}) is the residual mapping to be learnt [103]. The in-depth structure and parameters of three variations of ResNet models are shown in Table 7.2.

During the benchmark, ResNet with 50 layers is chosen, the final FNN layer was modified according to the type of datasets used while maintaining the exact residual structure and layer numbers of the model. The final fully connected layer is a single dense connection with around 128 nodes.

Table 7.2: 18-layer,34-layer and 50-layer ResNet Models

18-layer	34-layer	50-layer
conv7-64		
max pool, 3		
[con3-64, conv3-64]x3	[con3-64, conv3-64]x3	[conv1-64, conv3-64, conv1-256] x 3
[con3-128, conv3-128]x3	[con3-128, conv3-128]x3	[conv1-128, conv3-128, conv1-512] x 3
[con3-256, conv3-256]x3	[con3-256, conv3-256]x3	[conv1-256, conv3-256, conv1-1024] x 3
[con3-512, conv3-512]x3	[con3-512, conv3-512]x3	[conv1-512, conv3-512, conv1-2048] x 3
average pool		
maxpool		
FC-1000		
soft-max/sigmoid		

7.2.3 MobileNetV2

Introduced by Sandler and his team [104], MobileNetV2 is an improved version of MobilenetV1 and is a convolutional neural network used in mobile applications. Previously Version 1 used depthwise separable convolution to reduce the model complexity and size. The new version introduces an an added block termed inverted residual structure, removing non-linearities in narrow layers. Fig. 7.4 illustrates the MobileNetV2 architecture.

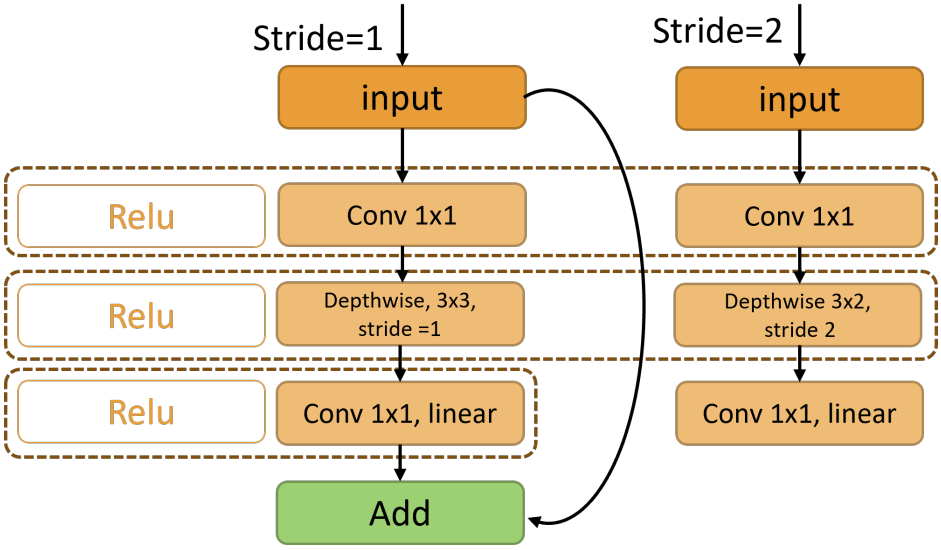


Figure 7.4: MobileNetV2 basic construction of two blocks [104].

In MobileNetV2, there are two types of blocks one of which is a residual block with a stride of 1. Stride is a function used in convolutional neural networks for compressing the images during fine-tuning. It modifies the amount of movement over the image. Another is a block with stride 2 for

downsizing. Each block consists of three different types of layers. The first layer is a 1x1 CovNet with Rectified Linear Unit (ReLU) as its activation function, and the second layer is the depthwise convolution taken from MobilNetV1. The final layer is another 1x1 convolution but without any non-linearity.

7.2.4 Dataset Generation and Pre-processing

In benchmarking the SABiNN model, two types of image sets are used to evaluate the versatility of the model architecture. Cifar10 colored image dataset and medical image dataset MedMNIST are used in this process. To successfully feed into the two models, both the datasets are set as 32x32 inputs where cifar10 is in RGB, having 10 classes, and MedMNIST is in Black and White having binary classes.

Cifar-10

The cifar-10 dataset contains 32x32 colored images. The sample size is around 60000 with 10 classes each. During the evaluation, 5000 were used in training and the rest as testing. The dataset is equally divided into five training batches and one test batch, where each has 1000 sample images. The test batch contains 1000 randomly sampled images to avoid bias training. All the 10 classes in the cifar-10 are mutually exclusive [105]. Fig. 7.5 shows a sample of the cifar-10 images with their labels.

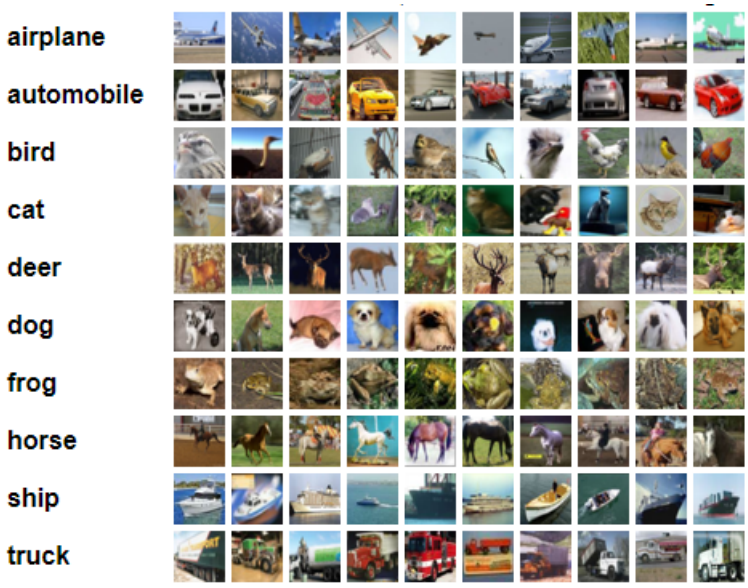
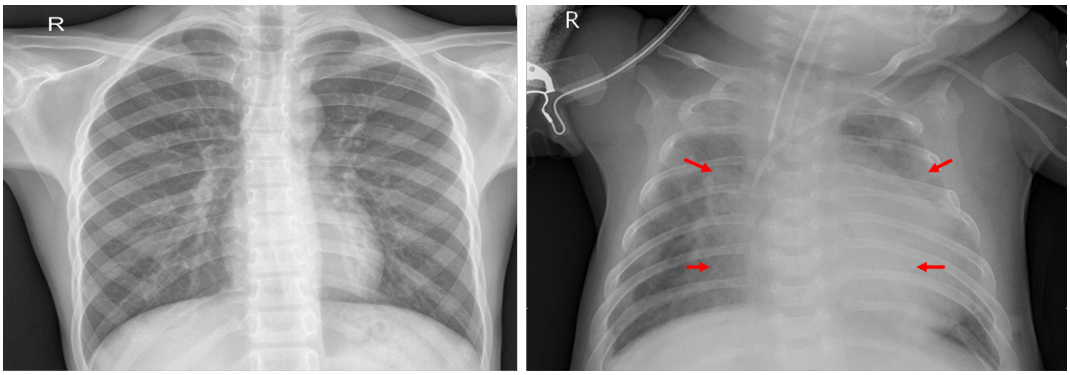


Figure 7.5: Sample of a cifar-10 dataset with image labels [105].

Medical MNIST: MedMNIST

MedMNIST is a large-scale MNIST-like dataset that includes a collection of standardized biomedical images. The dataset contains twelve 2D and six 3D sets [106]. All the collected images are 28x28 for 2D and 28x28x28 for 3D. Since the selected model uses 32x32 as input, the MedMNIST dataset is resized into 32x32 for accurate evaluation. The selected dataset for benchmarking the SABiNN model is the "peumoniaMNIST." PneumoniaMNIST is a 28x28 black-white (BW) 2D image dataset that detects white spots in a patient's lung called **infiltrates**. Fig. 7.6 compares a healthy lung and a pneumonic lung, where the red arrows indicate the white infiltrates.



(a) (b)

Figure 7.6: Sample of (a) healthy lung and (b) affected lung (pneumonia patient) [107].

The models were trained to do binary classification between healthy and pneumonic lungs where the healthy lung is labeled "0," and the pneumonic lung is labeled "1".

Transfer Learning on Deep Learning Models using Cifar10

Transfer learning on deep learning models (DL) is a machine-learning method where the hyperparameters of a specific model are previously trained with a different dataset. Thus, the model parameters are re-used for training on another dataset. It is a widely used method, especially in computer vision and natural language processing applications. Reusing pre-trained models significantly reduces the computational and time resources compared to building models from the ground up. This optimization technique allows rapid progress and improved performance when modeling the second task. However, transfer learning is valid if the DL model is trained on features used in general tasks and the dataset was previously trained on a similar type. For example, when using cifar10 dataset for training and evaluation, it is beneficial to reuse the model parameters trained on image datasets rather than scripts in computer vision [108]

During the benchmarking of SABiNN, parameter-based transfer learning is utilized during the training process. Parameter-based transfer learning is an approach that transfers knowledge at the parametric level, such as the values of weights and biases.

The idea involves transferring knowledge through the shared parameters of the source and target domain learner models. If the model is well-trained on the source domain with a well-defined structure and two related tasks, then the structure can be transferred to the target model.

There are two ways to share the weights in DL models: soft weight sharing and hard weight sharing [109].

Soft weight sharing: The model is expected to be close to the already learned features and penalized if its weights deviate significantly from a given set of weights.

Hard weight sharing: share the exact weights among different models.

This research uses soft weight sharing to benchmark the SABiNN method—the cifar10 image dataset is used during training and validation where the per-trained weights used are "imageNET". Originally the models were trained on ImageNET dataset, and the hyperparameter collected were learned weights that were then used for training on cifar10.

7.2.5 Binarizing Dense Layers

The SABiNN method is implemented on the proposed models: VGG19, ResNet50, and MobileNetV2 where the process is graphically showcased in Fig 7.7. During cifar training and benchmarking, transfer learning techniques reduce training complexity and memory usage in software. "ImageNET" pre-trained weights are utilized because the dimensions and characteristics of cifar10 and ImageNet are similar. Both datasets are 32x32 RGB (3-channel) images. Therefore, the model is initially trained with "ImageNet" After acquiring acceptable test results, each **Dense** layer is extracted, and the weights are further binarized for SABiNN implementation. For MedMNIST dataset, the models are trained with initial randomized weights and then extracted for binarization after attaining acceptable accuracy.

7.3 Training and Evaluation

For cifar10 dataset, categorical cross entropy was used as a loss function, and for binary classification on MedMNIST, binary cross entropy was used. The training features are presented in Table 7.3.

Stochastic Gradient Descent In short, SGD is a popular and standard algorithm used in neural network training for optimizing the model. Gradient descent means descending a slope to

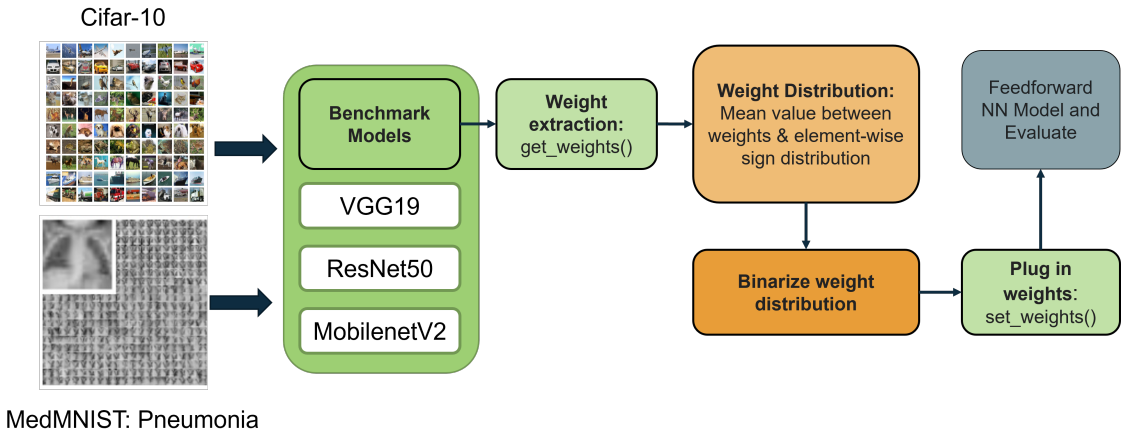


Figure 7.7: Algorithm for binarizing each layer through layer extraction using SABiNN method.

Table 7.3: Training Features for Cifar10 and MedMNIST Datasets

Optimizer	**Loss-Function**	**Activation Function**
Cifar-10 Dataset [105]		
Sparse categorical cross entropy	Stochastic Gradient Descent	Hidden: ReLU, Output: Softmax
MedMNIST: Pneumonia Dataset [106]		
Binary cross entropy	Stochastic Gradient Descent	Hidden: ReLU, Output: Sigmoid

reach the lowest point on that surface. It is an iterative algorithm that starts from a random point on a function and travels down the slope until it reaches its lowest values. The steps in the algorithm are: 1. Finding the slope of the function for each feature or parameter. This step computes the gradient function. 2. Picking a random initial value (stochastic) and updating the gradient function by plugging in the random values. 3. Calculation of the step sizes with a learning rate of each feature, and 4. calculation of the new parametric values where new params = old params-step size. Finally 5, repeating steps 3 and 5 until the gradient is 0. SGD is used for selecting data points at each step to calculate the derivatives. SGD randomly picks one data point from each set at each iteration to reduce computational complexity. The SGD formula is shown in the equation:

$$\theta = \theta - \alpha * \nabla\theta J(\theta; x(I); y(I)) \tag{7.2}$$

Where θ is the current set of model parameters, α is the learning rate that controls the step size taken during optimization in each iteration and x(i) and y(i) are training examples.

Categorical Cross Entropy used in cifar10 image classification, also called the categorical logarithmic loss, adjusts the model weights during training, and the perfect cross-entropy loss is ”0”. Each predicted class probability is compared to the desired multiclass. A score/loss is calculated that

penalizes the probability based on how far or close it is to the actual expected value. The categorical cross entropy function is used when the softmax is the output activation function. The equation shows the calculation of the categorical cross-entropy shown in equation 7.2 [110]:

$$L_{\mathrm{CCE}} = -\sum_{i=1}^{n} t_{\mathrm{i}} log(_{\mathrm{i}}), fornclasses \tag{7.3}$$

Where $\mathrm{L_{CCE}}$ = Categorical crosses entropy loss function, $\mathrm{p_i}$ is the softmax probability for $\mathrm{i^{ith}}$ class and $\mathrm{t_i}$ is the truth label. The true labels (expected output values) are one-hot encoded in categorical cross-entropy. For example, if we have a 3-class classification problem, then the output array will be [1,0,0],[0,1,0], and [0,0,1].

Binary Cross Entropy used in MedMNIST image dataset works with a classification task having two classes, "0" or "1." shown in equation 7.3. Binary cross-entropy is calculated as the average across all data samples shown in equation 7.4. [110]:

$$L = -\sum_{i=1}^{2} t_{\mathrm{i}} log(_{\mathrm{i}}) = -[t_1 log(p_{1)} + t_2 log(p_2) = -[tlog(p) + (1-t)log(1-p)] \tag{7.4}$$

$$L = -1/N[t_{\mathrm{j}} log(p_{\mathrm{j}}) + (1-t_{\mathrm{j}})log(1-p_{\mathrm{j}}) \tag{7.5}$$

Both softmax and sigmoid can be used as the output activation function for classification. In this research during binary classification, the sigmoid activation function was implemented as the last classification layer.

Tables 7.4 and 7.5 represent the evaluation metrics of SABiNN model benchmarked in VGG19 [102], ResNet50 [103] and MobilenNetV2, [104].In Table 7.4, the cifar-10 is a multi-class classification; thus, the evaluation metric is between model accuracy on the test dataset and the top 5% accuracy. Top 5% accuracy describes the model's top 5 highest probability answers that match the expected answer. The classification is correct if the five predictions match the targetted label. It is a widely used metric used for multiclass classification.

For MedMNIST, which performs binary classification, four popular metric calculations are presented in Table 7.5: precision, recall, F1-score, and AUC. **Area Under the ROC Curve (AUC)** measures the entire two-dimensional area underneath the entire ROC curve from (0,0) to (1,1). It represents the measure of performance across all possible classification thresholds. The mathematical equation for accuracy, precision, recall, and F1-score are given below in 7.2, 7.3, 7.4, and 7.5, respectively.

Table 7.4: Evaluation Metric: Cifar10

Classifier	Accuracy: Top 1%	Accuracy: Top 5%
Original Weights		
VGG19	86.61	98.92
ResNet50	95.33	60.65
MobileNetV2	70.32	48.32
Binarized weights		
VGG19	85.08	100
ResNet50	93.17	100
MobileNetV2	67.63	62.83

$$accuracy = TN + TP/TN + FP + TP + FN \tag{7.6}$$

$$precision = TP/TP + FP \tag{7.7}$$

$$recall = TP/TP + FN \tag{7.8}$$

$$f1 - score = 2 * ((precision*)/(precision + recall)) \tag{7.9}$$

Table 7.5: Evaluation Metric: MedMNIST-pneumonia

Classifier	Accuracy (%)	Precision (%)	Recall%	F1-Score%	AUC%
Original Weights					
VGG19	89	93	89	89	93.98
ResNet50	85	87	85	85	92.81
MobileNetV2	77	82	77	74	93.64
Binarized Weights					
VGG19	88	89	89	89	85.768
ResNet50	85	85	85	85	83.912
MobileNetV2	79.33	86	80	82	80.90

According to Tables 7.4 and 7.5, the overall accuracy degradation when converting from float32 weight value to binarized is around 1%-2.5%, which is relatively low compared to other widely used compression techniques [74].

The ROC-AUC, termed the Area Under Receiving Operating Curve, is shown in Fig 7.8. A receiver operating characteristic curve, or ROC curve, is a graphical plot illustrating the ability to

classify binary classification adequately. The ROC is a relationship between sensitivity and specificity. According to the AUC the accuracy degraded only 2.5% while maintaining the discriminator percentage to over 80%. If ROC-AUC percentage is in between 80% to 90% then the classifier or model is considered an excellent discriminator which is acceptable in optimization and compression. Due to sparsity reduction, the AUC curve showcases a straightforward learning curve.

According to the evaluation metrics, the transformed models showcase acceptable accuracy. To estimate the model's total energy during testing in each sample, calculating the layer-wise energy can contribute to calculating the total energy consumption of a given model shown in equation 7.6 [111]:

$$E_{\text{layer}} = E_{\text{data\_flow}} + E_{\text{computational}} \tag{7.10}$$

Where E_{layer} is the total energy of each model layer, $\text{E}_{\text{data\_flow}}$ is the data flow, which includes the off-chip on-chip data transmission and memory access read/write, and $\text{E}_{\text{computational}}$ is the computational energy of the performing MACs of a given layer. The main objective of this research focuses on decreasing the energy of the performing MACs by reducing the sparsity of a given Deep Neural Network (DNN) model. The total MAC units in each layer can be calculated by the shape of the filter and the number of filters in a convolutional layer, and the number of neurons for a fully connected layer.

Equation 7.7 calculates the total MAC unit per convolutional layer.

$$Conv\_Layer = (H_{\text{w}} * H_{\text{h}} * N_{\text{i-1}} + 1) * N_{\text{i}} \tag{7.11}$$

Where H_{w} is the width of the filter, H_{h} is the height of the filter, $\text{N}_{\text{i-1}}$ is the number of filters in the previous layer, and N_{i} is the number of filters in the current layer. Since the proposed model did not consider bias then, the equation stands as 7.8:

$$Conv\_Layer = H_{\text{w}} * H_{\text{h}} * N_{\text{i-1}} * N_{\text{i}} \tag{7.12}$$

Equation 7.9 calculates the total MAC unit per fully connected layer, FC_Layer.

$$FC\_Layer = (n_{\text{i}} * n_{\text{(i-1)}}) + 1 * n_{\text{i}} \tag{7.13}$$

Where n_{i} is the current layer neurons and $\text{n}_{\text{(i-1)}}$ is the previous layer neurons. Since bias is discarded, then the final formula is given in 7.10:

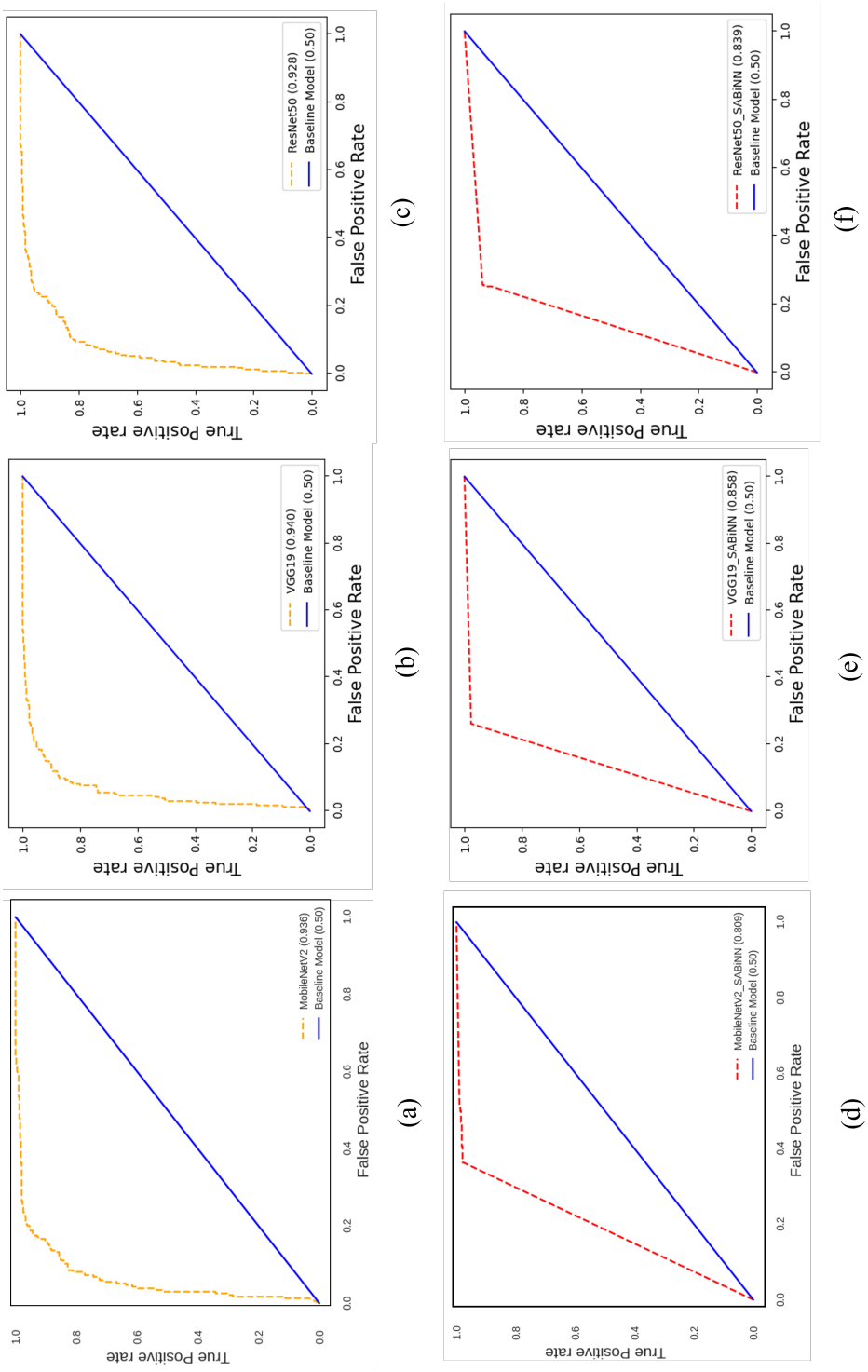


Figure 7.8: Simulation test result demonstrating the hardware prediction of diabetes prediction (1: diabetes predicted and 0: normal condition) by using unseen test data set.

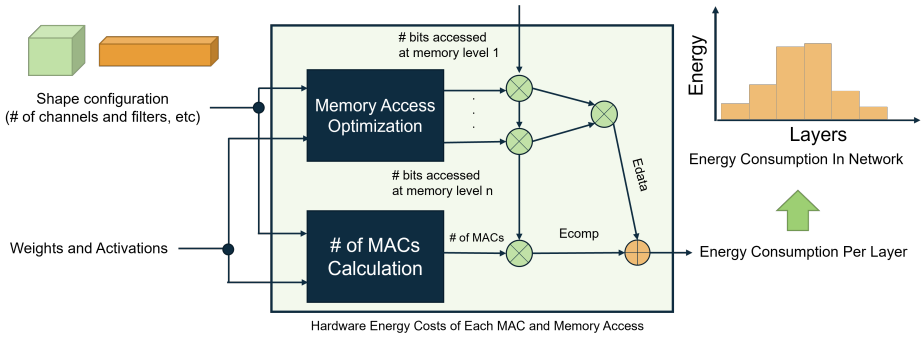


Figure 7.9: Energy estimation methodology [111] where E_{comp} is the computation energy being consumed and E_{data} is the energy per data passing and access.

$$FC\_Layer = (n_{\mathrm{i}} * n_{(\mathrm{i\text{-}1})}) \tag{7.14}$$

The classifiers used in the SABiNN benchmark were VGG19, ResNet50, and MobileNetV2, and Table 7.6 presents the total MAC units of each classifier when bias is discarded. Each of the models has a combination of Conv_Layer and FC_Layer.

Table 7.6: Total Number of MAC Units

DL Model	**Total Number of MAC Units**	**Conv_Layer MAC Unit**	**FC_Layer MAC Unit**
VGG19	20351040	20023232	327808
ResNet50	23790590	23581440	262272
MobileNetV2	2387264	2257408	163968

Each MAC unit comprises two FLOPs. FLOPs are the number of floating point operations a computing entity can perform in one second. The FLOPs quantify the model performance on hardware and represent the number of floating point operations required for a single forward pass. Therefore, FLOPs estimate the model performance of each forward pass for each sample given to the model. Since MAC is an operation that does an addition and a multiplication, thus it requires two FLOP operations shown in equation 7.11:

$$MAC = 2 * FLOPs = 1MULT + 1ADD \tag{7.15}$$

If the models are designed on a 45nm CMOS process which is an industry-standard silicon process and voltage supply of 0.9V is supplied, then the traditional energy rate of multipliers, adders, and 2s complement are presented in Table 7.7.

Table 7.7: Energy Rate of each Logic Block used in MAC and BAC

Bit	Multiplier [112]	Adder [112]	2s Complement [113]
8-bit	0.2pJ	0.03 pJ	0.06 pJ
16-bit	0.4pJ	0.06pJ	0.12pJ
32-bit	3.1pJ	0.1pJ	0.23pJ
16-bit float	1.1pJ	0.4pJ	-
32-bit float	3.7pJ	0.9pJ	-

An n-bit barrel shifter is estimated to have an energy consumption of around 1.28pJ on a 45nm CMOS process requiring a supply voltage of 0.8-0.9V. [114]. Then the estimated energy between original classifiers, DeepSAC, and SABiNN-based models are presented in Table 7.8. DeepSAC and SABiNN are calculated using 32-bit integer logic units (i.e., 32-bit multiplier and 32-bit adder), as the model architecture uses n-bit quantized values during inference testing.

Table 7.8: Estimated energy consumption Rate between original, DeepSAC, and SABiNN-based classifiers

Classifiers	Original	DeepSAC-Based	SABiNN-Based
VGG19	93 mJ	0.645 μ J	0.64 μ J
ResNet50	109mJ	0.75 μ J	0.84 μ J
MobileNetV2	11mJ	0.744 μ J	0.727 μ J

According to Table 7.8, implementing DeepSAC-based models on the original classifiers reduced the energy consumption by nearly 3x, and SABiNN-Based models decreased the energy efficiency by 14x, which is a significant reduction in energy consumption rate. This concludes that both DeepSAC and SABiNN model architectures can be utilized in developing energy-efficient deep neural networks while maintaining their accuracy rate.

7.4 Conclusion

In this chapter, the proper evaluation and benchmark of SABiNN model architecture are executed. Three widely used and popular deep learning models such as VGG19 [102], ResNet50 [103], and MobileNetV2 [104] are carefully chosen to add versatility to the benchmark. VGG19 is a deep convolutional neural network, whereas ResNet50 is a deep residual neural network that optimizes deep neural nets' backpropagation and training complexity. MobileNetV2 is a neural net consisting of convolutional and residual networks designed for mobile and embedded system applications. To add another dimension of versatility, two different image datasets are used during training and

evaluation. Cifar10 is an RGB dataset consisting of multi-class labels, and MedMNIST: Pneumonia is a biomedical image data in BW format consisting of binary class. The SABiNN model architecture evaluation metrics on the proposed models showed promising results. The accuracy degradation was less than 2.5% which is significantly lower than currently documented optimization and compression techniques used in NN models. The energy consumption rate of the computational unit of both DeepSAC-Based and SABiNN-Based reduced significantly by 3x-14x times in contrast with the original classifiers.

Chapter 8

CONCLUSION

In recent years, almost all intelligent systems in various applications, from healthy harvesting in agriculture to detecting complex diseases in healthcare, use Artificial-Intelligence (AI)/ Machine Learning (ML) models. The market size of AI/ML is estimated to reach its compound annual growth (CAGR) from USD 94 billion as of 2021-2022 to USD 1 trillion in 2030, an estimated increase of 38.1% [115]. The competition in creating and developing highly complex deep neural networks is rising. Technological companies and industries are constantly battling to design revolutionary AI models that can predict, generate and classify complex problems that the average human brain cannot. However, the deployment of such sophisticated models on edge for real-time applications is challenging due to the existence of limited hardware resources. Moreover, running and training complex algorithms requires a higher power budget to achieve a high accuracy rate which exhausts the computational power of both CPUs and GPUs. Due to Moore's Law being stagnant, the amount of memory and existing hardware technology is inadequate to drive and execute advanced high computing models. The lack of advanced hardware technology creates a massive gap between model computation and available hardware resources. Future technological advancement in healthcare, agriculture, and even solutions to the environmental and economic crisis will be stagnating if proper steps are not taken. This dissertation aims to reduce this gap by answering how to effectively deploy computationally intensive models on edge in the post-Moore era- providing re-designing solutions to available hardware resources that will maximize memory utilization while maintaining a high accuracy rate. A software-hardware co-simulation is developed that walks through the process from neural network training on software to inference on CMOS platform, ensuring low error margin in designing and fabrication cost shown in Fig. 8.1.

Two energy-efficient, hardware-friendly models are developed: **DeepSAC:** Shift accumulate-based Deep Neural Network and **SABiNN:** Shift accumulate-based binarized neural networks.

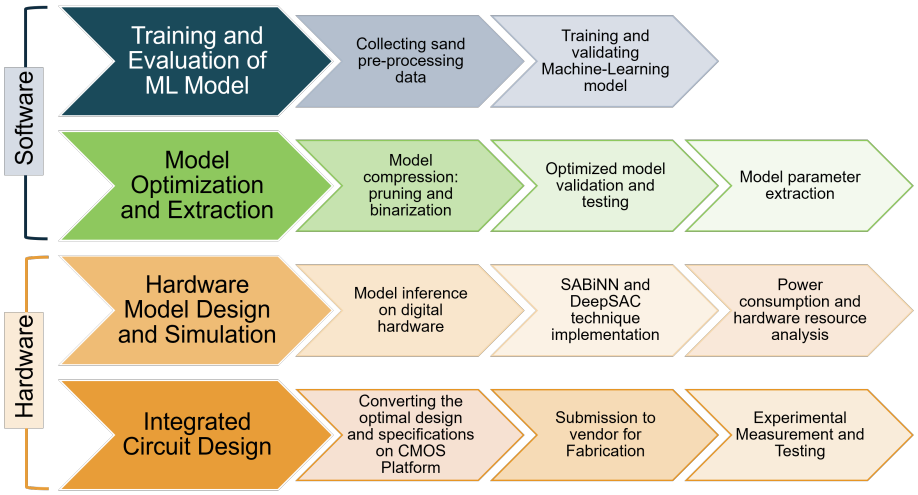


Figure 8.1: Software-Hardware co-simulation method [16].

The DeepSAC method uses compression and optimization techniques, such as pruning and n-bit quantization, and replaces multipliers with shifters by transforming the weights in the multiple of 2s. This technique is used in developing a diabetes prediction model for pregnant women that can be used as a mobile application that will showcase prediction results in real-time. Another application DeepSAC uses is a wearable sleep apnea classification device for adults that can detect real-time obstructive sleep apnea. Both the design architecture was tested on reconfigurable hardware (Nexys Artix-7 FPGA) to ensure energy efficiency and precision rate. The SABiNN model is used to develop a wearable healthcare device that detects sleep apnea among adults with minimal sensor involvement. The model is tested on reconfigurable hardware, and the final trained model is integrated into a 180 nm and 130 nm CMOS platforms. Furthermore, the SABiNN model is evaluated by benchmarking the design technique on various popularly used deep neural network modes such as VGG19, ResNet50, and MobileNetV2 and validating it on two different image datasets cifar10 and MedMNIST: pneumonia. Both the models DeepSAC and SABiNN showcase promising accuracy and energy efficiency results. SABiNN and DeepSAC reduced the power consumption rate by 13x times from a regular deep neural network model using traditional matrix multiplication calculation.

The central concept of this dissertation is the development of low-power, energy-efficient circuit design techniques for embedding ML models, specifically deep neural networks (DNN) on edge. The dissertation can be represented in a four-dimensional matrix inspired by Song Han's [1] dissertation summary shown in Fig. 8.2. Thus, the ultimate goal is to reduce sparsity and increase efficiency. Reducing sparsity translates to reducing components and parameters that do not directly impact

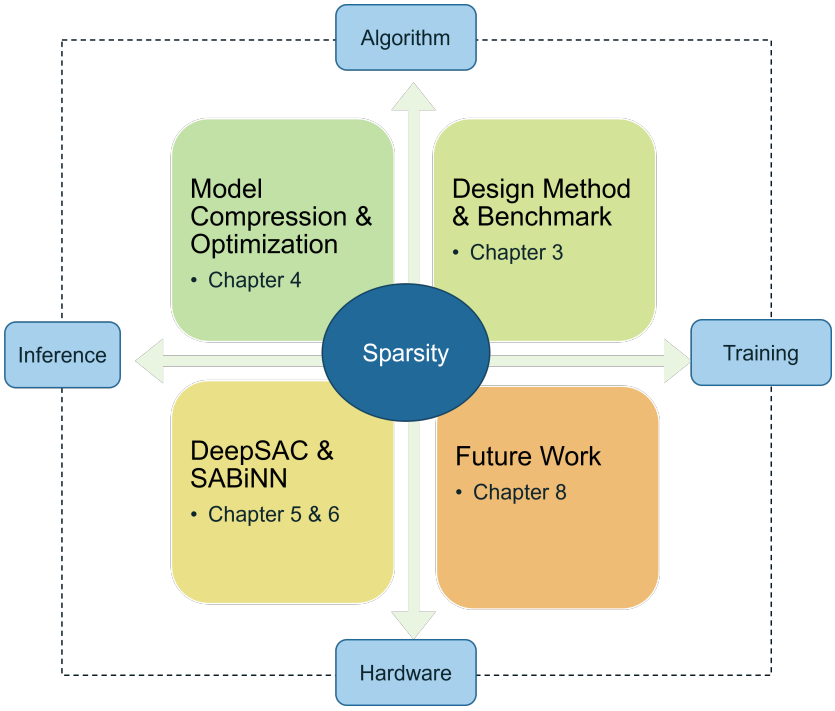


Figure 8.2: Summary of the thesis.

the calculation in a neural network model. Implementing pruning, 2s multiple conversion, and n-bit quantization on DeepSAC and binarization on SABiNN greatly reduces the model size and utilizes the memory allocation when working with integer and binary values. Thus, resulting in replacing multipliers with shifters and XNOR gates.

- **Design Method and Benchmark**: To ensure a low error rate and avoid significant degradation of accuracy, the software-hardware co-simulation method presents a seamless software-to-edge implementation during the training and testing phase. The algorithms of the proposed models are benchmarked during training to ensure versatility and robustness.

- **Model Compression and Optimization** After properly training and validating the model, further optimization and compression are done by introducing and replacing logic blocks such as multipliers and n-bit floating points (n = 16,32) to shifters, 2s complement, and n-bit integers (n = 8, 16). The proposed and implemented methods significantly reduced the size and power consumption rate by maintaining a balanced accuracy across different types of applications.

- **DeepSAC and SABiNN** DeepSAC and SABiNN are two proposed model architectures developed when embedding a trained deep neural network model on hardware. These models are extensively used during the inference phase, ensuring a low power consumption rate, high

energy efficiency, and precision rate.

Currently, the main application of this research includes designing hardware architectures for biomedical applications such as sleep apnea (SA) detection. The detection system uses minimal yet essential biophysical sensors such as an ECG (Electrocardiogram) patch to detect heart rate and a finger-tip pulse oximeter to measure blood saturation level for SA screening. The targeted power consumption rate is below 50 μW, enabling the device to operate efficiently with a coin-sized lithium battery.

8.1 Future Work

This dissertation can be divided into two potential future directions: (i) developing methods for designing low-power, energy-efficient deep learning models on edge and (ii) designing and creating low-cost, wearable, and smart healthcare devices. The emergence of neuromorphic computing has sparked interest in the device and electronics field, which aids in highly complex and high-speed computation. Alternatively, it introduces energy efficiency, precision, and durability challenges.

- **Deep-Learning Model in Analog Computation (ML-on-Chip):** Electronics industries are gaining new interest in neuromorphic computing as this could be the future in high-speed, high-precision, and low-power computing. Various neuromorphic chips and neural engine frameworks (NEF) are developed for high-processing computation [116,117]. But it is yet to be used in a wide range of applications. There is a huge opportunity to solve efficient ways to design high computational models with various analog device components such as memristors, amplifiers, converters, etc.

- **Deep-Learning Model in Digital Computation (TinyML):** Challenges still exist in realizing parallel computing using digital logic units. Deep-learning scientists and AI-powered industries still leverage GPUs and their ASIC-based TPUs to train intensive and complex ML models requiring high power and energy. Thus, the carbon footprint of such models is becoming a significant concern. Developing and exploring efficient AI/ML inference techniques and re-designing digital logic blocks can be a potential future green technology direction.

- **Energy-Efficient Embedded System Development for Healthcare:** There is a significant need for hardware resources to run high computational models that can diagnose complex diseases. Developing and constructing energy-efficient models during the inference phase will minimize the energy efficiency challenge in AI/ML.

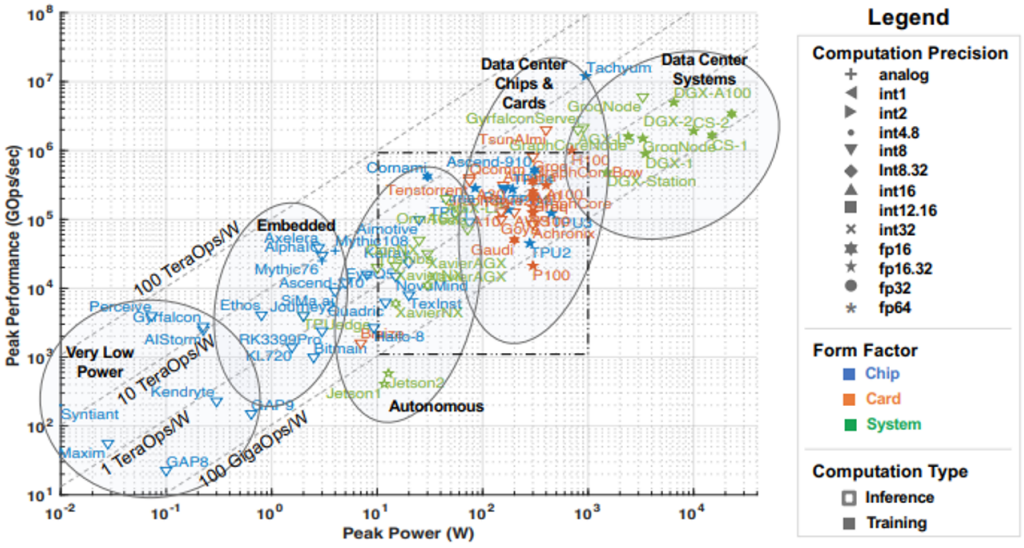


Figure 8.3: 2022 Artificial Intelligence Accelerators Surveys and Trends [118].

The ultimate goal is to deliver energy-efficient solutions in redesigning and developing AI/ML-based hardware accelerators that can seamlessly run highly complex models without exhausting the available resources. Currently, the AI accelerator market where accelerators labeled as **very low power** are inference and integer types shown in Fig 8.3. Introducing an ML-on-chip solution where the AI/ML models are trained and updated on the chip in real-time without exhausting the hardware resources can significantly change resource utilization and reduce carbon emissions produced by the current AI models in the market. AI/ML is on the rise of technological advancement, and providing energy-efficient solutions will make the advancement faster. The proposed methods and model architectures can be a stepping stone into democratizing AI in the future and open spaces for applications requiring real-time detection and prediction.

Appendix A

PYTHON CODE: FNN, DEEPSAC AND SABINN

```
import numpy as np

#Data Normalization and Pre-Processing
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
# define min max scaler
scaler = MinMaxScaler()
xs = scaler.fit_transform(x)
print(xs)

#train test split
from sklearn.model_selection import train_test_split
x_train, x_val, y_train, y_val = train_test_split(xs, y, test_size=0.30)
print(x_train.shape)
print(y_train.shape)
print(x_val.shape)
print(y_val.shape)

#model build and train
from sklearn.model_selection import StratifiedKFold
from keras.layers import LeakyReLU
```

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from keras.models import Model

# fix random seed for reproducibility
seed = 10
np.random.seed(seed)
# define 10-fold cross validation test harness
kfold = StratifiedKFold(n_splits=10  , shuffle=True, random_state=seed)
cvscores = []
for train, test in kfold.split(x_train, y_train):
  model = Sequential()
  model.add(Dense(4, input_dim = 2, activation = 'relu',use_bias= False))
  model.add(Dense(8, activation = 'relu',use_bias= False))
  model.add(Dense(6, activation = 'relu',use_bias= False))
  model.add(Dense(4, activation = 'relu',use_bias= False))
  model.add(Dense(1, activation = 'sigmoid',use_bias= False))

  #opt = SGD(lr=0.01, momentum=0.9)
  model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
        # Fit the model
  history= model.fit(x_train[train], y_train[train],
  validation_data= (x_train[test], y_train[test]),
  epochs=1000, batch_size=10, verbose=0)
        # evaluate the model
  scores = model.evaluate(x_train[test], y_train[test], verbose=0)
  print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
  cvscores.append(scores[1] * 100)
print("%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(cvscores)))
```

```
#evaluation
model.evaluate(x_val, y_val.round())
model.summary()

_, accuracy = model.evaluate(x_val, y_val)
print('test_Accuracy: %.2f' % (accuracy*100))
_, accuracy1 = model.evaluate(x_train, y_train.round())
print('val_Accuracy: %.2f' % (accuracy1*100))

#confusion metrix generation
y_pred = model.predict(x_val)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_val, y_pred.round(), normalize= None)
print(cm)

#precision
from sklearn.metrics import classification_report
print(classification_report(y_val, y_pred.round()))
#matrix = [TP, FP,
#          FN, TN ]

#DeepSAC Pruning and weight extraction for conversion

import pandas as pd

#weight pruning
all_weights = {}

for layer_no in range(total_no_layers - 1):
    layer_weights = (pd.DataFrame(
    trained_model.layers[layer_no].get_weights()[0]).stack()).to_dict()
```

```
    layer_weights = { (layer_no, k[0], k[1]): v for k, v in layer_weights.items() }
    all_weights.update(layer_weights)

all_weights_sorted = {k: v for k, v in sorted(all_weights.items(),
key=lambda item: abs(item[1]))}
total_no_weights = len(all_weights_sorted)
total_no_weights

weight_pruning_scores = []

for pruning_percent in K:

    new_model = 'load_the_model'
    new_weights = trained_model.get_weights().copy()

    prune_fraction = pruning_percent/100
    number_of_weights_to_be_pruned = int(prune_fraction*total_no_weights)
    weights_to_be_pruned =
    {k: all_weights_sorted[k] for k in list(all_weights_sorted)
    [ :  number_of_weights_to_be_pruned]}

    for k, v in weights_to_be_pruned.items():
        new_weights[k[0]][k[1], k[2]] = 0

    for layer_no in range(total_no_layers- 1) :
        new_layer_weights =
        new_weights[layer_no].reshape(1, new_weights[layer_no].shape[0],
        new_weights[layer_no].shape[1])
        new_model.layers[layer_no].set_weights(new_layer_weights)

    new_score  = new_model.evaluate(x_val, y_val, verbose=0)
    weight_pruning_scores .append(new_score[1])
```

```
#neuron pruning

all_neurons = {}

for layer_no in range(total_no_layers - 1):

    layer_neurons = {}
    layer_neurons_df = pd.DataFrame(trained_model.layers[layer_no].get_weights()[0])

    for i in range(len(layer_neurons_df.columns)):
        layer_neurons.update({ i : np.array( layer_neurons_df.iloc[:,i] ) })

    layer_neurons = { (layer_no, k): v for k, v in layer_neurons.items() }
    all_neurons.update(layer_neurons)

all_neurons_sorted = {k: v for k, v in sorted(all_neurons.items(),
key=lambda item: np.linalg.norm(item[1], ord=2, axis=0))}
total_no_neurons = len(all_neurons_sorted)
total_no_neurons

neuron_pruning_scores = []

for pruning_percent in K:

    new_model = 'load_the_model'

    prune_fraction = pruning_percent/100
    number_of_neurons_to_be_pruned = int(prune_fraction*total_no_neurons)
    neurons_to_be_pruned = {k: all_neurons_sorted[k]
    for k in list(all_neurons_sorted)[ : number_of_neurons_to_be_pruned]}
```

```
    for k, v in neurons_to_be_pruned.items():
        new_weights[k[0]][:, k[1]] = 0

    for layer_no in range(total_no_layers - 1) :
        new_layer_weights = new_weights[layer_no].reshape(1,
        new_weights[layer_no].shape[0], new_weights[layer_no].shape[1])
        new_model.layers[layer_no].set_weights(new_layer_weights)

    new_score  = new_model.evaluate(x_val, y_val, verbose=0)
    neuron_pruning_scores.append(new_score[1])

#extraction of the weights
w0= new_model.layers[0].get_weights()
print(w0)

#extraction of the weights
w1=new_model.layers[1].get_weights()
print(w1)

#extraction of the weights
w2 = new_model.layers[2].get_weights()
print(w2)

#extraction of the weights
w3= new_model.layers[3].get_weights()
print(w3)

#Binarization of weights for SABiNN

bin0=model.layers[0].get_weights()
bin1=model.layers[1].get_weights()
bin2=model.layers[2].get_weights()
```

```
bin3=model.layers[3].get_weights()
bin4=model.layers[4].get_weights()

#evaluating model with shifter
import tensorflow as tf

m = tf.keras.metrics.Mean()
b0 = m(tf.math.abs(bin0))*tf.keras.backend.sign(bin0)
b0= tf.reshape(b0, [2, 4]).numpy()
print(b0)

#evaluating model with shifter
b1 = m(tf.math.abs(bin1))*tf.keras.backend.sign(bin1)
b1= tf.reshape(b1, [4, 8]).numpy()
print(b1)

#evaluating model with shifter
b2 = m(tf.math.abs(bin2))*tf.keras.backend.sign(bin2)
b2= tf.reshape(b2, [8, 6]).numpy()
print(b2)

b3 = m(tf.math.abs(bin3))*tf.keras.backend.sign(bin3)
b3= tf.reshape(b3, [6, 4]).numpy()
print(b3)

b4 = m(tf.math.abs(bin4))*tf.keras.backend.sign(bin4)
b4= tf.reshape(b4, [4, 1]).numpy()
print(b4)

b10= b0/ 0.5886239 # mean value of the first layer
print(b10)
```

```
b11= b1/0.46663094
print(b11)

b12= b2/0.48054782
print(b12)

b13= b3/0.5224517
print(b13)

b14= b4/ 0.5637053
print(b14)

model_bin.layers[0].set_weights([b10])
model_bin.layers[1].set_weights([b11])
model_bin.layers[2].set_weights([b12])
model_bin.layers[3].set_weights([b13])
model_bin.layers[4].set_weights([b14])

model_bin.evaluate(x_val, y_val)
```

Appendix B

VHDL CODE: DEEPSAC FOR SA DETECTION

```
library ieee;
use ieee.std_logic_1164.all;

#top cell
entity neural_net_springer is
port (x1, x2: in std_logic_vector (8 downto 0); clk: std_logic;
y:out std_logic);
end neural_net_springer;

architecture hidden of neural_net_springer is

component hidden0 is
port (x1, x2: in std_logic_vector (8 downto 0); clk: std_logic; y1, y2, y3, y4,
y5, y6, y7, y8:out std_logic_vector(8 downto 0));
end component;

component hidden1 is
port (x1, x2, x3, x4, x5, x6, x7, x8: in std_logic_vector (8 downto 0);
clk: std_logic; y1, y2, y3, y4, y5, y6:out std_logic_vector(8 downto 0));
end component;

component hidden2 is
```

```
port (x1, x2, x3, x4, x5, x6: in std_logic_vector (8 downto 0);
clk: std_logic; y1, y2, y3, y4:out std_logic_vector(8 downto 0));
end component;

component hidden3 is
port (x1, x2, x3, x4: in std_logic_vector (8 downto 0); clk: std_logic;
y1:out std_logic_vector(8 downto 0));
end component;

component shift8_out is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
        dout: out std_logic);
end component;


signal h11, h12, h13, h14, h15, h16, h17, h18, h21, h22, h23, h24,
h25, h26, h31, h32, h33, h34, yout: std_logic_vector(8 downto 0);

begin

h0: hidden0 port map (x1, x2, clk, h11, h12, h13, h14, h15,
h16, h17, h18);
hl: hidden1 port map (h11, h12, h13, h14, h15, h16, h17, h18,
clk, h21, h22, h23, h24, h25, h26);
h2: hidden2 port map (h21, h22, h23, h24, h25, h26, clk, h31,
h32, h33, h34);
h3: hidden3 port map (h31, h32, h33, h34, clk, yout);
s: shift8_out port map (yout, clk, y);

end hidden;

# hidden 0
```

```
entity hidden0 is
port (x1, x2: in std_logic_vector (8 downto 0); clk: std_logic; y1, y2,
y3, y4, y5, y6, y7, y8:out std_logic_vector(8 downto 0));
end hidden0;

architecture hidden of hidden0 is

component shift1 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
        dout: out std_logic_vector(8 downto 0));
end component;

component shift3 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
        dout: out std_logic_vector(10 downto 0));
end component;

component shift4 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
        dout: out std_logic_vector(11 downto 0));
end component;

component shift5 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
        dout: out std_logic_vector(12 downto 0));
end component;

component shift6 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
        dout: out std_logic_vector(13 downto 0));
end component;
```

```
component shift7 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
        dout: out std_logic_vector(13 downto 0));
end component;

component S7 is
port (din: in std_logic_vector(13 downto 0); clk: in std_logic;
        dout: out std_logic_vector(8 downto 0));
end component;

component S6 is
port (din: in std_logic_vector(13 downto 0); clk: in std_logic;
        dout: out std_logic_vector(8 downto 0));
end component;

component S5 is
port (din: in std_logic_vector(12 downto 0); clk: in std_logic;
        dout: out std_logic_vector(8 downto 0));
end component;

component S4 is
port (din: in std_logic_vector(11 downto 0); clk: in std_logic;
        dout: out std_logic_vector(8 downto 0));
end component;

component S3 is
port (din: in std_logic_vector(10 downto 0); clk: in std_logic;
        dout: out std_logic_vector(8 downto 0));
end component;

component S1 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
```

```
        dout: out std_logic_vector(8 downto 0));
end component;

component add9 is
port (A,B: in std_logic_vector(8 downto 0); Cin: in std_logic; Co:
out std_logic; Sum: out std_logic_vector(8 downto 0));
end component;

component relu9 is
port (x : in std_logic_vector(8 downto 0);
y: out std_logic_vector(8 downto 0));
end component;

component compliment2s
port (Din: in std_logic_vector (8 downto 0); Co: out std_logic;
Do: out std_logic_vector (8 downto 0));
end component;

signal xs11, xs12, xs21, xs22, xs31, xs32, xs41, xs42, xs51, xs52, xs61,
xs62, xs71, xs72, xs81, xs82: std_logic_vector(8 downto 0);
signal xs82c,xs42c, xs52c, xs62c, xs72c, xs21c, xs31c,
xs71c : std_logic_vector(8 downto 0);

signal xs61s : std_logic_vector(8 downto 0); --1 shift
signal xs21s: std_logic_vector(10 downto 0); --3 shift
signal xs81s, xs82s: std_logic_vector (11 downto 0); --4 shift
signal xs11s, xs12s, xs31s, xs41s, xs42s, xs52s, xs62s, xs72s:
std_logic_vector(12 downto 0); --5 shift
signal xs22s, xs32s, xs51s: std_logic_vector(13 downto 0); --6 shift
signal xs71s: std_logic_vector(13 downto 0); --7 shift

signal Ca112, Ca212, Ca312, Ca412, Ca512, Ca612, Ca712, Ca812, Cab1,
```

```
Cab2, Cab3, Cab4, Cab5, Cab7, Cab8: std_logic;
signal co21, co31, co71, co42, co52, co62, co72, co82: std_logic;
signal ABo1, ABo2, ABo3, ABo4, ABo5, ABo6, ABo7, ABo8, Ao112,
Ao212, Ao312, Ao412, Ao512, Ao712, Ao812: std_logic_vector(8 downto 0);

begin
S11: shift5 port map (x1, clk, xs11s);
SS11: S5 port map (xs11s,clk,  xs11);
S12: shift5 port map (x2,clk,  xs12s);
SS12: S5 port map (xs12s,clk, xs12);

A112: add9 port map (xs12, xs11, '0', Ca112, Ao112);
AB1: add9 port map (Ao112,"111100000", '0', Cab1, ABo1);

S21: shift3 port map (x1,clk, xs21s);
SS21: S3 port map (xs21s,clk,xs21c);
C21: compliment2s port map (xs21c, co21, xs21);
S22: shift6 port map (x2,clk, xs22s);
SS22: S6 port map (xs22s,clk, xs22);

A212: add9 port map (xs22, xs21, '0', Ca212, Ao212);
AB2: add9 port map (Ao212, "111110000", '0', Cab2, ABo2);

S31: shift5 port map (x1,clk, xs31s);
SS31: S5 port map (xs31s,clk, xs31c);
C31: compliment2s port map (xs31c, co31,xs31);
S32: shift6 port map (x2, clk, xs32s);
SS32: S6 port map (xs32s, clk, xs32);

A312: add9 port map (xs31, xs32, '0', Ca312, Ao312);
AB3: add9 port map (Ao312, "111111100", '0', Cab3, ABo3);
```

```
S41: shift5 port map (x1, clk, xs41s);
SS41: S5 port map (xs41s, clk, xs41);
S42: shift5 port map (x2,clk, xs42s);
SS42: S5 port map (xs42s,clk, xs42c);
C42: compliment2s port map (xs42c, co42,xs42);

A412: add9 port map (xs42, xs41, '0', Ca412, Ao412);
AB4: add9 port map (Ao412, "000000100", '0', Cab4, ABo4);

S51: shift6 port map (x1, clk, xs51s);
SS51: S6 port map (xs51s, clk, xs51);
S52: shift5 port map (x2,clk, xs52s);
SS52: S5 port map (xs52s,clk, xs52c);
C53: compliment2s port map (xs52c, co52,xs52);

A512: add9 port map (xs52, xs51, '0', Ca512, Ao512);
AB5: add9 port map (Ao512, "000000001", '0', Cab5, ABo5);

S61: shift1 port map (x1, clk, xs61s);
SS61: S1 port map (xs61s, clk, xs61);
S62: shift5 port map (x2,clk, xs62s);
SS62: S5 port map (xs62s,clk, xs62c);
C61: compliment2s port map (xs62c, co62, xs62);

AB6: add9 port map (xs62, xs61, '0', Ca612, ABo6);

S71: shift7 port map (x1, clk, xs71s);
SS71: S7 port map (xs71s, clk, xs71c);
C71: compliment2s port map (xs71c, co71, xs71);
S72: shift5 port map (x2,clk, xs72s);
SS72: S5 port map (xs72s,clk, xs72c);
C73: compliment2s port map (xs72c, co72, xs72);
```

```
A712: add9 port map (xs72, xs71, '0', Ca712, Ao712);
AB7: add9 port map (Ao712,"000100000", '0',Cab7,ABo7);

S81: shift4 port map (x1, clk, xs81s);
SS81: S4 port map (xs81s, clk, xs81);
S82: shift4 port map (x2,clk, xs82s);
SS82: S4 port map (xs82s,clk, xs82c);
C82: compliment2s port map (xs82c, co82, xs82);

A812: add9 port map (xs82, xs81, '0', Ca812, Ao812);
AB8: add9 port map (Ao812,"111111100", '0',Cab8,ABo8);

RE1: relu9 port map (ABo1, y1);
RE2: relu9 port map (ABo2, y2);
RE3: relu9 port map (ABo3, y3);
RE4: relu9 port map (ABo4, y4);
RE5: relu9 port map (ABo5, y5);
RE6: relu9 port map (ABo6, y6);
RE7: relu9 port map (ABo7, y7);
RE8: relu9 port map (ABo8, y8);

end hidden;

#hidden 1

entity hidden1 is
port (x1, x2, x3, x4, x5, x6, x7, x8: in std_logic_vector
(8 downto 0); clk: std_logic; y1, y2, y3, y4, y5, y6:out
std_logic_vector(8 downto 0));
end hidden1;
```

```
architecture hidden of hidden1 is

component shift1 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
        dout: out std_logic_vector(8 downto 0));
end component;

component shift2 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
        dout: out std_logic_vector(9 downto 0));
end component;

component shift3 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
        dout: out std_logic_vector(10 downto 0));
end component;

component shift4 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
        dout: out std_logic_vector(11 downto 0));
end component;

component shift5 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
        dout: out std_logic_vector(12 downto 0));
end component;

component shift6 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
        dout: out std_logic_vector(13 downto 0));
end component;
```

```
component shift7 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
        dout: out std_logic_vector(13 downto 0));
end component;

component S7 is
port (din: in std_logic_vector(13 downto 0); clk: in std_logic;
        dout: out std_logic_vector(8 downto 0));
end component;

component S6 is
port (din: in std_logic_vector(13 downto 0); clk: in std_logic;
        dout: out std_logic_vector(8 downto 0));
end component;

component S5 is
port (din: in std_logic_vector(12 downto 0); clk: in std_logic;
        dout: out std_logic_vector(8 downto 0));
end component;

component S4 is
port (din: in std_logic_vector(11 downto 0); clk: in std_logic;
        dout: out std_logic_vector(8 downto 0));
end component;

component S3 is
port (din: in std_logic_vector(10 downto 0); clk: in std_logic;
        dout: out std_logic_vector(8 downto 0));
end component;

component S2 is
port (din: in std_logic_vector(9 downto 0); clk: in std_logic;
```

```
        dout: out std_logic_vector(8 downto 0));
end component;

component S1 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
        dout: out std_logic_vector(8 downto 0));
end component;

component add9 is
port (A,B: in std_logic_vector(8 downto 0); Cin: in std_logic; Co: out
std_logic; Sum: out std_logic_vector(8 downto 0));
end component;

component relu9 is
port (x : in std_logic_vector(8 downto 0); y: out std_logic_vector(8 downto 0));
end component;

component compliment2s
port (Din: in std_logic_vector (8 downto 0); Co: out std_logic; Do: out
std_logic_vector (8 downto 0));
end component;

signal xs11, xs12, xs13, xs14, xs15, xs16, xs17, xs18,
xs21, xs22, xs23, xs24,
xs25, xs26, xs27, xs28, xs31, xs32, xs33, xs34, xs35,
xs36, xs37, xs38, xs41,
xs42, xs43, xs44, xs45, xs46, xs47, xs48, xs51, xs52,
xs53, xs54, xs55, xs56,
xs57, xs58, xs61, xs62, xs63, xs64, xs65, xs66, xs67,
xs68: std_logic_vector(8 downto 0);
signal xs11c, xs13c, xs15c, xs16c, xs18c, xs24c, xs25c,
xs27c, xs28c, xs31c,
```

```
xs33c, xs34c, xs35c,xs36c, xs38c, xs42c, xs43c, xs46c,
xs54c, xs55c, xs56c,
xs61c, xs67c, xs68c: std_logic_vector(8 downto 0);


signal xs58s : std_logic_vector(9 downto 0); --2 shift
signal xs16s, xs21s, xs25s, xs31s, xs32s, xs36s, xs37s, xs46s
: std_logic_vector(10 downto 0); --3 shift
signal xs12s, xs18s, xs26s, xs28s, xs33s, xs34s, xs38s, xs48s, xs54s
: std_logic_vector (11 downto 0); --4 shift
signal xs11s, xs22s, xs35s, xs42s, xs52s, xs56s, xs63s, xs66s, xs67s,
xs68s: std_logic_vector(12 downto 0); --5 shift
signal xs14s, xs15s, xs24s, xs27s, xs41s, xs44s, xs53s, xs55s, xs57s,
xs62s, xs64s: std_logic_vector(13 downto 0); --6 shift
signal xs13s, xs17s, xs23s, xs43s, xs47s, xs51s, xs61s, xs65s
: std_logic_vector(13 downto 0); --7 shift

signal Ca112, Ca134, Ca156, Ca178, Ca114, Ca158, Ca118,Ca212, Ca234,
Ca256, Ca278, Ca214, Ca258, Ca218,Ca312, Ca334, Ca356, Ca378, Ca314,
Ca358, Ca318,Ca412, Ca434, Ca456, Ca478, Ca414, Ca458, Ca418,Ca512,
Ca534, Ca556, Ca578, Ca514, Ca558, Ca518,Ca612, Ca634, Ca656, Ca678,
Ca614, Ca658, Ca618,Cab1, Cab2, Cab3, Cab4, Cab5, Cab6: std_logic;
signal co11, co13, co16, co18, co24, co25, co27, co28, co31, co33,
co34, co35, co36, co38, co42,co43, co46, co54, co55, co56, co61, co67,
co68 : std_logic;
signal ABo1, ABo2, ABo3, ABo4, ABo5, ABo6, Ao112,Ao134, Ao156, Ao178,
Ao114, Ao158, Ao118, Ao212,Ao234, Ao256, Ao278, Ao214, Ao258, Ao218,
Ao312,Ao334, Ao356, Ao378, Ao314, Ao358, Ao318,Ao412,Ao434, Ao456,
Ao478, Ao414, Ao458, Ao418,Ao512,Ao534, Ao556, Ao578, Ao514, Ao558,
Ao518,Ao612,Ao634, Ao656, Ao678, Ao614, Ao658, Ao618
: std_logic_vector(8 downto 0);
```

```
begin
S11: shift5 port map (x1, clk, xs11s);
SS11: S5 port map (xs11s, clk, xs11c);
C11: compliment2s port map (xs11c, co11, xs11);
S12: shift4 port map (x2,clk, xs12s);
SS12: S4 port map (xs12s,clk, xs12);
S13: shift7 port map (x3,clk,  xs13s);
SS13: S7 port map (xs13s, clk, xs13c);
C13: compliment2s port map (xs13c, co13, xs13);
S14: shift6 port map (x4,clk,  xs14s);
SS14: S6 port map (xs14s, clk, xs14);
S15: shift6 port map (x5,clk,  xs15s);
SS15: S6 port map (xs15s, clk, xs15);
S16: shift3 port map (x6, clk,  xs16s);
SS16: S3 port map (xs16s, clk, xs16c);
C16: compliment2s port map (xs16c, co16, xs16);
S17: shift7 port map (x7, clk,  xs17s);
SS17: S7 port map (xs17s, clk, xs17);
S18: shift4 port map (x8, clk,  xs18s);
SS18: S4 port map (xs18s, clk, xs18c);
C18: compliment2s port map (xs18c, co18, xs18);

A112: add9 port map (xs12, xs11, '0', Ca112, Ao112);
A134: add9 port map (xs13, xs14, '0', Ca134, Ao134);
A156: add9 port map (xs16, xs15, '0', Ca156, Ao156);
A178: add9 port map (xs18, xs17, '0', Ca178, Ao178);
A114: add9 port map (Ao112, Ao134, '0', Ca114, Ao114);
A158: add9 port map (Ao156, Ao178, '0', Ca158, Ao158);
A118: add9 port map (Ao114, Ao158, '0', Ca118, Ao118);
AB1: add9 port map (Ao118,"000000100", '0', Cab1, ABo1);

S21: shift3 port map (x1, clk, xs21s);
```

```
SS21: S3 port map (xs21s, clk, xs21);
S22: shift5 port map (x2,clk, xs22s);
SS22: S5 port map (xs22s,clk, xs22);
S23: shift7 port map (x3,clk,  xs23s);
SS23: S7 port map (xs23s, clk, xs23);
S24: shift6 port map (x4,clk,  xs24s);
SS24: S6 port map (xs24s, clk, xs24c);
C24: compliment2s port map (xs24c, co24, xs24);
S25: shift3 port map (x5,clk,  xs25s);
SS25: S3 port map (xs25s, clk, xs25c);
C25: compliment2s port map (xs25c, co25, xs25);
S26: shift4 port map (x6, clk,  xs26s);
SS26: S4 port map (xs26s, clk, xs26);
S27: shift6 port map (x7, clk,  xs27s);
SS27: S6 port map (xs27s, clk, xs27c);
C27: compliment2s port map (xs27c, co27, xs27);
S28: shift4 port map (x8, clk,  xs28s);
SS28: S4 port map (xs28s, clk, xs28c);
C28: compliment2s port map (xs28c, co28, xs28);

A212: add9 port map (xs22, xs21, '0', Ca212, Ao212);
A234: add9 port map (xs23, xs24, '0', Ca234, Ao234);
A256: add9 port map (xs26, xs25, '0', Ca256, Ao256);
A278: add9 port map (xs28, xs27, '0', Ca278, Ao278);
A214: add9 port map (Ao212, Ao234, '0', Ca214, Ao214);
A258: add9 port map (Ao256, Ao278, '0', Ca258, Ao258);
A218: add9 port map (Ao214, Ao258, '0', Ca218, Ao218);
AB2: add9 port map (Ao218, "000001000", '0', Cab2, ABo2);

S31: shift3 port map (x1, clk, xs31s);
SS31: S3 port map (xs31s, clk, xs31c);
C31: compliment2s port map (xs31c, co31, xs31);
```

```
S32: shift3 port map (x2,clk, xs32s);
SS32: S3 port map (xs32s,clk, xs32);
S33: shift4 port map (x3,clk, xs33s);
SS33: S4 port map (xs33s, clk, xs33c);
C33: compliment2s port map (xs33c, co33, xs33);
S34: shift4 port map (x4,clk, xs34s);
SS34: S4 port map (xs34s, clk, xs34c);
C34: compliment2s port map (xs34c, co34, xs34);
S35: shift5 port map (x5,clk, xs35s);
SS35: S5 port map (xs35s, clk, xs35c);
C35: compliment2s port map (xs35c, co35, xs35);
S36: shift3 port map (x6, clk, xs36s);
SS36: S3 port map (xs36s, clk, xs36c);
C36: compliment2s port map (xs36c, co36, xs36);
S37: shift3 port map (x7, clk, xs37s);
SS37: S3 port map (xs37s, clk, xs37);
S38: shift4 port map (x8, clk, xs38s);
SS38: S4 port map (xs38s, clk, xs38c);
C38: compliment2s port map (xs38c, co38, xs38);

A312: add9 port map (xs32, xs31, '0', Ca312, Ao312);
A334: add9 port map (xs33, xs34, '0', Ca334, Ao334);
A356: add9 port map (xs36, xs35, '0', Ca356, Ao356);
A378: add9 port map (xs38, xs37, '0', Ca378, Ao378);
A314: add9 port map (Ao312, Ao334, '0', Ca314, Ao314);
A358: add9 port map (Ao356, Ao378, '0', Ca358, Ao358);
A318: add9 port map (Ao314, Ao358, '0', Ca318, Ao318);
AB3: add9 port map (Ao318, "111111110", '0', Cab3, ABo3);

S41: shift6 port map (x1, clk, xs41s);
SS41: S6 port map (xs41s, clk, xs41);
S42: shift5 port map (x2,clk, xs42s);
```

```
SS42: S5 port map (xs42s,clk, xs42c);
C42: compliment2s port map (xs42c, co42, xs42);
S43: shift7 port map (x3,clk, xs43s);
SS43: S7 port map (xs43s, clk, xs43c);
C43: compliment2s port map (xs43c, co43, xs43);
S44: shift6 port map (x4,clk, xs44s);
SS44: S6 port map (xs44s, clk, xs44);
S46: shift3 port map (x6, clk, xs46s);
SS46: S3 port map (xs46s, clk, xs46c);
C46: compliment2s port map (xs46c, co46, xs46);
S47: shift7 port map (x7, clk, xs47s);
SS47: S7 port map (xs47s, clk, xs47);
S48: shift4 port map (x8, clk, xs48s);
SS48: S4 port map (xs48s, clk, xs48);

A412: add9 port map (xs42, xs41, '0', Ca412, Ao412);
A434: add9 port map (xs43, xs44, '0', Ca434, Ao434);
A456: add9 port map (xs46, "000000000", '0', Ca456, Ao456);
A478: add9 port map (xs48, xs47, '0', Ca478, Ao478);
A414: add9 port map (Ao412, Ao434, '0', Ca414, Ao414);
A458: add9 port map (Ao456, Ao478, '0', Ca458, Ao458);
A418: add9 port map (Ao414, Ao458, '0', Ca418, Ao418);
AB4: add9 port map (Ao418, "000001000", '0', Cab4, ABo4);

S51: shift7 port map (x1, clk, xs51s);
SS51: S7 port map (xs51s, clk, xs51);
S52: shift5 port map (x2,clk, xs52s);
SS52: S5 port map (xs52s,clk, xs52);
S53: shift6 port map (x3,clk, xs53s);
SS53: S6 port map (xs53s, clk, xs53);
S54: shift4 port map (x4,clk, xs54s);
SS54: S4 port map (xs54s, clk, xs54c);
```

```
C54: compliment2s port map (xs54c, co54, xs54);
S55: shift6 port map (x4,clk,  xs55s);
SS55: S6 port map (xs55s, clk, xs55c);
C55: compliment2s port map (xs55c, co55, xs55);
S56: shift5 port map (x6, clk,  xs56s);
SS56: S5 port map (xs56s, clk, xs56c);
C56: compliment2s port map (xs56c, co56, xs56);
S57: shift6 port map (x7, clk,  xs57s);
SS57: S6 port map (xs57s, clk, xs57);
S58: shift2 port map (x8, clk,  xs58s);
SS58: S2 port map (xs58s, clk, xs58);

A512: add9 port map (xs52, xs51, '0', Ca512, Ao512);
A534: add9 port map (xs53, xs54, '0', Ca534, Ao534);
A556: add9 port map (xs56, xs55, '0', Ca556, Ao556);
A578: add9 port map (xs58, xs57, '0', Ca578, Ao578);
A514: add9 port map (Ao512, Ao534, '0', Ca514, Ao514);
A558: add9 port map (Ao556, Ao578, '0', Ca558, Ao558);
A518: add9 port map (Ao514, Ao558, '0', Ca518, Ao518);
AB5: add9 port map (Ao518, "000000010", '0', Cab5, ABo5);

S61: shift7 port map (x1, clk, xs61s);
SS61: S7 port map (xs61s, clk, xs61c);
C61: compliment2s port map (xs61c, co61, xs61);
S62: shift6 port map (x2,clk, xs62s);
SS62: S6 port map (xs62s,clk, xs62);
S63: shift5 port map (x3,clk,  xs63s);
SS63: S5 port map (xs63s, clk, xs63);
S64: shift6 port map (x4,clk,  xs64s);
SS64: S6 port map (xs64s, clk, xs64);
S65: shift7 port map (x4,clk,  xs65s);
SS65: S7 port map (xs65s, clk, xs65);
```

```
S66: shift5 port map (x6, clk,  xs66s);
SS66: S5 port map (xs66s, clk, xs66);
S67: shift5 port map (x7, clk,  xs67s);
SS67: S5 port map (xs67s, clk, xs67c);
C67: compliment2s port map (xs67c, co67, xs67);
S68: shift5 port map (x8, clk,  xs68s);
SS68: S5 port map (xs68s, clk, xs68c);
C68: compliment2s port map (xs68c, co68, xs68);

A612: add9 port map (xs62, xs61, '0', Ca612, Ao612);
A634: add9 port map (xs63, xs64, '0', Ca634, Ao634);
A656: add9 port map (xs66, xs65, '0', Ca656, Ao656);
A678: add9 port map (xs68, xs67, '0', Ca678, Ao678);
A614: add9 port map (Ao612, Ao634, '0', Ca614, Ao614);
A658: add9 port map (Ao656, Ao678, '0', Ca658, Ao658);
A618: add9 port map (Ao614, Ao658, '0', Ca618, Ao618);
AB6: add9 port map (Ao618, "111111000", '0', Cab6, ABo6);


RE1: relu9 port map (ABo1, y1);
RE2: relu9 port map (ABo2, y2);
RE3: relu9 port map (ABo3, y3);
RE4: relu9 port map (ABo4, y4);
RE5: relu9 port map (ABo5, y5);
RE6: relu9 port map (ABo6, y6);


end hidden;

#hidden 2
entity hidden2 is
port (x1, x2, x3, x4, x5, x6: in std_logic_vector (8 downto 0);
```

```
clk: std_logic; y1, y2, y3, y4:out std_logic_vector(8 downto 0));
end hidden2;

architecture hidden of hidden2 is

component shift1 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
        dout: out std_logic_vector(8 downto 0));
end component;

component shift2 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
        dout: out std_logic_vector(9 downto 0));
end component;

component shift3 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
        dout: out std_logic_vector(10 downto 0));
end component;

component shift4 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
        dout: out std_logic_vector(11 downto 0));
end component;

component shift5 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
        dout: out std_logic_vector(12 downto 0));
end component;

component shift6 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
```

```
        dout: out std_logic_vector(13 downto 0));
end component;

component shift7 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
        dout: out std_logic_vector(13 downto 0));
end component;

component S7 is
port (din: in std_logic_vector(13 downto 0); clk: in std_logic;
        dout: out std_logic_vector(8 downto 0));
end component;

component S6 is
port (din: in std_logic_vector(13 downto 0); clk: in std_logic;
        dout: out std_logic_vector(8 downto 0));
end component;

component S5 is
port (din: in std_logic_vector(12 downto 0); clk: in std_logic;
        dout: out std_logic_vector(8 downto 0));
end component;

component S4 is
port (din: in std_logic_vector(11 downto 0); clk: in std_logic;
        dout: out std_logic_vector(8 downto 0));
end component;

component S3 is
port (din: in std_logic_vector(10 downto 0); clk: in std_logic;
        dout: out std_logic_vector(8 downto 0));
end component;
```

```
component S2 is
port (din: in std_logic_vector(9 downto 0); clk: in std_logic;
        dout: out std_logic_vector(8 downto 0));
end component;

component S1 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
        dout: out std_logic_vector(8 downto 0));
end component;

component add9 is
port (A,B: in std_logic_vector(8 downto 0);
Cin: in std_logic; Co: out std_logic;
Sum: out std_logic_vector(8 downto 0));
end component;

component relu9 is
port (x : in std_logic_vector(8 downto 0);
y: out std_logic_vector(8 downto 0));
end component;

component compliment2s
port (Din: in std_logic_vector (8 downto 0);
Co: out std_logic; Do: out std_logic_vector (8 downto 0));
end component;

signal xs11, xs12, xs13, xs14, xs15, xs16,xs21, xs22,
xs23, xs24, xs25, xs26, xs31, xs32, xs33, xs34, xs35,
xs36, xs41, xs42, xs43, xs44, xs45, xs46 : std_logic_vector(8 downto 0);
signal xs13c, xs14c, xs15c, xs16c, xs24c, xs25c, xs33c,
xs44c, xs46c: std_logic_vector(8 downto 0);
```

```
signal xs16s, xs31s : std_logic_vector(10 downto 0); --3 shift
signal xs11s, xs14s, xs46s: std_logic_vector (11 downto 0); --4 shift
signal xs12s, xs13s,  xs22s, xs26s, xs32s, xs34s,
xs36s: std_logic_vector(12 downto 0); --5 shift
signal xs15s, xs21s, xs25s, xs35s, xs41s, xs42s,
xs45s : std_logic_vector(13 downto 0); --6 shift
signal xs23s, xs24s, xs33s, xs43s,
xs44s: std_logic_vector(13 downto 0); --7 shift

signal Ca112, Ca134, Ca156, Ca114, Ca116,Ca212, Ca234,
Ca256, Ca214, Ca216,Ca312, Ca334, Ca356, Ca314, Ca316,
Ca412, Ca434, Ca456, Ca414, Ca416, Cab1, Cab2, Cab3, Cab4 : std_logic;
signal co13, co14, co15, co16, co24, co25, co33, co44, co46  : std_logic;

signal ABo1, ABo2, ABo3, ABo4, Ao112,Ao134, Ao156,
Ao114, Ao116,Ao212,Ao234, Ao256, Ao214, Ao216, Ao312,
Ao334, Ao356, Ao314, Ao316, Ao412,Ao434, Ao456, Ao414,
Ao416 : std_logic_vector(8 downto 0);

begin
S11: shift4 port map (x1, clk, xs11s);
SS11: S4 port map (xs11s, clk, xs11);
S12: shift5 port map (x2,clk, xs12s);
SS12: S5 port map (xs12s,clk, xs12);
S13: shift5 port map (x3,clk,  xs13s);
SS13: S5 port map (xs13s, clk, xs13c);
C13: compliment2s port map (xs13c, co13, xs13);
S14: shift4 port map (x4,clk,  xs14s);
SS14: S4 port map (xs14s, clk, xs14c);
```

```
C14: compliment2s port map (xs14c, co14, xs14);
S15: shift6 port map (x5,clk,  xs15s);
SS15: S6 port map (xs15s, clk, xs15c);
C15: compliment2s port map (xs15c, co15, xs15);
S16: shift3 port map (x6, clk,  xs16s);
SS16: S3 port map (xs16s, clk, xs16c);
C16: compliment2s port map (xs16c, co16, xs16);

A112: add9 port map (xs12, xs11, '0', Ca112, Ao112);
A134: add9 port map (xs13, xs14, '0', Ca134, Ao134);
A156: add9 port map (xs16, xs15, '0', Ca156, Ao156);
A114: add9 port map (Ao112, Ao134, '0', Ca114, Ao114);
A116: add9 port map (Ao114, Ao156, '0', Ca116, Ao116);
AB1: add9 port map (Ao116,"111111100", '0', Cab1, ABo1);

S21: shift6 port map (x1, clk, xs21s);
SS21: S6 port map (xs21s, clk, xs21);
S22: shift5 port map (x2,clk, xs22s);
SS22: S5 port map (xs22s,clk, xs22);
S23: shift7 port map (x3,clk,  xs23s);
SS23: S7 port map (xs23s, clk, xs23);
S24: shift7 port map (x4,clk,  xs24s);
SS24: S7 port map (xs24s, clk, xs24c);
C24: compliment2s port map (xs24c, co24, xs24);
S25: shift6 port map (x5,clk,  xs25s);
SS25: S6 port map (xs25s, clk, xs25c);
C25: compliment2s port map (xs25c, co25, xs25);
S26: shift5 port map (x6, clk,  xs26s);
SS26: S5 port map (xs26s, clk, xs26);

A212: add9 port map (xs22, xs21, '0', Ca212, Ao212);
A234: add9 port map (xs23, xs24, '0', Ca234, Ao234);
```

```
A256: add9 port map (xs26, xs25, '0', Ca256, Ao256);
A214: add9 port map (Ao212, Ao234, '0', Ca214, Ao214);
A216: add9 port map (Ao214, Ao256, '0', Ca216, Ao216);
AB2: add9 port map (Ao216, "111111100", '0', Cab2, ABo2);

S31: shift3 port map (x1, clk, xs31s);
SS31: S3 port map (xs31s, clk, xs31);
S32: shift5 port map (x2,clk, xs32s);
SS32: S5 port map (xs32s,clk, xs32);
S33: shift7 port map (x3,clk,  xs33s);
SS33: S7 port map (xs33s, clk, xs33c);
C33: compliment2s port map (xs33c, co33, xs33);
S34: shift5 port map (x4,clk,  xs34s);
SS34: S5 port map (xs34s, clk, xs34);
S35: shift6 port map (x5,clk,  xs35s);
SS35: S6 port map (xs35s, clk, xs35);
S36: shift5 port map (x6, clk,  xs36s);
SS36: S5 port map (xs36s, clk, xs36);


A312: add9 port map (xs32, xs31, '0', Ca312, Ao312);
A334: add9 port map (xs33, xs34, '0', Ca334, Ao334);
A356: add9 port map (xs36, xs35, '0', Ca356, Ao356);
A314: add9 port map (Ao312, Ao334, '0', Ca314, Ao314);
A316: add9 port map (Ao314, Ao356, '0', Ca316, Ao316);
AB3: add9 port map (Ao316, "111111110", '0', Cab3, ABo3);

S41: shift6 port map (x1, clk, xs41s);
SS41: S6 port map (xs41s, clk, xs41);
S42: shift6 port map (x2,clk, xs42s);
SS42: S6 port map (xs42s,clk, xs42);
S43: shift7 port map (x3,clk,  xs43s);
```

```
SS43: S7 port map (xs43s, clk, xs43);
S44: shift7 port map (x4,clk,  xs44s);
SS44: S7 port map (xs44s, clk, xs44c);
C44: compliment2s port map (xs44c, co44, xs44);
S45: shift6 port map (x5, clk,  xs45s);
SS45: S6 port map (xs45s, clk, xs45);
S46: shift4 port map (x6, clk,  xs46s);
SS46: S4 port map (xs46s, clk, xs46c);
C46: compliment2s port map (xs46c, co46, xs46);


A412: add9 port map (xs42, xs41, '0', Ca412, Ao412);
A434: add9 port map (xs43, xs44, '0', Ca434, Ao434);
A456: add9 port map (xs46, "000000000", '0', Ca456, Ao456);
A414: add9 port map (Ao412, Ao434, '0', Ca414, Ao414);
A416: add9 port map (Ao414, Ao456, '0', Ca416, Ao416);
AB4: add9 port map (Ao416, "000000101", '0', Cab4, ABo4);


RE1: relu9 port map (ABo1, y1);
RE2: relu9 port map (ABo2, y2);
RE3: relu9 port map (ABo3, y3);
RE4: relu9 port map (ABo4, y4);

end hidden;

#hidden 3
library ieee;
use ieee.std_logic_1164.all;

entity hidden3 is
port (x1, x2, x3, x4: in std_logic_vector (8 downto 0);
```

```
clk: std_logic; y1:out std_logic_vector(8 downto 0));
end hidden3;

architecture hidden of hidden3 is


component shift7 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
        dout: out std_logic_vector(13 downto 0));
end component;

component S7 is
port (din: in std_logic_vector(13 downto 0); clk: in std_logic;
        dout: out std_logic_vector(8 downto 0));
end component;

component add9 is
port (A,B: in std_logic_vector(8 downto 0); Cin: in std_logic;
Co: out std_logic; Sum: out std_logic_vector(8 downto 0));
end component;

component sig9 is
port (x: in std_logic_vector (8 downto 0); clk: std_logic;
sig: out std_logic_vector(8 downto 0));
end component;

component compliment2s
port (Din: in std_logic_vector (8 downto 0); Co: out std_logic;
Do: out std_logic_vector (8 downto 0));
end component;

signal xs11, xs12, xs13, xs14,xs21, xs22, xs23, xs24, xs31, xs32,
```

```
xs33, xs34, xs41, xs42, xs43, xs44 : std_logic_vector(8 downto 0);
signal xs11c: std_logic_vector(8 downto 0);

signal xs11s, xs12s, xs13s,
xs14s: std_logic_vector(13 downto 0); --7 shift

signal Ca112, Ca134, Ca114, Cab1 : std_logic;
signal co11  : std_logic;

signal ABo1, Ao112, Ao134, Ao114: std_logic_vector(8 downto 0);

begin
S11: shift7 port map (x1, clk, xs11s);
SS11: S7 port map (xs11s, clk, xs11c);
C11: compliment2s port map (xs11c, co11, xs11);
S12: shift7 port map (x2,clk, xs12s);
SS12: S7 port map (xs12s,clk, xs12);
S13: shift7 port map (x3,clk,  xs13s);
SS13: S7 port map (xs13s, clk, xs13);
S14: shift7 port map (x4,clk,  xs14s);
SS14: S7 port map (xs14s, clk, xs14);

A112: add9 port map (xs12, xs11, '0', Ca112, Ao112);
A134: add9 port map (xs13, xs14, '0', Ca134, Ao134);
A114: add9 port map (Ao112, Ao134, '0', Ca114, Ao114);
AB1: add9 port map (Ao114,"111111001", '0', Cab1, ABo1);

RE1: sig9 port map (ABo1,clk, y1);


end hidden;
```

```
#output

entity shift8_out is
port (din: in std_logic_vector(8 downto 0);
clk: in std_logic;
dout: out std_logic);
end shift8_out;

architecture arch of shift8_out is

begin
process(clk)
begin
if(clk'event and clk='1') then dout <= din(8);
end if;
end process;
end arch;
```

Appendix C

VERILOG CODE: SABINN FOR SA DETECTION

```
module SaBiNN_sleep(
output class,
input clk,
input rst,
input [15:0] x1,
input [15:0] x2
);

// Signals for Layer 1
wire [15:0] x1c, x2c;
wire Ca_L1_o1, Ca_L1_o2, Ca_L1_o3, Ca_L1_o4, Ca_L1_o5,
Ca_L1_o6, Ca_L1_o7, Ca_L1_o8;
wire Ca_L2_o1, Ca_L2_o2, Ca_L2_o3, Ca_L2_o4, Ca_L2_o5,
Ca_L2_o6;
wire [15:0] AB_L1_o1, AB_L1_o2, AB_L1_o3, AB_L1_o4,
AB_L1_o5, AB_L1_o6, AB_L1_o7, AB_L1_o8;
wire [15:0] y1, y2, y3, y4, y5, y6, y7, y8;

// Signals for Layer 2
```

```
wire [15:0] y1c, y2c, y3c, y4c, y5c, y6c, y7c, y8c ;
wire [15:0] AB_L2_o1, AB_L2_o2, AB_L2_o3, AB_L2_o4,
AB_L2_o5, AB_L2_o6;;

wire Ca_L2_1_12, Ca_L2_1_34, Ca_L2_1_56, Ca_L2_1_78,
Ca_L2_1_1234, Ca_L2_1_5678 ;
wire [15:0] AB_L2_1_12, AB_L2_1_34, AB_L2_1_56,
AB_L2_1_78, AB_L2_1_1234, AB_L2_1_5678 ;

wire Ca_L2_2_12, Ca_L2_2_34, Ca_L2_2_56, Ca_L2_2_78,
Ca_L2_2_1234, Ca_L2_2_5678 ;
wire [15:0] AB_L2_2_12, AB_L2_2_34, AB_L2_2_56,
AB_L2_2_78, AB_L2_2_1234, AB_L2_2_5678 ;

wire Ca_L2_3_12, Ca_L2_3_34, Ca_L2_3_56, Ca_L2_3_78,
Ca_L2_3_1234, Ca_L2_3_5678 ;
wire [15:0] AB_L2_3_12, AB_L2_3_34, AB_L2_3_56,
AB_L2_3_78, AB_L2_3_1234, AB_L2_3_5678 ;

wire Ca_L2_4_12, Ca_L2_4_34, Ca_L2_4_56, Ca_L2_4_78,
Ca_L2_4_1234, Ca_L2_4_5678 ;
wire [15:0] AB_L2_4_12, AB_L2_4_34, AB_L2_4_56,
AB_L2_4_78, AB_L2_4_1234, AB_L2_4_5678 ;

wire Ca_L2_5_12, Ca_L2_5_34, Ca_L2_5_56, Ca_L2_5_78,
Ca_L2_5_1234, Ca_L2_5_5678 ;
wire [15:0] AB_L2_5_12, AB_L2_5_34, AB_L2_5_56,
AB_L2_5_78, AB_L2_5_1234, AB_L2_5_5678 ;

wire Ca_L2_6_12, Ca_L2_6_34, Ca_L2_6_56, Ca_L2_6_78,
Ca_L2_6_1234, Ca_L2_6_5678 ;
wire [15:0] AB_L2_6_12, AB_L2_6_34, AB_L2_6_56,
```

```
AB_L2_6_78, AB_L2_6_1234, AB_L2_6_5678 ;


//layer 3 connections
wire [15:0] n1,n2,n3,n4,n5,n6;
wire [15:0] n1c,n2c,n3c,n4c,n5c,n6c;

wire co112, co123,co134,co145,co156;
wire [15:0] s131,s132,s133,s134,s145,s156;

wire co212, co223,co234,co245,co256;
wire [15:0] s231,s232,s233,s234,s245,s256;

wire co312, co323,co334,co345,co356;
wire [15:0] s331,s332,s333,s334,s345,s356;

wire co412, co423,co434,co445,co456;
wire [15:0] s431,s432,s433,s434,s445,s456;

wire co512, co523,co534,co545,co556;
wire [15:0] s531,s532,s533,s534,s545,s556;

wire co612, co623,co634,co645,co656;
wire [15:0] s631,s632,s633,s634,s645,s656;

// layer 4 connections
wire[15:0] y31,y32,y33,y34,y35,y36;
wire[15:0] y31c,y32c,y33c,y34c,y35c,y36c;

wire co6112, co6123,co6134,co6145,co6156;
wire [15:0] s6131,s6132,s6133,s6134,s6145;
```

```
wire co6212, co6223,co6234,co6245,co6256;
wire [15:0] s6231,s6232,s6233,s6234,s6245;

wire co6312, co6323,co6334,co6345,co6356;
wire [15:0] s6331,s6332,s6333,s6334,s6345;

wire co6412, co6423,co6434,co6445,co6456;
wire [15:0] s6431,s6432,s6433,s6434,s6445;


//output layer connection
wire[15:0] y41,y42,y43,y44;

wire co012, co023,co034;
wire [15:0] s01,s02,s03;
wire [15:0] sig_out;

    //------------------Hidden Layer1------------
    // Layer 1 Complements
complement_rejfdb CP_L1_x1 (x1, x1c);
complement_rejfdb CP_L1_x2 (x2, x2c);


    // Layer 1 Adders
clockadder_rejfdb CA_L1_1(Ca_L1_o1, AB_L1_o1, x1c, x2c, 1'b0);
clockadder_rejfdb CA_L1_2(Ca_L1_o2, AB_L1_o2, x1c, x2c, 1'b0);
clockadder_rejfdb CA_L1_3(Ca_L1_o3, AB_L1_o3, x1, x2c, 1'b0);
clockadder_rejfdb CA_L1_4(Ca_L1_o4, AB_L1_o4, x1c, x2, 1'b0);
clockadder_rejfdb CA_L1_5(Ca_L1_o5, AB_L1_o5, x1c, x2, 1'b0);
clockadder_rejfdb CA_L1_6(Ca_L1_o6, AB_L1_o6, x1c, x2c, 1'b0);
clockadder_rejfdb CA_L1_7(Ca_L1_o7, AB_L1_o7, x1c, x2, 1'b0);
clockadder_rejfdb CA_L1_8(Ca_L1_o8, AB_L1_o8, x1c, x2, 1'b0);
```

```
    // Layer 1 Activation Function
relu16 RE_L1_o1(AB_L1_o1, y1);
relu16 RE_L1_o2(AB_L1_o2, y2);
relu16 RE_L1_o3(AB_L1_o3, y3);
relu16 RE_L1_o4(AB_L1_o4, y4);
relu16 RE_L1_o5(AB_L1_o5, y5);
relu16 RE_L1_o6(AB_L1_o6, y6);
relu16 RE_L1_o7(AB_L1_o7, y7);
relu16 RE_L1_o8(AB_L1_o8, y8);

    //------------------------- Hidden Layer2-----------------
    // Layer 2 Complements
complement_rejfdb CP_L2_y1 (y1, y1c);
complement_rejfdb CP_L2_y2 (y2, y2c);
complement_rejfdb CP_L2_y3 (y3, y3c);
complement_rejfdb CP_L2_y4 (y4, y4c);
complement_rejfdb CP_L2_y5 (y5, y5c);
complement_rejfdb CP_L2_y6 (y6, y6c);
complement_rejfdb CP_L2_y7 (y7, y7c);
complement_rejfdb CP_L2_y8 (y8, y8c);

    // Layer 2 Adders

    // Neuron 1
clockadder_rejfdb CA_L2_11(Ca_L2_1_12, AB_L2_1_12, y1, y2c, 1'b0);
clockadder_rejfdb CA_L2_12(Ca_L2_1_34, AB_L2_1_34, y3, y4c, 1'b0);
clockadder_rejfdb CA_L2_13_56(Ca_L2_1_56, AB_L2_1_56, y5c, y6,1'b0);
clockadder_rejfdb CA_L2_14_78(Ca_L2_1_78, AB_L2_1_78, y7, y8c,1'b0);

clockadder_rejfdb CA_L2_15(Ca_L2_1_1234, AB_L2_1_1234,
AB_L2_1_12, AB_L2_1_34, 1'b0);
```

```
clockadder_rejfdb CA_L2_16(Ca_L2_1_5678, AB_L2_1_5678,
AB_L2_1_56, AB_L2_1_78, 1'b0);
clockadder_rejfdb CA_L2_17(Ca_L2_o1, AB_L2_o1, AB_L2_1_1234,
AB_L2_1_5678, 1'b0);

    // Neuron 2
clockadder_rejfdb CA_L2_21(Ca_L2_2_12, AB_L2_2_12, y1, y2c, 1'b0);
clockadder_rejfdb CA_L2_22(Ca_L2_2_34, AB_L2_2_34, y3c, y4c, 1'b0);
clockadder_rejfdb CA_L2_23(Ca_L2_2_56, AB_L2_2_56, y5, y6c,  1'b0);
clockadder_rejfdb CA_L2_24(Ca_L2_2_78, AB_L2_2_78, y7, y8, 1'b0);

clockadder_rejfdb CA_L2_25(Ca_L2_2_1234, AB_L2_2_1234, AB_L2_2_12,
AB_L2_2_34, 1'b0);
clockadder_rejfdb CA_L2_26(Ca_L2_2_5678, AB_L2_2_5678, AB_L2_2_56,
AB_L2_2_78, 1'b0);
clockadder_rejfdb CA_L2_27(Ca_L2_o2, AB_L2_o2, AB_L2_2_1234,
AB_L2_2_5678, 1'b0);

    // Neuron 3
clockadder_rejfdb CA_L2_31(Ca_L2_3_12, AB_L2_3_12, y1, y2c, 1'b0);
clockadder_rejfdb CA_L2_32(Ca_L2_3_34, AB_L2_3_34, y3, y4c, 1'b0);
clockadder_rejfdb CA_L2_33(Ca_L2_3_56, AB_L2_3_56, y5c, y6, 1'b0);
clockadder_rejfdb CA_L2_34(Ca_L2_3_78, AB_L2_3_78, y7c, y8c, 1'b0);

clockadder_rejfdb CA_L2_35(Ca_L2_3_1234, AB_L2_3_1234, AB_L2_3_12,
AB_L2_3_34, 1'b0);
clockadder_rejfdb CA_L2_36(Ca_L2_3_5678, AB_L2_3_5678, AB_L2_3_56,
AB_L2_3_78, 1'b0);
clockadder_rejfdb CA_L2_37(Ca_L2_o3, AB_L2_o3, AB_L2_3_1234,
AB_L2_3_5678, 1'b0);

    // Neuron 4
```

```
clockadder_rejfdb CA_L2_41(Ca_L2_4_12, AB_L2_4_12, y1, y2c, 1'b0);
clockadder_rejfdb CA_L2_42(Ca_L2_4_34, AB_L2_4_34, y3c, y4c, 1'b0);
clockadder_rejfdb CA_L2_43(Ca_L2_4_56, AB_L2_4_56, y5, y6, 1'b0);
clockadder_rejfdb CA_L2_44(Ca_L2_4_78, AB_L2_4_78, y7, y8, 1'b0);

clockadder_rejfdb CA_L2_45(Ca_L2_4_1234, AB_L2_4_1234, AB_L2_4_12,
AB_L2_4_34, 1'b0);
clockadder_rejfdb CA_L2_46(Ca_L2_4_5678, AB_L2_4_5678, AB_L2_4_56,
AB_L2_4_78, 1'b0);
clockadder_rejfdb CA_L2_47(Ca_L2_o4, AB_L2_o4, AB_L2_4_1234,
AB_L2_4_5678, 1'b0);

    // Neuron 5
clockadder_rejfdb CA_L2_51(Ca_L2_5_12, AB_L2_5_12, y1, y2,  1'b0);
clockadder_rejfdb CA_L2_52(Ca_L2_5_34, AB_L2_5_34, y3, y4, 1'b0);
clockadder_rejfdb CA_L2_53(Ca_L2_5_56, AB_L2_5_56, y5c, y6, 1'b0);
clockadder_rejfdb CA_L2_54(Ca_L2_5_78, AB_L2_5_78, y7c, y8c, 1'b0);

clockadder_rejfdb CA_L2_55(Ca_L2_5_1234, AB_L2_5_1234, AB_L2_5_12,
AB_L2_5_34, 1'b0);
clockadder_rejfdb CA_L2_56(Ca_L2_5_5678, AB_L2_5_5678, AB_L2_5_56,
AB_L2_5_78, 1'b0);
clockadder_rejfdb CA_L2_57(Ca_L2_o5, AB_L2_o5, AB_L2_5_1234,
AB_L2_5_5678, 1'b0);

    // Neuron 6
clockadder_rejfdb CA_L2_61(Ca_L2_6_12, AB_L2_6_12, y1c, y2, 1'b0);
clockadder_rejfdb CA_L2_62(Ca_L2_6_34, AB_L2_6_34, y3, y4, 1'b0);
clockadder_rejfdb CA_L2_63(Ca_L2_6_56, AB_L2_6_56, y5c, y6c, 1'b0);
clockadder_rejfdb CA_L2_64(Ca_L2_6_78, AB_L2_6_78, y7c, y8, 1'b0);

clockadder_rejfdb CA_L2_65(Ca_L2_6_1234, AB_L2_6_1234, AB_L2_6_12,
```

```
AB_L2_6_34, 1'b0);
clockadder_rejfdb CA_L2_66(Ca_L2_6_5678, AB_L2_6_5678, AB_L2_6_56,
AB_L2_6_78, 1'b0);
clockadder_rejfdb CA_L2_67(Ca_L2_o6, AB_L2_o6, AB_L2_6_1234,
AB_L2_6_5678, 1'b0);


// Layer 2 Activation Function
relu16 RE_L2_o1(AB_L2_o1, n1);
relu16 RE_L2_o2(AB_L2_o2, n2);
relu16 RE_L2_o3(AB_L2_o3, n3);
relu16 RE_L2_o4(AB_L2_o4, n4);
relu16 RE_L2_o5(AB_L2_o5, n5);
relu16 RE_L2_o6(AB_L2_o6, n6);

//------------------- Hidden Layer3-----------------------------------
//compliment the 12 inputs
complement_rejfdb CP_L3_n1 (n1, n1c);
complement_rejfdb CP_L3_n2 (n2, n2c);
complement_rejfdb CP_L3_n3 (n3, n3c);
complement_rejfdb CP_L3_n4 (n4, n4c);
complement_rejfdb CP_L3_n5 (n5, n5c);
complement_rejfdb CP_L3_n6 (n6, n6c);

//layer 3
//Neuron1
clockadder_rejfdb CA_L3_6_11(co112, s131, n1c, n2c,  1'b0);
clockadder_rejfdb CA_L3_6_12(co123, s132, s131, n3, 1'b0);
clockadder_rejfdb CA_L3_6_13(co134, s133, s132, n4c, 1'b0);
clockadder_rejfdb CA_L3_6_14(co145, s134, s133, n5c, 1'b0);
clockadder_rejfdb CA_L3_6_15(co156, s145, s134, n6, 1'b0);
```

```
//Neuron 2
clockadder_rejfdb CA_L3_6_21(co212, s231, n1c, n2, 1'b0);
clockadder_rejfdb CA_L3_6_22(co223, s232, s231, n3, 1'b0);
clockadder_rejfdb CA_L3_6_23(co234, s233, s232, n4c, 1'b0);
clockadder_rejfdb CA_L3_6_24(co245, s234, s233, n5c, 1'b0);
clockadder_rejfdb CA_L3_6_25(co256, s245, s234, n6, 1'b0);


//Neuron3
clockadder_rejfdb CA_L3_6_31(co312, s331, n1c, n2, 1'b0);
clockadder_rejfdb CA_L3_6_32(co323, s332, s331, n3c, 1'b0);
clockadder_rejfdb CA_L3_6_33(co334, s333, s332, n4c, 1'b0);
clockadder_rejfdb CA_L3_6_34(co345, s334, s333, n5, 1'b0);
clockadder_rejfdb CA_L3_6_35(co356, s345, s334, n6c, 1'b0);


//Neuron4
clockadder_rejfdb CA_L3_6_41(co412, s431, n1, n2c, 1'b0);
clockadder_rejfdb CA_L3_6_42(co423, s432, s431, n3,  1'b0);
clockadder_rejfdb CA_L3_6_43(co434, s433, s432, n4, 1'b0);
clockadder_rejfdb CA_L3_6_44(co445, s434, s433, n5, 1'b0);
clockadder_rejfdb CA_L3_6_45(co456, s445, s434, n6c, 1'b0);


//Neuron5
clockadder_rejfdb CA_L3_6_51(co512, s531, n1c, n2c,  1'b0);
clockadder_rejfdb CA_L3_6_52(co523, s532, s531, n3c,  1'b0);
clockadder_rejfdb CA_L3_6_53(co534, s533, s532, n4,  1'b0);
clockadder_rejfdb CA_L3_6_54(co545, s534, s533, n5,  1'b0);
clockadder_rejfdb CA_L3_6_55(co556, s545, s534, n6,  1'b0);
clockadder_rejfdb CA_L3_6_56(co567, s556, s545, n7,  1'b0);
```

```
clockadder_rejfdb CA_L3_6_57(co578, s567, s556, n8c, 1'b0);
clockadder_rejfdb CA_L3_6_58(co589, s578, s567, n9,  1'b0);
clockadder_rejfdb CA_L3_6_59(co5910,s589, s578, n10,  1'b0);
clockadder_rejfdb CA_L3_6_510(co51011,s5910, s589, n11, 1'b0);
clockadder_rejfdb CA_L3_6_511(co51112, s51011, s5910, n12,  1'b0);

//Neuron6
clockadder_rejfdb CA_L3_6_61(co612, s631, n1c, n2c, 1'b0);
clockadder_rejfdb CA_L3_6_62(co623, s632, s631, n3,  1'b0);
clockadder_rejfdb CA_L3_6_63(co634, s633, s632, n4,  1'b0);
clockadder_rejfdb CA_L3_6_64(co645, s634, s633, n5c,  1'b0);
clockadder_rejfdb CA_L3_6_65(co656, s645, s634, n6c,  1'b0);
clockadder_rejfdb CA_L3_6_66(co667, s656, s645, n7,  1'b0);
clockadder_rejfdb CA_L3_6_67(co678, s667, s656, n8,  1'b0);
clockadder_rejfdb CA_L3_6_68(co689, s678, s667, n9c,  1'b0);
clockadder_rejfdb CA_L3_6_69(co6910,s689, s678, n10c,  1'b0);
clockadder_rejfdb CA_L3_6_610(co61011,s6910, s689, n11,  1'b0);
clockadder_rejfdb CA_L3_6_611(co61112, s61011, s6910, n12, 1'b0);

//relu out
relu16 RE_L3_o1(s11011, y31);
relu16 RE_L3_o2(s21011, y32);
relu16 RE_L3_o3(s31011, y33);
relu16 RE_L3_o4(s41011, y34);
relu16 RE_L3_o5(s51011, y35);
relu16 RE_L3_o6(s61011, y36);

//------------------------- Hidden Layer 4------------
//compliment the 6 input
complement_rejfdb CP_L4_n1 (y31, y31c);
complement_rejfdb CP_L4_n2 (y32, y32c);
complement_rejfdb CP_L4_n3 (y33, y33c);
```

```
complement_rejfdb CP_L4_n4 (y34, y34c);
complement_rejfdb CP_L4_n5 (y35, y35c);
complement_rejfdb CP_L4_n6 (y36, y36c);

//Neuron 1
clockadder_rejfdb CA_L3_4_11(co6112, s6131, y31c, y32c,  1'b0);
clockadder_rejfdb CA_L3_4_12(co6123, s6132, s6131, y33c,  1'b0);
clockadder_rejfdb CA_L3_4_13(co6134, s6133, s6132, y34c,  1'b0);
clockadder_rejfdb CA_L3_4_14(co6145, s6134, s6133, y35c, 1'b0);
clockadder_rejfdb CA_L3_4_15(co6156, s6145, s6134, y36c,  1'b0);

//Neuron 2
clockadder_rejfdb CA_L3_4_21(co6212, s6231, y31c, y32c, 1'b0);
clockadder_rejfdb CA_L3_4_22(co6223, s6232, s6231, y33c, 1'b0);
clockadder_rejfdb CA_L3_4_23(co6234, s6233, s6232, y34c, 1'b0);
clockadder_rejfdb CA_L3_4_24(co6245, s6234, s6233, y35c,  1'b0);
clockadder_rejfdb CA_L3_4_25(co6256, s6245, s6234, y36c,  1'b0);

//Neuron3
clockadder_rejfdb CA_L3_4_31(co6312, s6331, y31, y32,  1'b0);
clockadder_rejfdb CA_L3_4_32(co6323, s6332, s6331, y33c, 1'b0);
clockadder_rejfdb CA_L3_4_33(co6334, s6333, s6332, y34c,  1'b0);
clockadder_rejfdb CA_L3_4_34(co6345, s6334, s6333, y35,  1'b0);
clockadder_rejfdb CA_L3_4_35(co6356, s6345, s6334, y36c, 1'b0);

//Neuron4
clockadder_rejfdb CA_L3_4_41(co6412, s6431, y31, y32,  1'b0);
clockadder_rejfdb CA_L3_4_42(co6423, s6432, s6431, y33c, 1'b0);
clockadder_rejfdb CA_L3_4_43(co6434, s6433, s6432, y34c, 1'b0);
clockadder_rejfdb CA_L3_4_44(co6445, s6434, s6433, y35,  1'b0);
clockadder_rejfdb CA_L3_4_45(co6456, s6445, s6434, y36c, 1'b0);
```

```
//relu out
relu16 RE_L4_o1(s6145, y41);
relu16 RE_L4_o2(s6245, y42);
relu16 RE_L4_o3(s6345, y43);
relu16 RE_L4_o4(s6445, y44);

//---------- output Layer----------

//Neuron1
clockadder_rejfdb CA_L3_out_41(co012, s01, y41, y42, 1'b0);
clockadder_rejfdb CA_L3_out_42(co023, s02, s01, y43,  1'b0);
clockadder_rejfdb CA_L3_out_43(co034, s03, s02, y44,  1'b0);

//sigmoid out
sigmoid_ohbk4 SIG_L_out(sig_out, clk,rst, s03);

//decision
convert16_to_1 conv16(sig_out, class);

endmodule

//sigmoid function
module sigmoid_ohbk4(sig, clk, rst, x);
input [15:0] x;
input clk;
input rst;
output[15:0] sig;
wire [1:0] s;
wire [15:0] x1,x2,x3;
wire [15:0] c1,c2;
wire co;
```

```
begin
shiftback2bit S2(x1, x, clk,rst);
shiftback3bit S3(x2, x, clk, rst);
shiftback5bit S5(x3, x, clk, rst);
ifstatement IS(x, s);
compare comp(s,x1,x2,x3,c1);
compare2 comp2(s, c2);
clockadder_rejfdb ad(co, sig, c1, c2,  1'b0);
end

endmodule

// 5bit shifter

module shiftback5bit(
dout,
din,
clk,
rst
);
output reg [15:0] dout;
input [15:0] din;
input clk;
input rst;
always @(posedge clk)
begin
if (rst == 1'b1) dout<= 16'b0000000000000000;
else if (clk == 1'b1) dout[10:0] <= din[15:5];
dout[15:11] <= 5'b00000;
end

endmodule
```

```
// 2bit shifter
module shiftback2bit(
dout,
din,
clk,
rst
);
output reg [15:0] dout;
input [15:0] din;
input  clk, rst;
always @(posedge clk)
begin
if (rst == 1'b1) dout <= 16'b0000000000000000;
else if (clk == 1'b1) dout[13:0] <= din[15:2];
dout [15:14] <= 2'b00;
end

endmodule

// 3bit shifter
module shiftback3bit(
dout,
din,
clk,
rst
);

output reg [15:0] dout;
input [15:0] din;
input clk,rst;
always @(posedge clk)
```

```
begin
if (rst == 1'b1) dout <= 16'b0000000000000000;
else if (clk == 1'b1) dout[12:0] <= din[15:3];
dout[15:13] <= 3'b000;
end

endmodule

//compare module
module compare(
s, x1, x2, x3, y
    );
input [1:0] s;
input [15:0] x1,x2,x3;
output reg[15:0] y;



always @(s,x1,x2,x3)
begin
case(s)
2'b00: y = x1; //2bit shifter
2'b01: y = x2; //3bit shifter
2'b10: y = x3; //5bit shifter
default: y = 0;
endcase
end
endmodule

//2nd compare
module compare2(s, y);

input [1:0] s;
```

```
output reg [15:0] y;
always @(s)
begin
case(s)
2'b00: y = 16'b0000001000000000; //2bit shifter
2'b01: y = 16'b0000001010000000; //3bit shifter
2'b10: y = 16'b0000001101100000; //5bit shifter
2'b11: y = 16'b0000010000000000; //direct 1024
default: y = 0;
endcase
end
endmodule

// if-statement
module ifstatement(x,y);

input [15:0] x;
output reg [1:0] y;

always @(x)
if (x > 16'b0000000000000000 &&  x< 16'b0000010000000000) y <= 2'b00 ;
else if (x == 16'b0000000000000000) y <= 2'b00 ;
else if (x > 16'b0000010000000000 &&  x< 16'b0000100110000000) y <= 2'b01 ;
else if (x > 16'b0000100110000000 &&  x< 16'b0001010000000000) y <= 2'b10 ;
else if (x > 16'b0001010000000000) y <= 2'b11 ;

endmodule

//adder 16 bit
module clockadder_rejfdb(
    output cout,
    output [15:0]s,
```

```
    input [15:0]a,
    input [15:0]b,
    input cin
    );

wire [15:0] bin;
    assign bin[0]=b[0]^cin;
    assign bin[1]=b[1]^cin;
    assign bin[2]=b[2]^cin;
    assign bin[3]=b[3]^cin;
    assign bin[4]=b[4]^cin;
    assign bin[5]=b[5]^cin;
    assign bin[6]=b[6]^cin;
    assign bin[7]=b[7]^cin;
    assign bin[8]=b[8]^cin;
    assign bin[9]=b[9]^cin;
    assign bin[10]=b[10]^cin;
    assign bin[11]=b[11]^cin;
    assign bin[12]=b[12]^cin;
    assign bin[13]=b[13]^cin;
    assign bin[14]=b[14]^cin;
    assign bin[15]=b[15]^cin;

wire [15:1] carry;
 Full_Adder FA0(carry[1],s[0],a[0],bin[0],cin);
 Full_Adder FA1(carry[2],s[1],a[1],bin[1],carry[1]);
 Full_Adder FA2(carry[3],s[2],a[2],bin[2],carry[2]);
 Full_Adder FA3(carry[4],s[3],a[3],bin[3],carry[3]);
 Full_Adder FA4(carry[5],s[4],a[4],bin[4],carry[4]);
 Full_Adder FA5(carry[6],s[5],a[5],bin[5],carry[5]);
 Full_Adder FA6(carry[7],s[6],a[6],bin[6],carry[6]);
 Full_Adder FA7(carry[8],s[7],a[7],bin[7],carry[7]);
```

```
 Full_Adder FA8(carry[9],s[8],a[8],bin[8],carry[8]);
 Full_Adder FA9(carry[10],s[9],a[9],bin[9],carry[9]);
 Full_Adder FA10(carry[11],s[10],a[10],bin[10],carry[10]);
 Full_Adder FA11(carry[12],s[11],a[11],bin[11],carry[11]);
 Full_Adder FA12(carry[13],s[12],a[12],bin[12],carry[12]);
 Full_Adder FA13(carry[14],s[13],a[13],bin[13],carry[13]);
 Full_Adder FA14(carry[15],s[14],a[14],bin[14],carry[14]);
 Full_Adder FA15(cout,s[15],a[15],bin[15],carry[15]);

endmodule

module Full_Adder(
    cout,
    sum,
    ain,
    bin,
    cin
    );

output reg sum, cout;
input wire ain, bin, cin;



always @(ain, bin, cin)
begin
sum = ain^bin^cin;
cout=  (ain&bin)|(ain&cin)|(bin&cin);

end
endmodule

// 2s compliment
```

```
module complement_rejfdb(in,out);
    input [15:0]in;
    output [15:0]out;
    assign out=(~in);
endmodule

//relu
module relu16(
input [15:0] din_relu,
output[15:0] dout_relu
    );
    assign dout_relu = (din_relu[15]==0)? din_relu: 0;
endmodule

//decision function
module convert16_to_1(sig_out, class);

input [15:0] sig_out;
output reg class;

always @(sig_out)
if(sig_out< 16'b0000001000000000 ||
sig_out == 16'b0000001000000000) class <=1'b0;
else if (sig_out > 16'b0000001000000000) class <= 1'b1;
endmodule
```

Appendix D

VHDL CODE: DEEPSAC FOR DIABETES PREDICTION

```
    # top level code

    library ieee;
use ieee.std_logic_1164.all;

entity neural_net is
port (x1, x2, x3, x4, x5, x6, x7, x8: in std_logic_vector (7 downto 0);
clk: std_logic; sys_out :out std_logic);
end neural_net;

architecture neural of neural_net is

component hiddenshift is
port (x1, x2, x3, x4, x5, x6, x7, x8: in std_logic_vector (7 downto 0);
clk: std_logic; y1, y2, y3, y4, y5, y6, y7, y8, y9, y10, y11,
y12:out std_logic_vector(7 downto 0));
end component;

component Hidden_Layer2 is
port (x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11,
x12: in std_logic_vector (7 downto 0); clk: std_logic; y1, y2, y3,
y4, y5, y6, y7, y8:out std_logic_vector(7 downto 0));
```

```
end component;

component Hidden_Layer3 is
port (x1, x2, x3, x4, x5, x6, x7, x8: in std_logic_vector (7 downto 0);
clk: std_logic; y1, y2, y3, y4 :out std_logic_vector(7 downto 0));
end component;

component Out_Layer is
port (x1, x2, x3, x4 : in std_logic_vector (7 downto 0);
clk: std_logic; y1 :out std_logic_vector(7 downto 0));
end component;

component sft8 is
port (din: in std_logic_vector(7 downto 0); clk: in std_logic;
        dout: out std_logic);
end component;

signal h11, h12, h13, h14, h15, h16, h17, h18, h19, h110,
        h111, h112 : std_logic_vector (7 downto 0);
signal h21, h22, h23, h24, h25, h26, h27, h28 : std_logic_vector (7 downto 0);
signal h31, h32, h33, h34, y1 : std_logic_vector (7 downto 0);



begin
HL1: hiddenshift port map (x1, x2, x3, x4, x5, x6, x7, x8, clk,
h11, h12, h13, h14, h15,
        h16, h17, h18, h19, h110, h111, h112);

HL2: Hidden_Layer2 port map (h11, h12, h13, h14, h15, h16, h17,
h18, h19, h110, h111, h112,
        clk, h21, h22, h23, h24, h25, h26, h27, h28);
```

```
HL3: Hidden_Layer3 port map (h21, h22, h23, h24, h25, h26, h27, h28,
        clk, h31, h32, h33, h34);

HO: Out_Layer port map (h31, h32, h33, h34, clk, y1);

S_OUT: sft8 port map(y1, clk, sys_out);

end neural;

#hidden shift

entity hiddenshift is
port (x1, x2, x3, x4, x5, x6, x7, x8: in std_logic_vector
(7 downto 0); clk: std_logic; y1, y2, y3, y4, y5, y6, y7, y8, y9,
y10, y11, y12:out std_logic_vector(7 downto 0));
end hiddenshift;

architecture hidden of hiddenshift is

component shift1 is
port (din: in std_logic_vector(7 downto 0); clk: in std_logic;
        dout: out std_logic_vector(8 downto 0));
end component;

component shift2 is
port (din: in std_logic_vector(7 downto 0); clk: in std_logic;
        dout: out std_logic_vector(9 downto 0));
end component;

component shift3 is
port (din: in std_logic_vector(7 downto 0); clk: in std_logic;
        dout: out std_logic_vector(10 downto 0));
```

```
end component;

component shift4 is
port (din: in std_logic_vector(7 downto 0); clk: in std_logic;
        dout: out std_logic_vector(11 downto 0));
end component;

component shift5 is
port (din: in std_logic_vector(7 downto 0); clk: in std_logic;
        dout: out std_logic_vector(12 downto 0));
end component;

component shift6 is
port (din: in std_logic_vector(7 downto 0); clk: in std_logic;
        dout: out std_logic_vector(13 downto 0));
end component;

component S6 is
port (din: in std_logic_vector(13 downto 0); clk: in std_logic;
        dout: out std_logic_vector(7 downto 0));
end component;

component S5 is
port (din: in std_logic_vector(12 downto 0); clk: in std_logic;
        dout: out std_logic_vector(7 downto 0));
end component;

component S4 is
port (din: in std_logic_vector(11 downto 0); clk: in std_logic;
        dout: out std_logic_vector(7 downto 0));
end component;
```

```
component S3 is
port (din: in std_logic_vector(10 downto 0); clk: in std_logic;
        dout: out std_logic_vector(7 downto 0));
end component;

component S2 is
port (din: in std_logic_vector(9 downto 0); clk: in std_logic;
        dout: out std_logic_vector(7 downto 0));
end component;

component S1 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
        dout: out std_logic_vector(7 downto 0));
end component;

component add8 is
port (A,B: in std_logic_vector(7 downto 0); Cin: in std_logic;
Co: out std_logic; Sum: out std_logic_vector(7 downto 0));
end component;

component relu8 is
port (x : in std_logic_vector(7 downto 0); y:
out std_logic_vector(7 downto 0));
end component;

component compliment2s
port (Din: in std_logic_vector (7 downto 0); Co: out std_logic;
Do: out std_logic_vector (7 downto 0));
end component;

signal xs72s: std_logic_vector(13 downto 0); -- 6 shift
signal xs127s,xs121s,xs117s, xs115s,xs118s,xs112s,xs108s,xs105s,xs106s,
```

```
xs96s,xs91s,xs86s,xs87s,
xs88s,xs83s,xs81s,xs78s,xs71s,xs64s,xs66s,xs67s,xs62s,xs63s,xs51s,
xs48s,xs46s,xs44s,xs41s,xs12s,xs17s,xs21s,xs22s,xs24s,xs25s,xs26s,
xs27s,xs28s,xs33s,xs35s : std_logic_vector(12 downto 0); -- 5 shift
signal xs128s,xs125s,xs122s,xs114s,xs107s,xs104s,xs101s,xs98s,xs93s,xs94s,
xs95s,xs84s,xs85s,xs73s,xs74s,xs75s,xs58s,xs55s,xs56s,xs52s,xs47s,
xs42s,xs43s, xs36s, xs38s: std_logic_vector(11 downto 0); -- 4 shift
signal xs123s,xs102s,xs97s,xs92s,xs77s,xs54s,xs11s,xs16s,xs23s,xs32s,
xs37s: std_logic_vector(10 downto 0); -- 3 shift
signal xs113s,xs111s,xs103s,xs82s,xs13s, xs14s, xs15s,
xs18s : std_logic_vector(9 downto 0); -- 2 shift
signal xs124s,xs116s,xs76s,xs65s,xs68s,xs61s,xs57s,xs53s,xs45s,
xs34s: std_logic_vector(8 downto 0); -- 1 shift

signal xs11c, xs12c, xs15c, xs16c, xs17c, xs18c, xs11, xs12, xs13,
xs14, xs15, xs16, xs17, xs18: std_logic_vector(7 downto 0);
signal Ao112, Ao134, Ao156, Ao178, Ao11234, Ao15678,
ABo1: std_logic_vector (7 downto 0);
signal co11, co12, co15, co16, co17, co18, Ca112, Ca134,
Ca156, Ca178, Ca11234, Ca15678, Cao1: std_logic;
signal xs24c, xs25c, xs21, xs22, xs23, xs24, xs25, xs26,
xs27, xs28: std_logic_vector(7 downto 0);
signal Ao212, Ao234, Ao256, Ao278, Ao21234, Ao25678, ABo2,
AAo2: std_logic_vector (7 downto 0);
signal co24, co25, Ca212, Ca234, Ca256, Ca278, Ca21234,
Ca25678, Cao2, Cab2: std_logic;
signal xs36c, xs32, xs33, xs34, xs35, xs36, xs37,
xs38: std_logic_vector(7 downto 0);
signal Ao312, Ao334, Ao356, Ao378, Ao31234, Ao35678,
AAo3: std_logic_vector (7 downto 0);
signal co36, Ca312, Ca334, Ca356, Ca378, Ca31234, Ca35678,
Cao3, Cab3: std_logic;
```

```
signal xs43c,xs45c, xs48c, xs41, xs42, xs43, xs44, xs45, xs46,
xs47, xs48: std_logic_vector(7 downto 0);
signal Ao412, Ao434, Ao456, Ao478, Ao41234, Ao45678,
AAo4: std_logic_vector (7 downto 0);
signal co43,co45,co48,Ca412, Ca434, Ca456, Ca478, Ca41234,
Ca45678, Cao4, Cab4: std_logic;
signal xs53c, xs55c,xs56c, xs57c, xs58c, xs51, xs52, xs53,
xs54, xs55, xs56, xs57, xs58: std_logic_vector(7 downto 0);
signal Ao512, Ao534, Ao556, Ao578, Ao51234, Ao55678,
AAo5: std_logic_vector (7 downto 0);
signal co53,co55,co56,co57,co58,Ca512, Ca534, Ca556,
Ca578, Ca51234, Ca55678, Cao5, Cab5: std_logic;
signal xs61c,xs61, xs62, xs63, xs64, xs65, xs66, xs67,
xs68: std_logic_vector(7 downto 0);
signal Ao612, Ao634, Ao656, Ao678, Ao61234,
Ao65678: std_logic_vector (7 downto 0);
signal co61, Ca612, Ca634, Ca656, Ca678, Ca61234,
Ca65678, Cao6: std_logic;
signal xs71c,xs73c,xs76c, xs71, xs72, xs73, xs74,
xs75, xs76, xs77, xs78: std_logic_vector(7 downto 0);
signal Ao712, Ao734, Ao756, Ao778, Ao71234, Ao75678,
AAo7: std_logic_vector (7 downto 0);
signal co71,co73, co76, Ca712, Ca734, Ca756, Ca778, Ca71234,
Ca75678, Cao7, Cab7: std_logic;
signal xs82c,xs84c,xs87c,xs88c, xs81, xs82, xs83, xs84,
xs85, xs86, xs87, xs88: std_logic_vector(7 downto 0);
signal Ao812, Ao834, Ao856, Ao878, Ao81234, Ao85678,
AAo8: std_logic_vector (7 downto 0);
signal co82,co84,co87,co88, Ca812, Ca834, Ca856, Ca878,
Ca81234, Ca85678, Cao8, Cab8: std_logic;
signal xs92c, xs93c, xs95c, xs96c, xs97c, xs98c, xs91,
xs92, xs93, xs94, xs95, xs96, xs97, xs98: std_logic_vector(7 downto 0);
```

```
signal Ao912, Ao934, Ao956, Ao978, Ao91234,
Ao95678: std_logic_vector (7 downto 0);
signal co92,co93,co97,co98,co95,co96, Ca912, Ca934, Ca956,
Ca978, Ca91234, Ca95678, Cao9: std_logic;
signal xs101c, xs102c, xs104c, xs101, xs102, xs103, xs104,
xs105, xs106, xs107, xs108: std_logic_vector(7 downto 0);
signal Ao1012, Ao1034, Ao1056, Ao1078, Ao101234, Ao105678,
AAo10: std_logic_vector (7 downto 0);
signal co101,co102,co104,Ca1012, Ca1034, Ca1056, Ca1078,
Ca101234, Ca105678, Cao10, Cab10: std_logic;
signal xs111c, xs112c, xs115c, xs117c, xs111, xs112, xs113,
xs114, xs115, xs116, xs117, xs118: std_logic_vector(7 downto 0);
signal Ao1112, Ao1134, Ao1156, Ao1178, Ao111234, Ao115678,
AAo11: std_logic_vector (7 downto 0);
signal co111,co112,co115,co117,Ca1112, Ca1134, Ca1156, Ca1178,
Ca111234, Ca115678, Cao11, Cab11: std_logic;
signal xs122c, xs124c, xs125c, xs128c, xs121, xs122, xs123, xs124,
xs125, xs127, xs128: std_logic_vector(7 downto 0);
signal Ao1212, Ao1234, Ao1256, Ao1278, Ao121234, Ao125678,
AAo12: std_logic_vector (7 downto 0);
signal co122,co124,co125,co128,Ca1212, Ca1234, Ca1256,
Ca1278, Ca121234, Ca125678, Cao12, Cab12: std_logic;
signal ABo3, ABo4, ABo5, ABo6, ABo7, ABo8, ABo9, ABo10, ABo11,
ABo12: std_logic_vector(7 downto 0);
begin
S11: shift3 port map (x1, clk, xs11s);
SS11: S3 port map (xs11s,clk,  xs11c);
C11: compliment2s port map (xs11c,co11, xs11);
S12: shift5 port map (x2,clk,  xs12s);
SS12: S5 port map (xs12s,clk, xs12c);
C12: compliment2s port map (xs12c, co12, xs12);
S13: shift2 port map (x3,clk, xs13s);
```

```
SS13: S2 port map (xs13s,clk, xs13);
S14: shift2 port map (x4,clk, xs14s);
SS14: S2 port map (xs14s,clk, xs14);
S15: shift2 port map (x5,clk, xs15s);
SS15: S2 port map (xs15s,clk, xs15c);
C15: compliment2s port map (xs15c,co15, xs15);
S16: shift3 port map (x6,clk, xs16s);
SS16: S3 port map (xs16s,clk, xs16c);
C16: compliment2s port map (xs16c,co16, xs16);
S17: shift5 port map (x7,clk, xs17s);
SS17: S5 port map (xs17s,clk, xs17c);
C17: compliment2s port map (xs17c,co17, xs17);
S18: shift2 port map (x8,clk, xs18s);
SS18: S2 port map (xs18s,clk, xs18c);
C18: compliment2s port map (xs18c,co18, xs18);

A112: add8 port map (xs12, xs11, '0', Ca112, Ao112);
A134: add8 port map (xs13, xs14, '0', Ca134, Ao134);
A156: add8 port map (xs15, xs16, '0', Ca156, Ao156);
A178: add8 port map (xs17, xs18, '0', Ca178, Ao178);

A11234: add8 port map (Ao112, Ao134, '0', Ca11234, Ao11234);
A15678: add8 port map (Ao156, Ao178, '0', Ca15678, Ao15678);
AAo1: add8 port map (Ao11234, Ao15678, '0', Cao1, ABo1);

S21: shift5 port map (x1,clk, xs21s);
SS21: S5 port map (xs21s,clk,xs21);
S22: shift5 port map (x2,clk, xs22s);
SS22: S5 port map (xs22s,clk, xs22);
S23: shift3 port map (x3, clk, xs23s);
SS23: S3 port map (xs23s, clk, xs23);
S24: shift5 port map (x4, clk, xs24s);
```

```
SS24: S5 port map (xs24s,clk, xs24c);
C24: compliment2s port map (xs24c, co24, xs24);
S25: shift5 port map (x5, clk, xs25s);
SS25: S5 port map (xs25s,clk, xs25c);
C25: compliment2s port map (xs25c, co25, xs25);
S26: shift5 port map (x6,clk, xs26s);
SS26:S5 port map (xs26s,clk, xs26);
S27: shift5 port map (x7,clk, xs27s);
SS27:S5 port map (xs27s,clk, xs27);
S28: shift5 port map (x8, clk, xs28s);
SS28:S5 port map (xs28s,clk, xs28);

A212: add8 port map (xs22, xs21, '0', Ca212, Ao212);
A234: add8 port map (xs23, xs24, '0', Ca234, Ao234);
A256: add8 port map (xs25, xs26, '0', Ca256, Ao256);
A278: add8 port map (xs27, xs28, '0', Ca278, Ao278);

A21234: add8 port map (Ao212, Ao234, '0', Ca21234, Ao21234);
A25678: add8 port map (Ao256, Ao278, '0', Ca25678, Ao25678);
AA02: add8 port map (Ao21234, Ao25678, '0', Cao2, AAo2);
AB2: add8 port map (AAo2, "11111000", '0', Cab2, ABo2);

S32: shift3 port map (x2,clk, xs32s);
SS32: S3 port map (xs32s,clk, xs32);
S33: shift5 port map (x3, clk, xs33s);
SS33: S5 port map (xs33s, clk, xs33);
S34: shift1 port map (x4, clk, xs34s);
SS34: S1 port map (xs34s,clk, xs34);
S35: shift5 port map (x5, clk, xs35s);
SS35: S5 port map (xs35s,clk, xs35);
S36: shift4 port map (x6,clk, xs36s);
SS36:S4 port map (xs36s,clk, xs36c);
```

```
C36: compliment2s port map (xs36c, co36,xs36);
S37: shift3 port map (x7,clk, xs37s);
SS37:S3 port map (xs37s,clk, xs37);
S38: shift4 port map (x8, clk, xs38s);
SS38:S4 port map (xs38s,clk, xs38);

A312: add8 port map (xs32, x1, '0', Ca312, Ao312);
A334: add8 port map (xs33, xs34, '0', Ca334, Ao334);
A356: add8 port map (xs35, xs36, '0', Ca356, Ao356);
A378: add8 port map (xs37, xs38, '0', Ca378, Ao378);

A31234: add8 port map (Ao312, Ao334, '0', Ca31234, Ao31234);
A35678: add8 port map (Ao356, Ao378, '0', Ca35678, Ao35678);
AA03: add8 port map (Ao31234, Ao35678, '0', Cao3, AAo3);
AB3: add8 port map (AAo3, "00001000", '0', Cab3, ABo3);

S41: shift5 port map (x1, clk, xs41s);
SS41: S5 port map (xs41s, clk, xs41);
S42: shift4 port map (x2,clk, xs42s);
SS42: S4 port map (xs42s,clk, xs42);
S43: shift4 port map (x3, clk, xs43s);
SS43: S4 port map (xs43s, clk, xs43c);
C43: compliment2s port map (xs43c, co43,xs43);
S44: shift5 port map (x4, clk, xs44s);
SS44: S5 port map (xs44s,clk, xs44);
S45: shift1 port map (x5, clk, xs45s);
SS45: S1 port map (xs45s,clk, xs45c);
C45: compliment2s port map (xs45c, co45,xs45);
S46: shift5 port map (x6,clk, xs46s);
SS46:S5 port map (xs46s,clk, xs46);
S47: shift4 port map (x7,clk, xs47s);
SS47:S4 port map (xs47s,clk, xs47);
```

```
S48: shift5 port map (x8, clk, xs48s);
SS48:S5 port map (xs48s,clk, xs48c);
C48: compliment2s port map (xs48c, co48, xs48);

A412: add8 port map (xs42, xs41, '0', Ca412, Ao412);
A434: add8 port map (xs43, xs44, '0', Ca434, Ao434);
A456: add8 port map (xs45, xs46, '0', Ca456, Ao456);
A478: add8 port map (xs47, xs48, '0', Ca478, Ao478);

A41234: add8 port map (Ao412, Ao434, '0', Ca41234, Ao41234);
A45678: add8 port map (Ao456, Ao478, '0', Ca45678, Ao45678);
AA04: add8 port map (Ao41234, Ao45678, '0', Cao4, AAo4);
AB4: add8 port map (AAo4, "00000100", '0', Cab4, ABo4);

S51: shift5 port map (x1, clk, xs51s);
SS51: S5 port map (xs51s, clk, xs51);
S52: shift4 port map (x2,clk, xs52s);
SS52: S4 port map (xs52s,clk, xs52);
S53: shift1 port map (x3, clk, xs53s);
SS53: S1 port map (xs53s, clk, xs53c);
C53: compliment2s port map (xs53c, co53,xs53);
S54: shift3 port map (x4, clk, xs54s);
SS54: S3 port map (xs54s,clk, xs54);
S55: shift4 port map (x5, clk, xs55s);
SS55: S4 port map (xs55s,clk, xs55c);
C55: compliment2s port map (xs55c, co55,xs55);
S56: shift4 port map (x6,clk,xs56s);
SS56:S4 port map (xs56s,clk, xs56c);
C56: compliment2s port map (xs56c, co56,xs56);
S57: shift1 port map (x7,clk, xs57s);
SS57:S1 port map (xs57s,clk, xs57c);
C57: compliment2s port map (xs57c, co57,xs57);
```

```
S58: shift4 port map (x8, clk, xs58s);
SS58:S4 port map (xs58s,clk, xs58c);
C58: compliment2s port map (xs58c, co58, xs58);

A512: add8 port map (xs52, xs51, '0', Ca512, Ao512);
A534: add8 port map (xs53, xs54, '0', Ca534, Ao534);
A556: add8 port map (xs55, xs56, '0', Ca556, Ao556);
A578: add8 port map (xs57, xs58, '0', Ca578, Ao578);

A51234: add8 port map (Ao512, Ao534, '0', Ca51234, Ao51234);
A55678: add8 port map (Ao556, Ao578, '0', Ca55678, Ao55678);
AA05: add8 port map (Ao51234, Ao55678, '0', Cao5, AAo5);
AB5: add8 port map (AAo5, "11111000", '0', Cab5, ABo5);

S61: shift1 port map (x1, clk, xs61s);
SS61: S1 port map (xs61s, clk, xs61c);
C61: compliment2s port map (xs61c, co61, xs61);
S62: shift5 port map (x2,clk, xs62s);
SS62: S5 port map (xs62s,clk, xs62);
S63: shift5 port map (x3, clk, xs63s);
SS63: S5 port map (xs63s, clk, xs63);
S64: shift5 port map (x4, clk, xs64s);
SS64: S5 port map (xs64s,clk, xs64);
S65: shift1 port map (x5, clk, xs65s);
SS65: S1 port map (xs65s,clk, xs65);
S66: shift5 port map (x6,clk,xs66s);
SS66:S5 port map (xs66s,clk, xs66);
S67: shift5 port map (x7,clk, xs67s);
SS67:S5 port map (xs67s,clk, xs67);
S68: shift1 port map (x8, clk, xs68s);
SS68:S1 port map (xs68s,clk, xs68);
```

```
A612: add8 port map (xs62, xs61, '0', Ca612, Ao612);
A634: add8 port map (xs63, xs64, '0', Ca634, Ao634);
A656: add8 port map (xs65, xs66, '0', Ca656, Ao656);
A678: add8 port map (xs67, xs68, '0', Ca678, Ao678);

A61234: add8 port map (Ao612, Ao634, '0', Ca61234, Ao61234);
A65678: add8 port map (Ao656, Ao678, '0', Ca65678, Ao65678);
AA06: add8 port map (Ao61234, Ao65678, '0', Cao6, ABo6);

S71: shift5 port map (x1, clk, xs71s);
SS71: S5 port map (xs71s, clk, xs71c);
C71: compliment2s port map (xs71c, co71, xs71);
S72: shift6 port map (x2,clk, xs72s);
SS72: S6 port map (xs72s,clk, xs72);
S73: shift4 port map (x3, clk, xs73s);
SS73: S4 port map (xs73s, clk, xs73c);
C73: compliment2s port map (xs73c, co73, xs73);
S74: shift4 port map (x4, clk, xs74s);
SS74: S4 port map (xs74s,clk, xs74);
S75: shift4 port map (x5, clk, xs75s);
SS75: S4 port map (xs75s,clk, xs75);
S76: shift1 port map (x6,clk,xs76s);
SS76:S1 port map (xs76s,clk, xs76c);
C76: compliment2s port map (xs76c, co76, xs76);
S77: shift3 port map (x7,clk, xs77s);
SS77:S3 port map (xs77s,clk, xs77);
S78: shift5 port map (x8, clk, xs78s);
SS78:S5 port map (xs78s,clk, xs78);

A712: add8 port map (xs72, xs71, '0', Ca712, Ao712);
A734: add8 port map (xs73, xs74, '0', Ca734, Ao734);
A756: add8 port map (xs75, xs76, '0', Ca756, Ao756);
```

```
A778: add8 port map (xs77, xs78, '0', Ca778, Ao778);

A71234: add8 port map (Ao712, Ao734, '0', Ca71234, Ao71234);
A75678: add8 port map (Ao756, Ao778, '0', Ca75678, Ao75678);
AA07: add8 port map (Ao71234, Ao75678, '0', Cao7, AAo7);
AB7: add8 port map (AAo7,"11111110", '0',Cab7,ABo7);

S81: shift5 port map (x1, clk, xs81s);
SS81: S5 port map (xs81s, clk, xs81);
S82: shift2 port map (x2,clk, xs82s);
SS82: S2 port map (xs82s,clk, xs82c);
C82: compliment2s port map (xs82c, co82, xs82);
S83: shift5 port map (x3, clk, xs83s);
SS83: S5 port map (xs83s, clk, xs83);
S84: shift4 port map (x4, clk, xs84s);
SS84: S4 port map (xs84s,clk, xs84c);
C84: compliment2s port map (xs84c, co84, xs84);
S85: shift4 port map (x5, clk, xs85s);
SS85: S4 port map (xs85s,clk, xs85);
S86: shift5 port map (x6,clk,xs86s);
SS86:S5 port map (xs86s,clk, xs86);
S87: shift5 port map (x7,clk, xs87s);
SS87:S5 port map (xs87s,clk, xs87c);
C87: compliment2s port map (xs87c, co87, xs87);
S88: shift5 port map (x8, clk, xs88s);
SS88:S5 port map (xs88s,clk, xs88c);
C88: compliment2s port map (xs88c, co88, xs88);

A812: add8 port map (xs82, xs81, '0', Ca812, Ao812);
A834: add8 port map (xs83, xs84, '0', Ca834, Ao834);
A856: add8 port map (xs85, xs86, '0', Ca856, Ao856);
A878: add8 port map (xs87, xs88, '0', Ca878, Ao878);
```

```
A81234: add8 port map (Ao812, Ao834, '0', Ca81234, Ao81234);
A85678: add8 port map (Ao856, Ao878, '0', Ca85678, Ao85678);
AA08: add8 port map (Ao81234, Ao85678, '0', Cao8, AAo8);
AB8: add8 port map (AAo8,"00000010", '0',Cab8,ABo8);

S91: shift5 port map (x1, clk, xs91s);
SS91: S5 port map (xs91s, clk, xs91);
S92: shift3 port map (x2,clk, xs92s);
SS92: S3 port map (xs92s,clk, xs92c);
C92: compliment2s port map (xs92c, co92, xs92);
S93: shift4 port map (x3, clk, xs93s);
SS93: S4 port map (xs93s, clk, xs93c);
C93: compliment2s port map (xs93c, co93, xs93);
S94: shift4 port map (x4, clk, xs94s);
SS94: S4 port map (xs94s,clk, xs94);
S95: shift4 port map (x5, clk, xs95s);
SS95: S4 port map (xs95s,clk, xs95c);
C95: compliment2s port map (xs95c, co95, xs95);
S96: shift5 port map (x6,clk,xs96s);
SS96:S5 port map (xs96s,clk, xs96c);
C96: compliment2s port map (xs96c, co96, xs96);
S97: shift3 port map (x7,clk, xs97s);
SS97:S3 port map (xs97s,clk, xs97c);
C97: compliment2s port map (xs97c, co97, xs97);
S98: shift4 port map (x8, clk, xs98s);
SS98:S4 port map (xs98s,clk, xs98c);
C98: compliment2s port map (xs98c, co98, xs98);

A912: add8 port map (xs92, xs91, '0', Ca912, Ao912);
A934: add8 port map (xs93, xs94, '0', Ca934, Ao934);
A956: add8 port map (xs95, xs96, '0', Ca956, Ao956);
```

```
A978: add8 port map (xs97, xs98, '0', Ca978, Ao978);

A91234: add8 port map (Ao912, Ao934, '0', Ca91234, Ao91234);
A95678: add8 port map (Ao956, Ao978, '0', Ca95678, Ao95678);
AA09: add8 port map (Ao91234, Ao95678, '0', Cao9, ABo9);

S101: shift4 port map (x1, clk, xs101s);
SS101: S4 port map (xs101s, clk, xs101c);
C101: compliment2s port map (xs101c, co101, xs101);
S102: shift3 port map (x2,clk, xs102s);
SS102: S3 port map (xs102s,clk, xs102c);
C102: compliment2s port map (xs102c, co102, xs102);
S103: shift2 port map (x3, clk, xs103s);
SS103: S2 port map (xs103s, clk, xs103);
S104: shift4 port map (x4, clk, xs104s);
SS104: S4 port map (xs104s,clk, xs104c);
C104: compliment2s port map (xs104c, co104, xs104);
S105: shift5 port map (x5, clk, xs105s);
SS105: S5 port map (xs105s,clk, xs105);
S106: shift5 port map (x6,clk,xs106s);
SS106:S5 port map (xs106s,clk, xs106);
S107: shift4 port map (x7,clk, xs107s);
SS107:S4 port map (xs107s,clk, xs107);
S108: shift5 port map (x8, clk, xs108s);
SS108:S5 port map (xs108s,clk, xs108);

A1012: add8 port map (xs102, xs101, '0', Ca1012, Ao1012);
A1034: add8 port map (xs103, xs104, '0', Ca1034, Ao1034);
A1056: add8 port map (xs105, xs106, '0', Ca1056, Ao1056);
A1078: add8 port map (xs107, xs108, '0', Ca1078, Ao1078);

A101234: add8 port map (Ao1012, Ao1034, '0', Ca101234, Ao101234);
```

```
A105678: add8 port map (Ao1056, Ao1078, '0', Ca105678, Ao105678);
AA010: add8 port map (Ao101234, Ao105678, '0', Cao10, AAo10);
AB10: add8 port map (AAo10, "00000100", '0', Cab10, ABo10);

S111: shift2 port map (x1, clk, xs111s);
SS111: S2 port map (xs111s, clk, xs111c);
C111: compliment2s port map (xs111c, co111, xs111);
S112: shift5 port map (x2,clk, xs112s);
SS112: S5 port map (xs112s,clk, xs112c);
C112: compliment2s port map (xs112c, co112, xs112);
S113: shift2 port map (x3, clk, xs113s);
SS113: S2 port map (xs113s, clk, xs113);
S114: shift4 port map (x4, clk, xs114s);
SS114: S4 port map (xs114s,clk, xs114);
S115: shift5 port map (x5, clk, xs115s);
SS115: S5 port map (xs115s,clk, xs115c);
C115: compliment2s port map (xs115c, co115, xs115);
S116: shift1 port map (x6,clk,xs116s);
SS116:S1 port map (xs116s,clk, xs116);
S117: shift5 port map (x7,clk, xs117s);
SS117:S5 port map (xs117s,clk, xs117c);
C117: compliment2s port map (xs117c, co117, xs117);
S118: shift5 port map (x8, clk, xs118s);
SS118:S5 port map (xs118s,clk, xs118);

A1112: add8 port map (xs112, xs111, '0', Ca1112, Ao1112);
A1134: add8 port map (xs113, xs114, '0', Ca1134, Ao1134);
A1156: add8 port map (xs115, xs116, '0', Ca1156, Ao1156);
A1178: add8 port map (xs117, xs118, '0', Ca1178, Ao1178);

A111234: add8 port map (Ao1112, Ao1134, '0', Ca111234, Ao111234);
A115678: add8 port map (Ao1156, Ao1178, '0', Ca115678, Ao115678);
```

```
AA011: add8 port map (Ao111234, Ao115678, '0', Cao11, AAo11);
AB11: add8 port map (AAo11, "11111000", '0', Cab11, ABo11);

S121: shift5 port map (x1, clk, xs121s);
SS121: S5 port map (xs121s, clk, xs121);
S122: shift4 port map (x2,clk, xs122s);
SS122: S4 port map (xs122s,clk, xs122c);
C122: compliment2s port map (xs122c, co122, xs122);
S123: shift3 port map (x3, clk, xs123s);
SS123: S3 port map (xs123s, clk, xs123);
S124: shift1 port map (x4, clk, xs124s);
SS124: S1 port map (xs124s,clk, xs124c);
C124: compliment2s port map (xs124c, co124, xs124);
S125: shift4 port map (x5, clk, xs125s);
SS125: S4 port map (xs125s,clk, xs125c);
C125: compliment2s port map (xs125c, co125, xs125);
S127: shift5 port map (x7,clk, xs127s);
SS127:S5 port map (xs127s,clk, xs127);
S128: shift4 port map (x8, clk, xs128s);
SS128:S4 port map (xs128s,clk, xs128c);
C128: compliment2s port map (xs128c, co128, xs128);

A1212: add8 port map (xs122, xs121, '0', Ca1212, Ao1212);
A1234: add8 port map (xs123, xs124, '0', Ca1234, Ao1234);
A1256: add8 port map (xs125, x6, '0', Ca1256, Ao1256);
A1278: add8 port map (xs127, xs128, '0', Ca1278, Ao1278);

A121234: add8 port map (Ao1212, Ao1234, '0', Ca121234, Ao121234);
A125678: add8 port map (Ao1256, Ao1278, '0', Ca125678, Ao125678);
AA012: add8 port map (Ao121234, Ao125678, '0', Cao12, AAo12);
AB12: add8 port map (AAo12, "00001000", '0', Cab12, ABo12);
```

```
RE1: relu8 port map (ABo1, y1);
RE2: relu8 port map (ABo2, y2);
RE3: relu8 port map (ABo3, y3);
RE4: relu8 port map (ABo4, y4);
RE5: relu8 port map (ABo5, y5);
RE6: relu8 port map (ABo6, y6);
RE7: relu8 port map (ABo7, y7);
RE8: relu8 port map (ABo8, y8);
RE9: relu8 port map (ABo9, y9);
RE10: relu8 port map (ABo10, y10);
RE11: relu8 port map (ABo11, y11);
RE12: relu8 port map (ABo12, y12);

end hidden;
entity Hidden_Layer2 is
port (x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11,
x12: in std_logic_vector (7 downto 0); clk: std_logic; y1, y2, y3, y4, y5,
y6, y7, y8:out std_logic_vector(7 downto 0));
end Hidden_Layer2;

architecture hidden2 of Hidden_Layer2 is

component shift1 is
port (din: in std_logic_vector(7 downto 0); clk: in std_logic;
        dout: out std_logic_vector(8 downto 0));
end component;

component shift2 is
port (din: in std_logic_vector(7 downto 0); clk: in std_logic;
        dout: out std_logic_vector(9 downto 0));
end component;
```

```
component shift3 is
port (din: in std_logic_vector(7 downto 0); clk: in std_logic;
        dout: out std_logic_vector(10 downto 0));
end component;

component shift4 is
port (din: in std_logic_vector(7 downto 0); clk: in std_logic;
        dout: out std_logic_vector(11 downto 0));
end component;

component shift5 is
port (din: in std_logic_vector(7 downto 0); clk: in std_logic;
        dout: out std_logic_vector(12 downto 0));
end component;

component shift6 is
port (din: in std_logic_vector(7 downto 0); clk: in std_logic;
        dout: out std_logic_vector(13 downto 0));
end component;

component S6 is
port (din: in std_logic_vector(13 downto 0); clk: in std_logic;
        dout: out std_logic_vector(7 downto 0));
end component;

component S5 is
port (din: in std_logic_vector(12 downto 0); clk: in std_logic;
        dout: out std_logic_vector(7 downto 0));
end component;

component S4 is
port (din: in std_logic_vector(11 downto 0); clk: in std_logic;
```

```
        dout: out std_logic_vector(7 downto 0));
end component;

component S3 is
port (din: in std_logic_vector(10 downto 0); clk: in std_logic;
        dout: out std_logic_vector(7 downto 0));
end component;

component S2 is
port (din: in std_logic_vector(9 downto 0); clk: in std_logic;
        dout: out std_logic_vector(7 downto 0));
end component;

component S1 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
        dout: out std_logic_vector(7 downto 0));
end component;

component add8 is
port (A,B: in std_logic_vector(7 downto 0); Cin: in std_logic;
Co: out std_logic; Sum: out std_logic_vector(7 downto 0));
end component;

component relu8 is
port (x : in std_logic_vector(7 downto 0); y: out std_logic_vector(7 downto 0));
end component;

component compliment2s
port (Din: in std_logic_vector (7 downto 0); Co: out std_logic;
Do: out std_logic_vector (7 downto 0));
end component;
```

```
-- 6 shift
signal xs211s, xs711s : std_logic_vector(13 downto 0); -- 6 shift
-- 5 shift
signal xs13s, xs14s, xs110s, xs22s, xs25s, xs27s, xs29s, xs210s,
xs212s, xs31s, xs36s, xs38s, xs43s, xs44s, xs45s, xs46s, xs47s,
         xs49s, xs51s, xs52s, xs57s, xs511s, xs512s, xs64s, xs66s,
  xs611s, xs74s, xs78s, xs79s,
         xs81s, xs82s, xs85s, xs86s, xs88s,
  xs89s : std_logic_vector(12 downto 0); -- 5 shift
-- 4 shift
signal xs11s, xs12s, xs17s, xs18s, xs111s, xs112s, xs23s, xs24s,
xs26s, xs28s, xs34s, xs310s, xs312s, xs41s, xs42s, xs54s, xs55s,
         xs58s, xs59s, xs510s, xs61s, xs62s, xs67s, xs68s, xs69s,
  xs610s, xs72s, xs75s, xs76s, xs77s, xs710s, xs712s, xs811s,
  xs812s : std_logic_vector(11 downto 0); -- 4 shift
-- 3 shift
signal xs15s, xs35s, xs48s, xs410s, xs411s, xs63s,
xs84s : std_logic_vector(10 downto 0); -- 3 shift
-- 2 shift
signal  xs16s, xs19s, xs21s, xs311s, xs412s, xs56s, xs612s, xs73s,
xs87s  : std_logic_vector(9 downto 0); -- 2 shift
-- 1 shift
signal xs32s, xs33s, xs37s, xs39s, xs65s, xs71s, xs83s,
xs810s : std_logic_vector(8 downto 0); -- 1 shift

signal xs12c, xs14c, xs15c, xs16c, xs17c, xs19c, xs110c,
xs112c, xs11, xs12, xs13, xs14, xs15, xs16, xs17, xs18,
xs19, xs110, xs111, xs112: std_logic_vector(7 downto 0);
signal Ao112, Ao134, Ao156, Ao178, Ao1910, Ao11112, Ao11234,
Ao15678, Ao19101112, Ao11_8, AAo1, ABo1: std_logic_vector (7 downto 0);
signal co12, co14, co15, co16, co17, co19, co110, co112,
```

```
Ca112, Ca134, Ca156, Ca178, Ca1910, Ca11112, Ca11234,
Ca15678, Ca19101112, Ca11_8, Cao1, Cab1: std_logic;

signal xs23c, xs28c, xs21, xs22, xs23, xs24, xs25, xs26,
xs27, xs28, xs29, xs210, xs211, xs212: std_logic_vector(7 downto 0);
signal Ao212, Ao234, Ao256, Ao278, Ao2910, Ao21112, Ao21234,
Ao25678, Ao29101112, Ao21_8, ABo2, AAo2: std_logic_vector (7 downto 0);
signal co23, co28, Ca212, Ca234, Ca256, Ca278, Ca2910,
Ca21112, Ca21234, Ca25678, Ca29101112, Ca21_8, Cao2, Cab2: std_logic;

signal xs310c, xs311c, xs31, xs32, xs33, xs34, xs35, xs36, xs37,
xs38, xs39, xs310, xs311, xs312 : std_logic_vector(7 downto 0);
signal Ao312, Ao334, Ao356, Ao378, Ao3910, Ao31112, Ao31234,
Ao35678, Ao39101112, Ao31_8, AAo3: std_logic_vector (7 downto 0);
signal co310, co311, Ca312, Ca334, Ca356, Ca378, Ca3910, Ca31112,
Ca31234, Ca35678, Ca39101112, Ca31_8, Cao3, Cab3: std_logic;

signal xs43c,xs44c, xs45c, xs46c, xs48c, xs49c, xs411c, xs41, xs42,
xs43, xs44, xs45, xs46, xs47, xs48, xs49, xs410, xs411,
xs412  : std_logic_vector(7 downto 0);
signal Ao412, Ao434, Ao456, Ao478, Ao4910, Ao41112, Ao41234,
Ao45678, Ao49101112, Ao41_8, AAo4: std_logic_vector (7 downto 0);
signal co43, co44, co45, co46, co48, co49, co411, Ca412, Ca434,
Ca456, Ca478, Ca4910, Ca41112, Ca41234, Ca45678, Ca49101112,
Ca41_8, Cao4, Cab4: std_logic;

signal xs51c, xs511c, xs51, xs52, xs54, xs55, xs56, xs57, xs58,
xs59, xs510, xs511, xs512: std_logic_vector(7 downto 0);
signal Ao512, Ao534, Ao556, Ao578, Ao5910, Ao51112, Ao51234,
Ao55678, Ao59101112, Ao51_8, AAo5: std_logic_vector (7 downto 0);
signal co51, co511, Ca512, Ca534, Ca556, Ca578, Ca5910, Ca51112,
Ca51234, Ca55678, Ca59101112, Ca51_8, Cao5, Cab5: std_logic;
```

```
signal xs61c, xs62c, xs64c, xs66c, xs68c, xs69c, xs610c, xs611c,
xs612c, xs61, xs62, xs63, xs64, xs65, xs66, xs67, xs68,xs69, xs610,
xs611, xs612: std_logic_vector(7 downto 0);
signal Ao612, Ao634, Ao656, Ao678, Ao6910, Ao61112, Ao61234, Ao65678,
Ao69101112, Ao61_8, AAo6: std_logic_vector (7 downto 0);
signal co61, co62, co64, co66, co68, co69, co610, co611, co612, Ca612,
Ca634, Ca656, Ca678, Ca6910, Ca61112, Ca61234, Ca65678,Ca69101112,
Ca61_8, Cao6, Cab6 : std_logic;


signal xs71c, xs75c, xs76c, xs78c, xs710c, xs712c, xs71, xs72, xs73,
xs74, xs75, xs76, xs77, xs78,  xs79, xs710, xs711,
xs712: std_logic_vector(7 downto 0);
signal Ao712, Ao734, Ao756, Ao778, Ao7910, Ao71112, Ao71234, Ao75678,
Ao79101112, Ao71_8, AAo7: std_logic_vector (7 downto 0);
signal co71,co75, co76, co78, co710, co712, Ca712, Ca734, Ca756, Ca778,
Ca7910, Ca71112, Ca71234, Ca75678, Ca79101112, Ca71_8, Cao7, Cab7: std_logic;

signal xs81c, xs83c, xs84c, xs85c, xs86c, xs87c, xs88c, xs89c, xs810c,
xs811c, xs812c, xs81, xs82, xs83, xs84, xs85, xs86, xs87, xs88, xs89,
xs810, xs811, xs812: std_logic_vector(7 downto 0);
signal Ao812, Ao834, Ao856, Ao878, Ao8910, Ao81112, Ao81234, Ao85678,
Ao89101112, Ao81_8: std_logic_vector (7 downto 0);
signal co81, co83, co84, co85, co86, co87,co88, co89, co810, co811,
co812, Ca812, Ca834, Ca856, Ca878, Ca8910, Ca81112, Ca81234, Ca85678,
Ca89101112, Ca81_8, Cbo8: std_logic;

signal ABo3, ABo4, ABo5, ABo6, ABo7, ABo8 :std_logic_vector(7 downto 0);

begin
S11: shift4 port map (x1, clk, xs11s);
```

```
SS11: S4 port map (xs11s,clk,  xs11);
S12: shift4 port map (x2,clk,  xs12s);
SS12: S4 port map (xs12s,clk, xs12c);
C12: compliment2s port map (xs12c,co12, xs12);
S13: shift5 port map (x3,clk, xs13s);
SS13: S5 port map (xs13s,clk, xs13);
S14: shift5 port map (x5,clk, xs14s);
SS14: S5 port map (xs14s,clk, xs14c);
C14: compliment2s port map (xs14c,co14, xs14);
S15: shift3 port map (x5,clk, xs15s);
SS15: S3 port map (xs15s,clk, xs15c);
C15: compliment2s port map (xs15c,co15, xs15);
S16: shift2 port map (x6,clk, xs16s);
SS16: S2 port map (xs16s,clk, xs16c);
C16: compliment2s port map (xs16c,co16, xs16);
S17: shift4 port map (x7,clk, xs17s);
SS17: S4 port map (xs17s,clk, xs17c);
C17: compliment2s port map (xs17c,co17, xs17);
S18: shift4 port map (x8,clk, xs18s);
SS18: S4 port map (xs18s,clk, xs18);
S19: shift2 port map (x9,clk, xs19s);
SS19: S2 port map (xs19s,clk, xs19c);
C19: compliment2s port map (xs19c,co19, xs19);
S110: shift5 port map (x10,clk, xs110s);
SS110: S5 port map (xs110s,clk, xs110c);
C110: compliment2s port map (xs110c,co110, xs110);
S111: shift4 port map (x11,clk, xs111s);
SS111: S4 port map (xs111s,clk, xs111);
S112: shift4 port map (x12,clk, xs112s);
SS112: S4 port map (xs112s,clk, xs112c);
C112: compliment2s port map (xs112c,co112, xs112);
```

```
A112: add8 port map (xs12, xs11, '0', Ca112, Ao112);
A134: add8 port map (xs13, xs14, '0', Ca134, Ao134);
A156: add8 port map (xs15, xs16, '0', Ca156, Ao156);
A178: add8 port map (xs17, xs18, '0', Ca178, Ao178);
A1910: add8 port map (xs19, xs110, '0', Ca1910, Ao1910);
A11112: add8 port map (xs111, xs112, '0', Ca11112, Ao11112);

A11234: add8 port map (Ao112, Ao134, '0', Ca11234, Ao11234);
A15678: add8 port map (Ao156, Ao178, '0', Ca15678, Ao15678);
A19101112: add8 port map (Ao1910, Ao11112, '0', Ca19101112, Ao19101112);

A11_8: add8 port map (Ao11234, Ao15678, '0', Ca11_8, Ao11_8);
AA01: add8 port map (Ao11_8, Ao19101112, '0', Cao1, AAo1);
AB1: add8 port map (AAo1, "00010000", '0', Cab1, ABo1);



S21: shift2 port map (x1,clk, xs21s);
SS21: S2 port map (xs21s,clk,xs21);
S22: shift5 port map (x2,clk, xs22s);
SS22: S5 port map (xs22s,clk, xs22);
S23: shift4 port map (x3, clk, xs23s);
SS23: S4 port map (xs23s, clk, xs23c);
C23: compliment2s port map (xs23c, co23, xs23);
S24: shift4 port map (x4, clk, xs24s);
SS24: S4 port map (xs24s,clk, xs24);
S25: shift5 port map (x5, clk, xs25s);
SS25: S5 port map (xs25s,clk, xs25);
S26: shift4 port map (x6,clk, xs26s);
SS26:S4 port map (xs26s,clk, xs26);
S27: shift5 port map (x7,clk, xs27s);
```

```
SS27:S5 port map (xs27s,clk, xs27);
S28: shift4 port map (x8, clk, xs28s);
SS28:S4 port map (xs28s,clk, xs28c);
C28: compliment2s port map (xs28c, co28, xs28);
S29: shift5 port map (x9, clk, xs29s);
SS29:S5 port map (xs29s,clk, xs29);
S210: shift5 port map (x10, clk, xs210s);
SS210:S5 port map (xs210s,clk, xs210);
S211: shift6 port map (x11, clk, xs211s);
SS211:S6 port map (xs211s,clk, xs211);
S212: shift5 port map (x12, clk, xs212s);
SS212:S5 port map (xs212s,clk, xs212);


A212: add8 port map (xs22, xs21, '0', Ca212, Ao212);
A234: add8 port map (xs23, xs24, '0', Ca234, Ao234);
A256: add8 port map (xs25, xs26, '0', Ca256, Ao256);
A278: add8 port map (xs27, xs28, '0', Ca278, Ao278);
A2910: add8 port map (xs29, xs210, '0', Ca2910, Ao2910);
A21112: add8 port map (xs211, xs212, '0', Ca21112, Ao21112);

A21234: add8 port map (Ao212, Ao234, '0', Ca21234, Ao21234);
A25678: add8 port map (Ao256, Ao278, '0', Ca25678, Ao25678);
A29101112: add8 port map (Ao2910, Ao21112, '0', Ca29101112, Ao29101112);

A21_8: add8 port map (Ao21234, Ao25678, '0', Ca21_8, Ao21_8);
AA02: add8 port map (Ao21_8, Ao29101112, '0', Cao2, AAo2);
AB2: add8 port map (AAo2, "11111000", '0', Cab2, ABo2);

S31: shift5 port map (x1,clk, xs31s);
SS31: S5 port map (xs31s,clk, xs31);
S32: shift1 port map (x2,clk, xs32s);
```

```
SS32: S1 port map (xs32s,clk, xs32);
S33: shift1 port map (x3, clk, xs33s);
SS33: S1 port map (xs33s, clk, xs33);
S34: shift4 port map (x4, clk, xs34s);
SS34: S4 port map (xs34s,clk, xs34);
S35: shift3 port map (x5, clk, xs35s);
SS35: S3 port map (xs35s,clk, xs35);
S36: shift5 port map (x6,clk, xs36s);
SS36:S5 port map (xs36s,clk, xs36);
S37: shift1 port map (x7,clk, xs37s);
SS37:S1 port map (xs37s,clk, xs37);
S38: shift5 port map (x8, clk, xs38s);
SS38:S5 port map (xs38s, clk, xs38);
S39: shift1 port map (x9, clk, xs39s);
SS39:S1 port map (xs39s, clk, xs39);
S310: shift4 port map (x10, clk, xs310s);
SS310:S4 port map (xs310s, clk, xs310c);
C310: compliment2s port map (xs310c,co310, xs310);
S311: shift2 port map (x11, clk, xs311s);
SS311:S2 port map (xs311s, clk, xs311c);
C311: compliment2s port map (xs311c,co311, xs311);
S312: shift4 port map (x12, clk, xs312s);
SS312:S4 port map (xs312s, clk, xs312);


A312: add8 port map (xs32, x1, '0', Ca312, Ao312);
A334: add8 port map (xs33, xs34, '0', Ca334, Ao334);
A356: add8 port map (xs35, xs36, '0', Ca356, Ao356);
A378: add8 port map (xs37, xs38, '0', Ca378, Ao378);
A3910: add8 port map (xs39, xs310, '0', Ca3910, Ao3910);
A31112: add8 port map (xs311, xs312, '0', Ca31112, Ao31112);
```

```
A31234: add8 port map (Ao312, Ao334, '0', Ca31234, Ao31234);
A35678: add8 port map (Ao356, Ao378, '0', Ca35678, Ao35678);
A39101112: add8 port map (Ao3910, Ao31112, '0', Ca39101112, Ao39101112);

A31_8: add8 port map (Ao31234, Ao35678, '0', Ca31_8, Ao31_8);
AA03: add8 port map (Ao31_8, Ao39101112, '0', Cao3, AAo3);
AB3: add8 port map (AAo3, "00010000", '0', Cab3, ABo3);

S41: shift4 port map (x1, clk, xs41s);
SS41: S4 port map (xs41s, clk, xs41);
S42: shift4 port map (x2,clk, xs42s);
SS42: S4 port map (xs42s,clk, xs42);
S43: shift5 port map (x3, clk, xs43s);
SS43: S5 port map (xs43s, clk, xs43c);
C43: compliment2s port map (xs43c, co43,xs43);
S44: shift5 port map (x4, clk, xs44s);
SS44: S5 port map (xs44s,clk, xs44c);
C44: compliment2s port map (xs44c, co44,xs44);
S45: shift5 port map (x5, clk, xs45s);
SS45: S5 port map (xs45s,clk, xs45c);
C45: compliment2s port map (xs45c, co45,xs45);
S46: shift5 port map (x6,clk, xs46s);
SS46:S5 port map (xs46s,clk, xs46c);
C46: compliment2s port map (xs46c, co46,xs46);
S47: shift5 port map (x7,clk, xs47s);
SS47:S5 port map (xs47s,clk, xs47);
S48: shift3 port map (x8, clk, xs48s);
SS48:S3 port map (xs48s,clk, xs48c);
C48: compliment2s port map (xs48c, co48, xs48);
S49: shift5 port map (x9, clk, xs49s);
SS49:S5 port map (xs49s,clk, xs49c);
C49: compliment2s port map (xs49c, co49, xs49);
```

```
S410: shift3 port map (x10, clk, xs410s);
SS410:S3 port map (xs410s,clk, xs410);
S411: shift3 port map (x11, clk, xs411s);
SS411:S3 port map (xs411s,clk, xs411c);
C411: compliment2s port map (xs411c, co411, xs411);
S412: shift2 port map (x12, clk, xs412s);
SS412:S2 port map (xs412s,clk, xs412);



A412: add8 port map (xs42, xs41, '0', Ca412, Ao412);
A434: add8 port map (xs43, xs44, '0', Ca434, Ao434);
A456: add8 port map (xs45, xs46, '0', Ca456, Ao456);
A478: add8 port map (xs47, xs48, '0', Ca478, Ao478);
A4910: add8 port map (xs49, xs410, '0', Ca4910, Ao4910);
A41112: add8 port map (xs411, xs412, '0', Ca41112, Ao41112);

A41234: add8 port map (Ao412, Ao434, '0', Ca41234, Ao41234);
A45678: add8 port map (Ao456, Ao478, '0', Ca45678, Ao45678);
A49101112: add8 port map (Ao4910, Ao41112, '0', Ca49101112, Ao49101112);

A41_8: add8 port map (Ao41234, Ao45678, '0', Ca41_8, Ao41_8);
AA04: add8 port map (Ao41_8, Ao49101112, '0', Cao4, AAo4);
AB4: add8 port map (AAo4, "00000000", '0', Cab4, ABo4);

S51: shift5 port map (x1, clk, xs51s);
SS51: S5 port map (xs51s, clk, xs51c);
C51: compliment2s port map (xs51c, co51,xs51);
S52: shift5 port map (x2,clk, xs52s);
SS52: S5 port map (xs52s,clk, xs52);
S54: shift4 port map (x4, clk, xs54s);
SS54: S4 port map (xs54s,clk, xs54);
S55: shift4 port map (x5, clk, xs55s);
```

```
SS55: S4 port map (xs55s,clk, xs55);
S56: shift2 port map (x6,clk,xs56s);
SS56:S2 port map (xs56s,clk, xs56);
S57: shift5 port map (x7,clk, xs57s);
SS57:S5 port map (xs57s,clk, xs57);
S58: shift4 port map (x8, clk, xs58s);
SS58:S4 port map (xs58s,clk, xs58);
S59: shift4 port map (x9, clk, xs59s);
SS59:S4 port map (xs59s,clk, xs59);
S510: shift4 port map (x10, clk, xs510s);
SS510:S4 port map (xs510s,clk, xs510);
S511: shift5 port map (x11, clk, xs511s);
SS511:S5 port map (xs511s,clk, xs511c);
C511: compliment2s port map (xs511c, co511,xs511);
S512: shift5 port map (x12, clk, xs512s);
SS512:S5 port map (xs512s,clk, xs512);

A512: add8 port map (xs52, xs51, '0', Ca512, Ao512);
A534: add8 port map (x3, xs54, '0', Ca534, Ao534);
A556: add8 port map (xs55, xs56, '0', Ca556, Ao556);
A578: add8 port map (xs57, xs58, '0', Ca578, Ao578);
A5910: add8 port map (xs59, xs510, '0', Ca5910, Ao5910);
A51112: add8 port map (xs511, xs512, '0', Ca51112, Ao51112);

A51234: add8 port map (Ao512, Ao534, '0', Ca51234, Ao51234);
A55678: add8 port map (Ao556, Ao578, '0', Ca55678, Ao55678);
A59101112: add8 port map (Ao5910, Ao51112, '0', Ca59101112, Ao59101112);

A51_8: add8 port map (Ao51234, Ao55678, '0', Ca51_8, Ao51_8);
AA05: add8 port map (Ao51_8, Ao59101112, '0', Cao5, AAo5);
AB5: add8 port map (AAo5, "11110000", '0', Cab5, ABo5);
```

```
S61: shift4 port map (x1, clk, xs61s);
SS61: S4 port map (xs61s, clk, xs61c);
C61: compliment2s port map (xs61c, co61, xs61);
S62: shift4 port map (x2,clk, xs62s);
SS62: S4 port map (xs62s,clk, xs62c);
C62: compliment2s port map (xs62c, co62, xs62);
S63: shift3 port map (x3, clk, xs63s);
SS63: S3 port map (xs63s, clk, xs63);
S64: shift5 port map (x4, clk, xs64s);
SS64: S5 port map (xs64s,clk, xs64c);
C64: compliment2s port map (xs64c, co64, xs64);
S65: shift1 port map (x5, clk, xs65s);
SS65: S1 port map (xs65s,clk, xs65);
S66: shift5 port map (x6,clk,xs66s);
SS66:S5 port map (xs66s,clk, xs66c);
C66: compliment2s port map (xs66c, co66, xs66);
S67: shift4 port map (x7,clk, xs67s);
SS67:S4 port map (xs67s,clk, xs67);
S68: shift4 port map (x8, clk, xs68s);
SS68:S4 port map (xs68s,clk, xs68c);
C68: compliment2s port map (xs68c, co68, xs68);
S69: shift4 port map (x9, clk, xs69s);
SS69:S4 port map (xs69s,clk, xs69c);
C69: compliment2s port map (xs69c, co69, xs69);
S610: shift4 port map (x10, clk, xs610s);
SS610:S4 port map (xs610s,clk, xs610c);
C610: compliment2s port map (xs610c, co610, xs610);
S611: shift5 port map (x11, clk, xs611s);
SS611:S5 port map (xs611s,clk, xs611c);
C611: compliment2s port map (xs611c, co611, xs611);
S612: shift2 port map (x12, clk, xs612s);
```

```
SS612:S2 port map (xs612s,clk, xs612c);
C612: compliment2s port map (xs612c, co612, xs612);


A612: add8 port map (xs62, xs61, '0', Ca612, Ao612);
A634: add8 port map (xs63, xs64, '0', Ca634, Ao634);
A656: add8 port map (xs65, xs66, '0', Ca656, Ao656);
A678: add8 port map (xs67, xs68, '0', Ca678, Ao678);
A6910: add8 port map (xs69, xs610, '0', Ca6910, Ao6910);
A61112: add8 port map (xs611, xs612, '0', Ca61112, Ao61112);


A61234: add8 port map (Ao612, Ao634, '0', Ca61234, Ao61234);
A65678: add8 port map (Ao656, Ao678, '0', Ca65678, Ao65678);
A69101112: add8 port map (Ao6910, Ao61112, '0', Ca69101112, Ao69101112);

A61_8: add8 port map (Ao61234, Ao65678, '0', Ca61_8, Ao61_8);
AA06: add8 port map (Ao61_8, Ao69101112, '0', Cao6, AAo6);
AB6: add8 port map (AAo6, "11111110", '0', Cab6, ABo6);


S71: shift1 port map (x1, clk, xs71s);
SS71: S1 port map (xs71s, clk, xs71c);
C71: compliment2s port map (xs71c, co71, xs71);
S72: shift4 port map (x2,clk, xs72s);
SS72: S4 port map (xs72s,clk, xs72);
S73: shift2 port map (x3, clk, xs73s);
SS73: S2 port map (xs73s, clk, xs73);
S74: shift5 port map (x4, clk, xs74s);
SS74: S5 port map (xs74s,clk, xs74);
S75: shift4 port map (x5, clk, xs75s);
SS75: S4 port map (xs75s,clk, xs75c);
```

```
C75: compliment2s port map (xs75c, co75, xs75);
S76: shift4 port map (x6,clk,xs76s);
SS76:S4 port map (xs76s,clk, xs76c);
C76: compliment2s port map (xs76c, co76, xs76);
S77: shift4 port map (x7,clk, xs77s);
SS77:S4 port map (xs77s,clk, xs77);
S78: shift5 port map (x8, clk, xs78s);
SS78:S5 port map (xs78s,clk, xs78c);
C78: compliment2s port map (xs78c, co78, xs78);
S79: shift5 port map (x9, clk, xs79s);
SS79:S5 port map (xs79s,clk, xs79);
S710: shift4 port map (x10, clk, xs710s);
SS710:S4 port map (xs710s,clk, xs710c);
C710: compliment2s port map (xs710c, co710, xs710);
S711: shift6 port map (x11, clk, xs711s);
SS711:S6 port map (xs711s,clk, xs711);
S712: shift4 port map (x12, clk, xs712s);
SS712:S4 port map (xs712s,clk, xs712c);
C712: compliment2s port map (xs712c, co712, xs712);

A712: add8 port map (xs72, xs71, '0', Ca712, Ao712);
A734: add8 port map (xs73, xs74, '0', Ca734, Ao734);
A756: add8 port map (xs75, xs76, '0', Ca756, Ao756);
A778: add8 port map (xs77, xs78, '0', Ca778, Ao778);
A7910: add8 port map (xs79, xs710, '0', Ca7910, Ao7910);
A71112: add8 port map (xs711, xs712, '0', Ca71112, Ao71112);

A71234: add8 port map (Ao712, Ao734, '0', Ca71234, Ao71234);
A75678: add8 port map (Ao756, Ao778, '0', Ca75678, Ao75678);
A79101112: add8 port map (Ao7910, Ao71112, '0', Ca79101112, Ao79101112);

A71_8: add8 port map (Ao71234, Ao75678, '0', Ca71_8, Ao71_8);
```

```
AA07: add8 port map (Ao71_8, Ao79101112, '0', Cao7, AAo7);
AB7: add8 port map (AAo7,"11111110", '0',Cab7,ABo7);



S81: shift5 port map (x1, clk, xs81s);
SS81: S5 port map (xs81s, clk, xs81c);
C81: compliment2s port map (xs81c, co81, xs81);
S82: shift5 port map (x2,clk, xs82s);
SS82: S5 port map (xs82s,clk, xs82);
S83: shift1 port map (x3, clk, xs83s);
SS83: S1 port map (xs83s, clk, xs83c);
C83: compliment2s port map (xs83c, co83, xs83);
S84: shift3 port map (x4, clk, xs84s);
SS84: S3 port map (xs84s,clk, xs84c);
C84: compliment2s port map (xs84c, co84, xs84);
S85: shift5 port map (x5, clk, xs85s);
SS85: S5 port map (xs85s,clk, xs85c);
C85: compliment2s port map (xs85c, co85, xs85);
S86: shift5 port map (x6,clk,xs86s);
SS86:S5 port map (xs86s,clk, xs86c);
C86: compliment2s port map (xs86c, co86, xs86);
S87: shift2 port map (x7,clk, xs87s);
SS87:S2 port map (xs87s,clk, xs87c);
C87: compliment2s port map (xs87c, co87, xs87);
S88: shift5 port map (x8, clk, xs88s);
SS88:S5 port map (xs88s,clk, xs88c);
C88: compliment2s port map (xs88c, co88, xs88);
S89: shift5 port map (x9, clk, xs89s);
SS89:S5 port map (xs89s,clk, xs89c);
C89: compliment2s port map (xs89c, co89, xs89);
S810: shift1 port map (x10, clk, xs810s);
```

```
SS810:S1 port map (xs810s,clk, xs810c);
C810: compliment2s port map (xs810c, co810, xs810);
S811: shift4 port map (x11, clk, xs811s);
SS811:S4 port map (xs811s,clk, xs811c);
C811: compliment2s port map (xs811c, co811, xs811);
S812: shift4 port map (x12, clk, xs812s);
SS812:S4 port map (xs812s,clk, xs812c);
C812: compliment2s port map (xs812c, co812, xs812);



A812: add8 port map (xs82, xs81, '0', Ca812, Ao812);
A834: add8 port map (xs83, xs84, '0', Ca834, Ao834);
A856: add8 port map (xs85, xs86, '0', Ca856, Ao856);
A878: add8 port map (xs87, xs88, '0', Ca878, Ao878);
A8910: add8 port map (xs89, xs810, '0', Ca8910, Ao8910);
A81112: add8 port map (xs811, xs812, '0', Ca81112, Ao81112);


A81234: add8 port map (Ao812, Ao834, '0', Ca81234, Ao81234);
A85678: add8 port map (Ao856, Ao878, '0', Ca85678, Ao85678);
A89101112: add8 port map (Ao8910, Ao81112, '0', Ca89101112, Ao89101112);

A81_8: add8 port map (Ao81234, Ao85678, '0', Ca81_8, Ao81_8);
AA08: add8 port map (Ao81_8, Ao89101112, '0', Cbo8, ABo8);



RE1: relu8 port map (ABo1, y1);
RE2: relu8 port map (ABo2, y2);
RE3: relu8 port map (ABo3, y3);
```

```
RE4: relu8 port map (ABo4, y4);
RE5: relu8 port map (ABo5, y5);
RE6: relu8 port map (ABo6, y6);
RE7: relu8 port map (ABo7, y7);
RE8: relu8 port map (ABo8, y8);


end hidden2;

#hidden 3
entity Hidden_Layer3 is
port (x1, x2, x3, x4, x5, x6, x7, x8: in std_logic_vector (7 downto 0);
clk: std_logic; y1, y2, y3, y4 :out std_logic_vector(7 downto 0));
end Hidden_Layer3;

architecture hidden3 of Hidden_Layer3 is

component shift1 is
port (din: in std_logic_vector(7 downto 0); clk: in std_logic;
        dout: out std_logic_vector(8 downto 0));
end component;

component shift2 is
port (din: in std_logic_vector(7 downto 0); clk: in std_logic;
        dout: out std_logic_vector(9 downto 0));
end component;

component shift3 is
port (din: in std_logic_vector(7 downto 0); clk: in std_logic;
        dout: out std_logic_vector(10 downto 0));
end component;
```

```
component shift4 is
port (din: in std_logic_vector(7 downto 0); clk: in std_logic;
        dout: out std_logic_vector(11 downto 0));
end component;

component shift5 is
port (din: in std_logic_vector(7 downto 0); clk: in std_logic;
        dout: out std_logic_vector(12 downto 0));
end component;

component shift6 is
port (din: in std_logic_vector(7 downto 0); clk: in std_logic;
        dout: out std_logic_vector(13 downto 0));
end component;

component S6 is
port (din: in std_logic_vector(13 downto 0); clk: in std_logic;
        dout: out std_logic_vector(7 downto 0));
end component;

component S5 is
port (din: in std_logic_vector(12 downto 0); clk: in std_logic;
        dout: out std_logic_vector(7 downto 0));
end component;

component S4 is
port (din: in std_logic_vector(11 downto 0); clk: in std_logic;
        dout: out std_logic_vector(7 downto 0));
end component;

component S3 is
port (din: in std_logic_vector(10 downto 0); clk: in std_logic;
```

```
        dout: out std_logic_vector(7 downto 0));
end component;

component S2 is
port (din: in std_logic_vector(9 downto 0); clk: in std_logic;
        dout: out std_logic_vector(7 downto 0));
end component;

component S1 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
        dout: out std_logic_vector(7 downto 0));
end component;

component add8 is
port (A,B: in std_logic_vector(7 downto 0); Cin: in std_logic;
Co: out std_logic; Sum: out std_logic_vector(7 downto 0));
end component;

component relu8 is
port (x : in std_logic_vector(7 downto 0); y: out std_logic_vector(7 downto 0));
end component;

component compliment2s
port (Din: in std_logic_vector (7 downto 0); Co: out std_logic;
Do: out std_logic_vector (7 downto 0));
end component;



-- 6 shift
signal  xs11s, xs15s, xs17s, xs21s, xs22s, xs31s, xs32s, xs37s,
xs42s, xs47s : std_logic_vector(13 downto 0); -- 6 shift
-- 5 shift
```

```
signal  xs12s, xs13s, xs14s, xs18s, xs23s, xs24s, xs33s, xs35s,
xs38s, xs46s : std_logic_vector(12 downto 0); -- 5 shift
-- 4 shift
signal  xs25s, xs28s, xs44s, xs45s : std_logic_vector(11 downto 0); -- 4 shift
-- 3 shift
signal  xs34s, xs43s : std_logic_vector(10 downto 0); -- 3 shift
-- 2 shift
signal xs26s, xs36s, xs41s : std_logic_vector(9 downto 0); -- 2 shift
-- 1 shift
signal xs16s  : std_logic_vector(8 downto 0); -- 1 shift

signal xs11c, xs14c, xs16c, xs17c, xs18c, xs11, xs12, xs13, xs14,
xs15, xs16, xs17, xs18 : std_logic_vector(7 downto 0);
signal Ao112, Ao134, Ao156, Ao178, Ao11234, Ao15678, AAo1,
ABo1: std_logic_vector (7 downto 0);
signal co11, co14, co16, co17, co18, Ca112, Ca134, Ca156, Ca178,
Ca11234, Ca15678, Cao1, Cab1: std_logic;

signal xs22c, xs25c, xs26c, xs28c, xs21, xs22, xs23, xs24, xs25,
xs26, xs28 : std_logic_vector(7 downto 0);
signal Ao212, Ao234, Ao256, Ao278, Ao21234, Ao25678, ABo2,
AAo2: std_logic_vector (7 downto 0);
signal co22, co25, co26, co28, Ca212, Ca234, Ca256, Ca278,
Ca21234, Ca25678, Cao2, Cab2: std_logic;

signal xs34c, xs35c, xs36c, xs38c, xs31, xs32, xs33, xs34, xs35,
xs36, xs37, xs38 : std_logic_vector(7 downto 0);
signal Ao312, Ao334, Ao356, Ao378, Ao31234, Ao35678, AAo3,
ABo3: std_logic_vector (7 downto 0);
signal co34, co35, co36, co38, Ca312, Ca334, Ca356, Ca378,
Ca31234, Ca35678, Cao3, Cab3: std_logic;
```

```
signal xs44c, xs47c, xs41, xs42, xs43, xs44, xs45, xs46,
xs47  : std_logic_vector(7 downto 0);
signal Ao412, Ao434, Ao456, Ao478, Ao41234, Ao45678, AAo4,
ABo4 : std_logic_vector (7 downto 0);
signal co44, co47, Ca412, Ca434, Ca456, Ca478, Ca41234, Ca45678,
Cao4, Cab4 : std_logic;

begin
S11: shift6 port map (x1, clk, xs11s);
SS11: S6 port map (xs11s,clk,  xs11c);
C11: compliment2s port map (xs11c,co11, xs11);
S12: shift5 port map (x2,clk,  xs12s);
SS12: S5 port map (xs12s,clk, xs12);
S13: shift5 port map (x3,clk, xs13s);
SS13: S5 port map (xs13s,clk, xs13);
S14: shift5 port map (x5,clk, xs14s);
SS14: S5 port map (xs14s,clk, xs14c);
C14: compliment2s port map (xs14c, co14, xs14);
S15: shift6 port map (x5,clk, xs15s);
SS15: S6 port map (xs15s,clk, xs15);
S16: shift1 port map (x6,clk, xs16s);
SS16: S1 port map (xs16s,clk, xs16c);
C16: compliment2s port map (xs16c,co16, xs16);
S17: shift6 port map (x7,clk, xs17s);
SS17: S6 port map (xs17s,clk, xs17c);
C17: compliment2s port map (xs17c,co17, xs17);
S18: shift5 port map (x8,clk, xs18s);
SS18: S5 port map (xs18s,clk, xs18c);
C18: compliment2s port map (xs18c,co18, xs18);

A112: add8 port map (xs12, xs11, '0', Ca112, Ao112);
A134: add8 port map (xs13, xs14, '0', Ca134, Ao134);
```

```
A156: add8 port map (xs15, xs16, '0', Ca156, Ao156);
A178: add8 port map (xs17, xs18, '0', Ca178, Ao178);

A11234: add8 port map (Ao112, Ao134, '0', Ca11234, Ao11234);
A15678: add8 port map (Ao156, Ao178, '0', Ca15678, Ao15678);
AA01: add8 port map (Ao11234, Ao15678, '0', Cao1, AAo1);
AB1: add8 port map (AAo1, "11111000", '0', Cab1, ABo1);



S21: shift6 port map (x1,clk, xs21s);
SS21: S6 port map (xs21s,clk,xs21);
S22: shift6 port map (x2,clk, xs22s);
SS22: S6 port map (xs22s,clk, xs22c);
C22: compliment2s port map (xs22c, co22, xs22);
S23: shift5 port map (x3, clk, xs23s);
SS23: S5 port map (xs23s, clk, xs23);
S24: shift5 port map (x4, clk, xs24s);
SS24: S5 port map (xs24s,clk, xs24);
S25: shift4 port map (x5, clk, xs25s);
SS25: S4 port map (xs25s,clk, xs25c);
C25: compliment2s port map (xs25c, co25, xs25);
S26: shift2 port map (x6,clk, xs26s);
SS26: S2 port map (xs26s,clk, xs26c);
C26: compliment2s port map (xs26c, co26, xs26);
S28: shift4 port map (x8, clk, xs28s);
SS28:S4 port map (xs28s,clk, xs28c);
C28: compliment2s port map (xs28c, co28, xs28);

A212: add8 port map (xs22, xs21, '0', Ca212, Ao212);
A234: add8 port map (xs23, xs24, '0', Ca234, Ao234);
A256: add8 port map (xs25, xs26, '0', Ca256, Ao256);
```

```
A278: add8 port map (x7, xs28, '0', Ca278, Ao278);

A21234: add8 port map (Ao212, Ao234, '0', Ca21234, Ao21234);
A25678: add8 port map (Ao256, Ao278, '0', Ca25678, Ao25678);
AA02: add8 port map (Ao21234, Ao25678, '0', Cao2, AAo2);
AB2: add8 port map (AAo2, "00100000", '0', Cab2, ABo2);


S31: shift6 port map (x1,clk, xs31s);
SS31: S6 port map (xs31s,clk, xs31);
S32: shift6 port map (x2,clk, xs32s);
SS32: S6 port map (xs32s,clk, xs32);
S33: shift5 port map (x3, clk, xs33s);
SS33: S5 port map (xs33s, clk, xs33);
S34: shift3 port map (x4, clk, xs34s);
SS34: S3 port map (xs34s,clk, xs34c);
C34: compliment2s port map (xs34c, co34, xs34);
S35: shift5 port map (x5, clk, xs35s);
SS35: S5 port map (xs35s,clk, xs35c);
C35: compliment2s port map (xs35c, co35, xs35);
S36: shift2 port map (x6,clk, xs36s);
SS36: S2 port map (xs36s,clk, xs36c);
C36: compliment2s port map (xs36c, co36, xs36);
S37: shift6 port map (x7,clk, xs37s);
SS37: S6 port map (xs37s,clk, xs37);
S38: shift5 port map (x8, clk, xs38s);
SS38:S5 port map (xs38s, clk, xs38c);
C38: compliment2s port map (xs38c, co38, xs38);

A312: add8 port map (xs32, x1, '0', Ca312, Ao312);
A334: add8 port map (xs33, xs34, '0', Ca334, Ao334);
A356: add8 port map (xs35, xs36, '0', Ca356, Ao356);
```

```
A378: add8 port map (xs37, xs38, '0', Ca378, Ao378);

A31234: add8 port map (Ao312, Ao334, '0', Ca31234, Ao31234);
A35678: add8 port map (Ao356, Ao378, '0', Ca35678, Ao35678);
AA03: add8 port map (Ao31234, Ao35678, '0', Cao3, AAo3);
AB3: add8 port map (AAo3, "00100000", '0', Cab3, ABo3);



S41: shift2 port map (x1, clk, xs41s);
SS41: S2 port map (xs41s, clk, xs41);
S42: shift6 port map (x2,clk, xs42s);
SS42: S6 port map (xs42s,clk, xs42);
S43: shift3 port map (x3, clk, xs43s);
SS43: S3 port map (xs43s, clk, xs43);
S44: shift4 port map (x4, clk, xs44s);
SS44: S4 port map (xs44s,clk, xs44c);
C44: compliment2s port map (xs44c, co44,xs44);
S45: shift4 port map (x5, clk, xs45s);
SS45: S4 port map (xs45s,clk, xs45);
S46: shift5 port map (x6,clk, xs46s);
SS46:S5 port map (xs46s,clk, xs46);
S47: shift6 port map (x7,clk, xs47s);
SS47:S6 port map (xs47s,clk, xs47c);
C47: compliment2s port map (xs47c, co47, xs47);

A412: add8 port map (xs42, xs41, '0', Ca412, Ao412);
A434: add8 port map (xs43, xs44, '0', Ca434, Ao434);
A456: add8 port map (xs45, xs46, '0', Ca456, Ao456);
A478: add8 port map (xs47, x8, '0', Ca478, Ao478);

A41234: add8 port map (Ao412, Ao434, '0', Ca41234, Ao41234);
A45678: add8 port map (Ao456, Ao478, '0', Ca45678, Ao45678);
```

```
AA04: add8 port map (Ao41234, Ao45678, '0', Cao4, AAo4);
AB4: add8 port map (AAo4, "11110000", '0', Cab4, ABo4);

RE1: relu8 port map (ABo1, y1);
RE2: relu8 port map (ABo2, y2);
RE3: relu8 port map (ABo3, y3);
RE4: relu8 port map (ABo4, y4);

end hidden3;


#output layer

entity Out_Layer is
port (x1, x2, x3, x4 : in std_logic_vector (7 downto 0);
clk: std_logic; y1 :out std_logic_vector(7 downto 0));
end Out_Layer;

architecture Out_L of Out_Layer is

component shift1 is
port (din: in std_logic_vector(7 downto 0); clk: in std_logic;
        dout: out std_logic_vector(8 downto 0));
end component;

component shift2 is
port (din: in std_logic_vector(7 downto 0); clk: in std_logic;
        dout: out std_logic_vector(9 downto 0));
end component;

component shift3 is
port (din: in std_logic_vector(7 downto 0); clk: in std_logic;
```

```
        dout: out std_logic_vector(10 downto 0));
end component;

component shift4 is
port (din: in std_logic_vector(7 downto 0); clk: in std_logic;
        dout: out std_logic_vector(11 downto 0));
end component;

component shift5 is
port (din: in std_logic_vector(7 downto 0); clk: in std_logic;
        dout: out std_logic_vector(12 downto 0));
end component;

component shift6 is
port (din: in std_logic_vector(7 downto 0); clk: in std_logic;
        dout: out std_logic_vector(13 downto 0));
end component;

component S6 is
port (din: in std_logic_vector(13 downto 0); clk: in std_logic;
        dout: out std_logic_vector(7 downto 0));
end component;

component S5 is
port (din: in std_logic_vector(12 downto 0); clk: in std_logic;
        dout: out std_logic_vector(7 downto 0));
end component;

component S4 is
port (din: in std_logic_vector(11 downto 0); clk: in std_logic;
        dout: out std_logic_vector(7 downto 0));
end component;
```

```
component S3 is
port (din: in std_logic_vector(10 downto 0); clk: in std_logic;
        dout: out std_logic_vector(7 downto 0));
end component;

component S2 is
port (din: in std_logic_vector(9 downto 0); clk: in std_logic;
        dout: out std_logic_vector(7 downto 0));
end component;

component S1 is
port (din: in std_logic_vector(8 downto 0); clk: in std_logic;
        dout: out std_logic_vector(7 downto 0));
end component;

component add8 is
port (A,B: in std_logic_vector(7 downto 0); Cin: in std_logic;
Co: out std_logic; Sum: out std_logic_vector(7 downto 0));
end component;

component sig8 is
port (x: in std_logic_vector (7 downto 0); clk: std_logic;
sig: out std_logic_vector(7 downto 0));
end component;

component compliment2s
port (Din: in std_logic_vector (7 downto 0); Co: out std_logic;
Do: out std_logic_vector (7 downto 0));
end component;
```

```
-- 6 shift
signal  xs11s, xs12s, xs13s,
xs14s : std_logic_vector(13 downto 0); -- 6 shift


signal xs12c, xs13c, xs11, xs12, xs13,
xs14 : std_logic_vector(7 downto 0);
signal Ao112, Ao134, AAo1,
ABo1: std_logic_vector (7 downto 0);
signal co12, co13, Ca112, Ca134, Cao1, Cab1: std_logic;


begin
S11: shift6 port map (x1, clk, xs11s);
SS11: S6 port map (xs11s,clk,  xs11);
S12: shift6 port map (x2,clk,  xs12s);
SS12: S6 port map (xs12s,clk, xs12c);
C12: compliment2s port map (xs12c, co12, xs12);
S13: shift6 port map (x3,clk, xs13s);
SS13: S6 port map (xs13s,clk, xs13c);
C13: compliment2s port map (xs13c, co13, xs13);
S14: shift6 port map (x4,clk, xs14s);
SS14: S6 port map (xs14s,clk, xs14);

A112: add8 port map (xs12, xs11, '0', Ca112, Ao112);
A134: add8 port map (xs13, xs14, '0', Ca134, Ao134);

AA01: add8 port map (Ao112, Ao134, '0', Cao1, AAo1);
AB1: add8 port map (AAo1, "11110000", '0', Cab1, ABo1);

SIG1: sig8 port map (ABo1, clk, y1);
```

```
end Out_L;
```

Appendix E

PYTHON CODE: BENCHMARK OF SABINN

```
import keras,os
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPool2D , Flatten
from keras.preprocessing.image import ImageDataGenerator

# Classical array manipulation
import numpy as np

# Image manipulation | OpenCV
import cv2

# Showing images and evaluating model results
import matplotlib.pyplot as plt

# VGG19 Model
from keras.applications.vgg19 import VGG19

# Preparing VGG19 Model
from keras.layers import Dense,Flatten,Input
from keras.models import Sequential
```

```
# One hot label encoding
from keras.utils import to_categorical

# CIFAR10 dataset
from keras.datasets import cifar10
import keras as K


# Cifar10 Dataset Process
(x_train,y_train),(x_test,y_test) = cifar10.load_data()
print("Shape of x_train is ",x_train.shape)
print("Shape of y_train is ",y_train.shape)
print("Shape of x_test  is ",x_test.shape)
print("Shape of y_test  is",y_test.shape)

def resize_img(img):
    numberOfImage = img.shape[0]
    new_array = np.zeros((numberOfImage, 48,48,3))
    for i in range(numberOfImage):
        new_array[i] = cv2.resize(img[i,:,:,:],(48,48))
    return new_array

x_train = resize_img(x_train)
x_test = resize_img(x_test)
print("New shape of x_train is ",x_train.shape)
print("New shape of x_test  is ",x_test.shape)

# one hot encoding
y_train = to_categorical(y_train,num_classes=10)
y_test = to_categorical(y_test,num_classes=10)

print("New shape of y_train is ",y_train.shape)
```

```
print("New shape of y_test   is ",y_test.shape)

#MedMNIST Dataset
!pip install medmnist

import medmnist
from medmnist import INFO, Evaluator

data_flag = 'pneumoniamnist'
download = True

NUM_EPOCHS = 3
BATCH_SIZE = 128
lr = 0.0001

info = INFO[data_flag]
task = info['task']
n_channels = info['n_channels']
n_classes = len(info['label'])

DataClass = getattr(medmnist, info['python_class'])
# preprocessing
data_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[.5], std=[.5])
])

# load the data
train_dataset = DataClass(split='train',
transform=data_transform, download=download)
test_dataset = DataClass(split='test',
transform=data_transform, download=download)
```

```
pil_dataset = DataClass(split='train', download=download)

# encapsulate data into dataloader form
train_loader = data.DataLoader(dataset=train_dataset,
batch_size=BATCH_SIZE, shuffle=True)
train_loader_at_eval = data.DataLoader(dataset=train_dataset,
batch_size=2*BATCH_SIZE, shuffle=False)
test_loader = data.DataLoader(dataset=test_dataset,
batch_size=2*BATCH_SIZE, shuffle=False)

print(train_dataset)
print("==========")
print(test_dataset)

from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val =
train_test_split(train_dataset.imgs,train_dataset.labels ,test_size=0.2)
x_train = np.expand_dims(X_train, axis=-1)
x_val = np.expand_dims(X_val, axis=-1)

x_test = np.expand_dims(test_dataset.imgs, axis=-1)
y_test = np.expand_dims(test_dataset.labels, axis=-1)

X_train = tf.image.resize(x_train, [32, 32])
X_val = tf.image.resize(x_val, [32, 32])
X_test = tf.image.resize(x_test, [32, 32])
print(X_train.shape)
print(X_val.shape)
print(X_test.shape)
```

```
#VGG19
vgg_model = tf.keras.applications.VGG19(
    include_top=False,
    weights = None,
    input_shape=(32,32,1),
)

vgg_model.summary()

model = tf.keras.Sequential()
model.add(vgg_model)
model.add(Flatten())
model.add(Dense(512, activation = 'relu', use_bias =False))
model.add(Dense(128, activation = 'relu', use_bias =False))
model.add(Dense(1, activation = 's', use_bias =False))

model.summary()

BATCH_SIZE=10
EPOCHS = 25

optimizer = tf.keras.optimizers.SGD(
    learning_rate=0.0001,
    momentum=0.9)
acc = tf.keras.metrics.Accuracy(
    name='accuracy', dtype=None
)
prec = tf.keras.metrics.Precision(
    thresholds=None, top_k=None, class_id=None, name=None,
    dtype=None
)
bce = tf.keras.losses.BinaryCrossentropy(from_logits=False)
```

```
model.compile(loss=bce,
              optimizer=optimizer,
              metrics=acc)

history = model.fit(X_train, y_train, epochs=EPOCHS,
batch_size=BATCH_SIZE, validation_data = (X_val, y_val), verbose=0)

#for MEDMNIST [add this before fitting VGG19 model]
train_datagen = ImageDataGenerator(
    preprocessing_function = tf.keras.applications.vgg19.preprocess_input,
    rotation_range=10,
    zoom_range = 0.1,
    width_shift_range = 0.1,
    height_shift_range = 0.1,
    shear_range = 0.1,
    horizontal_flip = True
)
train_datagen.fit(X_train)

val_datagen = ImageDataGenerator(preprocessing_function
= tf.keras.applications.vgg19.preprocess_input)
val_datagen.fit(X_val)

#ResNET50
from keras.applications.resnet import ResNet50

res_model = ResNet50(include_top=False, weights=None,
input_shape=(32, 32, 1),classes=1)
res_model.summary()

model = tf.keras.Sequential()
model.add(res_model)
```

```
model.add(Flatten())
model.add(Dense(128, activation = 'relu', use_bias =False))
model.add(Dense(1, activation = 'sigmoid', use_bias =False))

model.summary()

#train
%%time

BATCH_SIZE=10
EPOCHS = 25

optimizer = tf.keras.optimizers.SGD(
    learning_rate=0.0001,
    momentum=0.9)
acc = tf.keras.metrics.Accuracy(
    name='accuracy', dtype=None
)
prec = tf.keras.metrics.Precision(
    thresholds=None, top_k=None, class_id=None, name=None, dtype=None
)
bce = tf.keras.losses.BinaryCrossentropy(from_logits=False)
model.compile(loss=bce,
              optimizer=optimizer,
              metrics=prec)

history = model.fit(X_train, y_train, epochs=EPOCHS,
batch_size=BATCH_SIZE, validation_data = (X_val, y_val), verbose=1)

#MobileNETV2

from keras.applications.mobilenet_v2 import MobileNetV2
```

```
mob_model = MobileNetV2(include_top=False, weights=None,
input_shape=(32, 32, 1),classes=1)
mob_model.summary()

model = tf.keras.Sequential()
model.add(mob_model)
model.add(Flatten())
model.add(Dense(128, activation = 'relu', use_bias =False))
model.add(Dense(1, activation = 'sigmoid', use_bias =False))



model.summary()

%%time

BATCH_SIZE=10
EPOCHS = 25

optimizer = tf.keras.optimizers.SGD(learning_rate=0.0001,momentum=0.9)
acc = tf.keras.metrics.Accuracy(name='accuracy', dtype=None)
prec = tf.keras.metrics.Precision(thresholds=None, top_k=None,
class_id=None, name=None, dtype=None)
bce = tf.keras.losses.BinaryCrossentropy(from_logits=False)
model.compile(loss=bce,optimizer=optimizer,metrics=acc)

history = model.fit(X_train, y_train, epochs=EPOCHS, batch_size=BATCH_SIZE,
validation_data = (X_val, y_val), verbose=0)



#evaluation and validation of the models
```

```
model.evaluate(X_test, y_test)
ys_test = np.squeeze(y_test)

#confusion metrix generation
from numpy import array
X_pred = array(X_test)
y_pred = model.predict(X_pred)

print(y_pred.shape,y_test.shape)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(ys_test,y_pred.round())
print(cm)

#precision
from sklearn.metrics import classification_report
print(classification_report(ys_test, y_pred.round()))
#matrix = [TP, FP,
#          FN, TN ]

from sklearn.metrics import roc_auc_score

# calculate roc auc
roc_auc = roc_auc_score(ys_test, y_pred)
print(roc_auc)

from sklearn.metrics import roc_curve
# roc curve for tpr = fpr
random_probs = [0 for i in range(len(ys_test))]
p_fpr, p_tpr, _ = roc_curve(ys_test, random_probs)

# roc curve for models
fpr, tpr, thresh = roc_curve(ys_test, y_pred)
```

```
# matplotlib
import matplotlib.pyplot as plt

# plot roc curves
plt.figure()
ax = plt.axes((0.1,0.1,0.7,0.8))
plt.plot(fpr, tpr, linestyle='—',color='orange',
label = 'VGG19 (%0.3f)' % roc_auc)
plt.plot(p_fpr, p_tpr, color='blue',
label = 'Baseline Model (0.50)')
# title
# x label

plt.xlabel('False Positive Rate', fontsize = 15)
# y label
plt.ylabel('True Positive rate', fontsize =15)

plt.legend(loc='best')
plt.show()

#binarization

#get weights from each layer = n
layer_name = layer's name
w_d= model.get_layer(layer_name).get_weights()
print(np.shape(w_d))

# Mean weight value calculation
m = tf.keras.metrics.Mean()
d1_w = m(tf.math.abs(bin1_squeezed))*tf.keras.backend.sign(bin1_squeezed)
d1_w= tf.reshape(d1_w, [N, N]).numpy() #where N is the Matrix
```

```
print(d1_w)

#binarize
b10_int= d1_w/M #mean value of each layer
b10 = b10_int.round()

#set the weights
layerName1 = layer name
b1= model.get_layer(layerName1).set_weights([b10])

#evaluation
_, accuracy = model.evaluate(X_test, ys_test)
print('test_Accuracy: %.2f' %(accuracy*100))

#confusion metrix generation
from numpy import array
X_pred = array(X_test)
yb_pred = modelb.predict(X_pred)

print(y_pred.shape, y_test.shape)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(ys_test, yb_pred.round())
print(cm)

#precision
from sklearn.metrics import classification_report
print(classification_report(ys_test, yb_pred.round()))
#matrix = [TP, FP,
#          FN, TN ]
```

BIBLIOGRAPHY

[1] S. Han, *Efficient methods and hardware for deep learning*. PhD thesis, Stanford University, 2017.

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[3] A. Page, N. Attaran, C. Shea, H. Homayoun, and T. Mohsenin, "Low-power manycore accelerator for personalized biomedical applications," in *Proceedings of the 26th edition on Great Lakes Symposium on VLSI*, pp. 63–68, 2016.

[4] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *ACM SIGARCH computer architecture news*, vol. 44, no. 3, pp. 367–379, 2016.

[5] E. L. Andrews, "Ai's carbon footprint problem," July 2020.

[6] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, *et al.*, "Mlperf inference benchmark," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 446–459, IEEE, 2020.

[7] G. E. Moore, "Lithography and the future of moore's law," in *Integrated Circuit Metrology, Inspection, and Process Control IX*, vol. 2439, pp. 2–17, SPIE, 1995.

[8] M. Shoaran, B. A. Haghi, M. Taghavi, M. Farivar, and A. Emami-Neyestanak, "Energy-efficient classification for resource-constrained biomedical applications," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 4, pp. 693–707, 2018.

[9] Y. Wei, J. Zhou, Y. Wang, Y. Liu, Q. Liu, J. Luo, C. Wang, F. Ren, and L. Huang, "A review of algorithm & hardware design for ai-based biomedical applications," *IEEE transactions on biomedical circuits and systems*, vol. 14, no. 2, pp. 145–163, 2020.

[10] C. Shea, A. Page, and T. Mohsenin, "Scalenet: A scalable low power accelerator for real-time embedded deep neural networks," in *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, pp. 129–134, 2018.

[11] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays*, pp. 161–170, 2015.

[12] A. Kulkarni, A. Page, N. Attaran, A. Jafari, M. Malik, H. Homayoun, and T. Mohsenin, "An energy-efficient programmable manycore accelerator for personalized biomedical applications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 1, pp. 96–109, 2017.

[13] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, "A 240 g-ops/s mobile coprocessor for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 682–687, 2014.

[14] O. Hassan, S. Shamsir, and S. K. Islam, "Machine learning based hardware model for a biomedical system for prediction of respiratory failure," in *2020 IEEE International Symposium on Medical Measurements and Applications (MeMeA)*, pp. 1–5, IEEE, 2020.

[15] O. Hassan, D. Parvin, and S. Kamrul, "Machine learning model based digital hardware system design for detection of sleep apnea among neonatal infants," in *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 607–610, IEEE, 2020.

[16] O. Hassan, T. Paul, M. H. Shuvo, D. Parvin, R. Thakker, M. Chen, A. S. M. Mosa, and S. K. Islam, "Energy efficient deep learning inference embedded on fpga for sleep apnea detection," *Journal of Signal Processing Systems*, pp. 1–11, 2022.

[17] M. M. H. Shuvo, O. Hassan, D. Parvin, M. Chen, and S. K. Islam, "An optimized hardware implementation of deep learning inference for diabetes prediction," in *2021 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, pp. 1–6, IEEE, 2021.

[18] L. D. Victor, "Obstructive sleep apnea," *American family physician*, vol. 60, no. 8, p. 2279, 1999.

[19] T. Young, L. Finn, P. E. Peppard, M. Szklo-Coxe, D. Austin, F. J. Nieto, R. Stubbs, and K. M. Hla, "Sleep disordered breathing and mortality: eighteen-year follow-up of the wisconsin sleep cohort," *Sleep*, vol. 31, no. 8, pp. 1071–1078, 2008.

[20] J. G. Suni Eric, "Sleep apnea: What it is, its risk factors, its health impacts, and how it can be treated," July 2021. [Online; posted 9-July-2021].

[21] S. S. Mostafa, J. P. Carvalho, F. Morgado-Dias, and A. Ravelo-García, "Optimization of sleep apnea detection using spo2 and ann," in *2017 XXVI international conference on information, communication and automation technologies (ICAT)*, pp. 1–6, IEEE, 2017.

[22] M. Cheng, W. J. Sori, F. Jiang, A. Khan, and S. Liu, "Recurrent neural network based classification of ecg signal features for obstruction of sleep apnea detection," in *2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, vol. 2, pp. 199–202, IEEE, 2017.

[23] D. Dey, S. Chaudhuri, and S. Munshi, "Obstructive sleep apnoea detection using convolutional neural network based deep learning framework," *Biomedical engineering letters*, vol. 8, no. 1, pp. 95–100, 2018.

[24] S. S. Mostafa, F. Mendonça, F. Morgado-Dias, and A. Ravelo-García, "Spo2 based sleep apnea detection using deep learning," in *2017 IEEE 21st international conference on intelligent engineering systems (INES)*, pp. 000091–000096, IEEE, 2017.

[25] H. Qin and G. Liu, "A dual-model deep learning method for sleep apnea detection based on representation learning and temporal dependence," *Neurocomputing*, vol. 473, pp. 24–36, 2022.

[26] N. Selvaraj and R. Narasimhan, "Automated prediction of the apnea-hypopnea index using a wireless patch sensor," in *2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 1897–1900, IEEE, 2014.

[27] M. Kopaczka, O. Oezkan, and D. Merhof, "Face tracking and respiratory signal analysis for the detection of sleep apnea in thermal infrared videos with head movement," in *New Trends in Image Analysis and Processing–ICIAP 2017: ICIAP International Workshops, WBICV, SSPandBE, 3AS, RGBD, NIVAR, IWBAAS, and MADiMa 2017, Catania, Italy, September 11-15, 2017, Revised Selected Papers 19*, pp. 163–170, Springer, 2017.

[28] X. Wang, M. Cheng, Y. Wang, S. Liu, Z. Tian, F. Jiang, and H. Zhang, "Obstructive sleep apnea detection using ecg-sensor with convolutional neural networks," *Multimedia Tools and Applications*, vol. 79, no. 23, pp. 15813–15827, 2020.

[29] Y. Zhou, D. Shu, H. Xu, Y. Qiu, P. Zhou, W. Ruan, G. Qin, J. Jin, H. Zhu, K. Ying, *et al.*, "Validation of novel automatic ultra-wideband radar for sleep apnea detection," *Journal of thoracic disease*, vol. 12, no. 4, p. 1286, 2020.

[30] S. Akbarian, N. M. Ghahjaverestan, A. Yadollahi, and B. Taati, "Noncontact sleep monitoring with infrared video data to estimate sleep apnea severity and distinguish between positional and nonpositional sleep apnea: Model development and experimental validation," *Journal of Medical Internet Research*, vol. 23, no. 11, p. e26524, 2021.

[31] O. Hassan, R. Thakker, T. Paul, D. Parvin, A. S. M. Mosa, and S. K. Islam, "Sabinn: Fpga implementation of shift accumulate binary neural network model for real-time automatic detection of sleep apnea," in *2022 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, pp. 1–6, IEEE, 2022.

[32] O. Hassan, T. Paul, N. Amin, T. Twisha, R. Thakker, D. Parvin, A. S. M. Mosa, and S. K. Islam, "An optimized hardware inference of sabinn: Shift-accumulate binarized neural network for sleep apnea detection," *IEEE Transactions on Instrumentation and Measurement*, 2023.

[33] P. Várady, T. Micsik, S. Benedek, and Z. Benyó, "A novel method for the detection of apnea and hypopnea events in respiration signals," *IEEE Transactions on Biomedical Engineering*, vol. 49, no. 9, pp. 936–942, 2002.

[34] J. V. Marcos, R. Hornero, D. Alvarez, F. del Campo, and C. Zamarrón, "Assessment of four statistical pattern recognition techniques to assist in obstructive sleep apnoea diagnosis from nocturnal oximetry," *Medical engineering & physics*, vol. 31, no. 8, pp. 971–978, 2009.

[35] P. De Chazal, C. Heneghan, E. Sheridan, R. Reilly, P. Nolan, and M. O'Malley, "Automated processing of the single-lead electrocardiogram for the detection of obstructive sleep apnoea," *IEEE transactions on biomedical engineering*, vol. 50, no. 6, pp. 686–696, 2003.

[36] D. Novák, K. Mucha, and T. Al-Ani, "Long short-term memory for apnea detection based on heart rate variability," in *2008 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 5234–5237, IEEE, 2008.

[37] B. Yılmaz, M. H. Asyalı, E. Arıkan, S. Yetkin, and F. Özgen, "Sleep stage and obstructive apneaic epoch classification using single-lead ecg," *Biomedical engineering online*, vol. 9, no. 1, pp. 1–14, 2010.

[38] R. Wei, X. Zhang, J. Wang, and X. Dang, "The research of sleep staging based on single-lead electrocardiogram and deep neural network," *Biomedical engineering letters*, vol. 8, no. 1, pp. 87–93, 2018.

[39] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," *Cited on*, vol. 14, no. 8, p. 2, 2012.

[40] T. Penzel, G. B. Moody, R. G. Mark, A. L. Goldberger, and J. H. Peter, "The apnea-ecg database," in *Computers in Cardiology 2000. Vol. 27 (Cat. 00CH37163)*, pp. 255–258, IEEE, 2000.

[41] A. L. Goldberger, L. A. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, "Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals," *circulation*, vol. 101, no. 23, pp. e215–e220, 2000.

[42] D. López-García, M. Ruz, J. Ramírez, and J. Górriz, "Automatic detection of sleep disorders: Multi-class automatic classification algorithms based on support vector machines," in *Conf. Time Ser. Forecast.(ITISE 2018)*, vol. 3, pp. 1270–1280, 2018.

[43] A. Rahimi, A. Safari, and M. Mohebbi, "Sleep stage classification based on ecg-derived respiration and heart rate variability of single-lead ecg signal," in *2019 26th National and 4th International Iranian Conference on Biomedical Engineering (ICBME)*, pp. 158–163, IEEE, 2019.

[44] Y. Zhao, J. Zhao, and Q. Li, "Derivation of respiratory signals from single-lead ecg," in *2008 International Seminar on Future BioMedical Information Engineering*, pp. 15–18, IEEE, 2008.

[45] A. Sawant, R. L. Smith, R. B. Venkat, L. Santanam, B. Cho, P. Poulsen, H. Cattell, L. J. Newell, P. Parikh, and P. J. Keall, "Toward submillimeter accuracy in the management of intrafraction motion: the integration of real-time internal position monitoring and multileaf collimator target tracking," *International Journal of Radiation Oncology* Biology* Physics*, vol. 74, no. 2, pp. 575–582, 2009.

[46] L. Liu, W. Chen, and G. Cao, "Prediction of neonatal amplitude-integrated eeg based on lstm method," in *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pp. 497–500, IEEE, 2016.

[47] M. J. Banner, N. R. Euliano, V. Brennan, C. Peters, A. J. Layon, and A. Gabrielli, "Power of breathing determined noninvasively with use of an artificial neural network in patients with respiratory failure," *Critical care medicine*, vol. 34, no. 4, pp. 1052–1059, 2006.

[48] B. R. F. F. P. E. M. A. M. E. B. J. R. Bataille, Benoit, "Integrated use of bedside lung ultrasound and echocardiography in acute respiratory failure: A prospective observational study in icu," *Chest*, vol. 146, pp. 1586–1593, 2014.

[49] I. Mahbub, M. S. Hasan, S. A. Pullano, F. Quaiyum, C. P. Stephens, S. K. Islam, A. S. Fiorillo, M. S. Gaylord, V. Lorch, and N. Beitel, "A low power wireless apnea detection system based on pyroelectric sensor," in *2015 IEEE Topical Conference on Biomedical Wireless Technologies, Networks, and Sensing Systems (BioWireleSS)*, pp. 1–3, IEEE, 2015.

[50] F. Mendonça, S. S. Mostafa, A. G. Ravelo-García, F. Morgado-Dias, and T. Penzel, "Devices for home detection of obstructive sleep apnea: A review," *Sleep medicine reviews*, vol. 41, pp. 149–160, 2018.

[51] S. Shamsir, S. H. Hesari, S. K. Islam, I. Mahbub, S. A. Pullano, and A. S. Fiorillo, "Instrumentation of a pyroelectric transducer based respiration monitoring system with wireless telemetry," in *2018 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, pp. 1–6, IEEE, 2018.

[52] S. A. Pullano, I. Mahbub, M. G. Bianco, S. Shamsir, S. K. Islam, M. S. Gaylord, V. Lorch, and A. S. Fiorillo, "Medical devices for pediatric apnea monitoring and therapy: past and new trends," *IEEE reviews in biomedical engineering*, vol. 10, pp. 199–212, 2017.

[53] S. Shamsir, O. Hassan, and S. K. Islam, "Smart infant-monitoring system with machine learning model to detect physiological activities and ambient conditions," in *2020 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, pp. 1–6, IEEE, 2020.

[54] A. Yüzer, H. Sümbül, and K. Polat, "A novel wearable real-time sleep apnea detection system based on the acceleration sensor," *Irbm*, vol. 41, no. 1, pp. 39–47, 2020.

[55] S. Kristiansen, K. Nikolaidis, T. Plagemann, V. Goebel, G. M. Traaen, B. Øverland, L. Aakerøy, T.-E. Hunt, J. P. Loennechen, S. L. Steinshamn, *et al.*, "Machine learning for sleep apnea detection with unattended sleep monitoring at home," *ACM Transactions on Computing for Healthcare*, vol. 2, no. 2, pp. 1–25, 2021.

[56] G. Ye, H. Yin, T. Chen, H. Chen, L. Cui, and X. Zhang, "Fenet: A frequency extraction network for obstructive sleep apnea detection," *IEEE Journal of Biomedical and Health Informatics*, vol. 25, no. 8, pp. 2848–2856, 2021.

[57] F. Mendonça, S. S. Mostafa, F. Morgado-Dias, and A. G. Ravelo-García, "An oximetry based wireless device for sleep apnea detection," *Sensors*, vol. 20, no. 3, p. 888, 2020.

[58] S. Hanson and L. Pratt, "Comparing biases for minimal network construction with back-propagation," *Advances in neural information processing systems*, vol. 1, 1988.

[59] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," *Advances in neural information processing systems*, vol. 2, 1989.

[60] B. Hassibi and D. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," *Advances in neural information processing systems*, vol. 5, 1992.

[61] F. Manessi, A. Rozza, S. Bianco, P. Napoletano, and R. Schettini, "Automated pruning for deep neural network compression," in *2018 24th International conference on pattern recognition (ICPR)*, pp. 657–664, IEEE, 2018.

[62] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu, "Discrimination-aware channel pruning for deep neural networks," *Advances in neural information processing systems*, vol. 31, 2018.

[63] A. Aghasi, A. Abdi, N. Nguyen, and J. Romberg, "Net-trim: Convex pruning of deep neural networks with performance guarantee," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[64] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems*, vol. 28, 2015.

[65] C. J. C. B. Yann LeCun, Corinns Cortes, "The mnist database."

[66] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[67] E. Ravussin, M. E. Valencia, J. Esparza, P. H. Bennett, and L. O. Schulz, "Effects of a traditional lifestyle on obesity in pima indians," *Diabetes care*, vol. 17, no. 9, pp. 1067–1074, 1994.

[68] C. Varon, A. Caicedo, D. Testelmans, B. Buyse, and S. Van Huffel, "A novel algorithm for the automatic detection of sleep apnea from single-lead ecg," *IEEE Transactions on Biomedical Engineering*, vol. 62, no. 9, pp. 2269–2278, 2015.

[69] C. Song, K. Liu, X. Zhang, L. Chen, and X. Xian, "An obstructive sleep apnea detection approach using a discriminative hidden markov model from ecg signals," *IEEE Transactions on Biomedical Engineering*, vol. 63, no. 7, pp. 1532–1542, 2015.

[70] Z. Zhang, "Improved adam optimizer for deep neural networks," in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pp. 1–2, IEEE, 2018.

[71] M. C. Mukkamala and M. Hein, "Variants of rmsprop and adagrad with logarithmic regret bounds," in *International conference on machine learning*, pp. 2545–2553, PMLR, 2017.

[72] Z. Zhang and M. Sabuncu, "Generalized cross entropy loss for training deep neural networks with noisy labels," *Advances in neural information processing systems*, vol. 31, 2018.

[73] C. J. Willmott and K. Matsuura, "Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance," *Climate research*, vol. 30, no. 1, pp. 79–82, 2005.

[74] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *Advances in neural information processing systems*, vol. 28, 2015.

[75] A. Bytyn, R. Leupers, and G. Ascheid, "Convaix: An application-specific instruction-set processor for the efficient acceleration of cnns," *IEEE Open Journal of Circuits and Systems*, vol. 2, pp. 3–15, 2020.

[76] C. Marimuthu, P. Thangaraj, and A. Ramesan, "Low power shift and add multiplier design," *arXiv preprint arXiv:1006.1179*, 2010.

[77] A. Hussein, V. Gaudet, H. Mostafa, and M. Elmasry, "A 16-bit high-speed low-power hybrid adder," in *2016 28th International Conference on Microelectronics (ICM)*, pp. 313–316, IEEE, 2016.

[78] R. Rafati, S. M. Fakhraie, and K. C. Smith, "A 16-bit barrel-shifter implemented in data-driven dynamic logic ($d3l$)," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 53, no. 10, pp. 2194–2202, 2006.

[79] A. Hazarika, A. Jain, S. Poddar, and H. Rahaman, "Shift and accumulate convolution processing unit," in *TENCON 2019-2019 IEEE Region 10 Conference (TENCON)*, pp. 914–919, IEEE, 2019.

[80] A.-M. Šimundić, "Measures of diagnostic accuracy: basic definitions," *ejifcc*, vol. 19, no. 4, p. 203, 2009.

[81] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[82] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "Optimizing loop operation and dataflow in fpga acceleration of deep convolutional neural networks," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 45–54, 2017.

[83] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, pp. 1–12, 2017.

[84] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*, pp. 525–542, Springer, 2016.

[85] D. Parvin, O. Hassan, T. Oh, and S. K. Islam, "Design of a smart maximum power point tracker (mppt) for rf energy harvester," *International Journal of High Speed Electronics and Systems*, vol. 29, no. 01n04, p. 2040006, 2020.

[86] R. T. Edwards, "Google/skywater and the promise of the open pdk,"

[87] "Github repository of sabinn model using google+skywater pdk," November 2021.

[88] M. Bahrami and M. Forouzanfar, "Sleep apnea detection from single-lead ecg: A comprehensive analysis of machine learning and deep learning algorithms," *IEEE Transactions on Instrumentation and Measurement*, vol. 71, pp. 1–11, 2022.

[89] S. Hu, W. Cai, T. Gao, and M. Wang, "A hybrid transformer model for obstructive sleep apnea detection based on self-attention mechanism using single-lead ecg," *IEEE Transactions on Instrumentation and Measurement*, vol. 71, pp. 1–11, 2022.

[90] M. Yeo, H. Byun, J. Lee, J. Byun, H.-Y. Rhee, W. Shin, and H. Yoon, "Robust method for screening sleep apnea with single-lead ecg using deep residual network: Evaluation with open database and patch-type wearable device data," *IEEE Journal of Biomedical and Health Informatics*, vol. 26, no. 11, pp. 5428–5438, 2022.

[91] M. M. Moussa, Y. Alzaabi, and A. H. Khandoker, "Explainable computer-aided detection of obstructive sleep apnea and depression," *IEEE Access*, vol. 10, pp. 110916–110933, 2022.

[92] M. H. Chyad, S. K. Gharghan, H. Q. Hamood, A. S. H. Altayyar, S. L. Zubaidi, and H. M. Ridha, "Hybridization of soft-computing algorithms with neural network for prediction obstructive sleep apnea using biomedical sensor measurements," *Neural Computing and Applications*, vol. 34, no. 11, pp. 8933–8957, 2022.

[93] K. Li, W. Pan, Y. Li, Q. Jiang, and G. Liu, "A method to detect sleep apnea based on deep neural network and hidden markov model using single-lead ecg signal," *Neurocomputing*, vol. 294, pp. 94–101, 2018.

[94] M. J. Lado, X. A. Vila, L. Rodríguez-Liñares, A. J. Méndez, D. N. Olivieri, and P. Félix, "Detecting sleep apnea by heart rate variability analysis: assessing the validity of databases and algorithms," *Journal of medical systems*, vol. 35, pp. 473–481, 2011.

[95] D. Álvarez, A. Cerezo-Hernández, A. Crespo, G. C. Gutiérrez-Tobal, F. Vaquerizo-Villar, V. Barroso-García, F. Moreno, C. A. Arroyo, T. Ruiz, R. Hornero, *et al.*, "A machine learning-based test for adult sleep apnoea screening at home using oximetry and airflow," *Scientific reports*, vol. 10, no. 1, pp. 1–12, 2020.

[96] L. Almazaydeh, K. Elleithy, M. Faezipour, and A. Abushakra, "Apnea detection based on respiratory signal classification," *Procedia Computer Science*, vol. 21, pp. 310–316, 2013.

[97] H. Luo, L. Zhang, L. Zhou, X. Lin, Z. Zhang, and M. Wang, "Design of real-time system based on machine learning for snoring and osa detection," in *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1156–1160, IEEE, 2022.

[98] M. Yeo, H. Byun, J. Lee, J. Byun, H.-Y. Rhee, W. Shin, and H. Yoon, "Respiratory event detection during sleep using electrocardiogram and respiratory related signals: Using polysomnogram and patch-type wearable device data," *IEEE Journal of Biomedical and Health Informatics*, vol. 26, no. 2, pp. 550–560, 2021.

[99] X. Yan, L. Wang, J. Zhu, S. Wang, Q. Zhang, and Y. Xin, "Automatic obstructive sleep apnea detection based on respiratory parameters in physiological signals," in *2022 IEEE International Conference on Mechatronics and Automation (ICMA)*, pp. 461–466, IEEE, 2022.

[100] A. John, K. K. Nundy, B. Cardiff, and D. John, "Multimodal multiresolution data fusion using convolutional neural networks for iot wearable sensing," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 15, no. 6, pp. 1161–1173, 2021.

[101] R. Parmar, M. Janveja, G. Trivedi, P. Jan, and Z. Nemec, "An area and power efficient vlsi architecture to detect obstructive sleep apnea for wearable devices," in *2022 32nd International Conference Radioelektronika (RADIOELEKTRONIKA)*, pp. 1–5, IEEE, 2022.

[102] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[103] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[104] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

[105] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[106] J. Yang, R. Shi, D. Wei, Z. Liu, L. Zhao, B. Ke, H. Pfister, and B. Ni, "Medmnist v2-a large-scale lightweight benchmark for 2d and 3d biomedical image classification," *Scientific Data*, vol. 10, no. 1, p. 41, 2023.

[107] R. Kundu, R. Das, Z. W. Geem, G.-T. Han, and R. Sarkar, "Pneumonia detection in chest x-ray images using an ensemble of deep learning models," *PloS one*, vol. 16, no. 9, p. e0256630, 2021.

[108] J. Brownlee, "Transfer learning for deep learning," September 2019.

[109] P. Baheti, "Kernel description," September 2019.

[110] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals of operations research*, vol. 134, pp. 19–67, 2005.

[111] T.-J. Yang, Y.-H. Chen, J. Emer, and V. Sze, "A method to estimate the energy consumption of deep neural networks," in *2017 51st asilomar conference on signals, systems, and computers*, pp. 1916–1920, IEEE, 2017.

[112] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 10–14, IEEE, 2014.

[113] V. Leon, S. Xydis, D. Soudris, and K. Pekmestzi, "Energy-efficient vlsi implementation of multipliers with double lsb operands," *IET Circuits, Devices & Systems*, vol. 13, no. 6, pp. 816–821, 2019.

[114] J. S. Sahoo and N. K. Rout, "Comparative study on low power barrel shifter/rotator at 45nm technology," *International Journal of Advanced Engineering and Nano Technology (IJAENT)*, vol. 2, no. 6, pp. 11–18, 2015.

[115] "Artificial intelligence market size report, 2022-2030," November 2022.

[116] T. Rizzo, S. Strangio, and G. Iannaccone, "Time domain analog neuromorphic engine based on high-density non-volatile memory in single-poly cmos," *IEEE Access*, vol. 10, pp. 49154–49166, 2022.

[117] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, *et al.*, "Guruguhanathan, venkataramanan, yi-hsin weng, andreas wild, yoonseok yang, and hong wang. 2018. loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[118] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Ai and ml accelerator survey and trends," in *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–10, IEEE, 2022.

VITA

Omiya Hassan is born and raised in Dhaka, Bangladesh. After completing her undergraduate studies at the United International University, Bangladesh, she traveled to the USA to pursue her doctorate in Electrical Engineering at the University of Missouri, Columbia. She worked as a research assistant in Prof. Syed Kamrul Islam's Analog/Mixed-Signal, VLSI and Devices Laboratory for two years and later as a graduate instructor in the Department of Electrical Engineering and Computer Science (EECS) at the University of Missouri.

Ms. Hassan's Ph. D. research topic focuses on developing and designing low-power integrated circuits (IC) while venturing through the field of AI/Machine-Learning techniques on edge. She is a 2021 Graduate Fellow of the IEEE Instrumentation and Measurement Society, a 2022 Rising Star of Electrical Engineering and Computer Science, and an NSF iRedefined fellow of 2023. She has won numerous awards throughout her career namely, the prestigious 2022 outstanding undergraduate research mentor of the Year from the University of Missouri, the 2021 outstanding doctoral student of Electrical and Computer Engineering, and the 1907 Women in Engineering Award from the College of Engineering at the University of Missouri.

Besides researching on developing future technology, she is a professionally trained vocalist in traditional South-Asian music and has experience freelancing for over five years in digital illustration.