APPROVAL SHEET

Title of Thesis: A Low Power On-board Processor for a Tongue Assistive Device

Name of Candidate: Sina Viseh

Master of Science, 2014

Thesis and Abstract Approved:

Dr.Tinoosh Mohsenin, Department of

Computer Science and

Electrical Engineering

Date Approved: 08/08/2019

Curriculum Vitae

Name: Sina Viseh.

Degree and date to be conferred: Master of Science, August, 2014.

Secondary education: SAS High School, Tehran, Iran.

Collegiate institutions attended:

University of Maryland Baltimore County, M.S Computer Engineering, 2014.

I.A.U South Tehran, B.C, 2012.
Major: Computer Engineering.

Professional Publications:

S.Viseh, A.Acevedo, M. Ghovanloo and T. Mohsenin
"Towards A Low Power FPGA Implementation for A Stand-Alone Intraoral Tongue Drive System"
39th Annual GOMACTech Conference, April 2014.

Professional positions held:

Graduate research assistant, University of Maryland Baltimore County.
(August 2013 - Present)

ABSTRACT

Title of Document: A Low Power On-board Processor for a Tongue Assistive Device

Sina Viseh, Master of Science, 2014

Directed By: Dr.Tinoosh Mohsenin
Department of Computer Science and Electrical Engineering

In biomedical wearable devices, patient's convenience and accuracy are the main priorities. To fulfill the patient's convenience requirement, the power consumption, which directly translates to the battery lifetime and size, must be kept as low as possible. Meanwhile, adopted improvements should not impact the accuracy. Therefore, focus on reducing the energy consumption within these devices has already been the subject of a significant amount of research in the past few years. In most wearable devices, all raw data is transmitted to a computer to carry out the required processing. This vast amount of communication leads to a considerable amount of power consumption and the need for a bulky battery, which hinders the device's practicality and patient's convenience. Tongue Drive System (TDS) is a new unobtrusive, wireless, and wearable assistive device that allows for real time tracking of the voluntary tongue motion in the oral space for communication, control, and navigation applications. The intraoral TDS clasps to the upper teeth and resists sensor

misplacement. However, the iTDS has more restrictions on its dimensions, limiting the battery size and consequently requiring a considerable reduction in its power consumption to operate over an extended period of two days on a single charge. In this thesis, we propose an ultra low power local processor for the TDS that performs all signals processing on the transmitter side, following the sensors. Implementing the computational engine reduces the data volume that needs to be wirelessly transmitted to a PC or smartphone by a factor of 30x, from 12 kbps to ~400 bps. The proposed design is implemented on an ultra low power IGLOO nano FPGA and is tested on AGLN250 prototype board. According to our post place and route results, implementing the engine on the FPGA significantly drops the required data transmission, while an ASIC implementation in 65 nm CMOS results in 0.128 mW power consumption and occupies a 0.02 mm^2 footprint. To explore a different architecture, we mapped our proposed TDS processor on the EEHPC many-core. The many-core has a flexible and time saving design procedure. As a result of having a local processor, the power consumption and size of the iTDS will be significantly reduced through the use of a much smaller rechargeable battery. Moreover, the system can operate longer following every recharge, improving the iTDS usability.

A Low Power On-board Processor for a Tongue Assistive Device

By

Sina Viseh

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, Baltimore County, in partial fulfillment
of the requirements for the degree of
Master of Science
2014

DEDICATION

I would like to dedicate this thesis to my family. My parents and sister have been very supportive and encouraging throughout my undergraduate and graduate career. They have helped me stay focused and determined. Without their unconditional love and support, I would not have been able to come this far.

ACKNOWLEDGMENTS

I owe gratitude to all those who have made this thesis possible. Firstly, I would like to thank my advisor, Dr. Tinoosh Mohsenin, for her guidance and support. Her experience and knowledge has helped me in my research work and I would like to thank her for giving me this opportunity to work with her. I would also like to thank my committee members, Dr. Tim Oates and Dr. Mohamed Younis for all their feedback and guidance. Their feedback has been very valuable to my research and myself. I would also like to thank Dr. Maysam Ghovanloo from Georgia Institute of Technology for giving me the opportunity to work on Tongue Drive System (TDS) for all his guidance and providing us the TDS test database. I would also like to thank all my friends at UMBC for all of their helps.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1. Motivation

In biomedical wearable devices, accuracy and the patient's convenience are the main priorities. To fulfill the patient's convenience requirement, the power consumption, which directly translates to the battery lifetime and size, must be kept as low as possible. Meanwhile, adopted improvements in power consumption should not impact the accuracy. Therefore, reducing the energy consumption of these devices has already been the subject of a significant amount of research in the past few years [1]–[5]. In wearable devices without a local processor, all raw data is transmitted to a computer to carry out the required processes. This vast amount of communication leads to considerable power consumption and the need for a bulky battery, which hinders the device's practicality and the patient's convenience. In a study in [6], it was shown that for continuous streaming of a single-lead ECG signal (with no on-board processing), the radio alone was responsible for up to 50% of the total power consumption (see Figure 1.1). However, having an on-board processor can reduce the radio power usage. The processor can be used to process the raw data and only transmit the important extracted features or transmit the final classification result. Applying a local processor to the system helps reduce the power consumption and, consequently, the size and weight of the device. Consequently, designing a very energy efficient on-board processor becomes challenging. This improvement also

helps the device's practicality by increasing the operation time, making it possible for the device to operate with every single recharge.

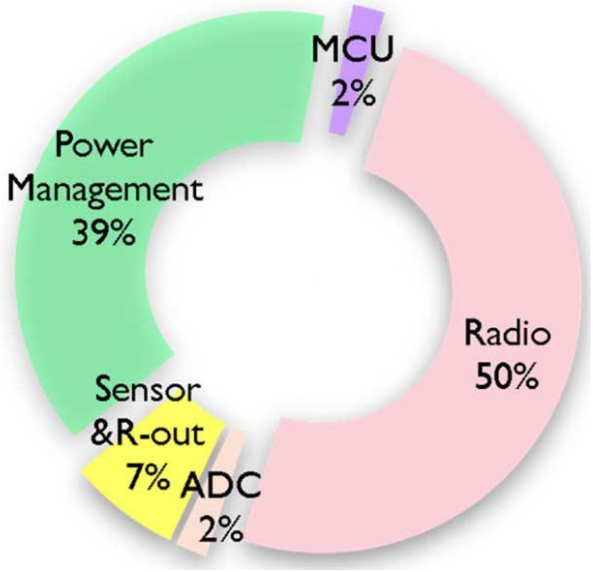


Figure 1.1. Power distribution of an ECG Necklace transmitting raw ECG data [6].

1.2. Contribution

In this thesis, we propose an on-board processor which performs all signals processing at the sensor level, instead of sending raw data for the case of a wearable assistive device called Tongue Drive System (TDS). TDS, developed at GT-BIONICS LAB of the Georgia Institute of Technology, is an unobtrusive, minimally invasive, and wireless tongue-operated assistive technology that can potentially replace some hand functions with tongue functions. The on-board processor reduces data transmission and therefore abates power consumption. In the previous versions of TDS, all sensed signals were transmitted to a computer to perform noise cancelation and classification algorithms.

We first carried out various analyses to perceive the causes of existing misclassification errors in TDS. Thereafter, we tested our proposed local processor with a data set, collected for TDS at the Georgia Institute of Technology and Northwestern University. We implemented our local processor on commercial FPGA, ASIC and a customized low power Many-Core Processor developed by EEHPC. Several optimizations are intended to make the processor very space and energy efficient. We selected the most suited classification algorithm with acceptable accuracy and reasonable area utilization. Afterwards, we optimized a number of required training samples. We shared the arithmetic logics between EMF cancelation and classification modules to save area. We also designed the arithmetic logics with specific bit resolution to enable sharing without affecting accuracy. The proposed local processing approach shows a promising impact on the data transmission by reducing the data transmission by a factor of 30 and consequently reducing the power consumption. Additionally, we mapped our proposed architecture on EEHPC Many-Core. Applied optimization reduced the battery size and consequently the device size, and it also augments the battery lifetime, which is beneficial to TDS practicality.

1.3. Organization of Thesis

In the second chapter, we discuss the issues with the existing TDS and compare different noise cancelation and classification algorithms. Chapter 3 is dedicated to FPGA implementation and results, while Chapter 4 explains the ASIC implementation results and ASIC comparison with FPGA results. Chapter 5 describes the implementation results on a Many-Core Processor and different implementations

with different numbers of cores. Chapter 6 concludes this thesis with the discussion of future work.

Chapter 2: Data Analysis and Proposed Architecture

2.1. TDS

Tongue motion as an untapped modality for motor function has the potential to serve as a control mechanism for disabled individuals. To this end, a few tongue-operated Assistive Technologies (AT), such as the Tongue-Touch-Keypad [1], Jouse2 [8], and Integra Mouse [9], have been developed. However, these technologies are limited due to their large size, requirements for specific head movements, and potential for causing fatigue. Tongue Drive System (TDS), developed at GT-BIONICS LAB at the Georgia Institute of Technology, is an unobtrusive, minimally invasive, and wireless tongue-operated AT that can potentially replace some hand functions with tongue functions [10]. The TDS architecture and performance by able-bodied subjects and those with severe physical disabilities (tetraplegia) have been previously evaluated and reported [16]-[10]. Prior versions of TDS are categorized into two main groups, iTDS and eTDS. The iTDS is in the shape of a dental retainer and is placed inside the mouth while the eTDS is in shape of a headset [15]. In all previous TDS versions, the raw sensed data is transmitted to a PC or a smartphone to perform all signal processing and command classification. This large amount of communication results in high power consumption; consequently, the device needs a bulky battery. Taking advantage of a local processor that can perform all signals processing at the sensors instead of sending raw data will significantly lower the power consumption and correspondingly augment the operation time and decrease the

battery size. In this thesis, we propose to implement a local processor for the current version of TDS. Figure 2.1 shows the eTDS appearance and its application in daily life [16].

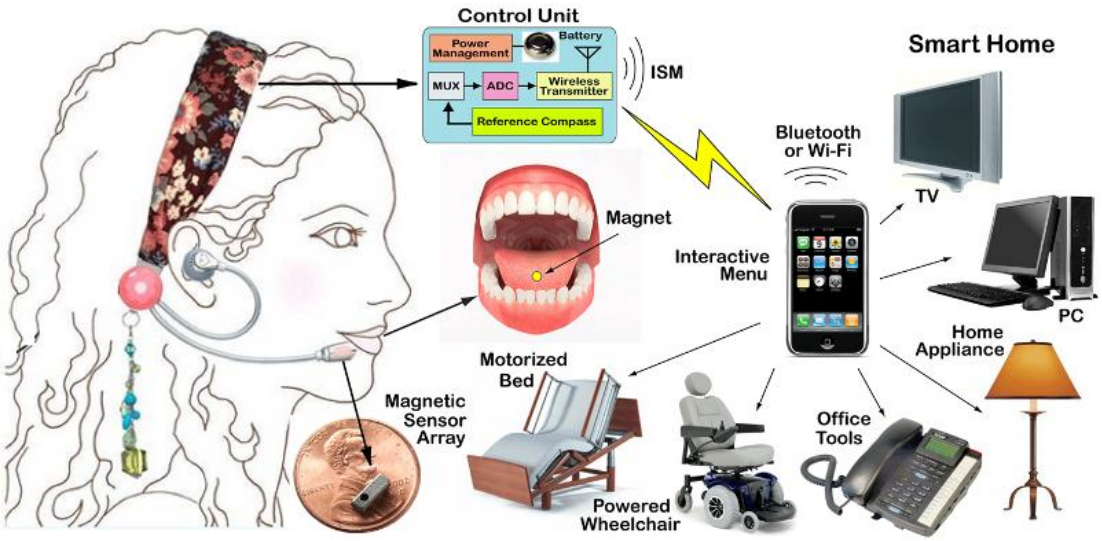


Figure 2.1. TDS and its applications [13].

Figure 2.2 shows the sensors' locations in the TDS. Two sensors are located in the front, and the two others are located in the rear. The TDS consists of four electromagnetic sensors, each of which returns three values for three axes. These twelve values are used to approximate noise and then classify the signals into seven discrete commands. Figure 2.3 shows the sensed signals of each sensor. Each sensor detects a permanent magnet's electromagnetic field and redundant electromagnetic fields. The redundant electromagnetic field consists of Earth's electromagnetic field and other environmental interferences. In the following sections, we investigate the cause of existing errors in the TDS. Figure 2.4 shows the twelve sensed signals from sensors before noise and classification algorithms are performed.

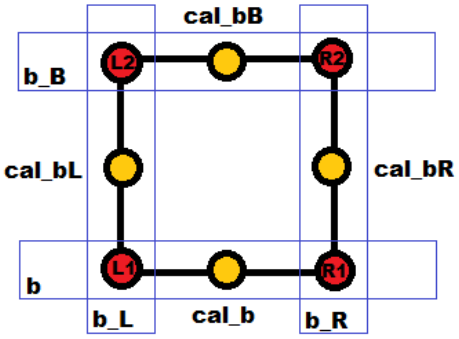


Figure 2.2. TDS Sensor alignment [16].

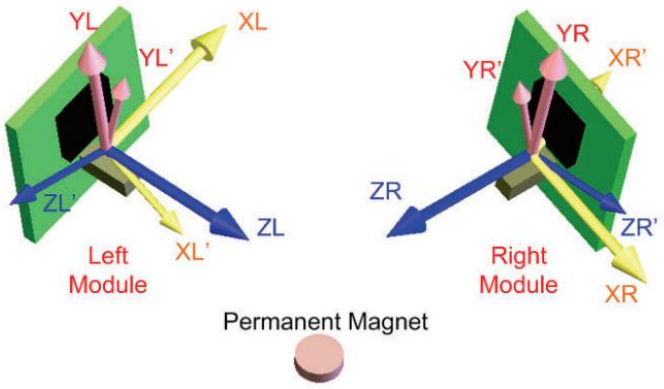


Figure. 2.3. Sensed signals in TDS [17].

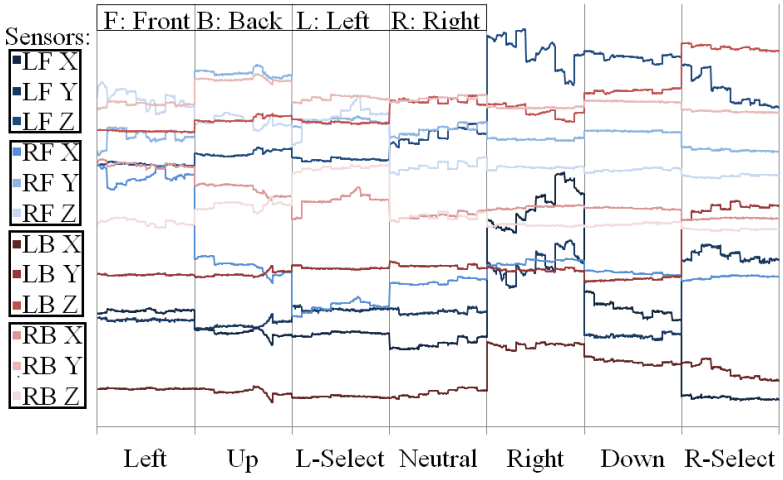


Figure 2.4. Signal waveform before noise cancelation and classification [18].

Figure 2.5 shows the seven discrete commands and the location of the tongue for issuing each of them.

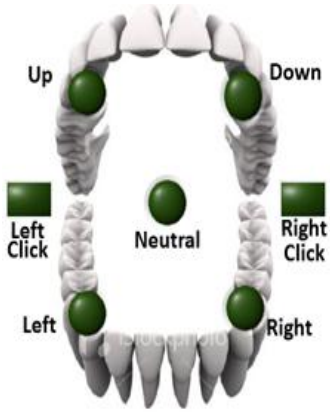


Figure 2.5. The seven discrete commands in TDS [19].

2.2. Investigating Training Error

In order to probe the training data and ensure that the current error rate is not caused by a failed training process, an analysis has been conducted to calculate the training error rate. First, we randomly selected 30% of training samples and used the remaining 70% to create a model. Subsequently, the formed model has been used to classify the selected 30%. The procedure was repeated 30 times to reach a meticulous average training error rate of 0.04%.

2.3. Visualizing the Data

In another investigation, we visualized a range of data to recognize the modality of the problem. The result showed that the range of training and test data are disparate in some commands. Figure 2.6 shows the range of training and test data in all seven commands in a data set with a considerable error rate of 55.17%, using conventional noise cancelation methods.

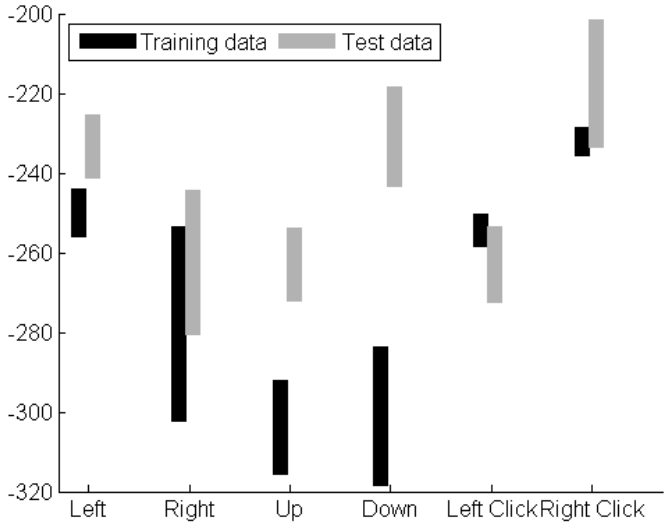


Figure 2.6. The range of training and test data in one single feature for one of the patient datasets after using a linear regression model to attenuate the Earth's electromagnetic interference.

This disparity can be a result of inexactitude in the current noise cancelation algorithm or a discrepancy in the location of the sensors. In the current noise cancelation method, a sample has been collected to create a model for the Earth's electromagnetic field [17]. The samples are processed by linear regression to determine four coefficients that will be used to approximate the Earth's electromagnetic field. Thereafter, the approximated electromagnetic field will be subtracted from sensed values to obtain the permanent magnet's electromagnetic field (see Equation 2.1).

$$X = X_{Left\ front\ sensor} - a_x X_{Left\ rear\ sensor} - b_x Y_{Left\ rear\ sensor} - c_x Z_{Left\ rear\ sensor} - d_x \quad (2.1)$$

The first speculation is that the discrepancy between the range of training and the test date is caused by the usual inexactitude in linear regression's computation. To probe

this speculation, one sensor in the rear was selected as a reference. Thereafter, it was subtracted from two front sensors. The notion behind this approach is that sensed values from each sensor can be modeled as a summation of data and noise (2.2). Subsequently, the subtraction of a reference value from another sensor's value results in the elimination of noise (see Equation 2.3).

$$X_{Left\ front} = X_{data_{Left\ front}} + X_{Noise} \tag{2.2}$$

$$X_{Right\ front} = X_{data_{Right\ front}} + X_{Noise}$$

$$X_{Reference} = X_{data_{Reference}} + X_{Noise}$$

$$X_{Left\ front} - X_{Reference} = X_{data_{Left\ front}} - X_{data_{Reference}}$$

$$X_{Right\ front} - X_{Reference} = X_{data_{Right\ front}} - X_{data_{Reference}} \tag{2.3}$$

The subtraction approach in Figure 2.7 does not succeed in eliminating the existing discrepancy between the test and training data in the same commands as Figure 2.6.

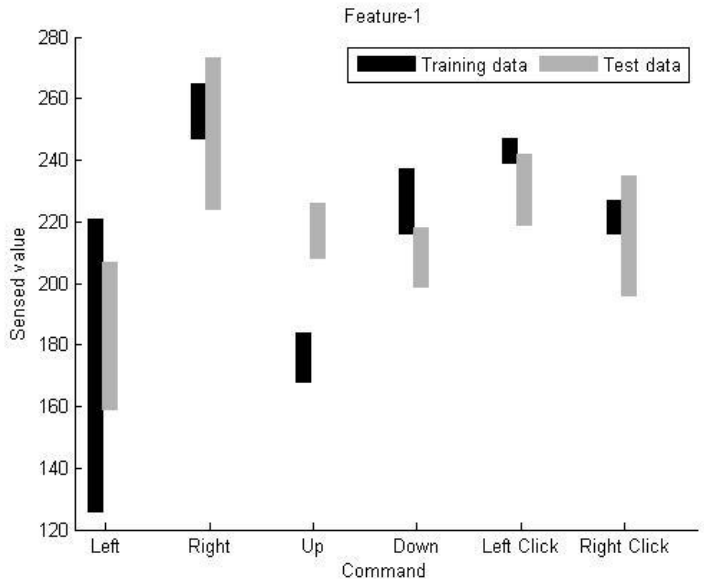


Figure 2.7. The range of training and test data in one single feature for the same patient dataset as Figure 2.6 after using the subtraction approach to attenuate the Earth's

electromagnetic interference.

The second speculation is that the discrepancy is caused by sensor misplacement. In the following sections, we investigate the problem from a domain change perspective.

2.4. Data Normalization

In another investigation, training and test data were normalized independently using standard deviation and mean (2.4). The classification result of normalized data showed that it not only improved the classification but also almost doubled the error rate, showing a 13.38% misclassification rate.

$$z = \frac{x - \mu}{\sigma} \tag{2.4}$$

2.5. Investigation of Domain Change

In pursuit of distribution change speculation, further experiments have been carried out. First, the training and test data have been incorporated and labeled as training and test. Afterwards, 30% of the shuffled incorporated data has been randomly selected to be used as a test set, while the remaining 70% is used to create a model. The created model has been tested to classify data into training and test data. The results showed that, with an uncomplicated classifier such as KNN, training and test data could be discriminated into training and test data with 99% accuracy. Figures 2.8 and 2.9 show the training and test data in the Up command; for one of the six features in two different data sets, one has a considerable error rate (Figure 2.8), and the other has 100% classification accuracy (Figure 2.9). In the data set with a high

error rate, the gap between the training and test data is wider by a factor of two.

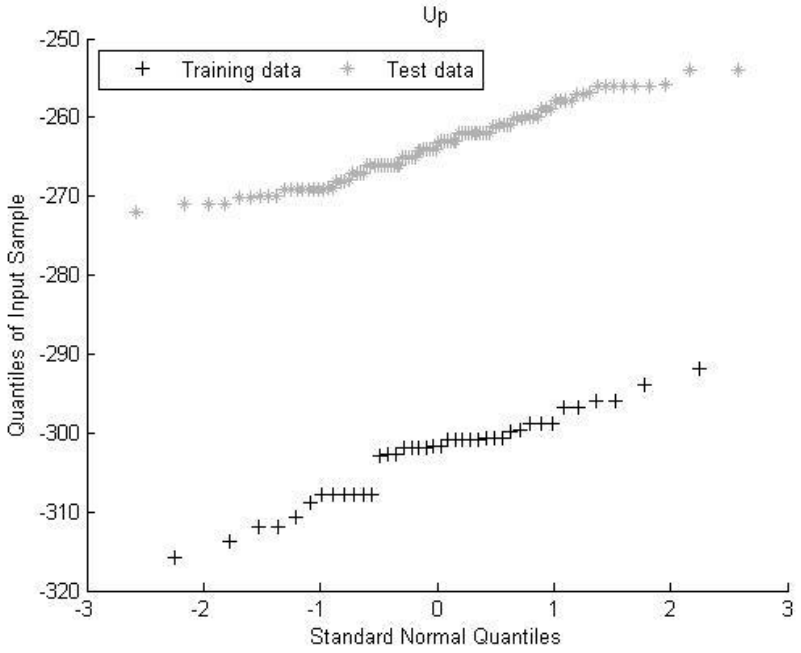


Figure 2.8. Distribution of training and test data in one feature for the Up command in a data set with 55.17% misclassification rate.

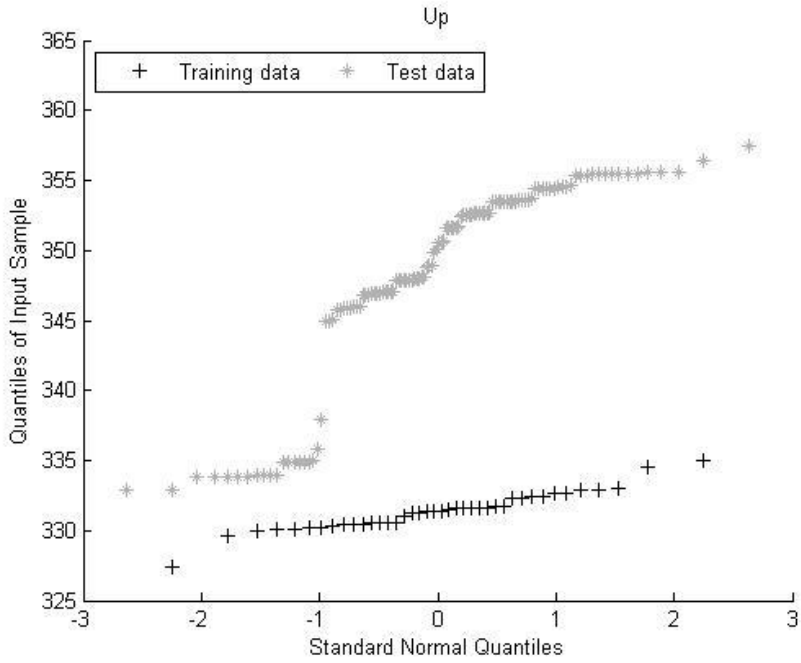


Figure 2.9. Distribution of training and test data in one feature for the Up command in

a data set with 100% accurate classification.

2.6. Analyzing Overfitting

In another investigation, we probed the affiliation of the number of training samples for every single command and error rate. First, 60 samples were selected randomly for every single command. Afterwards, we classified the test data using a specific number of training samples for every command, ranging from one to 60 samples. Figure 2.10 shows the affiliation of the number of training samples for each command and the misclassification rate. The plot almost flattened after 20 samples per command, and after that point, incrementing the number of samples representing any command does not cause a dramatic change in the error rate. Discussed results lead us to the conclusion that distribution discrepancy is the prime cause of the current error rate.

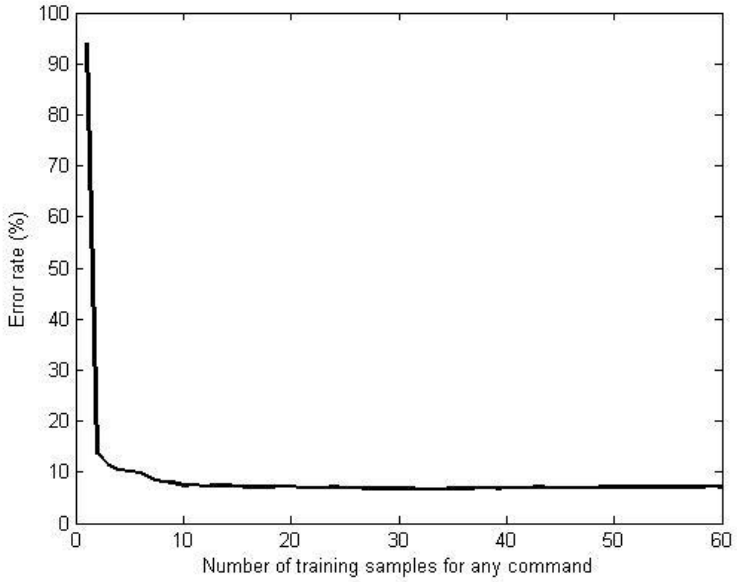


Figure 2.10.Affiliation of the number of training samples for every single command with misclassification rate.

2.7. Proposed Low Power On-Board Processor

Previous work has shown that the robustness of eTDS in real-life conditions could not be guaranteed because of the poor mechanical stability of the headset. Moreover, when using eTDS, the classification error with basic machine learning algorithms, such as K Nearest Neighbor (KNN), Quadratic, and Linear classifiers, could not be lowered any more than 4.8%; this accuracy is achieved by taking the majority vote among nine different classifiers which are not area and energy efficient for hardware implementation [20]. Our recent studies on the same 40 datasets, which were collected at the Georgia Institute of Technology and Northwestern University, also indicate that using other algorithms, such as Support Vector Machine (SVM), Naïve Bayes, and logistic regression, did not show a consistent and remarkable improvement across available patient datasets. Our investigations emphasize the importance of iTDS, which is potentially more mechanically stable and accurate.

2.7.1. Challenges

The key challenge to design a local processor in TDS is decreasing the power consumption and improving the battery lifetime while consequently reducing the battery size, which will be beneficial to the patient's comfort. In pursuit of this objective, the proposed processor should be small enough to fit in an ultra-low power FPGA, and its ASIC implementation should result in a very small footprint. Furthermore, applied hardware optimizations should not have an impact on the accuracy and device reliability. Consequently, classifier, data path word width design, resource allocation, and training data size should be arranged in a manner to satisfy

all of the restrictions. We initially designed the prototype on a very small and low-power FPGA and then implemented on ASIC 65 nm CMOS technology. The only available FPGA that can meet our design criteria is a low-power AGLN250 IGLOO Nano FPGA with a 6 mm by 6 mm package dimension. Figure 2.11 shows the existing iTDS prototype and its compact dimensions.

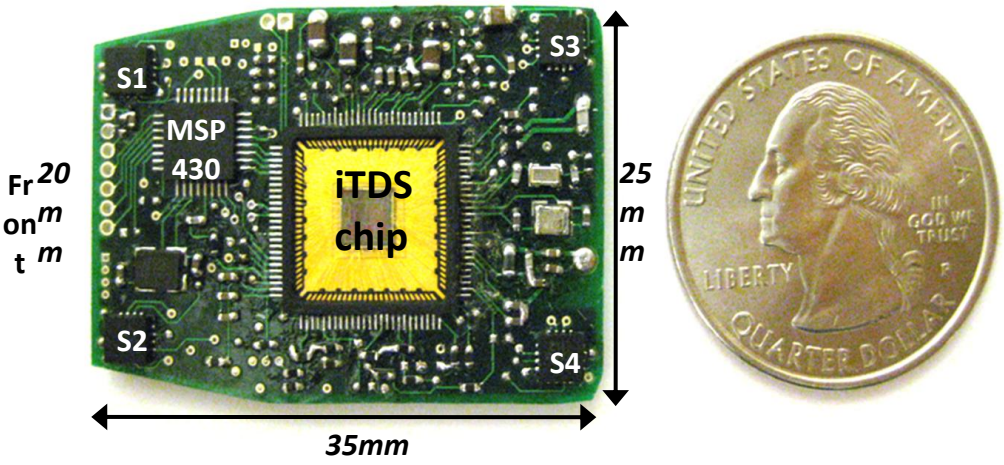


Figure 2.11. The iTDS existing prototype [21].

2.7.2. Classifier Algorithm Selection

Among different classification algorithms and configurations, such as single and double stage classifiers, single stage KNN Euclidean, SVM with Linear and Polynomial kernel, and a two-stage algorithm with Quadratic classifier as the first stage, and KNN Euclidean as the second stage, have reasonable accuracy. Figure 2.12 shows the accuracy of four different classification algorithms in four datasets and the average accuracy through all forty datasets. Although none of the four presented classifiers have a consistent accuracy advantage across all datasets, the KNN Euclidian and a double stage classifier, consisting of Quadratic and KNN Euclidean,

are more accurate, at 93.32%. The SVM with linear and polynomial kernels have 92.83% and 91.14% accuracy, respectively.

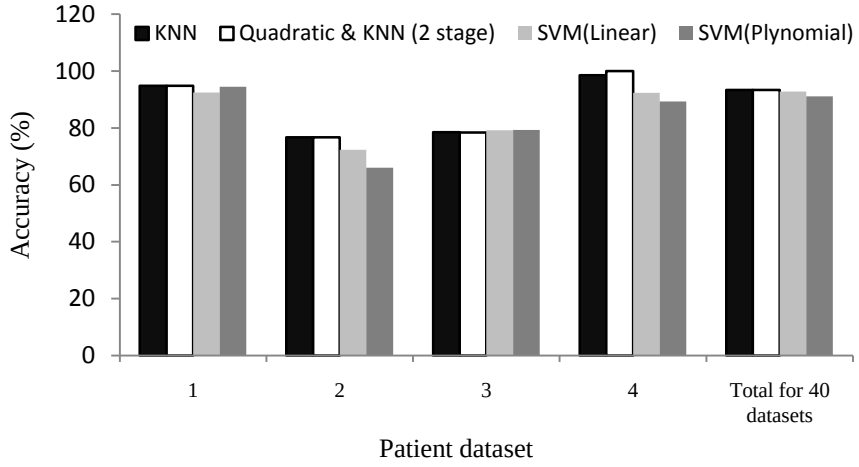


Figure 2.12. Command detection accuracy comparison of four different classifiers for sample datasets and the total average for 40 datasets.

In another investigation, we compared the device utilization of classifiers on an AGLN250 IGLOO FPGA. Figure 2.13 shows the accuracy and device utilization of single stage and double stage algorithms with different combinations of KNN and Quadratic classifiers. The single stage KNN Euclidean combined with the double stage algorithm with Quadratic classifier as the first stage and KNN Euclidean as the second stage classifier have 93.32% accuracy; however, the single stage KNN Euclidean utilizes four times fewer resources on AGLN250.

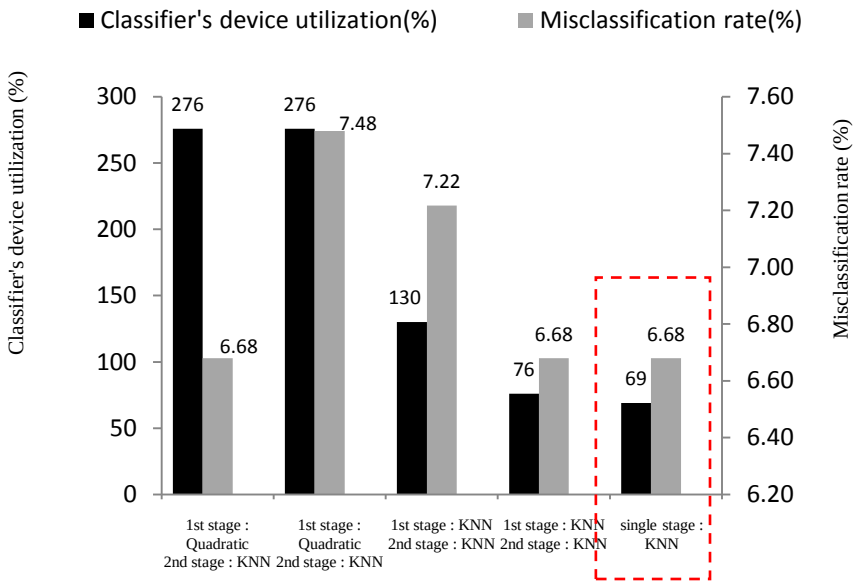
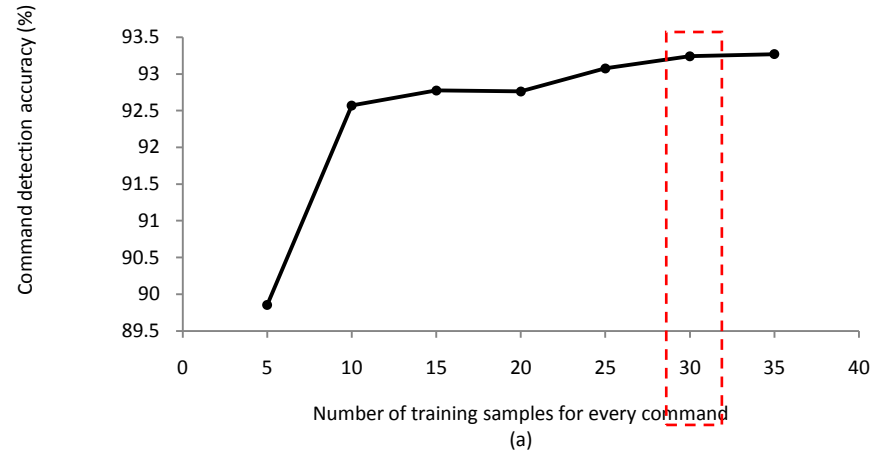


Figure 2.13. Comparison of device utilization on AGLN250 and accuracy in single stage and double stage classification algorithms. The dashed rectangle represents the design that is used.

2.7.3. Training Optimization

In another experiment, we investigated the impact of the number of training samples representing every single command in the training set on both accuracy and hardware complexity. Figure 2.14.a shows the impact of the training samples on accuracy. As shown in the plot, for fewer than ten samples, it has a considerable inaccuracy; however, it reaches an acceptable error rate at 25 samples. The accuracy plot flattened after 30 samples and reached 93.32% accuracy. The bar plots in Figure 2.13.b show the impact of increasing training size on the post place and route power consumption and area, respectively. Reduction of training samples results in a

considerable reduction in device utilization and also reduces the device utilization by 10% from 40 to 30 samples.



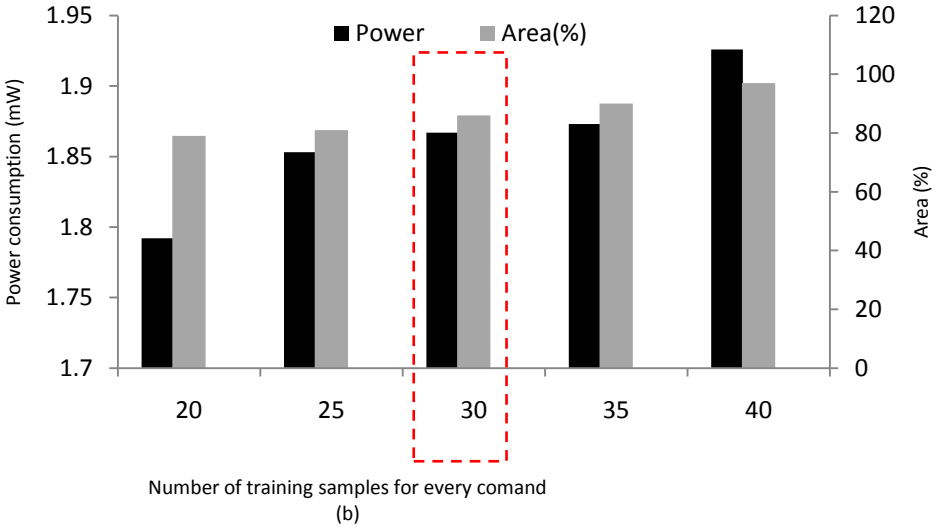


Figure 2.14. The impact of increasing training size on the command detection (a) and on the post place and route power consumption and area of the design on AGLN250 (b) with the optimal value of 30 samples, which is used for the design.

Our proposed architecture is shown in Figure 2.15. Our proposed architecture consists of a Serial Peripheral Interface, an Electromagnetic Field attenuation algorithm, and a KNN classifier [22].

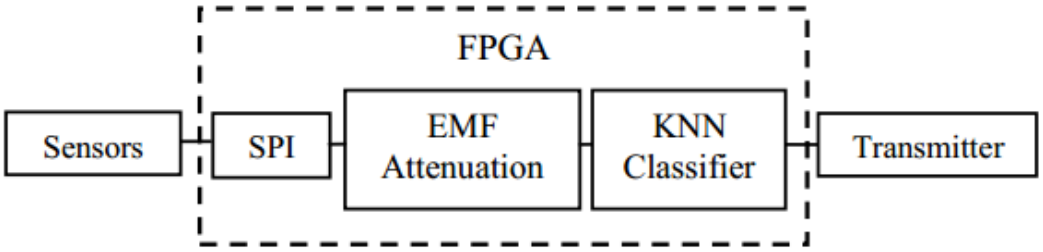


Figure 2.15. Basic block diagram for the local processor.

Chapter 3: FPGA Implementation and Results

3.1. Data Path Word Width Reduction and Bit Resolution Optimization

The data path word width of the processing blocks in the proposed architecture directly determines the required memory capacity, routing complexity, circuit area, and critical path delays. Moreover, it affects the amount of switching activity on wires and logic gates, and thus it affects the power dissipation. In order to measure the impact of word length on design size, ignoring the arithmetic unit and considering only the training lookup table, a training set with 12-bit data word has 20,160 bits, while the same training set with 11-bit data word has 18,480 bits. Figure 3.1 shows the impact of the number of ADC bits on the accuracy. Although the plot shows that with 10 bits ADC, we can achieve the same accuracy as 12 bits, the proposed architecture uses all 12 bits of ADC to make the system resistant to signal saturation. Figure 3.2 shows the impact of the number of decimal bits in noise cancelation coefficients and total error rate. Our proposed architecture has eight fixed decimal bits for noise cancelation coefficients.

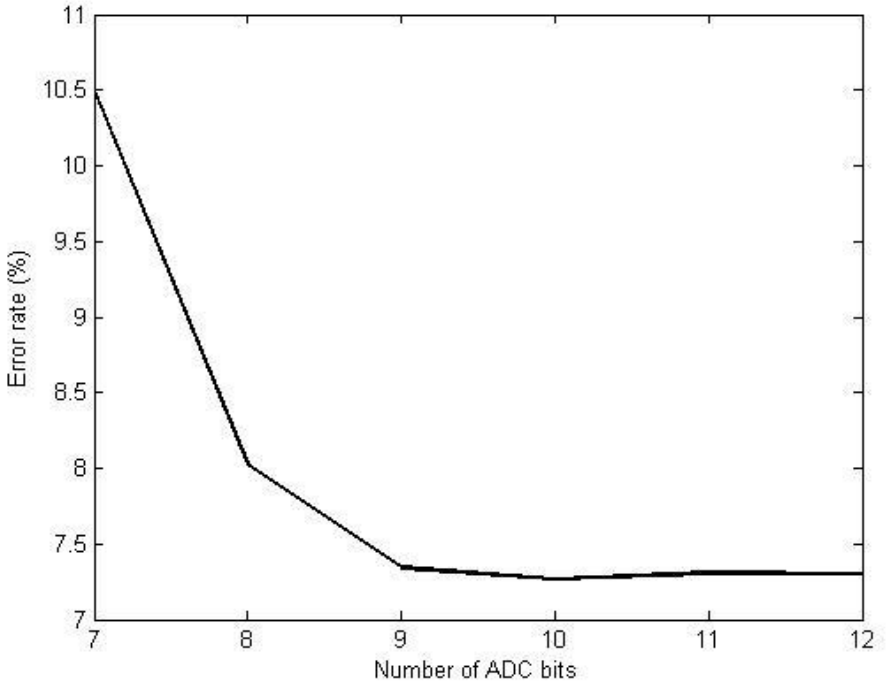


Figure 3.1. The affiliation of the number of integer bits in ADC and error rate.

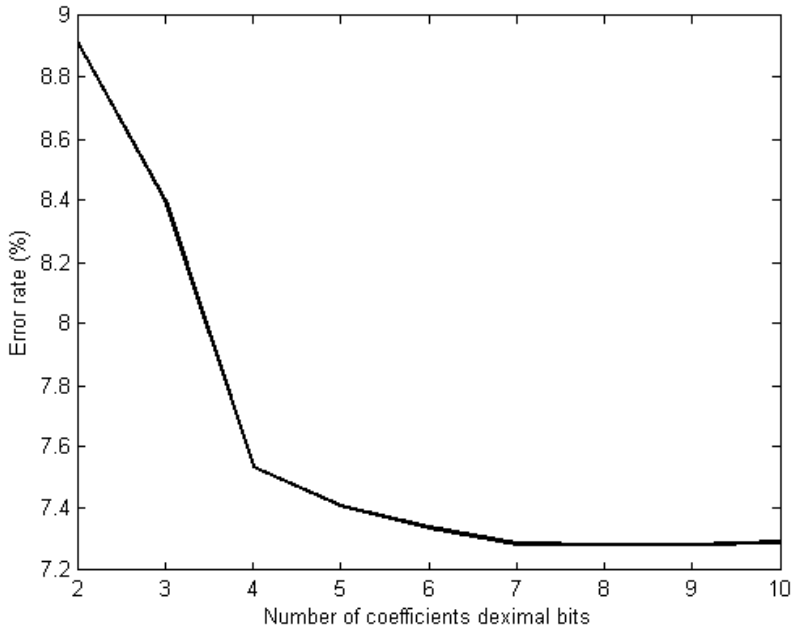


Figure 3.2. Affiliation of the number of decimal bits in noise cancelation coefficients and error rate.

3.2. Shared Processing

Solely relying on optimizing the classifier and data path word width does not provide enough space for all components to be implemented in the AGLN250 FPGA. The EMF cancellation unit and KNN classifier are using an adder and a multiplier such that, with a well-designed architecture, they can share these two resources in order to save space. As a result, the next important consideration is to choose the right multiplier and adder input length so that they can be used for both KNN classifier and EMF cancelation while not affecting the accuracy. Figure 3.3 shows a high-level block diagram of the design. The EMF cancelation lookup table is 480 bits, and the KNN training lookup table is 20,160 bits. The state machine is in charge of issuing all control signals. Sensed values are read from sensors through the SPI interface, and values are stored in FIFO; the SPI also checks sensors' IDs to guarantee that the received data is not corrupt and makes sure that there is no malfunction in the communication. The state machine issues the required signals to carry out calculations to extract the noise out of raw data. The output of EMF cancelation is fed into the KNN classifier, where the Euclidean distance between the input signal and all training data in the lookup table is calculated. The comparator stores only the three shortest Euclidean distances. After going through all the training data in the lookup table, a final decision is generated by using the vote majority on the labels of the three minimum distances. The final decision for each command is transmitted to the microcontroller to then be sent to the target device.

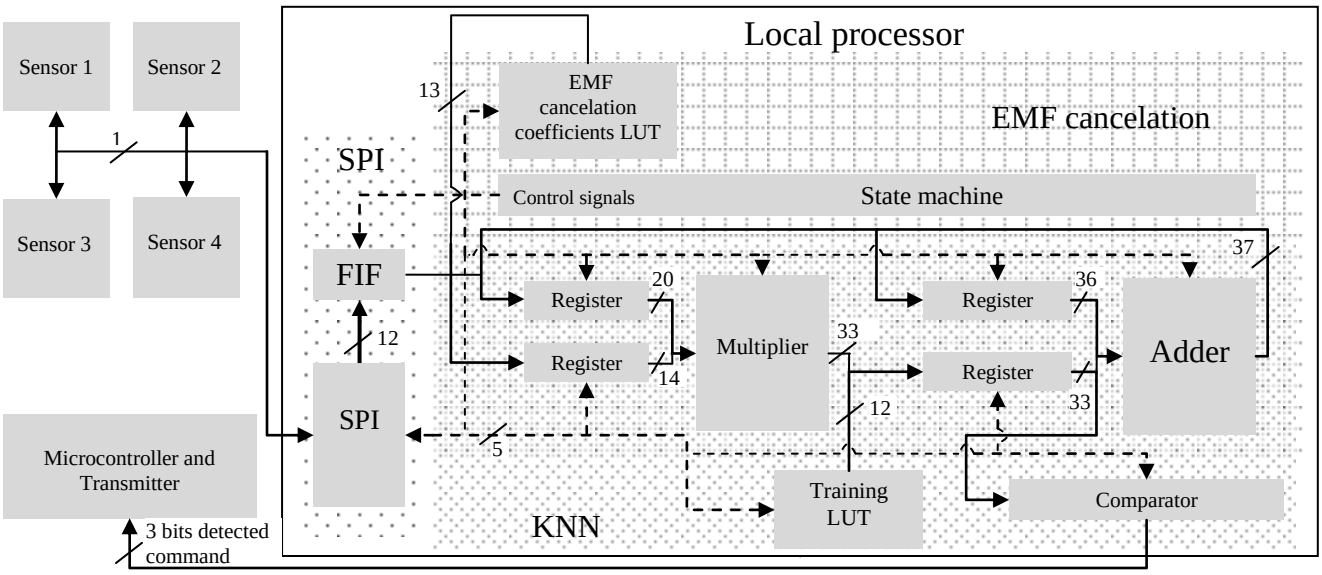


Figure 3.3. Top level block diagram of local processor and its interconnection with microcontroller and sensors. Dashed lines represent the control signals from state machine.

3.3. FPGA Implementation and Results

Table 3.1 and Figure 3.4 show the power breakdown of post place and route implementation on AGLN250. The low power consumption and low operation frequency keeps the package's temperature at the operation environment's temperature. As shown in the Figure 3.4, nearly 81% of power is consumed in the FPGA interconnection logic. In the next section, we show the impact of ASIC implementation on reducing the interconnection and overall power consumption.

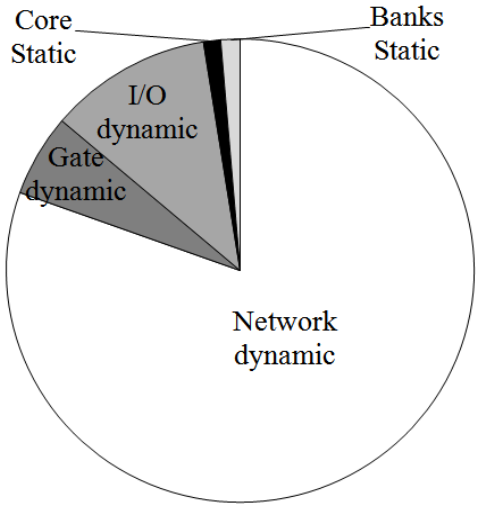


Figure 3.4. Power consumption breakdown of the design.

	Power(mw)	% of being active
Network dynamic	1.504	81.60%
Gate dynamic	0.107	5.60%
I/O dynamic	0.211	11.00%
Core static	0.022	0.80%
Banks static	0.025	0.90%
Total static	0.046	-
Total dynamic	1.821	-
Total	1.868	-

Table 3.1. Static, dynamic, and total power consumption of fully implemented design on AGLN250 with the operating frequency of 10 MHz.

Table 3.2 shows the device utilization on AGLN250 FPGA. Discussed architecture with a training dataset which has thirty samples representing each command, utilizes 88% of the device. Figure 3.5 shows the device utilization breakdown. The majority of the resources are utilized by the lookup tables, which are 20 Kbits total.

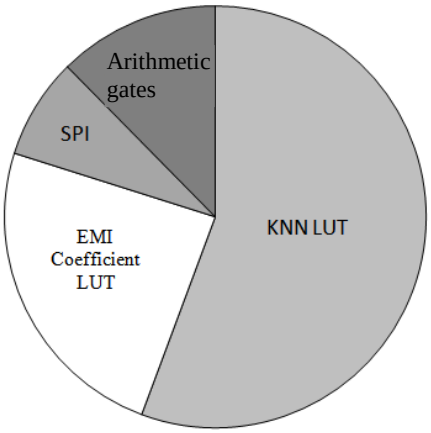


Figure 3.5. Device utilization breakdown of the design.

Area utilization			
	Used	Available	%
Core cells	5377	6144	88
CLKBUF	1	68	1.5
INBUF	2	68	3
OUTBUF	10	68	15
Total IO cells	13	68	19

Table 3.2. The FPGA implementation's device utilization of proposed TDS processor on AGLN 250.

Chapter 4: ASIC Implementation and Results

4.1. ASIC Layout

In order to lower the overall power consumption, we use a standard-cell RTL to GDSII flow, using synthesis and automatic place and route. The hardware was developed using Verilog to describe the architecture, synthesized with Synopsys Design Compiler, and placed and routed using Cadence SOC Encounter. Figure 4.1 shows the layout view of the TDS local processor in the 65 nm CMOS technology. The white dashed rectangle indicates the multiplier, adder, lookup tables, SPI, FIFO, and state machine. The processor is able to operate with 900 MHz clock frequency; however, the clock frequency has been reduced to abate the power consumption.

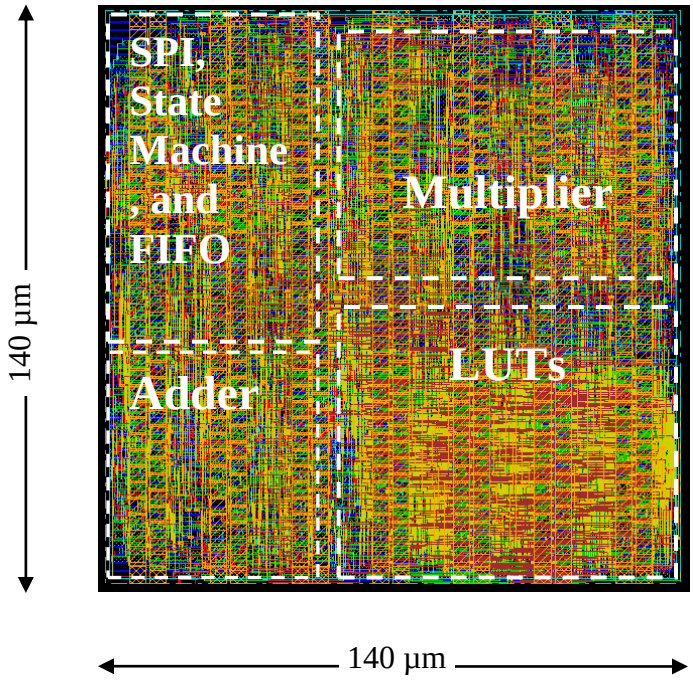


Figure 4.1. The layout view of the TDS local processor in 65 nm CMOS and statistics.

4.2. ASIC Power Consumption, Area, and Detection Latency

Implementation results	
Technology	65 nm, 1 V CMOS
Processor area (mm^2)	0.02
Frequency (MHz)	10
Latency to detect (mSec)	20
Power (mW)	0.128

Table 4.1. ASIC implementation results of the proposed TDS processor.

Table 4.1 shows the ASIC results of our proposed TDS processor. The 0.128 mW power consumption shows a dramatic improvement compared to the FPGA. Moreover, the footprint of the processor occupies only 0.02 mm^2, which opens up space for other components on the TDS board. The network power consumption leads us into ASIC implementation to reduce the power consumption. Figure 4.2 shows the percentage of network power consumption from total power in FPGA and ASIC. The network power consumption is 80% of the total in FPGA, while this number is less than 45% in ASIC.

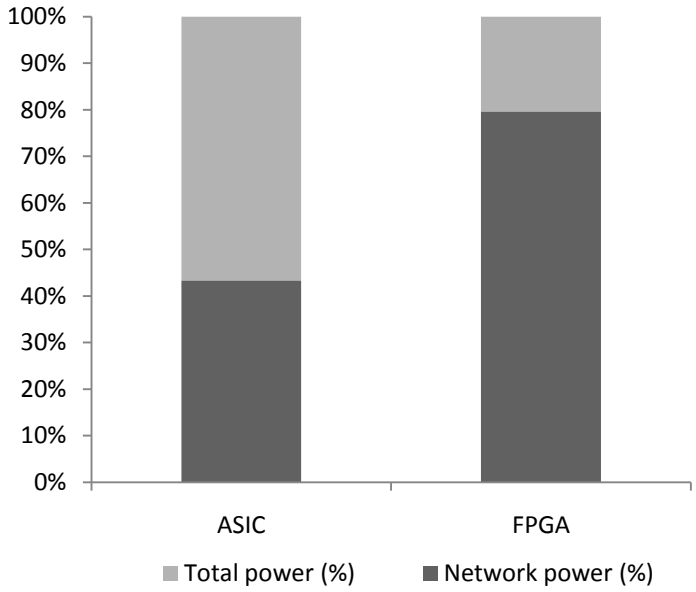


Figure 4.2. Percentage of network power consumption from total power consumption in FPGA and ASIC implementation.

Figure 4.3 shows the maximum post place and route delay in different paths, and Figure 4.4 shows the power breakdown of ASIC implementation.

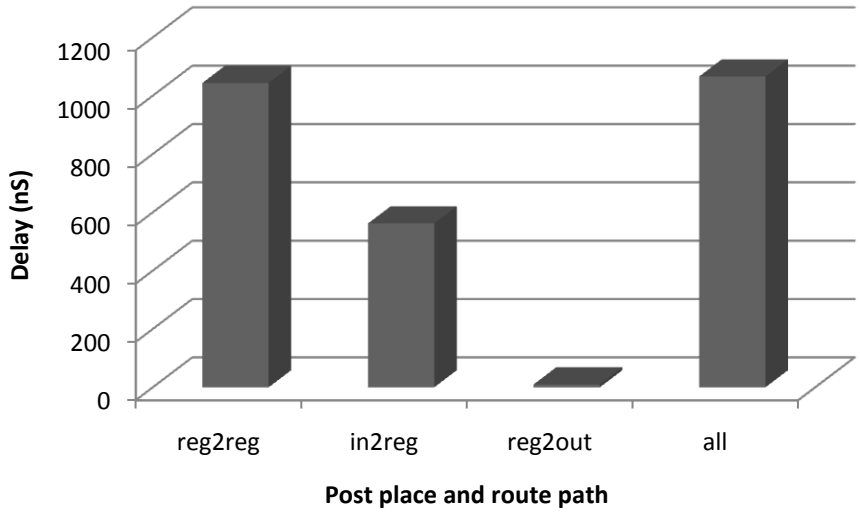


Figure 4.3. Post place and route maximum delay in different routes.

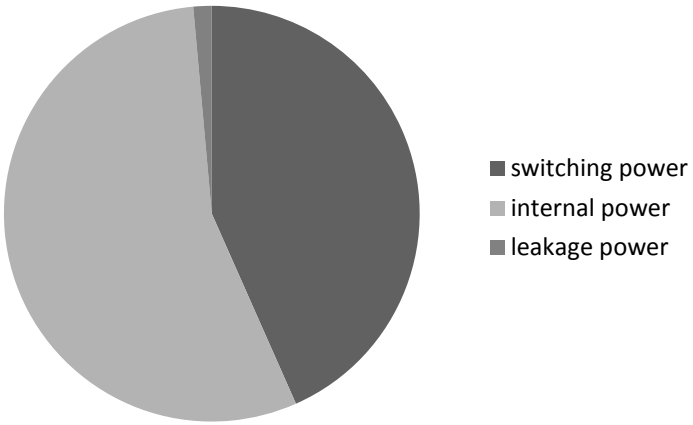


Figure 4.4. The power breakdown chart.

4.3. FPGA and ASIC Power Comparison

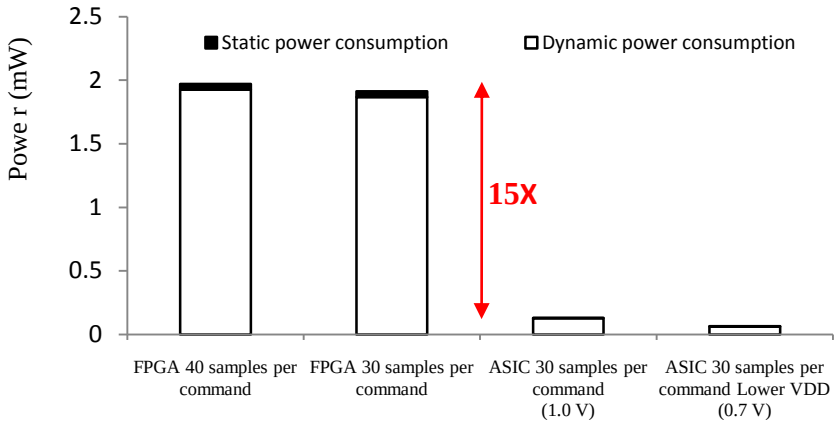


Figure 4.5. Comparison of power consumption in ASIC 65 nm and IGLOO AGLN250 FPGA implementation with the operation frequency of 10 MHz.

Figure 4.5 shows the power consumption (broken to static and dynamic) comparison of FPGA and ASIC implementation operating at 10 MHz. The ASIC implementation reduces the power consumption by a factor of 15, and voltage reduction decrements the power consumption by another factor of two.

Chapter 5: Many-Core Mapping and Results

5.1. EEHPC Many-Core

The EEHPC Many-Core is a custom many-core platform which uses low-power, modified reduced instruction set computing (RISC) processing cores. The processing cores are based on a six-stage modified RISC pipeline architecture. The core and router area footprints are 0.033 mm^2 and 0.035 mm^2, respectively, in 45nm technology, where each router is shared by four cores. The cores communicate through a simple, scalable hierarchical network that reduces the number of hops in communication. Each core operates at different clock rates through Globally Asynchronous Locally Synchronous (GALS) architecture [24], thereby eliminating global clock routing. All processors are assumed to communicate asynchronously through a 16-bit point-to-point network. The network-on-chip (NoC) is constructed as a hybrid concentrated 2D mesh and 4-ary tree topology where four cores are grouped into a cluster. Note that each core is quite small and contains dedicated 128word data and instruction memories [24]-[25]. Inter-core communication, which is crucial for enabling the processor cores to pass data between each other, is achieved by means of a simple hierarchical network (Figure 5.1).

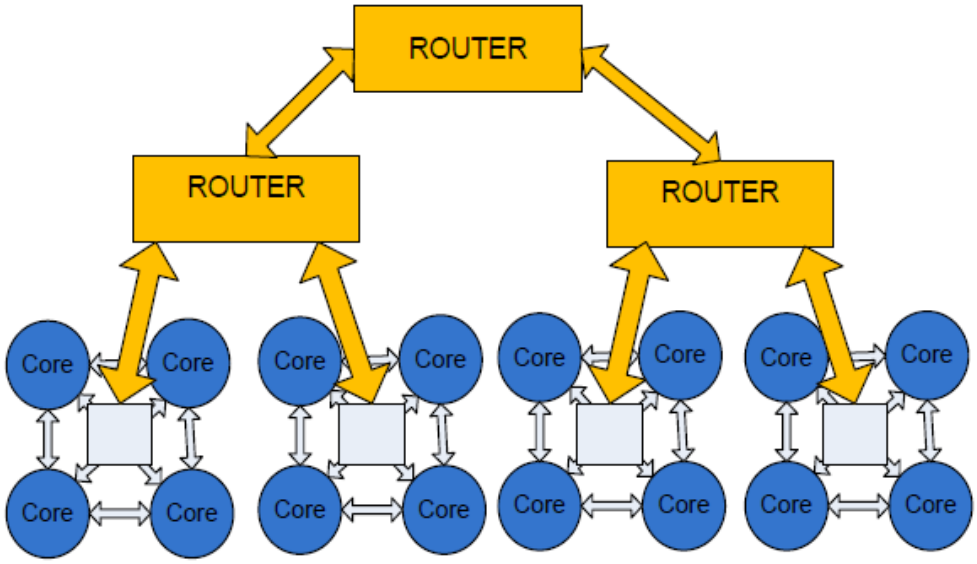


Figure 5.1. Hierarchical topology [26].

The hierarchical design allows for easy scaling of the many-core platform and reduces the number of routers that messages have to pass through in inter-core communication, when compared to a design that uses a flat topology (Figure 5.2).

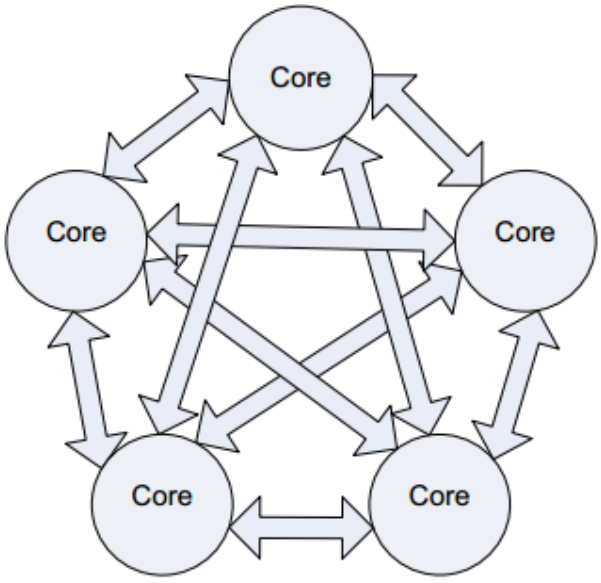


Figure 5.2. Flat topology [26].

The entire design was developed in the Verilog Hardware Descriptor Language (HDL); the many-core is scalable to any number of four. A four-core version is shown in Figure 5.3. Along with the processor platform, a compiler for translating assembly-language programs into binary (1s and 0s) that can be programmed onto the processor has already been developed [26]. The proposed design therefore combines the computing power of DSP processors, the high parallelism and re-configurability of FPGAs, and the ease of programming of general-purpose processors to deliver a novel and robust many-core platform.

For enabling inter-core communication in many-core architectures, two main approaches of flat and hierarchical topology are common. In flat topologies, each node or processor is connected to every other node. Flat topology is simple to implement, results in a smaller routing area, and has a maximum hop length of one between any two nodes. However, it becomes very cumbersome and difficult to maintain and troubleshoot as the number of cores increases. This topology is also not scalable on account of the physical wire connections between each node and every other node in the system.

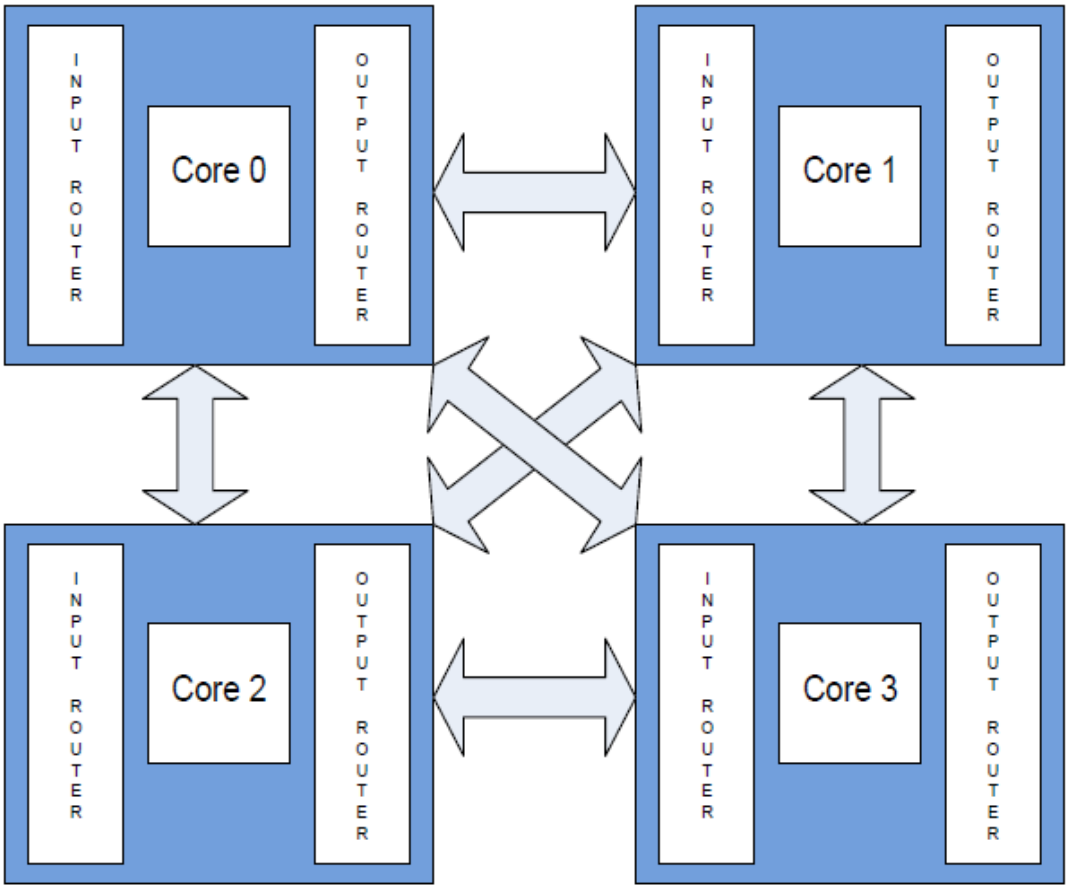


Figure 5.3. Four core cluster [26].

Figure 5.4 shows a sixteen-core processor and its interconnection. The many-core platform has been already successfully used for different applications, such as OMP compressive sensing and multi-channel seizure detection [24]-[25]-[27].

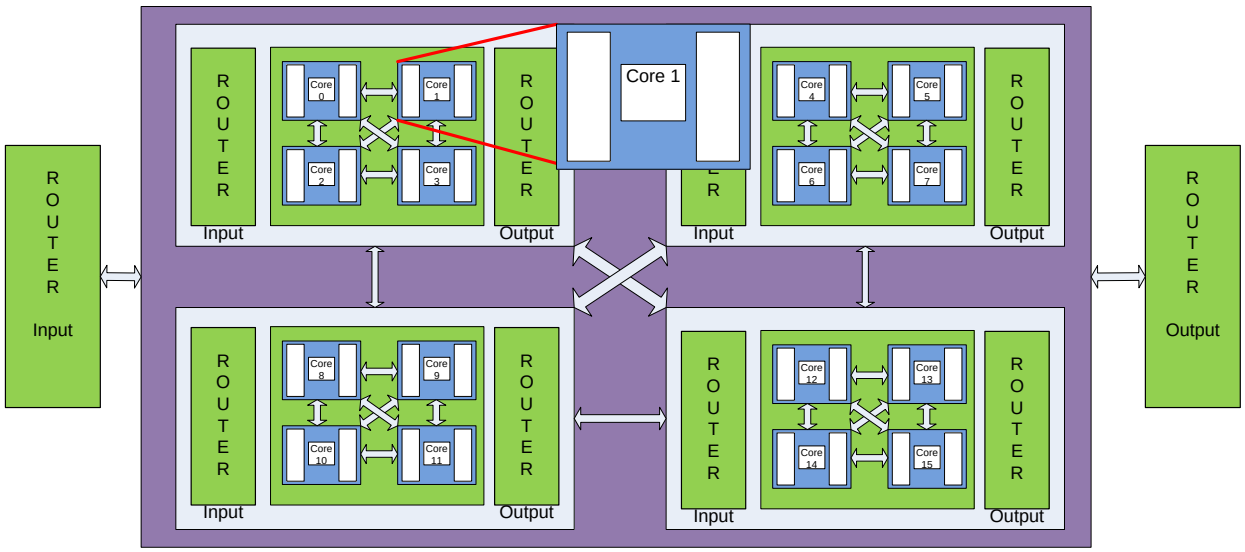


Figure 5.4. Sixteen-core cluster [26].

5.2. Mapping the KNN Classifier on Many-Core

In this thesis, we mapped our utilized KNN classifier in two different approaches. In the existing many-core, each core has a 2048-bit data memory to abate its power consumption. However, in the TDS, we require 26,880 bits to store training data. Equation 5.1 is the KNN with Euclidean distance, which is calculated in our classifier. Figure 5.5 shows the task graph of implementing the KNN on many-core. In TDS, we have six features and, as a result, we should repeat the first three steps six times; based on the number of utilized cores, each core is responsible for one section of training data. The training data contains 280 samples, and each sample has six features. Each feature can be represented with a 12-digit integer.

$$d = \sqrt{(X_{1_{test}} - X_{1_{train}})^2 + \cdots + (X_{n_{test}} - X_{n_{train}})^2} \tag{5.1}$$

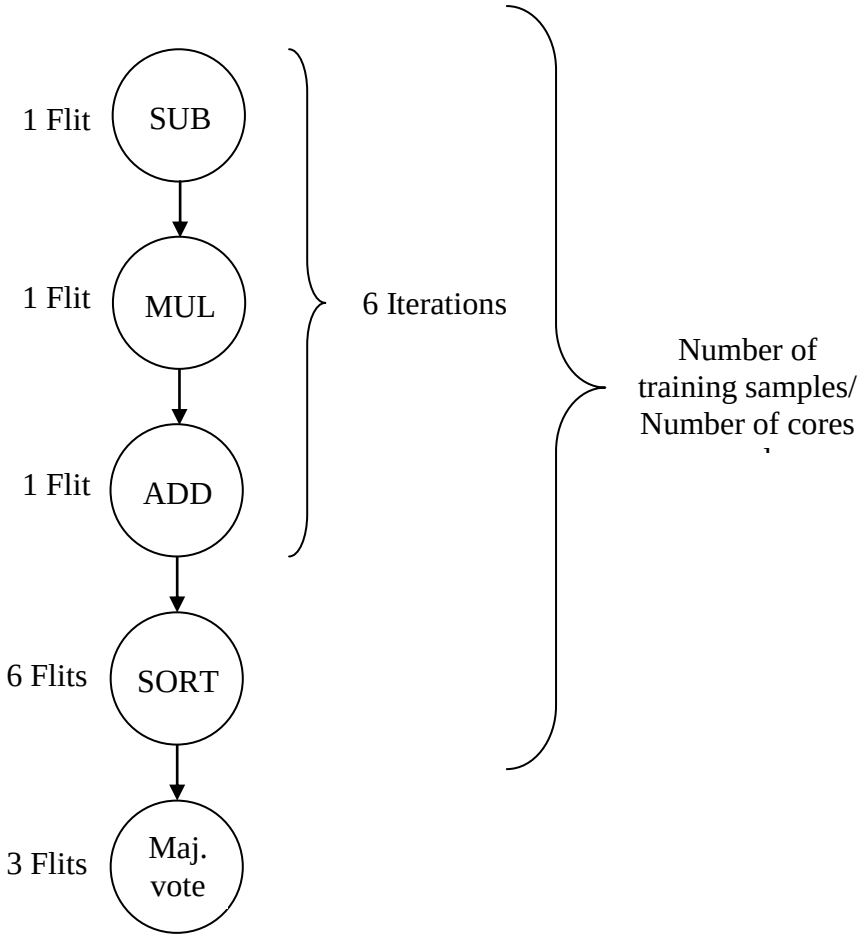


Figure 5.5. Task graph of the KNN classifier on the EEHPC many-core.

5.2.1. Mapping on Conventional Architecture

The first approach was to instantiate multiple numbers of cores to accommodate our design with enough data memory. For this purpose, we instantiate fourteen cores. Each core is in charge of calculating the Euclidean distance of sampled data and twenty samples of training data. In each core, we have 128 words of data memory. Each sample in the training data has six features. As a result, we can accommodate only 20 samples in each core (20 X 6). Afterwards, three minimum distances out of twenty calculated distances are transmitted to the core, which is then responsible for finding the three minimum distances across all 280 training samples.

Figure 5.6 shows the top-level block diagram of many-core with conventional architecture.

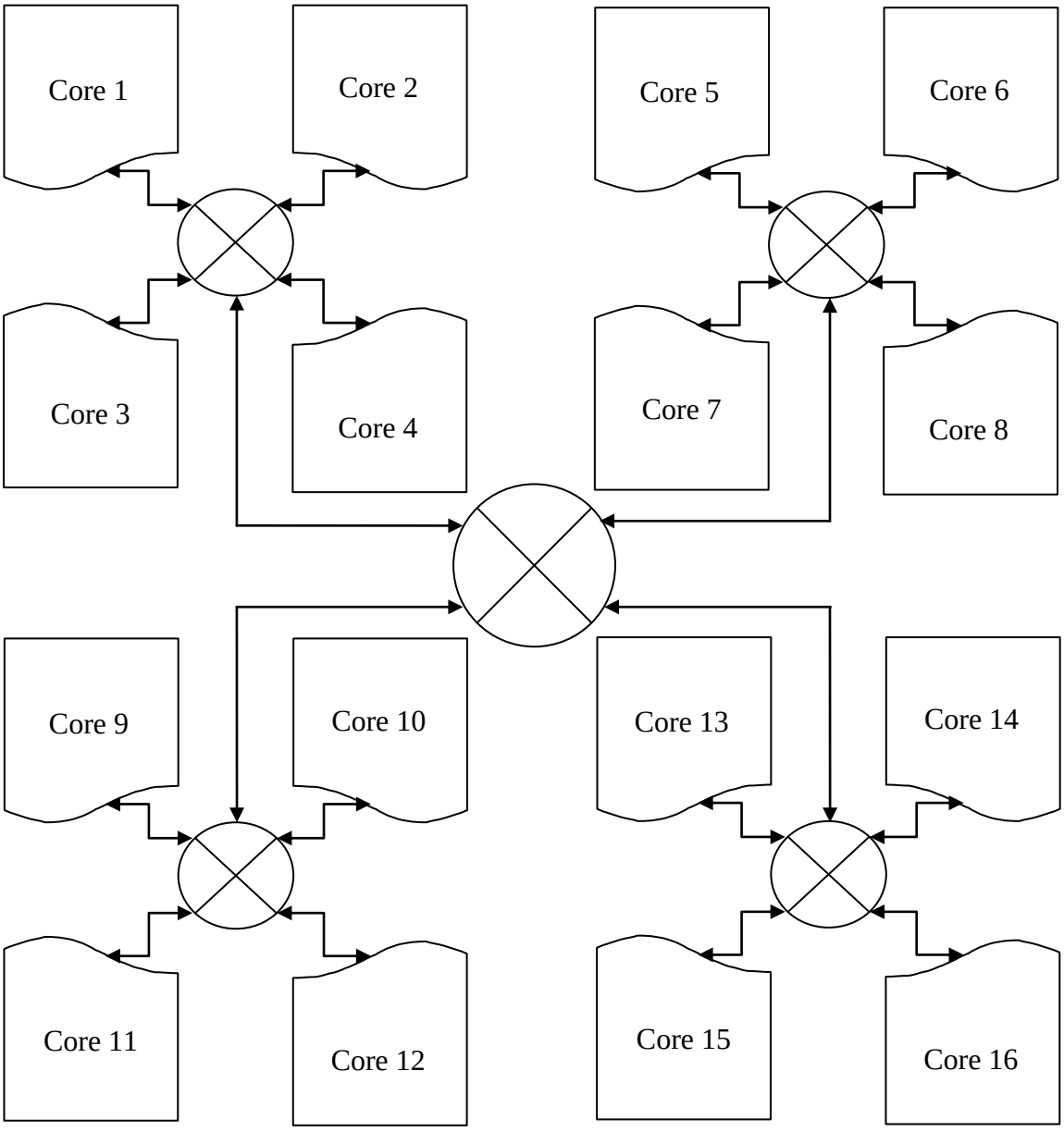


Figure 5.6. The top-level block diagram of many-core with conventional architecture. Each of cores 1 to 14 calculates three minimum distances among 20 samples, and core 15 receives all of these three minimum distances and calculates the three minimum distances among all 280 training samples.

5.2.2. Mapping on Architecture with Shared Memory

In another approach, we added a shared memory to a cluster of three cores. A 20,480-bit memory area footprint is $0.16mm^2$. Each core can access the memory through the existing routers. Figure 5.7 shows the top-level block diagram of Many-Core with shared data memory. In this architecture, each core calculates the Euclidean distance of sampled data and a specific number of training samples. The number of processed training samples in each core can be calculated by rounding the division of the total number of training samples by the number of utilized cores.

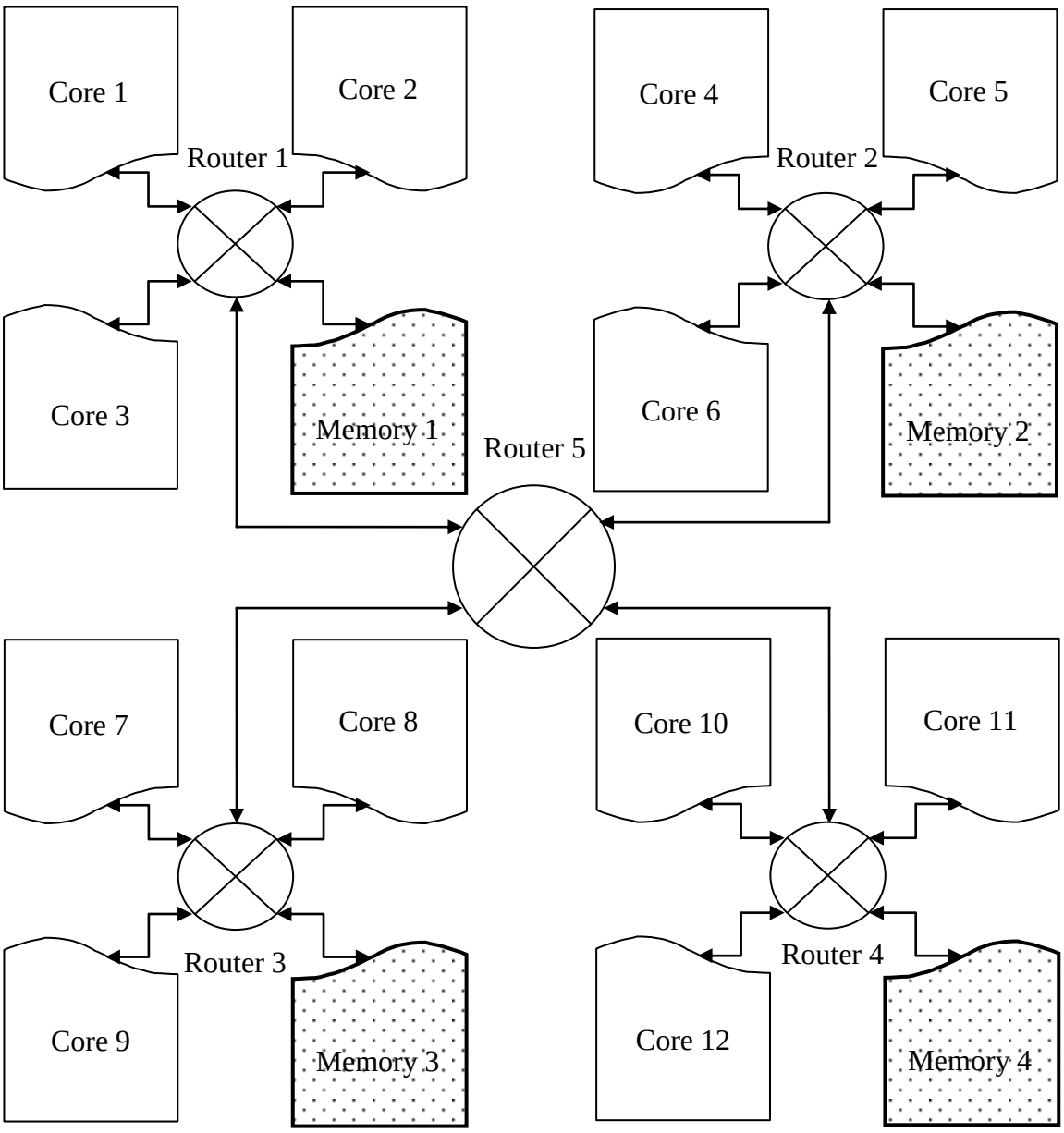


Figure 5.7. Top-level block diagram of many-core with shared memory and their interconnections. The number of processed samples in each core can be calculated by dividing the total number of training samples by the number of utilized cores.

As an example, Figure 5.8 shows the top-level block diagram of architecture using one core. In the case in which we utilize one core, it sends 280 X 6 messages to the

memory with its desired memory address and receives the same number of packets with the training data.

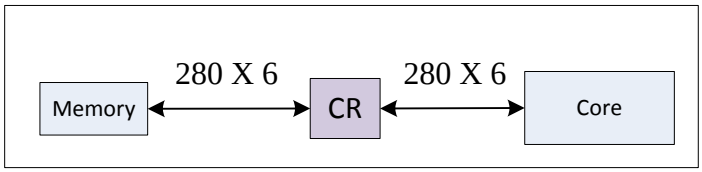


Figure 5.8. Top-level block diagram of architecture utilizing one core.

Figure 5.9 shows the top-level block diagram of architecture using three cores. When we utilize three cores, two cores send 2 X 40X 6, and the remaining core sends 3 X 40 X 6 messages to the memory with its desired memory address and receives the same number of packets with the training data.

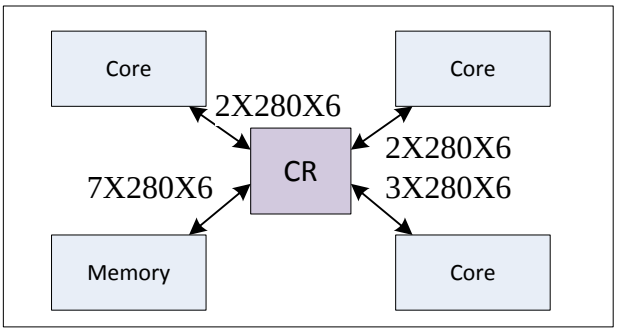


Figure 5.9. Top-level block diagram of architecture utilizing three cores.

5.3. Many-Core Mapping Results

Figure 5.10 shows the affiliation of the number of utilized cores and power consumption and latency in the processor with a shared memory. By increasing number of utilized cores, the power consumption increases. However, in the applications in which we have a deadline for the task, we should utilize as many cores as are required to meet the deadline.

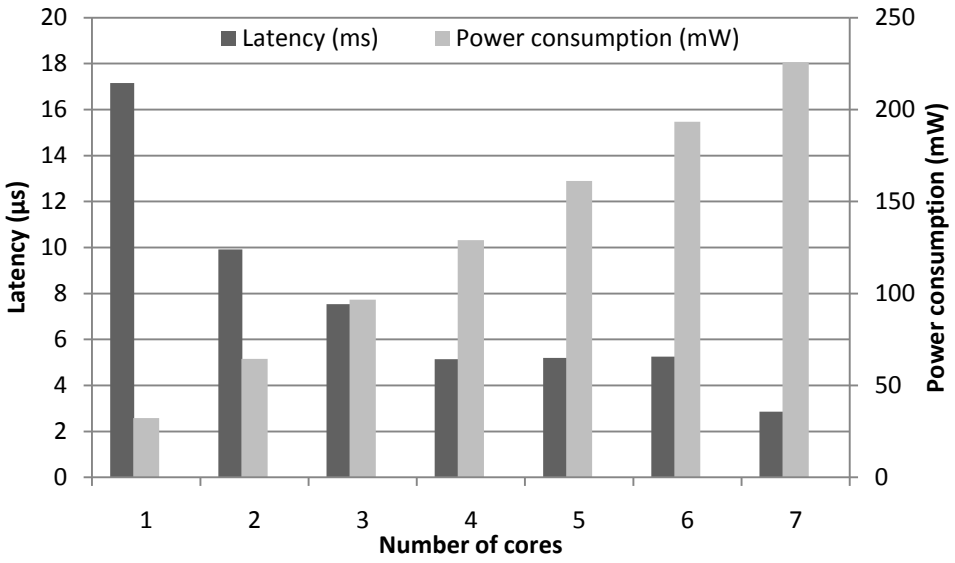


Figure 5.10. The impact of the number of utilized cores on power consumption and latency.

Table 5.1 shows the number of instructions and delays that are required in different mapping with different numbers of utilized cores. The number of communication instructions increases by incrementing the number of utilized cores. The architecture without applied shared memory shows a better performance in TDS application. The reason for the longer runtime in the architecture with a shared memory is the tremendous number of messages passing between cores and memory. In the applications with a considerable size of training data, the architecture without shared memory will not be a practical approach.

Architecture	Number of cores	ALU Inst.	Branch Inst.	Comm Inst.	No op Inst.	Total Inst.	Number of Required Cycles
With shared memory	1	9818	1963	3372	281	15434	15434
	4	9934	2026	3432	309	15701	4629
	7	10027	2077	3486	330	15920	2576
Without shared memory	14	5304	976	252	352	6884	985

Table 5.1. Different mapping of KNN on EEHPC Many-Core and number of instructions and runtime.

Architecture	Number of cores	Energy (µJ)	Runtime (µSec)
With shared memory	1	1.2	17.14
	4	1.63	5.14
	7	1.02	2.86
Without shared memory	14	0.73	1.09

Table 5.2. Different mapping of KNN on EEHPC Many-Core and its impact on energy dissipation in cores.

Chapter 6: Conclusion

6.1. Results Summary

The vast amount of power which is consumed in transmitting raw sensed data to a computer for processing leads us to propose a local processor. The proposed local processor performs all signal processing algorithms at the sensors and only transmits the final classification result.

6.1.1. Conventional TDS and TDS with Local Processor

Conventional TDS transmits all the sensor data to an external computer to detect a command, while a TDS with a local processor only transmits the detected command, thus scaling down the data transmission by a factor of 30. The local processor design is small enough to fit in our target ultra-low power FPGA (AGLN250) and a small footprint ASIC implementation. Furthermore, applied optimizations do not have any impact on the detection accuracy.

6.1.2. FPGA, ASIC, and Many-Core Results Comparison

FPGA implementation results show a considerable reduction of data communication, reduced from 12 Kbits/sec to 400 bits/sec. The implementation of the design on AGLN250 FPGA consumes 1.86 mW. The amount of power that is

consumed on network interconnect logic of FPGA directs us to ASIC implementation of the design. ASIC implementation in 65 nm CMOS technology reduces the power consumption by a factor of 15 when operating at 1 V, and by a factor of 30 when operating at 0.7 V, compared to the FPGA prototype. The ASIC footprint occupies 0.02 mm^2, and it satisfies the required command detection latency of 20 ms running at 10 MHz clock frequency. This optimization has a profound impact on the future of TDS, and it eliminates the need for a bulky battery and increases the operation time after every battery recharge. This improvement significantly reduces the TDS size and weight, and it improves the convenience and detection accuracy for the patient.

In another experiment, we implemented the proposed architecture on the EEHPC Many-Core processor. Table 6.1 shows the comparison of different implementation of our KNN classifier.

Implementation	Power (mW)	energy (µJ)	energy (meeting delay)(µJ)
FPGA (IGLOO nano)	1.775	2.218	2.218
ASIC	0.122	0.152	0.152
ASIC (Lower VDD)	0.060	0.075	0.075
Many-core (1-core)	32.24	1.2	1.2
Many-core (4-core)	128.94	1.02	1.02
Many-core (7-core)	225.65	0.73	0.73
Many-core(14-core)	483.56	0.56	0.56
FPGA (Virtex5)	1127.02	5.87	5.87

Table 6.1. Comparison of different implementations of utilized KNN classifiers in the proposed processor.

The implemented KNN classifier on AGLN250 consumes 2.22 µJ and shows a considerable improvement compared to the implementation results of Virtex5. The ASIC implementation shows the best results for power consumption. However, the Many-Core platform has reasonable power consumption while also having more

design flexibility and a shorter design and implementation process compared to the FPGA and ASIC. Figure 6.1 shows the comparison of utilized KNN classifiers in our proposed local processor on different platforms.

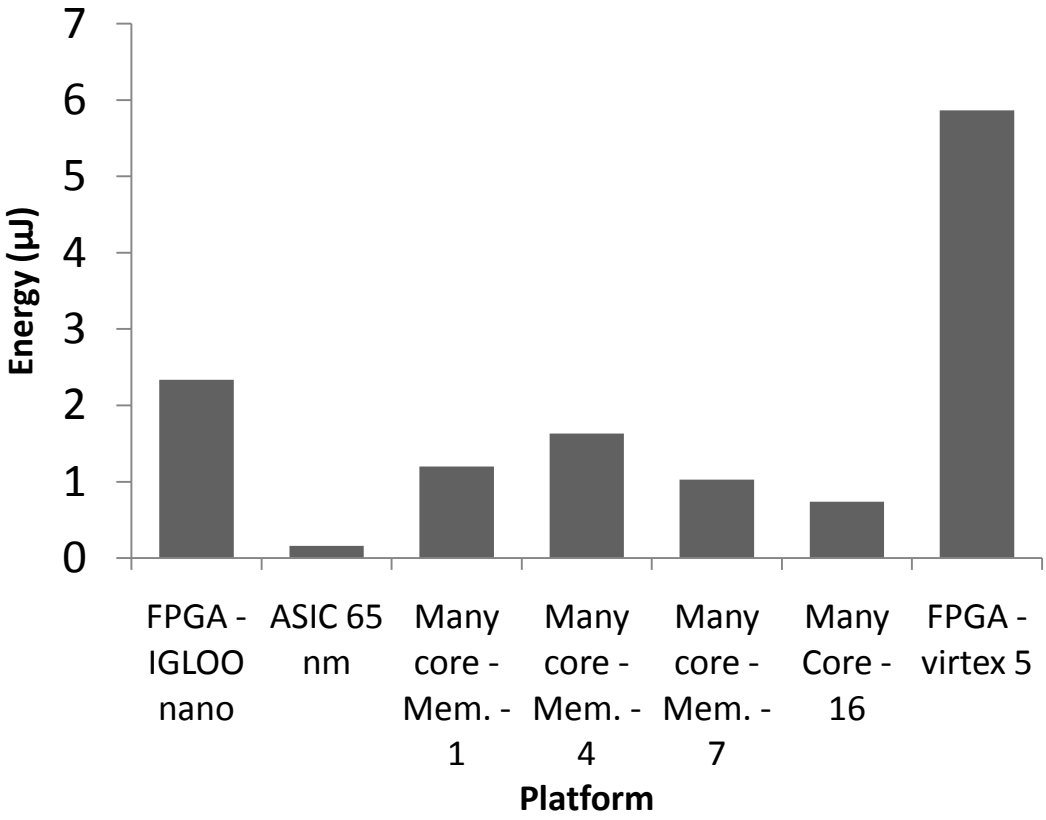


Figure 6.1. The comparison of energy in different implementations of KNN classifiers.

6.2. Future Work

There are several ways this work could be extended. To improve the liability of our local processor, we should add more features to our process. The new features could be a new physiological signal or a gyroscope. Adding a new sensor to the system that gives us a better understanding of the eTDS location can help to address

the domain shift issue in the eTDS, as discussed in Chapter 2. Collecting further patient data, especially by using new systems, would make our analysis more comprehensive. As a result, developing the new proposed system and collecting more data will help to increase the sensitivity and accuracy of the devices. By increasing sensitivity of devices, we can reduce the required size of the magnet in iTDS, and, instead of piercing, we can simply put the magnet in the tongue using a syringe. The modality of the TDS should make it resistant to deliberate interferences. Adding encryption to the data transmission and adding a new mechanism for protecting the system is another area for further improvement.

Bibliography

[1] Human++: Electronics for Health Care and Life Sciences, Nov.8, 2011 [Online]. http://www2.imec.be/be_en/research/human-biomedicalelectronics.html

[2] Center for Future Health Univ. of Rochester, Nov. 8, 2011 [Online]. http://www.urmc.rochester.edu/future-health/index.cfm

[3] The Assistive Cognition Project, Univ. of Washington, Nov. 8, 2011[Online]. http://www.cs.rochester.edu/u/kautz/ac

[4] The Aware Home, Georgia Institute of Technology, Nov. 8, 2011[Online]. http://www.cc.gatech.edu/fce/ahri/projects/index.html

[5] CodeBlue project: Wireless Sensor Networks for Medical Care, Harvard Univ., Nov. 8, 2011 [Online]. Available: http://fiji.eecs.harvard.edu/CodeBlue

[6] V. Pop *et al.*, "Human++: Wireless autonomous sensor technology for body area networks," in *Proc. Asia and South Pacific Design AutomationConf.*, 2011.

[7] TongueTouch Keypad™ (TTK), [Online]. Available: http://www.newabilities.com/

[8] Jouse2, Compusult Limited, [Online]. Available: http://www.jouse.com/

[9] USB Integra Mouse, Tash Inc., [Online]. Available: http://www.tashinc.com/catalog/ca_usb_integra_mouse.html

[10] X. Huo and M. Ghovanloo, "Tongue Drive: A wireless tongue-operated means for people with severe disabilities to communicate their intentions," *IEEE Communications Magazine*, vol. 50, no. 10, pp. 128–135, Oct. 2012.

[11] K. Bunton, and G. Weismer, "Evaluation of a reiterant force-impulse task in the tongue," *J Speech Hear Res*, vol. 37, pp 1020-1031, Oct. 1994.

[12] P. Svensson, A. Romaniello, K. Wang, L. Arendt-Nielsen and B.J. Sessle, "One_hour tongue-task training associated plasticity corticomotor control of the human tongue musculature", *Experimental Brain Research,* vol. 173, pp 165-173, Aug. 2006.

[13] X. Huo, and M. Ghovanloo, "Using unconstrained tongue motion as an alternative control mechanism for wheeled mobility" *IEEE Trans. Biomed. Eng.*, vol. 56, pp. 1719–1726, June 2009.

[14] X. Huo and M. Ghovanloo, "Evaluation of a wireless wearable tongue computer interface by individuals with high level spinal cord injuries," *Journal of Neural Engineering*, vol. 7, #026008, Mar. 2010.

[15] H. Park, M. Kiani, H.M. Lee, J. Kim, B. Gosselin, and M. Ghovanloo, "A wireless magnetoresistive sensing system for an intraoral tongue-computer interface," *IEEE Trans. on Biomed. Circuits and Systems*, vol. 6, no. 6, pp. 571–585, Dec. 2012.

[16] http://www.ece.gatech.edu/research/labs/gt-bionics/

[17] X. Huo, J.Wang and M. Ghovanloo, "A Wireless Tongue-Computer Interface Using Stereo Differential Magnetic Field Measurement," *IEEE EMBS*, pp. 5723–5726, Aug. 2007.

[18] Hangue Park ; Jeonghee Kim ; Ghovanloo, M. “Intraoral tongue drive system demonstration” Biomedical Circuits and Systems Conference (BioCAS), 2012 IEEE 10.1109/BioCAS.2012.6418503

[19] Yousefi, B. ; Xueliang Huo ; Ghovanloo, M., “Using Fitts's law for evaluating Tongue Drive System as a pointing device for computer access” Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE 10.1109/IEMBS.2010.5627130

[20] A. Ayala-Acevedo and M. Ghovanloo,” Quantitative Assessment of Magnetic Sensor Signal Processing Algorithms in a Wireless Tongue-Operated Assistive Technology” Engineering in Medicine and Biology Society (EMBC), 2012 Annual International Conference of the IEEE pp. 3692-3695

[21] Park, H ; Kiani, M. ; Hyung-Min Lee ; Jeonghee Kim ; Block, J. ; Gosselin, B. ; Ghovanloo, M. “A Wireless Magnetoresistive Sensing System for an Intraoral Tongue-Computer Interface“ Biomedical Circuits and Systems, IEEE Transactions, 10.1109/TBCAS.2012.2227962

[22] S.Viseh, A.Acevedo, M.Ghovanloo and T.Mohsenin, “Towards a Low Power FPGA Implementation for A Stand-Alone Intraoral Tongue Drive System” 39th Annual GOMACTech Conference, April 2014.

[23] S.Viseh, M. Ghovanloo and T. Mohsenin “ Towards an Ultra Low Power On-board Processor for Tongue Drive System“ TCAS-II,2014.(under review)

[24] Jordan Bisasky, James Darin Chandler, Jr. and Tinoosh Mohsenin "A Many-Core Platform Implemented for Multi-Channel Seizure Detection" *IEEE International Symposium on Circuits & Systems (ISCAS ' 12)*, May 2012.

[25] Jordan Bisasky, Houman Homayoun, Farhang Yazdani and Tinoosh Mohsenin, "A 64-core platform for biomedical signal processing" *The International Symposium on Quality Electronic Design (ISQED)*,March 2013.

[26] Julian Feild and Tinoosh Mohsenin. “A High-Performance, Reconfigurable Network for a 64-core Programmable Platform”, http://eehpc.csee.umbc.edu/publications-docs/URA_julian.pdf.

[27] Amey Kulkarni and Tinoosh Mohsenin "Parallel and Reconfigurable architectures for OMP compressive sensing reconstruction algorithm", *In proceedings of SPIE Sensing Technology and Applications, Baltimore, Maryland*, May 2014 .