A Study of the Power Trace Sampling Frequency Requirements in Non-Intrusive Program Tracing Through Power Consumption Monitoring

by

Ajay Kumar Singh

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2018

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

All the work presented in this thesis were completed under the supervision of Professor Sebastian Fischmeister in the ECE department at the University of Waterloo, who contributed with his ideas, reviews, and suggestions.

Hardware setup used in Chapter 3 for data capture was contributed by Dr. Carlos Moreno, who also contributed his ideas, reviews, and suggestions. Dr. Carlos Moreno is a Senior Research Associate under the supervision of Associate Professor Sebastian Fischmeister, in the Real-time Embedded Software Group at the University of Waterloo.

Abstract

In embedded systems, anomalies can be detected by monitoring the power consumption of the device. Recent literature has shown the use of this technique for the purpose of safety as well as security. However, on-the-fly power monitoring and classification becomes more complex as we increase the sampling rate of the power signal. This thesis presents a study where we experimentally evaluate the required sampling frequency, with respect to classification accuracy, as a function of clock frequency. We analyze the systems classification accuracy by running it at different clock speeds and varying the sampling frequency for power trace capture. We also show the effect of the sampling frequency on parameters such as training time and classification time.

Acknowledgements

I would like to thank all the people who made this thesis possible. First and foremost I would like to express my gratitude towards my supervisor Dr. Sebastian Fischmeister for all the guidance and opportunities he has given during this period. I learned a lot being a part of Real-time Embedded Software Group. Thank you for believing in me.

I would like to thank Dr. Carlos Moreno for his guidance and support throughout my masters. Thank you for answering all my questions, including the silly ones.

I would also like to thank Karim Elrayes, Sean Kauffman and Jack Morgan for their assistance and suggestions during this period.

Dedication

This thesis is dedicated to my parents for their endless love, support, and encouragement.

Table of Contents

List of Tables ix

List of Figures x

List of Symbols xii

1 Introduction 1

1.1 Related Work . . . 2

1.2 Problem Statement and Assumptions . . . 3

1.3 Our Contributions . . . 4

1.4 Organization of the Thesis . . . 5

2 Background 6

2.1 Power Consumption . . . 6

2.1.1 Power Consumption of a Micro-Controller . . . 7

2.1.2 Measuring the Power Consumption of a Micro-Controller . . . 7

2.1.3 Power Traces . . . 8

2.2 Basic Block (BB) . . . 9

2.2.1 Control Flow Graph (CFG) . . . 9

2.3 Time Series Classification . . . 11

3 System Model **13**

3.1 Powertraces Capture Setup . . . 14

3.1.1 Micro-controller Board . . . 15

3.1.2 Digitizer . . . 15

3.1.3 Port Bit Flip Markers . . . 16

3.2 Source Code Instrumentation . . . 18

3.3 Merging the Smallest Basic Blocks . . . 19

3.4 Digitizer Output Data Preprocessing . . . 19

3.5 Generating Training Data . . . 21

3.5.1 Variations in Samples in each Class . . . 21

3.5.2 Total Classes . . . 22

3.5.3 Training Dataset . . . 25

3.6 Testing Setup . . . 25

3.7 The Anomaly Detector . . . 27

4 Sampling Frequency **28**

4.1 Analytical Approach . . . 29

4.1.1 Nyquist Criterion . . . 29

4.1.2 Sampling Frequency for the Purpose of Anomaly Detection . . . 30

4.1.3 Our Approach to the Sampling Frequency . . . 31

4.2 Empirical Approach . . . 32

5 Conclusion and Future Work **41**

5.1 Conclusion . . . 41

5.2 Future Work . . . 42

References **43**

List of Tables

4.1 Accuracy and Standard deviation at different F_s, when $F_c = 250KHz$ . . . 34

4.2 Accuracy and Standard deviation at different F_s, when $F_c = 500KHz$ . . . 35

4.3 Accuracy and Standard deviation at different F_s, when $F_c = 1MHz$ 36

4.4 Accuracy and Standard deviation at different F_s, when $F_c = 4MHz$ 37

4.5 Accuracy and Standard deviation at different F_s, when $F_c = 8MHz$ 38

4.6 Accuracy and Standard deviation at different F_s, when $F_c = 12MHz$ . . . 39

List of Figures

2.1 Setup to measure power consumption of a MCU 8
2.2 Example of power traces for four blocks of code 8
2.3 Sample CFG with basic blocks 10
2.4 Categories of numeric time series distances 11
2.5 Two time series signals warped in time scale 12

3.1 Segments Powertrace Samples 13
3.2 Powertraces Capture Setup 15
3.3 Toggling Port pin added at the end of each basic block 17
3.4 Digitizer processed 2-channel data 18
3.5 Example of merging the CFG nodes 19
3.6 Removing noise from Channel-B signal using Schmitt trigger 20
3.7 Sample CFG with basic blocks 22
3.8 Average sample of each class 23
3.8 Average sample of each class 24
3.9 On-the-fly Anomaly detection 26

4.1 Signal sampling representation. 28
4.2 Sampling analog signal at different frequencies 30
4.3 Classification accuracy at different sampling frequencies, when $F_c = 250KHz$ 34
4.4 Classification accuracy at different sampling frequencies, when $F_c = 500KHz$ 35

4.5 Classification accuracy at different sampling frequencies, when $F_c = 1MHz$ 36
4.6 Classification accuracy at different sampling frequencies, when $F_c = 4MHz$ 37
4.7 Classification accuracy at different sampling frequencies, when $F_c = 8MHz$ 38
4.8 Classification accuracy at different sampling frequencies, when $F_c = 12MHz$ 39
4.9 Classification Accuracy Vs Sampling Frequency at different Clock Frequencies 40

List of Symbols

ADC Analog to Digital Conveter 15, 25, 40, 41

BB Basic Block 2, 7, 9, 16, 19, 31, 32, 40, 41

CFG Control Flow Graph 2, 4, 7, 9, 16, 19

CISC Complex Instruction Set Computer 42

CPU Central Processing Unit 33, 42

DTW Dynamic time warping 3, 11, 12, 42

ECG electrocardiogram (ECG) is a diagnostic tool to assess the electrical impulses from human muscles. 11

EEG Electroencephalography (EEG) is an electrophysiological monitoring method to record electrical activity of the brain. 11

IoT Internet of Things 1

MCU Micro-Controller Unit 1, 4, 7, 25, 26, 40–42

NCC Nearest Centroid classifier 21, 22

NNC Nearest Neighbor classifier 22

RISC Reduced Instruction Set Computer 42

Chapter 1

Introduction

Embedded system has become the necessity of the present world. It is at the core of all the modern high-tech gadgets ranging from the smart TV, smartphone, laptop, multimedia system, autonomous car, IoT devices, etc. And the future will witness more state-of-the-art development in the embedded systems. But with the increase in the usage of embedded systems, there is a need for its security. Especially with all the devices now connecting with the internet, the safety and the security of the device are more under threat [41]. Think of the consequences when an autonomous car connected to the internet has been hacked. The hacker could use the device at his will fulfilling his motive. He could also use this for terrorist activities. This is just one example, think of what could happen if the devices that we use on a day-to-day basis get control from unauthentic people. So, the safety and security of the embedded systems should be the utmost concern for the developers.

Apart from safety and security, real-time monitoring of the system is also an essential requirement. The embedded systems are deployed to work in a self-sufficient manner. So, these devices act like a black box taking inputs and providing some outputs according to the task. This opacity of the embedded system as a black box makes it difficult for the real-time monitoring and system tracing. To assist run-time debugging, one could write debugging information in the code during the development phase, but this practice is not practical because of the memory constraints of the micro-controllers. So, we need a technique which provides us with both, the real-time monitoring and the system security.

One of the recent and promising ways to provide both real-time monitoring and the system security is Power Trace Monitoring of the MCU [34][28][39][36][31][25]. This technique exploits the fact that there is a direct connection between the power consumption of the MCU and the instruction commands along with the data, it is executing. This connec-

tion allows researchers to correlate the power-traces from the system to the program tasks. Therefore, by analyzing the power consumption, researchers can do real-time monitoring of the system. Security researchers have developed side-channel analyses that use power consumption to determine the secret keys used in cryptographic routines. Existing power analysis techniques have found some success, but still, some questions are unsolved in this research domain.

In this thesis, we present a study where we evaluate the required sampling frequency, with respect to classification accuracy, as a function of system architecture and clock frequency. We analyze the systems classification accuracy by running the system at different clock speeds and varying the sampling frequency for power trace capture. We also show the effect of the sampling frequency on parameters such as classification time.

1.1 Related Work

Prior works have introduced the concept of power-trace analysis or more precisely, power-based program tracing. They have shown it to be feasible and reliable for security and debugging. Power-based program tracing is non-intrusive monitoring through reconstruction of program execution traces from power consumption measurements. Moreno et al. [34][28] showed a technique for non-intrusive program tracing and debugging through power-trace analysis where they use the power consumption characteristics of a system to identify which basic block BB of code is being executed. But they use system sound card to capture the power-traces. This limited the sampling frequency of the power-traces to around 150KHz and blocked the DC component of the signal. In [36], they improve performance by taking little help from the compiler that maximizes distinguish-ability of power-traces for different blocks of code. Liu et al. [32] took help from the CFG and source code information and got a very high classification accuracy. The main reason for high accuracy was the use of high-end oscilloscope with a sampling frequency of 1.25 Giga samples per seconds for sampling the power-traces. It is shown that this method can be used in anomaly detection; however, having the oscilloscope to capture the data and classifying the traces offline limits the usefulness of this work in real-life systems. Some researchers used very low sampling frequency and some used very high sampling frequency, which put a question on selection of sampling frequency for a good classification accuracy.

Eisenbarth et al. used hidden Markov models to trace assembly-level instructions using power consumption. At this fine granularity, the reported classification accuracy was too low for a practical application [48]. Msgna et al. [33] presented the idea of side channel control flow security in the embedded system where they collect several power-traces and

calculate the mean of traces to minimize the inherent and ambient noise introduced by the measurement setup. Clark et al. [10] tried to model permissible behavior by using a behavior monitoring system of a medical device. They tried to detect the deviation in the behavior which worked, however, was limited to the simple and highly repetitive operation of the device. The usefulness of non-intrusive systems has been highlighted in [3] where the authors presented a hardware-assisted paradigm to extract properties of an embedded program through static program analysis and used them to secure the system.

Many researchers have worked in the field of time-series power data classification and presented their work. Like Deng et al. [23], used Autoregressive-Moving-Average models (ARMA models) and used Euclidean distance as a distance measure to find the closest time-series and then performed a nearest neighbor classification for better results. An early approach to time-series classification using qualitative and quantitative methods are presented by Bakshi et al. in [5]. An overview of time-series knowledge extraction, data classification, data clustering and relationship finding is presented in [30] . In this paper, the author provides the analysis in different types of similarity measures in time-series data. Authors also focus on the critical issue of measuring the similarity between two sequences where the ability to deal with noise in the data, amplitude differences and gap in time axis are the primary problems. In [1], Nanopoulos et al. use neural networks on statistical features to perform time series classification.

Dynamic time warping DTW distance for classification has gained popularity in recent years in time series classification because of its immunity against signal warping in time-scale. DTW algorithm was first introduced by [6], and then gained popularity and used in many researches [16, 24, 49, 21, 43]. [12] has shown the efficient online sequence learning using unsupervised convolutional the neural network, but lately, DTW has gained in popularity in time series classification with the introduction of 1-Nearest Neighbor and faster computation with tighter lower bound [51, 40, 17]. [13, 52, 42, 4] have shown that DTW combined with 1 Nearest Neighbor (NN) is exceptionally difficult to beat in time series classification.

1.2 Problem Statement and Assumptions

The thesis addresses the following problem statement: Given a processor with a known clock speed, what should be the sampling frequency for power traces to perform anomaly detection based on power tracing to work at a given performance level?

In context of this work, we have made the following assumptions:

- Input Identification: All possible combination of execution paths were generated using the random input initialization and running the target program in multiple loops. This ensures generation of all valid paths in the CFG using random input initialization. The power-traces of same basic block may exhibits subtle variations due to the context in which it is executed. This is further discussed in Section 3.5.

- Non-preemptive system: The current system works with the non-preemptive model, i.e., no task can preempt any other task. For the particular research, we have used single process system, but this research can work with multi-processing non-preemptive systems as well.

- Control Flow Integrity: We assume that control flow integrity is maintained all the times. That means that we do not consider cases such as random execution due to memory leak or stack corruption, undefined behaviour due to segmentation errors, system crashes, etc.

1.3 Our Contributions

In embedded systems, one way to detect anomalies in the system is by observing the power consumption of the system. However if we increase the sampling rate for power traces, on-the-fly power monitoring and classification becomes difficult. The main contribution of the thesis is to address this question, of selecting the correct sampling frequency for capturing the power traces from the system.

- We draw a relationship between the clock frequency (F_c) of the system and the sampling frequency (F_s) of the power traces for that system, for the purpose of anomaly detection at a given performance level.

- We show the effect of the sampling frequency on parameters such as classification time (training and testing time).

- We show an intuitive analytical relationship between F_c, F_s and smallest segment size.

- We show the possibility of selecting the low cost MCUs for developing the anomaly detection system.

1.4 Organization of the Thesis

The rest of the thesis advances as follows: We discuss the background required for all the chapters in the Chapter 2. In Chapter 3, we present the details on the system model and all hardware-software setup required for data collection and processing. Chapter 4 contains information on calculating sampling frequency value for power-traces sampling, using analytical and empirical approach. Chapter 5 includes discussion and future work followed by some concluding remarks.

Chapter 2

Background

2.1 Power Consumption

In electrical engineering, the power consumption (P) of a system is defined as the total electrical energy (E) consumed per unit time.

$$P = \frac{E}{T} \tag{2.1}$$

It is measured in Watts (W) or kilowatts (kW). There are other variations of the formula for calculating the power consumption of the system. These are given below.

$$P = V * I \tag{2.2}$$

$$P = R * I^2 \tag{2.3}$$

$$P = \frac{V^2}{R} \tag{2.4}$$

where,
V - is the supply voltage of the system.
I - is the current consumed by the system.
R - is the total resistance of the system.

2.1.1 Power Consumption of a Micro-Controller

The total power consumption P_T by a micro-controller can be broken down into two powers: static power consumption P_S and dynamic power consumption P_D [15].

$$P_T = P_S + P_D \tag{2.5}$$

The static power consumption is when the system is powered but doing nothing, and is mainly due to the leakage current when all the inputs are at constant logic level and are not switching the logic level due to inactivity. The power consumption amount of P_S is very minimal. The Dynamic power consumption is the power consumption due to logic level switching and charging and discharging of capacitive loads [22]. It significantly contributes to the total power. Therefore, the average power P consumed by a micro-controller while executing a program is the product of the switching activity factor K, the load capacitance C_L, the clock frequency F_c, and the supply voltage squared V_{cc}^2 [9].

$$P = K * C_L * F_c * V_{cc}^2 \tag{2.6}$$

Given an MCU connected to a constant voltage supply and running at a fixed clock frequency F_c, the capacitive load will also be constant. Therefore, using equation 2.6, power P will only be dependent on the switching activity factor K. The switching activity factor is a function of the instructions executed and the data on which those instructions operate. Therefore, the power consumption of a MCU is a function of the instruction it is executing and the data on which the instruction operates. So, every instruction will have a different power trace depending on the data, and each BB 2.2 of the CFG will have a different power-trace signature, depending on the instructions and data in the BB.

2.1.2 Measuring the Power Consumption of a Micro-Controller

To measure the power consumption of a micro-controller unit MCU, we can put a shunt resistor of resistance R_L, in series with the MCU. The value of current is the same in a series circuit. So, the same current I flows in both, the shunt resistor and the MCU. The power consumption of the MCU is a multiplication of the total resistance of the MCU and square of the current passing through the MCU 2.3. And the total resistance of the MCU is constant. So, power is proportional to the current square. Therefore, current consumption in the MCU is a reflection of the power consumption by the MCU. The voltage V_L around the shunt resistor is $V_L = I * R_L$, where R_L is constant. So, V_L is proportional to the current I. Therefore, if we measure the voltage V_L around the shunt resistor, we'll get a power-trace, which is a reflection of the power consumption of the MCU.

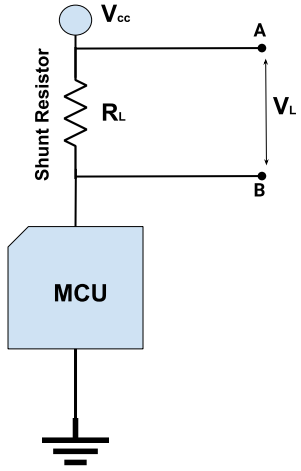


Figure 2.1: Setup to measure power consumption of a MCU

2.1.3 Power Traces

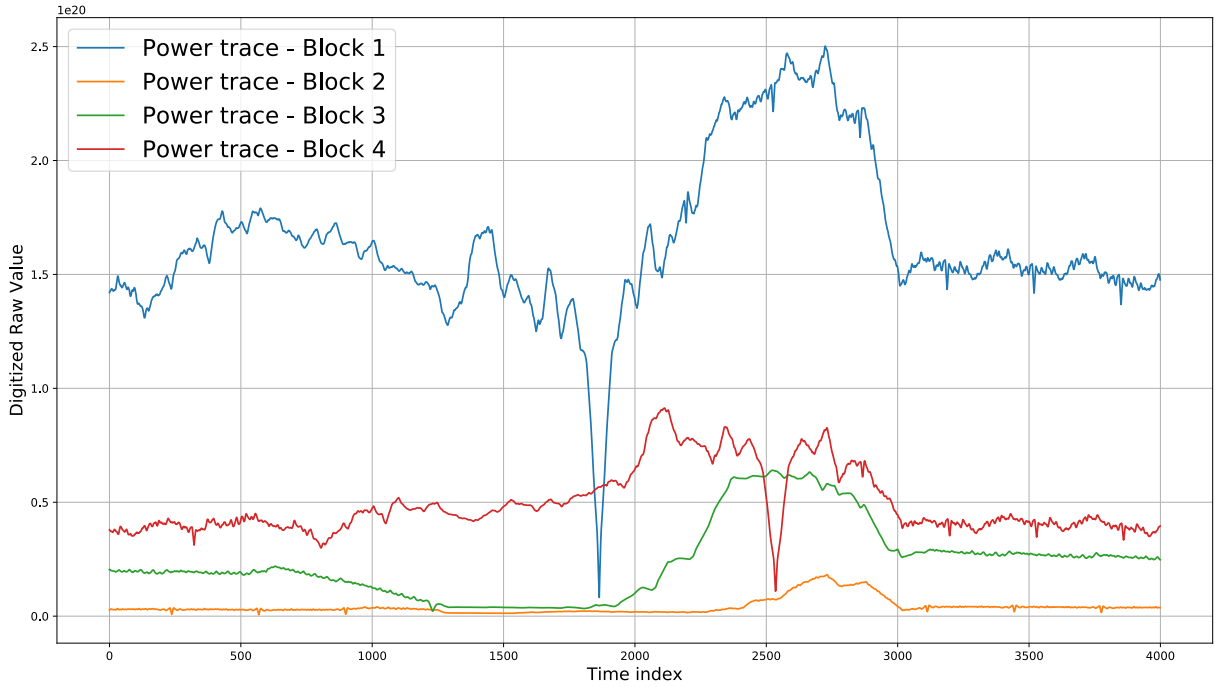


Figure 2.2: Example of power traces for four blocks of code

The power-trace $P = \langle V, t \rangle$, is the time series representation of a power consumption of a system as a function of time. Where V is the value of the power consumption at time t. Power consumption in terms of power-traces look like shown in Figure 2.2. Here, power-traces of 4 demo basic blocks BB are shown. As you can see, power-trace signals shape and values are all different, because power consumption is a function of the instructions and the data it's processing. So, it is different for different BB.

2.2 Basic Block (BB)

A complete code can be broken down into its basic blocks. A basic block is a sequence of instructions with single entry point at the beginning and single or multiple exit points at the end. A BB cannot have a branch in between. Basic block forms the nodes or vertices of a control flow graph 2.2.1. One BB is connected to another via an edge. Control from one BB can go to next BB, if there is no loop and to the same BB if there is a loop. Figure 2.3 shows the basic blocks in a control flow graph.

2.2.1 Control Flow Graph (CFG)

A piece of code can be represented as a directed graph with nodes and edges [2, 7, 8]. The Control Flow Graph $G = \langle V, E \rangle$ is a directed graph which represents the execution flow of a program. V is called vertex, and E is called edge in a CFG. Each vertex or node in a CFG can be considered as a basic block BB. Edges in a CFG are the paths from one node to another. Consider an edge between BB-1 and BB-2, from former to later, where $(BB-1, BB-2\ \epsilon\ V)$. This means the program has to go to basic block BB-2 after BB-1 is finished. Figure 2.3 shows a sample CFG with nodes as BB and edges as all possible paths, a program can follow.

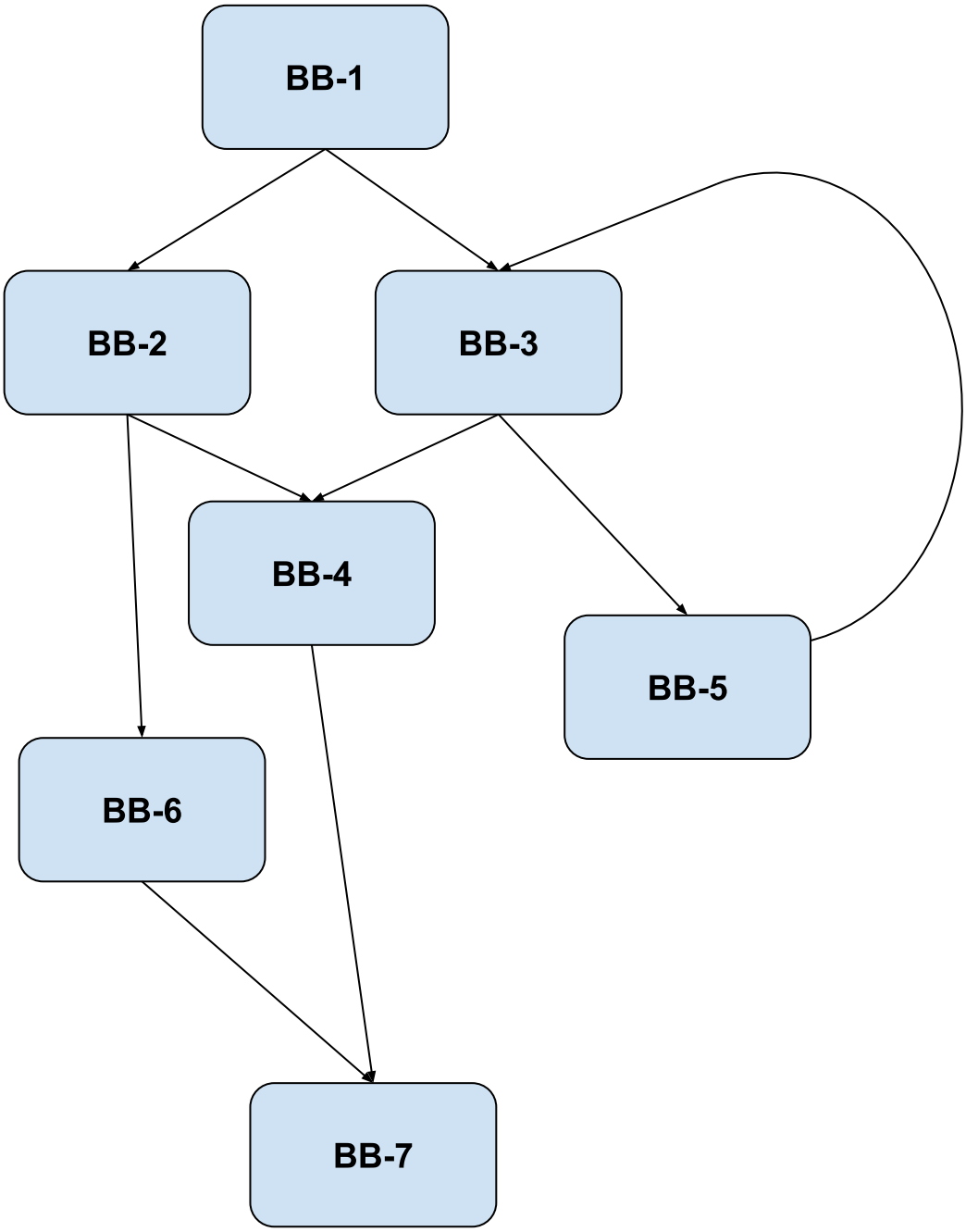


Figure 2.3: Sample CFG with basic blocks

2.3 Time Series Classification

As discussed above, a time series data D can be represented as $D = \langle V, t \rangle$, where V is the value of the time series at time t. Examples of time series data are - stocks data, weather data of a place, EEG, ECG, etc. We can compare two time-series signals by finding the numeric distances between them. Figure 2.4 shows a general view of distance measures in a time series data, that we can use. Broadly, there are four main categories: shape-based, feature-based, model-based and compression-based [29] [27] [37] [45] [14] [26]. In shape-based classification, we compare two time-series signals by their shapes. This can be done by normalizing both the signals and then computing the Euclidean distance between them. Suppose, we have two time series $D_1 = \langle V, t \rangle$ and $D_2 = \langle v, t \rangle$, then the Euclidean distance d between them will be:

$$d = \sqrt{(V_1 - v_1)^2 + (V_2 - v_2)^2 + ... + (V_t - v_t)^2} \tag{2.7}$$

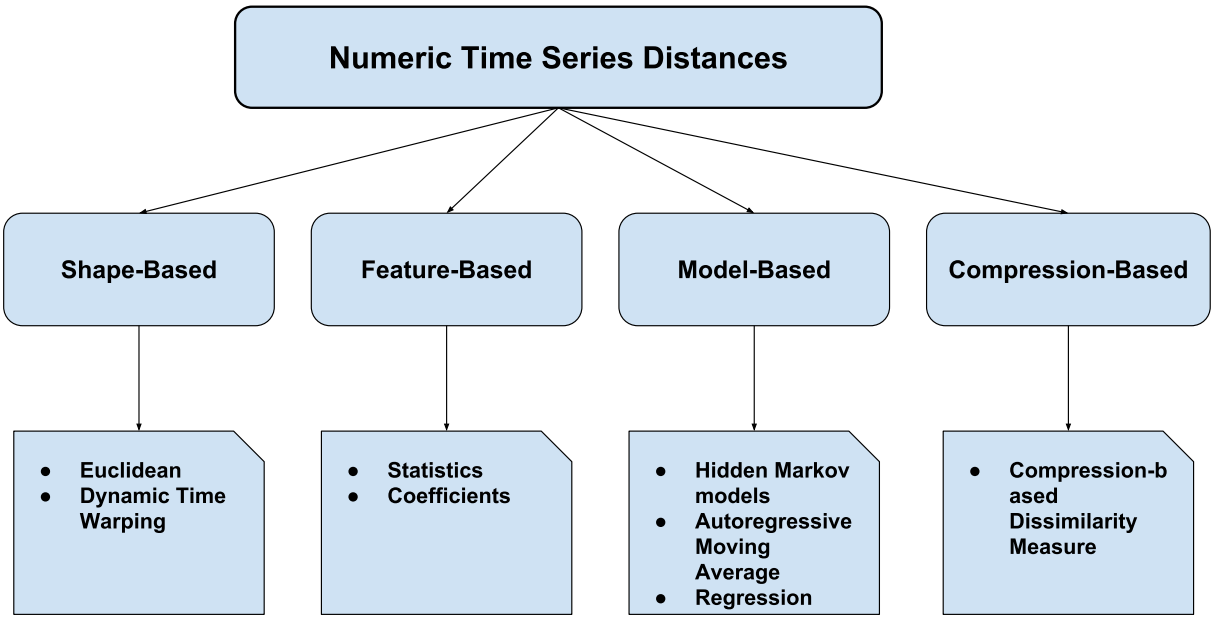


Figure 2.4: Categories of numeric time series distances

Another shape-based classification is dynamic time warping DTW [44] [19]. This classification method warps or scales one of the signal in time axis to match the other signal as close as possible. This can be of advantage when we have stretch and compression in a

signal during sampling. The Figure 2.5 shows two signals which are warped in timescale. The DTW classification method will find both of them to be equal.

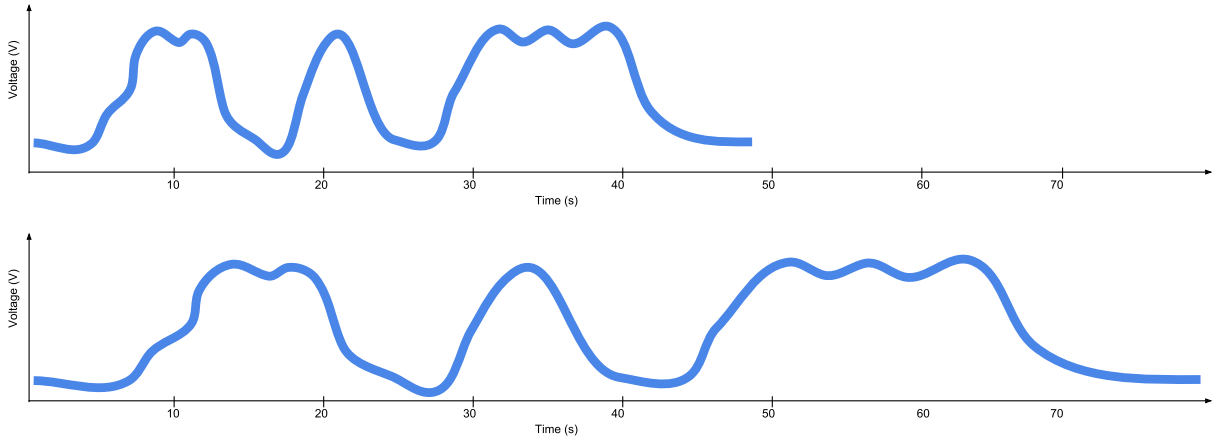
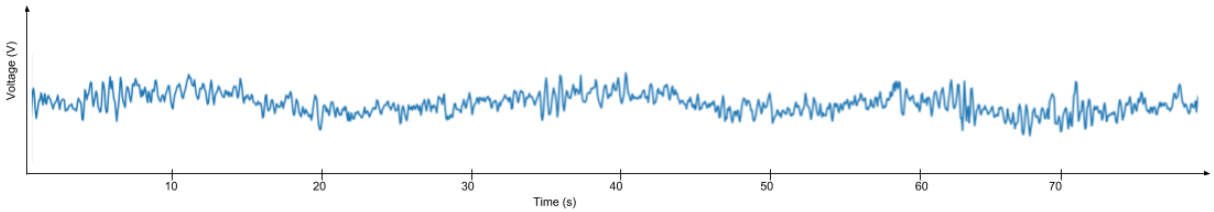


Figure 2.5: Two time series signals warped in time scale

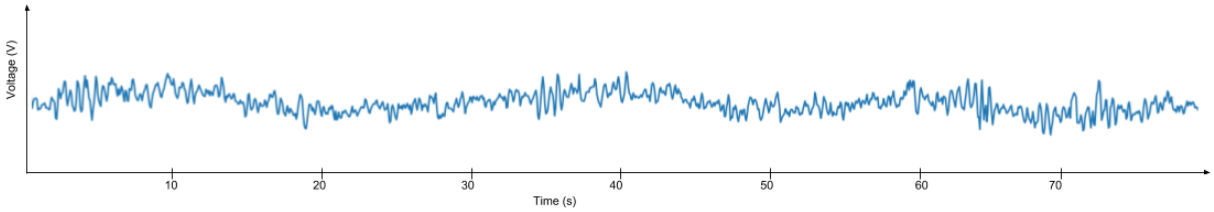
In feature-based classification, several features are collected from a time-series signal. Features like - length of a signal, changes in trend, max-min values, standard deviation, mean, etc. can be used. This reduces the dimensions in the training and testing data because earlier we were saving the complete trace of millions of data points, and now we are only saving few data points which are features from the complete trace. This also improves the classification time.

Chapter 3

System Model



(a) Sample Segment1 power-trace



(b) Sample Segment2 power-trace

Figure 3.1: Segments Powertrace Samples

As described in the section **text**, we use the concept of non-intrusive side channel power analysis to determine which segment of the code is being executed currently by the CPU. CPU doesn't consume constant power all the times. The power consumption of the CPU is dependent on the amount of work the CPU is doing at the instant. When CPU is busy executing some heavy tasks, it consumes more power than the times when it executes light tasks 2.1.1. So, if we divide our complete code into smaller chunks or segments and measure

power consumption for each segment, then each segment has different power consumption signatures, which would look like somewhat as shown in Figure 3.1. Here, each segment's power trace is a time series data. To get time series power traces of each segment from an MCU, we need some hardware setup and pre-processing, which has been discussed further.

3.1 Powertraces Capture Setup

We used custom-made Atmega328 microcontroller board 3.1.1 as our main target board running the standard CruiseControl code. Now, to peek into the power consumption of the MCU, we use a shunt resistor in series with the MCU board. If we now measure the voltage around the shunt resistor, we get a reading of voltage that is proportional to the current passing through the shunt resistor as $V = I * R_{shunt}$ and R_{shunt} is constant 2.1. The same current is passing through the MCU board, as current is the same in a series circuit. The power consumption of any circuit is $P = \frac{V^2}{R}$. Therefore, we measure the power consumption of the system in terms of the voltage change in the circuit.

The Figure 3.2 shows the block diagram of the complete hardware setup. As seen in the figure, the MCU power consumption (in terms of voltage) is fetched to the differential amplifier. The differential amplifier then amplifies the signal and fetches it to the channel-A of the Digitizer 3.1.2. At the same time, the signal from the MCU port pin in terms of a digital square wave is fetched to the channel-B of the Digitizer. Let's call this signal "Port_bit_flip_marker". The usage of this signal in the setup is discussed in Section 3.1.3. The output of the Digitizer is saved on the PC in binary files.

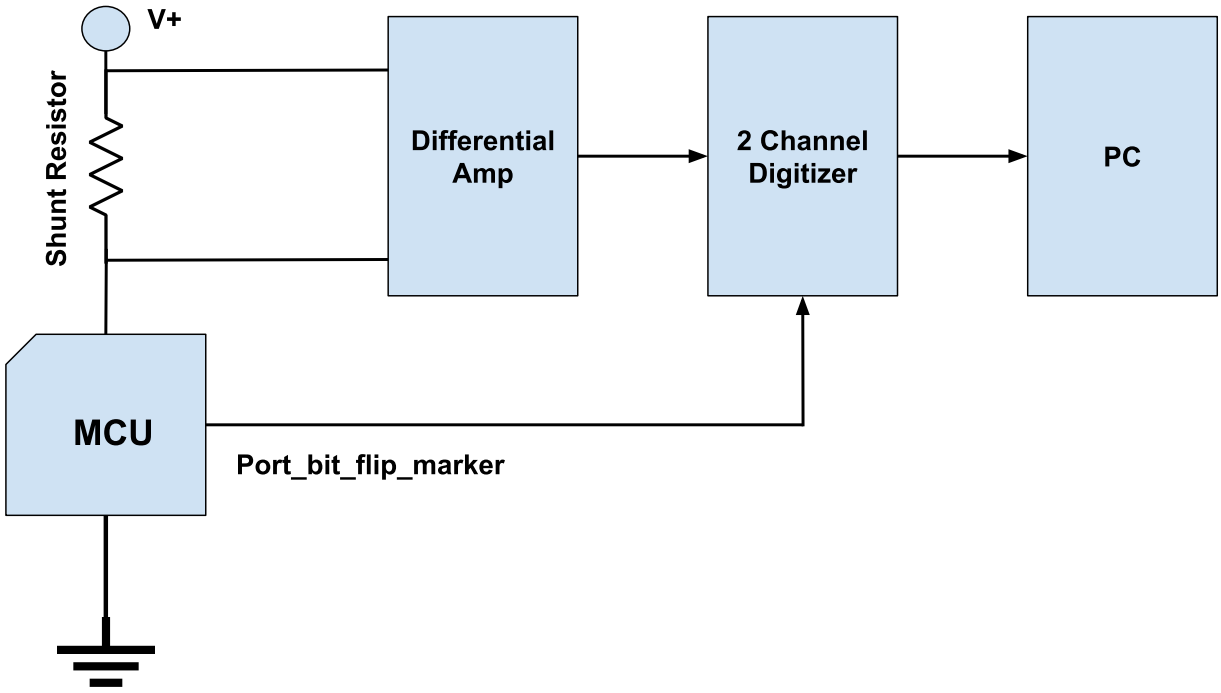


Figure 3.2: Powertraces Capture Setup

3.1.1 Micro-controller Board

We used Microchip's Atmega328p as the target microcontroller. The custom board designed for the target MCU has LEDs and push buttons connected to the peripherals, as needed by the target code to run properly. The port pin needed for the markers (see Section 3.1.3) was connected to the male pin connector, for easy connection with the Digitizer. The board also has a female pin connector for the shunt resistor and the crystal. The board has separate voltage source for the processing unit, I/O pins and the ADC.

3.1.2 Digitizer

The Digitizer is the proprietary of Alazar Technologies Inc. We use ATS9462 Digitizer, which is 16-bit, 2-channel, high-quality Analog to Digital Converter. It can sample up to 180MS/s (Mega-Samples/second) across two simultaneous inputs. We can vary sampling rate as well as other parameters. This ADC can sample from 1KS/s to 180MS/s.

3.1.3 Port Bit Flip Markers

Consider the CFG in Figure 2.3. Now, let's modify the BB to add one extra command at the end of each BB. This command is to toggle the logic level of a particular Port pin. The new CFG would look like as shown in Figure 3.3. This toggling method gives us another channel of data - mainly digital square wave, which can be used as markers to figure out power trace of start and end points in the complete power trace data in the channel-A. As mentioned in Section 3.1.1, we have a seperate power sources for I/O, so toggling port pin would not affect the actual power consumption of the system. Figure 3.4 shows the processed output of the Digitizer as channel-A and channel-B. You can see from the figure, how helpful are the markers in segmenting the power trace with respect to the BB. There are some noise peaks in channel-A exactly at the toggling of the markers, which we have removed by discarding the starting few sample points.

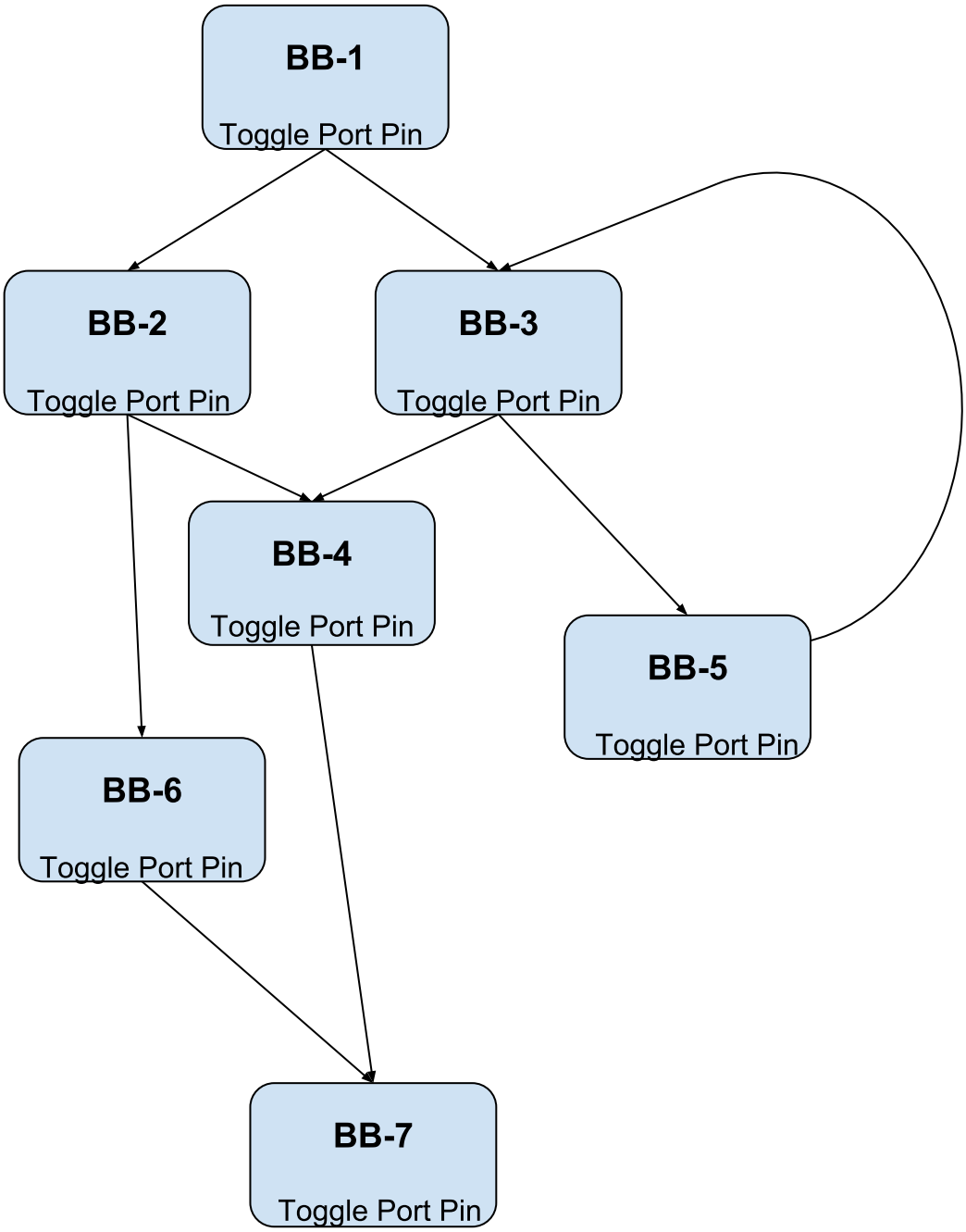
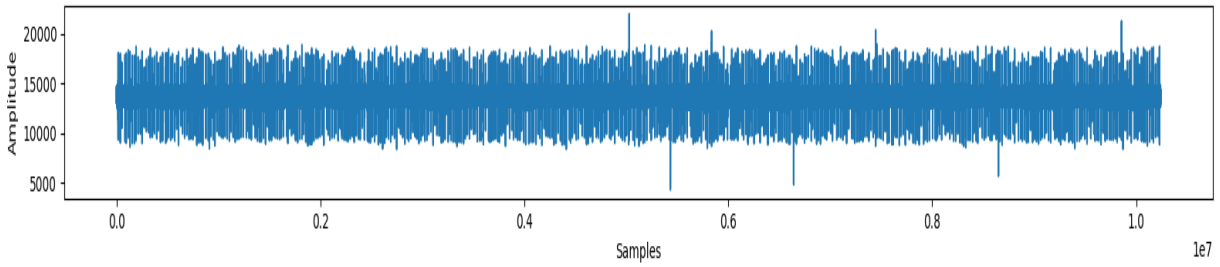
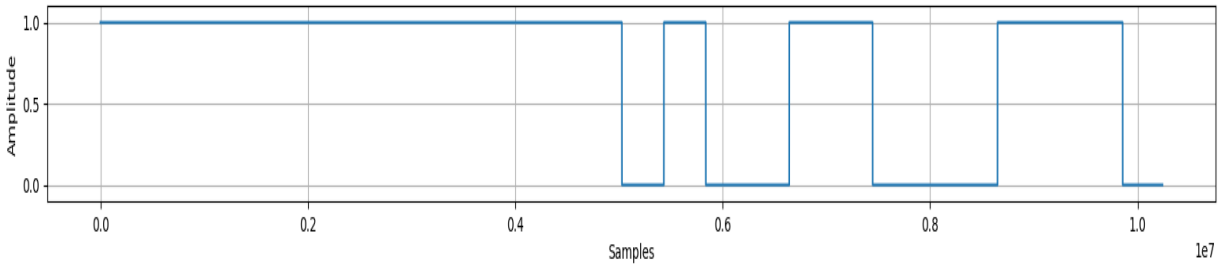


Figure 3.3: Toggling Port pin added at the end of each basic block



(a) Powertrace data in Channel-A



(b) Port_bit_flip_markers on channel-B

Figure 3.4: Digitizer processed 2-channel data

3.2 Source Code Instrumentation

Section 3.1.3 showed how to get power trace segment for each basic blocks with the help of markers. But the next issue now is that the power-traces are unlabeled. To address this issue, we instrumented the code into two versions. The first version is the original code that runs on the target MCU. Each Basic block in the 1st version has last line to toggle the port pin as shown above in Figure 3.3. The second version of the code is exactly like the first one except that it runs offline on the workstation, with the same data and random seed as the online version running on the MCU, but instead of the toggle pin command, it executes the *"printf"* command which prints the name of the current BB. This is how we get a sequence of labels which correspond to the sequence of segmented power-traces that we got earlier. Therefore, source code instrumentation method helps us to get the labeled data.

3.3 Merging the Smallest Basic Blocks

By running the CruiseControl program on the Atmega328 board running at clock frequency $F_c = 1MHz$, we got total 16 BB or classes, when the sampling rate $F_s = 1MS/s$. Out of the 16, there were 3 BB with the power trace sample points less than 100. These were the smallest segment lengths out of all 16 segments. Classifying them was resulting in very less accuracy. Also, with either a decrease in sampling frequency or increase in the clock frequency, these segments were getting even smaller. Some segments were even lost at $F_s = 100MS/s$. So, to resolve the issue, these smallest segments were merged into the parent segments, without disturbing the CFG of the code. Carlos et al. [35] showed the use of merging BB to its parent. The Figure 3.5 shows how the merging of the nodes can be done without disturbing the CFG. The left side of the image contains an example CFG with some nodes and edges. Suppose the node **B** marked in dark color is a very small segment. We need to merge it into its parent. The proper merging is shown on the right side of Figure 3.5. Now, the parent node **A** has become node **A'**, which is now bigger than the node-A.

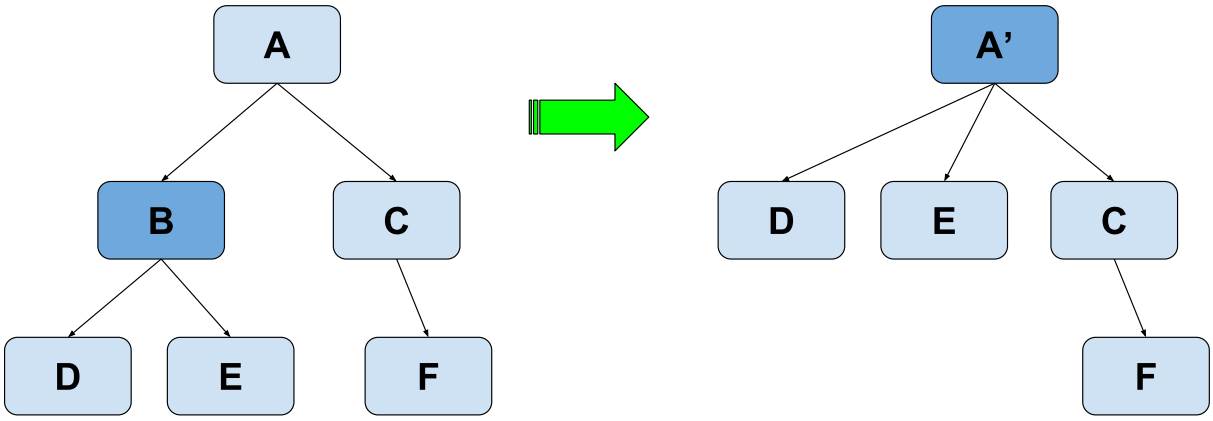
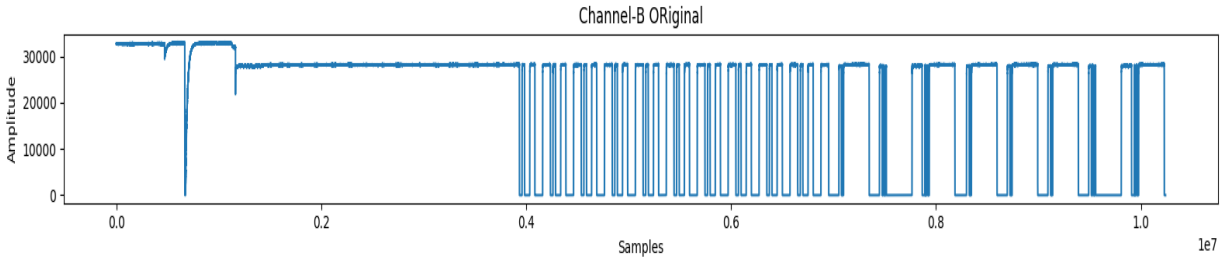


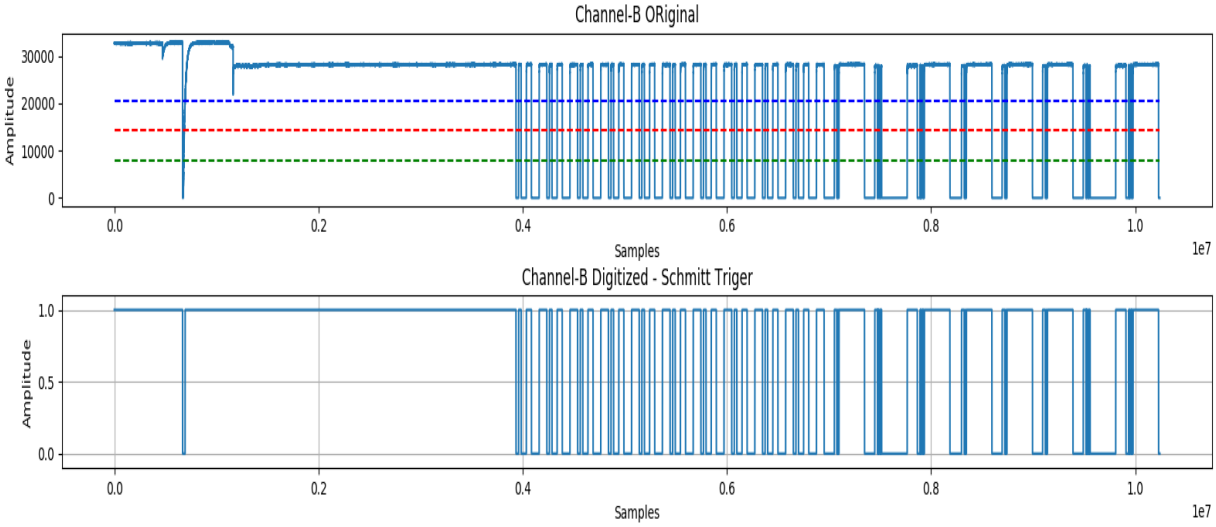
Figure 3.5: Example of merging the CFG nodes

3.4 Digitizer Output Data Preprocessing

Analog signals are prone to noises. These noises pass to the ADC and even reflected in the output digital data. The channel-B data from the Digitizer is shown in Figure 3.6a. Port_bit_flip_marker signal is very crucial, as channel-B is responsible for correct



(a) Channel-B signal affected with noise



(b) Channel-B noise removed using Schmitt trigger

Figure 3.6: Removing noise from Channel-B signal using Schmitt trigger

segmentation of the power-traces. Even a slight error like an extra toggle or a missing of a toggle signal can result in a garbage data. So, we need a very clean channel-B. We achieve this using the technique called Schmitt-trigger [46]. We have programmed the Schmitt-trigger filter with 3 thresholds - high-threshold, low-threshold and the normal threshold. The normal threshold is selected as the mean of the signal in channel-B. High-threshold,low-threshold are selected as normal threshold plus,minus one third of the difference between maximum value of signal and the mean. The working of the Schmitt-trigger is pretty simple. If there's a falling edge, then the signal should first cross high-threshold and then cross low-threshold in the order. If the order is not maintained then, we can consider it as a noise peak. Similarly, for the rising edge the order is first low-threshold and then high-threshold. Figure 3.6b shows the output of the channel-B data, after passing it through the Schmitt-trigger. You can clearly see the noise around $0.1x10^7$ (on x-axis), is filtered out easily.

3.5 Generating Training Data

In the following section, we will talk about the methods used to collect the training dataset, the variations in two instances of the same class and how the training dataset looks. We also talk about averaging the samples and NCC.

3.5.1 Variations in Samples in each Class

The complete target code was run 1000 times in a loop with different random numbers input initialization. This different random numbers data processing will produce different power-traces for the same basic block in different loops because each power-trace of the same BB depends on the data the BB is processing at that time. These (from the same BB) will follow a same the pattern and shape with slight differences because of different data inputs. Example of these is shown in Figure 3.7.

These four samples are from the same class - "RandomizeInputs". As you can see clearly, all the four samples are similar, but if you look carefully, you'll find subtle differences. The purpose of running the program in a long loop is to get all types of variations that we can get in a power-trace for the same class (same basic block). This variation is helpful in getting a good training data. Now for each class, we have at most 1000 samples. Each sample is a time series data. This is one dataset. We similarly captured 5 datasets. There are several machine learning algorithms that can be used with this training data.

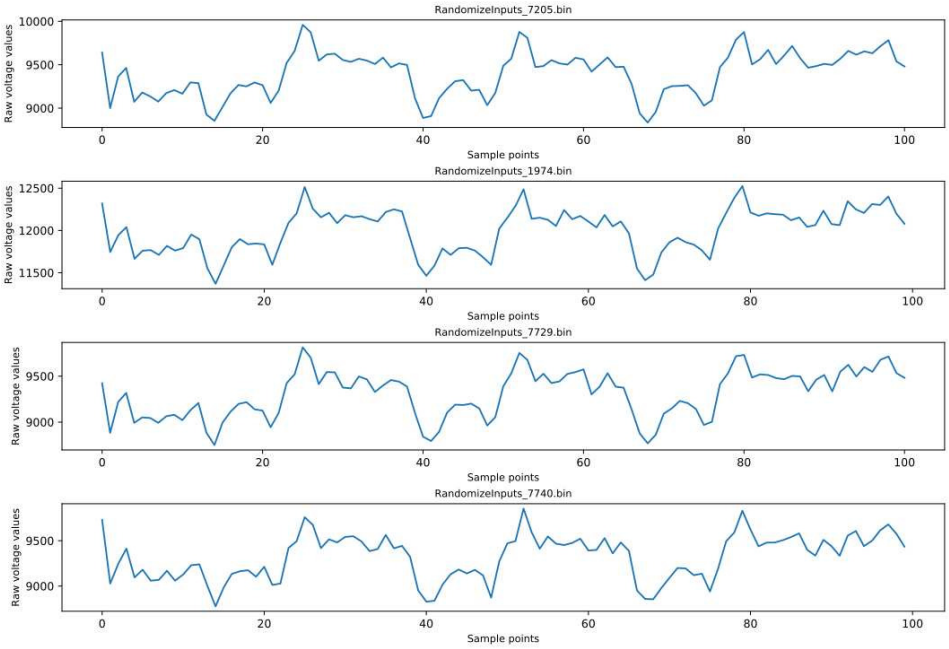


Figure 3.7: Sample CFG with basic blocks

3.5.2 Total Classes

As discussed in Section 3.3, we have a total of 13 classes. Each class has at most 1000 samples with slight variations. For fast processing during classification, we reduced the number of training samples by taking the averages of all the samples in a class. This gives us total of 13 samples for 13 classes (one sample for each class). These average samples for each class along with the Frequency components is shown in Figure 3.8. The mean of a set of samples is a better representation of the data or information, rather than the individual samples. Francis Galton showed this in his paper [18]. He noted that the crowd at a country fair accurately guess the average weight of an ox. The **average** of the weights given by people at the fair was far closer to the actual weight of the ox than the individual weights. This information is widely exploited in machine learning. For example, the Nearest Centroid classifier NCC [50] generalizes the Nearest neighbor classifier NNC [11] by replacing the set of neighbors with their centroid. There are two important reasons for selecting the former over the later. Firstly, it is faster, being only $O(1)$ instead of $O(n)$, assuming that the number of classes is bounded by a constant. Secondly, for some datasets, NCC is more accurate than NNC [20].

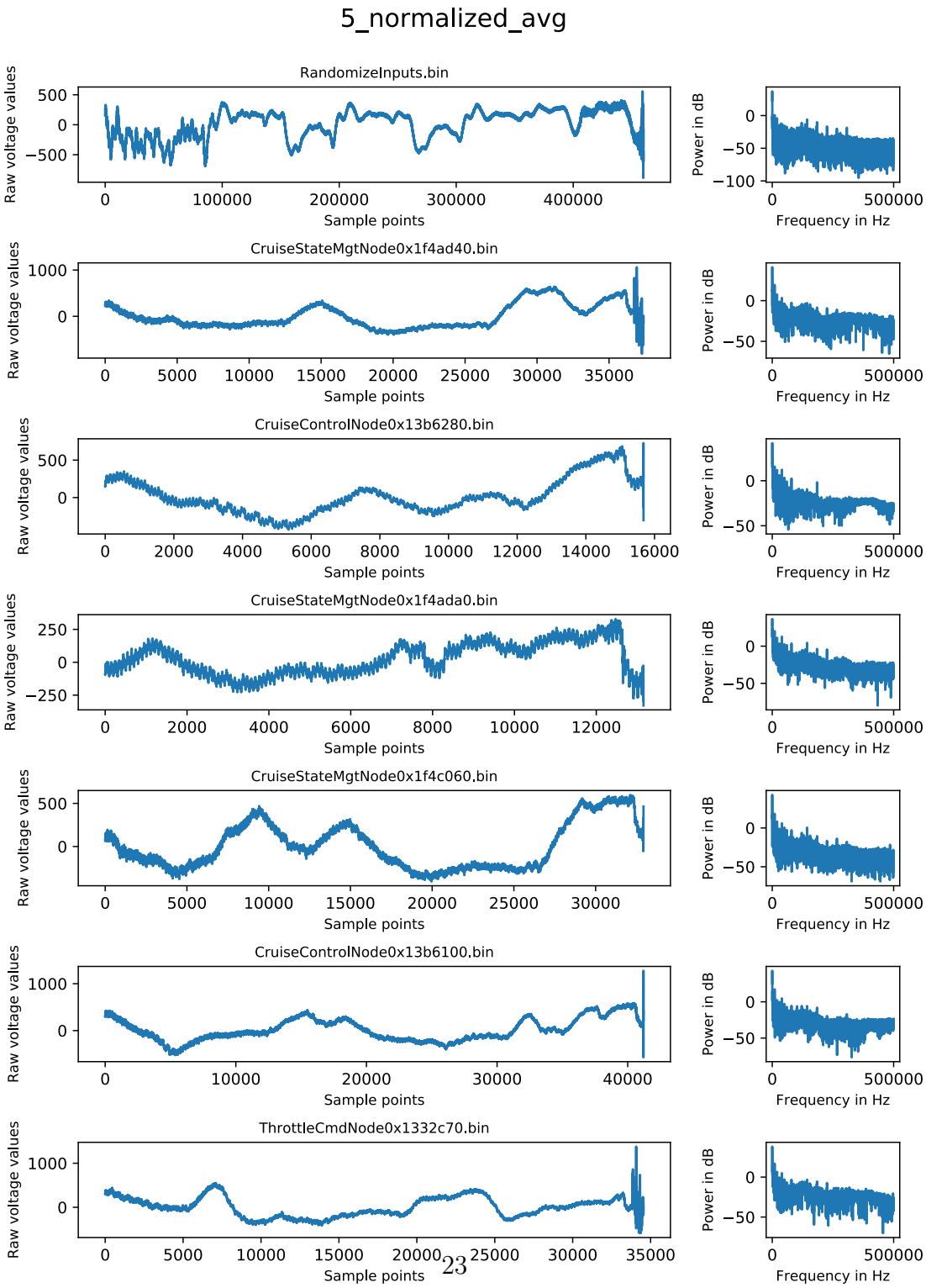


Figure 3.8: Average sample of each class

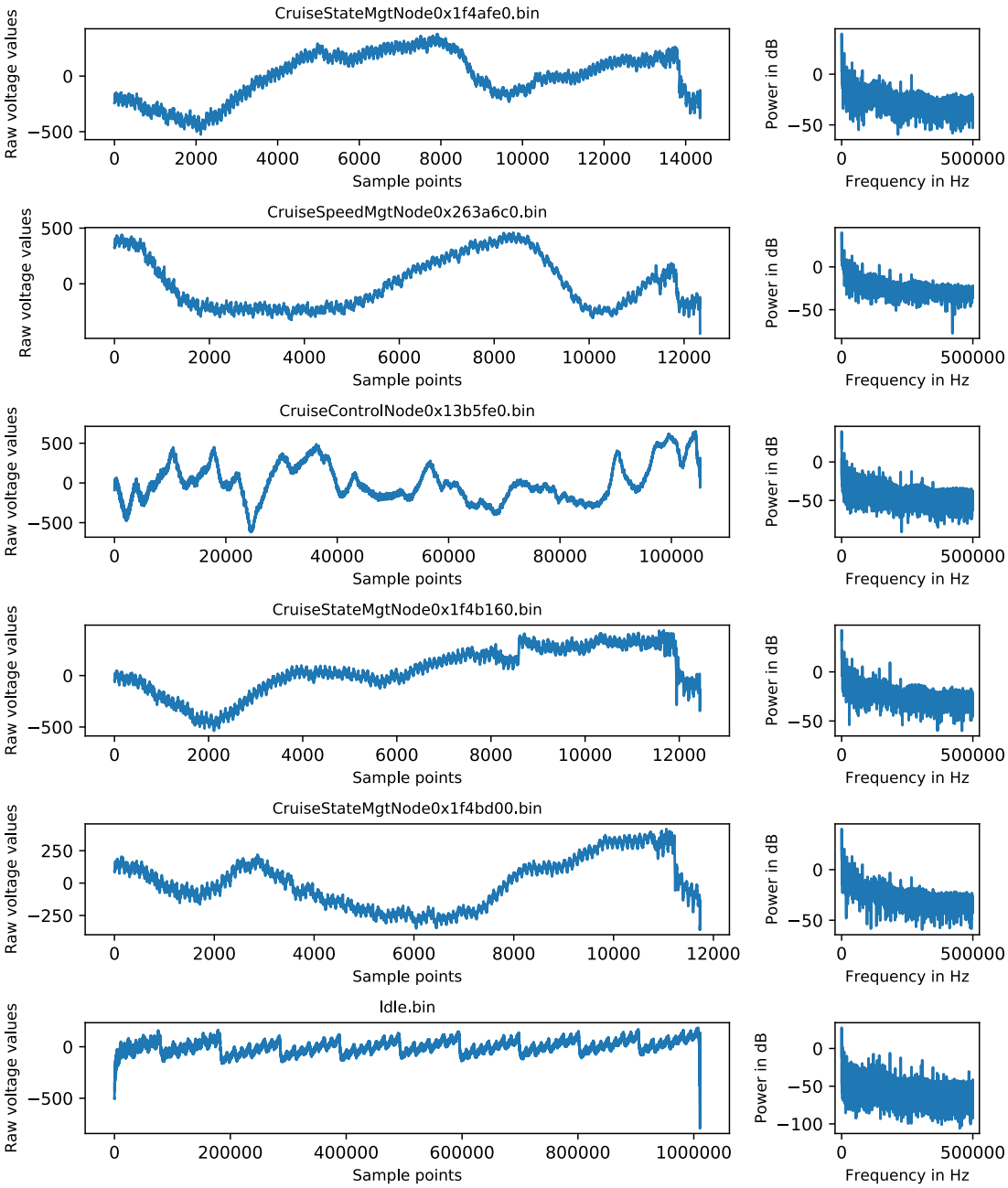


Figure 3.8: Average sample of each class

3.5.3 Training Dataset

For the purpose of the study, to find the optimal sampling frequency of power-traces of a system running at a clock frequency F_c, for classification, one needs a big dataset of power-traces. So, the micro-controller board mentioned in Section 3.1.1, is run on several clock frequencies - 250KHz, 500KHz, 1MHz, 4MHz, 8MHz and 12MHz. Then for each clock frequency, the dataset is collected in the same manner as mentioned in the steps in the above sections, at several sampling frequencies F_s (Digitizer sampling frequency 3.1.2). The several sampling frequencies selected were 100MS/s, 50MS/s, 25MS/s, 10MS/s, 5MS/s, 2.5MS/s, 1MS/s, 500KS/s, 250KS/s, 200KS/s, 100KS/s, 50KS/s, 40KS/s and 25KS/s [1]. Therefore, for each F_c we have a dataset at all the F_s mentioned above. Total datasets = number of F_c multiplied by the number of F_s multiplied by 5 (as we have five datasets in each combination 3.5.1).

$$\text{Total datasets} = 6 * 14 * 5 = 420.$$

Each dataset has 13 classes and each class has at most 1000 samples, which are averaged and combined into one sample. This is the all the dataset at several sampling and clock frequencies, that would be used to train the classifier for anomaly detection.

3.6 Testing Setup

The final testing setup for the purpose of anomaly detection would look like the one shown in Figure 3.9. Here the MCU-1 is being monitored for anomaly detection. This is done by observing the power consumption of the MCU-1. The MCU-1 is running the CruiseControl program, the same program used for collecting the training datasets. The monitoring system is used here for monitoring the MCU-1, running the anomaly detection program. The power consumption of the MCU-1 is fetched to the ADC of the monitoring system. The digital output data stream from the ADC is used as test data. The Train data is saved into the memory of the monitoring system. The CPU of the monitoring system is running the anomaly detection code. The similar setup has been used by Carlos et al [35].

[1] MS/s - Mega Samples per second, KS/s - Kilo Samples per second. One Sample is 16-bit long.

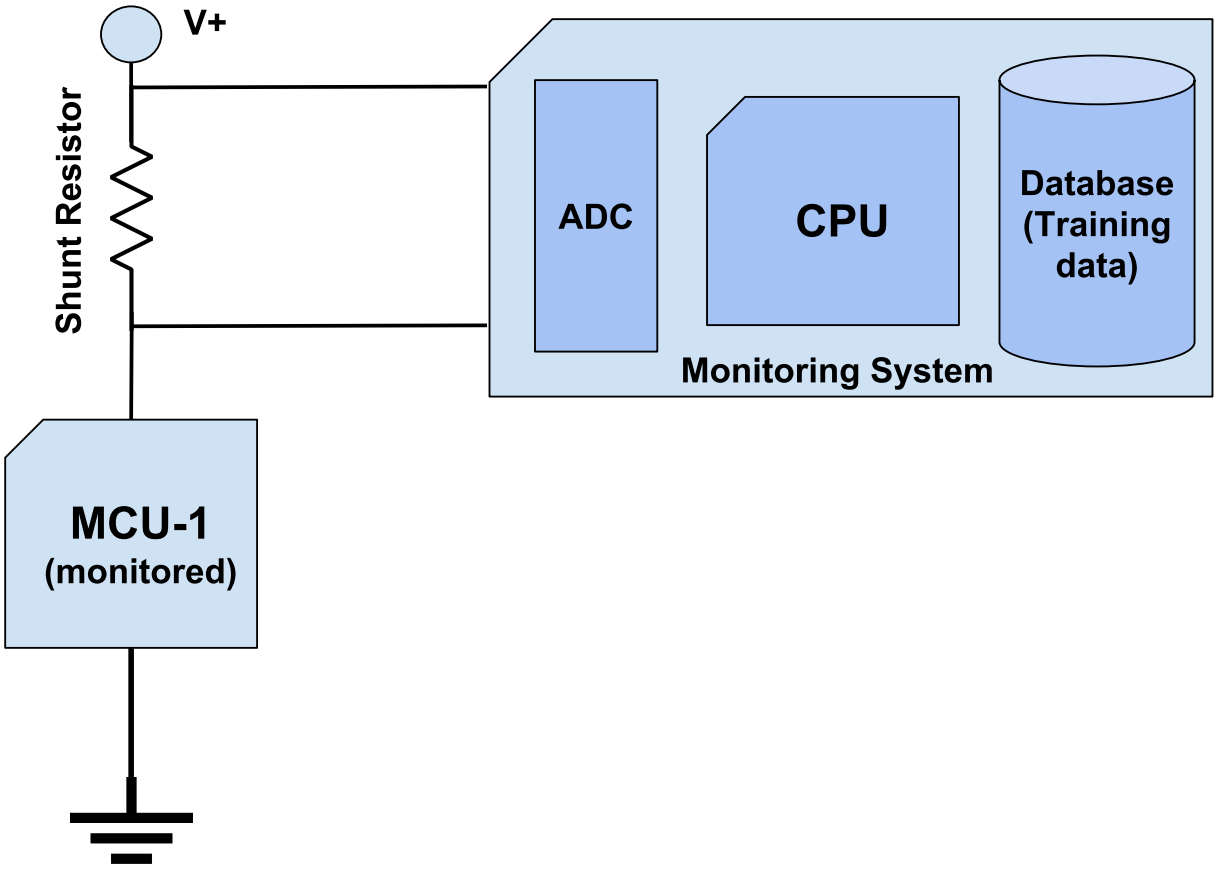


Figure 3.9: On-the-fly Anomaly detection

The benefits of having a separate monitoring system are as follows:

1. Even if the MCU-1 is hacked, the security of the Monitoring system is not compromised, as it's a separate system.
2. The cost of the monitoring system can be brought down if the training database is small enough to be fit in the default memory of the MCU.
3. The size of the monitoring system can be very small, depending on the database and the sampling frequency required for the anomaly detection.

3.7 The Anomaly Detector

The heart of the monitoring system is the anomaly detector that alerts the administrator if finds some anomaly in the system. The anomaly here can be anything, other than the usual CruiseControl code running in the system. The Power consumption by the default program is different than the power consumption by any other piece of code. The monitoring system is constantly monitoring the power consumption of the MCU-1. If it encounters any other power consumption signatures, it can raise a red flag for it. The power-traces of every possible segment of code are saved as the training dataset. The training data is coming from the ADC of the monitoring system. The anomaly detector compares the test-data with all the power-traces in the training dataset. If it matches with any of the training data samples, it labels it safe, otherwise raises a red flag. If it raises a red flag, a message is sent to the administrator about the detection of an anomaly. Then suitable measures can be taken by the administrator. These suitable measures can also be automated and coded in the monitoring system for removing the administrator level and making everything automatic. For the purpose of comparison between testing and training data samples, several metrices can be used as mentioned in Section 2.3. The classification method we use in out anomaly detector is a shape-based Euclidean Distance measure between the two samples. We also used FastDTW [44] for the purpose of testing, results of which are not included in the thesis.

Chapter 4

Sampling Frequency

In signal processing, sampling is the method of getting a discrete time signal from the continuous time signal. It is used mainly for storing and processing of real life data (continuous signal) into computers. For example, the conversion of sound wave (a continuous signal) to a sequence of samples (discrete signal) via microphone and sound card of a computer. An example is shown in Figure 4.1. Here the continuous signal is shown with the green colored curved line, while the discrete-signal is represented by blue vertical lines. Any continuous signal can be converted to its samples by the process of sampling.

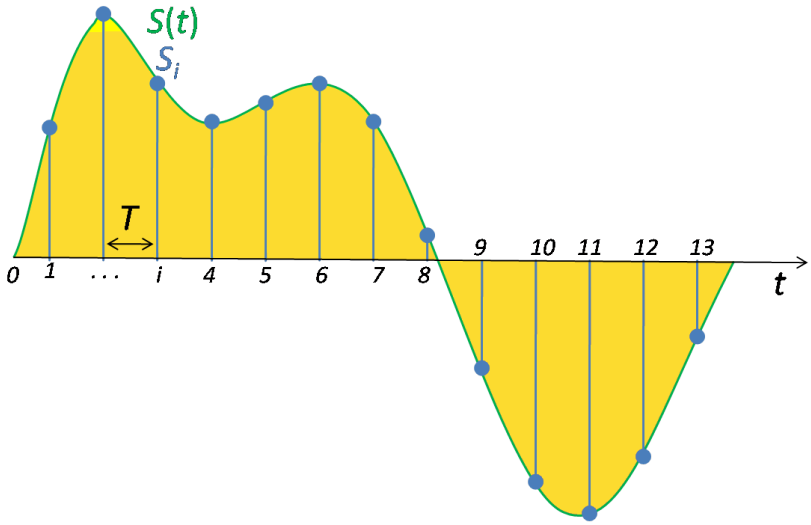


Figure 4.1: Signal sampling representation.

A sampling frequency F_s, is the frequency at which we sample a continuous signal. The choice of sampling frequency depends on several factors. Mainly, F_s should be such that it correctly samples the continuous signal, without loosing any of its information.

4.1 Analytical Approach

Several researcher and mathematicians have worked on finding the best sampling frequency for signal reconstruction. Here, we have discussed those analytical methods and the reason why they are not suited for the power-trace anomaly detection. Then, we have shown our intuitive relationship between the sampling frequency, the clock frequency, and the smallest segment signal length, for power-trace anomaly detection.

4.1.1 Nyquist Criterion

Many mathematicians have worked on the formula for finding the optimum sampling frequency of a continuous signal. The main work in the field is done by Harry Nyquist and Claude Shannon [38] [47]. They together gave the famous Nyquist-Shannon Sampling theorem. The theorem was also discovered independently by E. T. Whittaker, by Vladimir Kotelnikov, and by others. It is thus also known by the names NyquistShannonKotelnikov, WhittakerShannonKotelnikov, WhittakerNyquistKotelnikovShannon, and cardinal theorem of interpolation. Shannon's version of the theorem states [47]

> If a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced $1/(2B)$ seconds apart.

Therefore for a proper reconstruction of the analog signal (continuous signal) having the highest frequency component F_{max}, we need to samples it at sampling frequency higher than or equal to twice the F_{max}. This gives us the famous equation:

$$F_s \geq 2 * F_{max} \tag{4.1}$$

If you use the sampling frequency using the Nyquist criterion, then we are surely certain that the waveform reconstructed will contains all the features from the original signal and nothing has been lost in the process of analog-to-digital conversion. If we do sample at lower frequency than the given by th formula 4.1, then we will get a signal that is

low resolution and might have lost some of the important features. Consider Figure 4.2, showing an analog sinusoidal signal at a frequency f. When the signal is sampled at f, at same frequency, the reconstructed signal is a straight line. Thus, we completely lost the actual signal. Similarly, if we sample the signal exactly at $2f$, the sinusoidal signal now becomes a triangular wave. Therefore, we need to follow Nyquist criterion, if we want to properly reconstruct the actual signal.

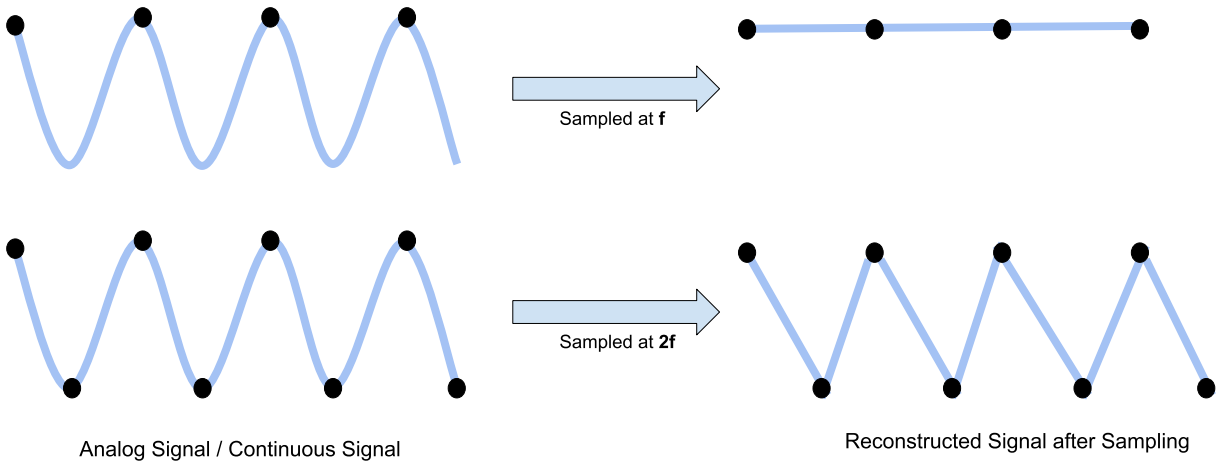


Figure 4.2: Sampling analog signal at different frequencies

4.1.2 Sampling Frequency for the Purpose of Anomaly Detection

For the purpose of anomaly detection, we don't need to reconstruct the power-trace exactly to the original, as our motive is not to get all the detailed features, but to get a good classification rate while detecting anomalies in the system. We can even accept a low quality of the power-trace, if it is showing some reasonable amount of features, which will not affect our classification rate. Following the Nyquist criterion, for the purpose of anomaly detection, will have the following issues:

1. Large size of training dataset.
2. Large size of testing signal.
3. High processing power required for classification.
4. More time required for processing.

Getting a complete comprehensive picture of the power-trace is no doubt is a best thing, but with it, it brings some drawbacks as discussed above. Bigger value of sampling frequency results in more memory and processing power of the anomaly detection system. More data, more memory, more processing power. Getting a little low sampled power-trace will not make a huge difference in anomaly detection, as it will make for example saving low sampled music files in a MP3 player. The application and the purpose make a huge difference in selecting the best sampling frequency for a signal. But then the question arises, "How low shall we go?". We cannot go too low otherwise we will loose all the features of the segment. Also, if we go for a very low sampling frequency, there are chances that we completely loose some of the very small segments in the power-trace dataset. Therefore, we need to find the optimum sampling frequency range, which is not too high (for avoiding the drawbacks given above) and not too low (for a good classification accuracy).

4.1.3 Our Approach to the Sampling Frequency

The equation 4.1 can be written as equation 4.2 in terms of time period.

$$F_s \geq 2 * \frac{1}{T_c} \tag{4.2}$$

Where, T_c is the time period of the system running at clock frequency F_c. We know that the time period, $T_c = 1/F_c$. Lets suppose the MCU in Figure 3.2, is running at a clock frequency $F_c = 1MHz$. And we collect a power-trace signal from the MCU for 1 second. And the source code instrumentation 3.2 gives us 10 segments or BB within this 1 second time frame. So, we have 10 power-trace segments. The number of sample points in the complete power-trace for 1 seconds will be 1 million. The 10 power-traces are sub-signals or sub-segments from the whole complete power-trace of 1 million points. For $F_c = 1MHz$, the time period $T_c = 1/F_c = 1us$. Therefore, the smallest segment or BB cannot be smaller than $1us$. Suppose the smallest segment is $1us$ long i.e., 1 sample point long. Therefore, to prevent the smallest sample from getting lost we need to sample it at greater than or equal to Nyquist frequency 4.1. Now suppose, the smallest segment is not 1 sample point long, it has L_{min} sample points (out of 1 million sample points). To prevent it from getting lost while sampling, we need atleast 1 sample point out of L_{min}. And then, if we follow Nyquist criterion, we can prevent that sample from getting lost. That means, the sampling frequency in the equation 4.1 is divided by a factor of L_{min}. So, if we put our scaling in the equation 4.1, we get the following equation:

$$F_s \geq 2 * \frac{F_c}{L_{min}} \tag{4.3}$$

But scaling the L_{min} sample points to 1 sample point, of course leads to loss of information. Also classifying a segment of only 1 sample point doesn't make any sense. We need atleast **"m"** sample points out of L_{min} sample points for correctly classifying the power-trace segment. Adding the factor **"m"** in the equation 4.3, and re-arranging the variables gives the equation 4.4.

$$\boxed{\frac{F_s}{F_c} \geq \frac{2 * m}{L_{min}}} \tag{4.4}$$

As we go low in clock frequency or high in sampling frequency, we get more features in power-traces which are hidden at higher clock or lower sampling frequencies. Let's define N as the ratio of sampling over clock frequency.

$$N = \frac{F_s}{F_c} \tag{4.5}$$

Therefore, for a good amount of features in a power-trace signals, we need high value of N in the equation 4.5. Combining the equation 4.5 and the equation 4.4, we get equation 4.6.

$$\boxed{N \geq \frac{2 * m}{L_{min}}} \tag{4.6}$$

where,
N - is the ratio of F_s and F_c.
F_s - is the sampling frequency of the anomaly detector.
F_c - is the clock frequency of the system.
L_{min} - is the length of the smallest segment of code (in sample points).
m - is a scaling factor constant, required for correct classification.

This final formula is based on our intuition and is open for future research. Therefore, analytically, the ratio of sampling frequency over clock frequency should be greater or equal to ratio of two times the scaling factor m and length of smallest BB.

4.2 Empirical Approach

To find the optimum sampling frequency of the power-traces for the purpose of anomaly detection for a system running at a particular clock frequency, we need a large comprehensive dataset. The training dataset 3.5.3 comprising of different combinations of sampling

and clock frequencies was used to train the system and was used in the classification 3.7 with the test-data 3.6. The classification module also measured the training-time, testing-time and standard deviation, along with the accuracy. All these results with the different sampling and clock frequencies are shown and discussed further.

The graph 4.3 shows the plot between classification accuracy and sampling frequency. The shaded boundary around the accuracy is the standard deviation i.e., the max region, accuracy can deviate from its mean value. The Table 4.1 shows accuracy percentage, standard deviation (STD), smallest segment length (L_{min}), ratio $N = (\frac{F_s}{F_c})$, calculated value of the constant m and CPU total time taken for classification (in seconds) at different sampling frequencies. Similar convention is followed by other graphs and tables for classification accuracy and sampling frequency.

The Table 4.1 shows a reasonable amount of accuracy even down to $F_s = 25KS/s$, and the smallest segment has twelve sample points at that sampling frequency. The reason for such a good accuracy even down to $F_s = 25KS/s$ is because at lower clock frequencies, the size or length of the segments are bigger. The reason being the system is running slower than the normal, so we capture more data at the same sampling frequency. The Table 4.2 shows a reasonable amount of accuracy down to $F_s = 40KS/s$, and below that sampling frequency accuracy drops significantly. One can easily see that the length of smallest segment is very small below $F_s = 40KS/s$, which is one of the factor of less accuracy. The Table 4.3 shows a reasonable amount of accuracy down to $F_s = 50KS/s$. The Table 4.4 shows a reasonable amount of accuracy down to $F_s = 250KS/s$, and below that sampling frequency accuracy drops significantly. Again, the length of the smallest segment is very small below $F_s = 250KS/s$. The Table 4.5 shows the accuracy starts to decrease at $F_s = 250KS/s$. The Table 4.6 shows a reasonable amount of accuracy down to $F_s = 500KS/s$, and below this sampling frequency the value of L_{min} comes to three sample points. Again, very less features to do a good classification. Therefore, for a good accuracy, the selection of the sampling frequency should be such that, even the smallest segment length should be big enough to have sufficient features.

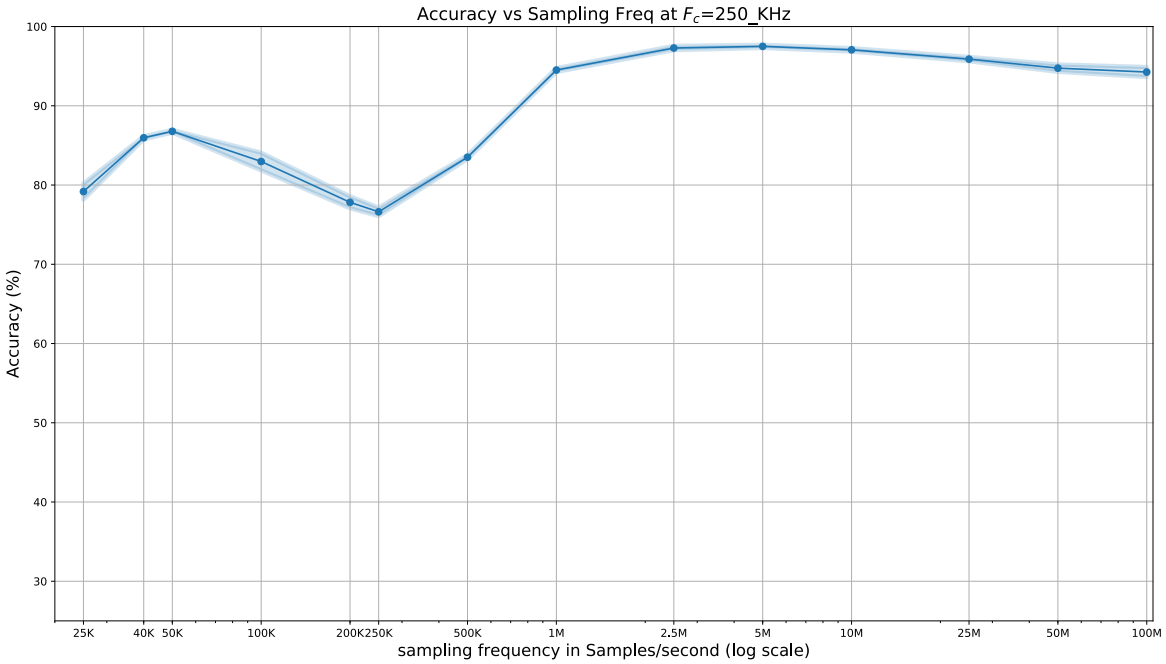


Figure 4.3: Classification accuracy at different sampling frequencies, when $F_c = 250KHz$

F_s **(Samples/s)**	**Accuracy**(%)	**STD**	$Lmin$	$\frac{F_s}{F_c}$	**m**	**CPU time(s)**
100M	94.26	0.633	44802	400	8.9604e+06	5377.26
50M	94.75	0.466	22401	200	2.2401e+06	2648.92
25M	95.89	0.272	11201	100	560050	1369.98
10M	97.06	0.217	4481	40	89620	587.66
5M	97.5	0.203	2241	20	22410	342.09
2.5M	97.29	0.269	1121	10	5605	190.03
1M	94.51	0.238	449	4	898	101.59
500K	83.51	0.251	225	2	225	41.78
250K	76.62	0.532	113	1	56.5	31.02
200K	77.82	0.76	90	0.8	36	26.79
100K	82.97	1.117	45	0.4	9	16.81
50K	86.78	0.17	23	0.2	2.3	14.71
40K	85.96	0.192	18	0.16	1.44	12.97
25K	79.18	1.019	12	0.1	0.6	12.68

Table 4.1: Accuracy and Standard deviation at different F_s, when $F_c = 250KHz$

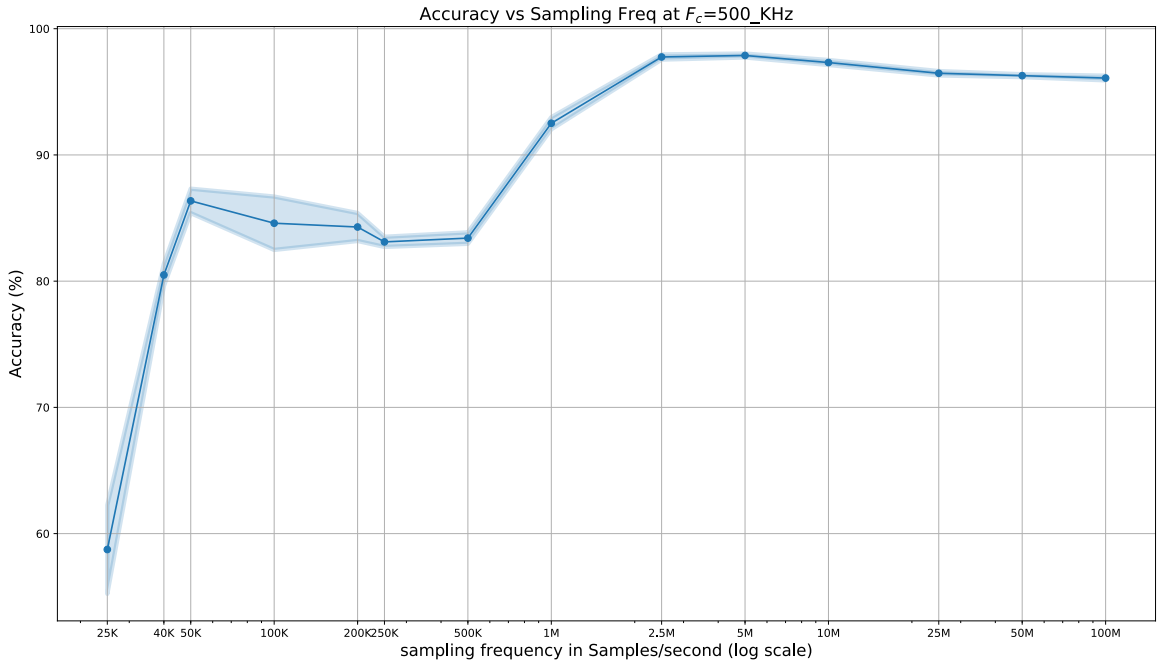


Figure 4.4: Classification accuracy at different sampling frequencies, when $F_c = 500KHz$

F_s **(Samples/s)**	**Accuracy**(%)	**STD**	$Lmin$	$\frac{F_s}{F_c}$	**m**	**CPU time(s)**
100M	96.09	0.185	22404	200	2.2404e+06	2633.95
50M	96.28	0.114	11202	100	560100	1368.96
25M	96.47	0.173	5601	50	140025	764.55
10M	97.32	0.197	2241	20	22410	349.73
5M	97.88	0.156	1121	10	5605	171.83
2.5M	97.76	0.206	561	5	1402.5	110.79
1M	92.51	0.48	225	2	225	46.85
500K	83.41	0.453	113	1	56.5	33.16
250K	83.11	0.402	57	0.5	14.25	17.45
200K	84.29	1.108	45	0.4	9	15.34
100K	84.59	2.115	23	0.2	2.3	12.27
50K	86.36	0.969	12	0.1	0.6	11.28
40K	80.49	0.876	9	0.08	0.36	11.1
25K	58.74	3.513	6	0.05	0.15	10.82

Table 4.2: Accuracy and Standard deviation at different F_s, when $F_c = 500KHz$

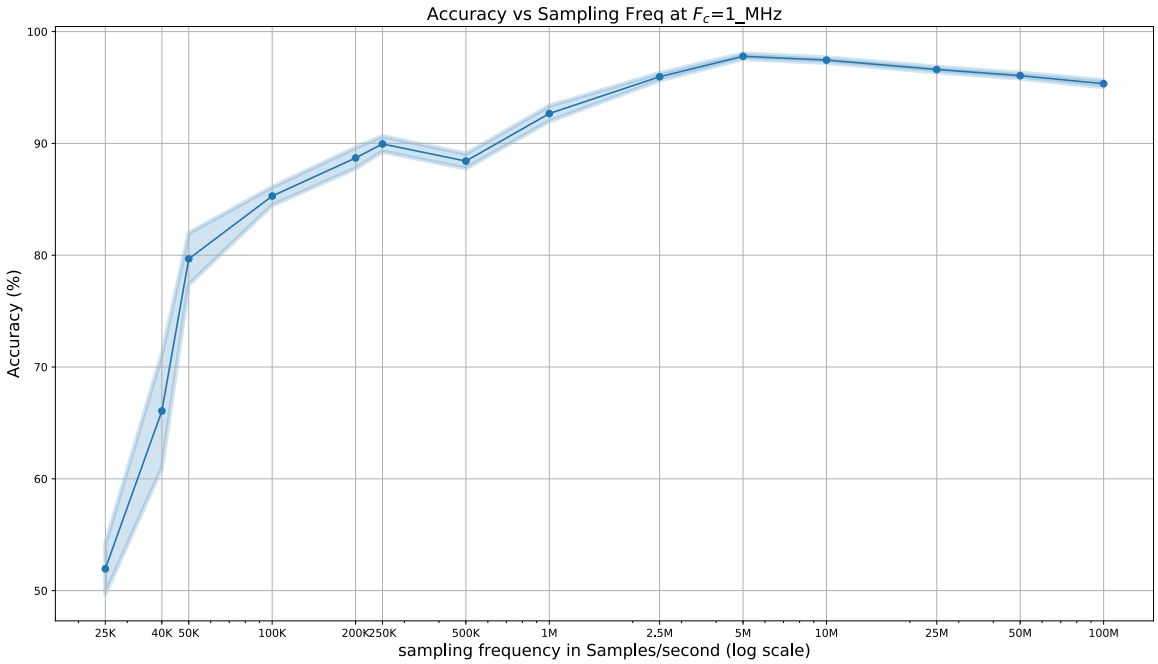


Figure 4.5: Classification accuracy at different sampling frequencies, when $F_c = 1MHz$

F_s **(Samples/s)**	**Accuracy**(%)	**STD**	$Lmin$	$\frac{F_s}{F_c}$	**m**	**CPU time(s)**
100M	95.34	0.307	11203	100	560150	1400.79
50M	96.06	0.252	5602	50	140050	742.05
25M	96.61	0.243	2801	25	35012.5	393.9
10M	97.45	0.234	1121	10	5605	180.39
5M	97.79	0.244	561	5	1402.5	98.48
2.5M	95.96	0.327	281	2.5	351.25	62.1
1M	92.67	0.691	113	1	56.5	28.24
500K	88.42	0.656	57	0.5	14.25	18.05
250K	89.95	0.69	29	0.25	3.625	15.21
200K	88.7	0.934	23	0.2	2.3	12.99
100K	85.29	0.821	12	0.1	0.6	12.91
50K	79.67	2.303	6	0.05	0.15	12.01
40K	66.07	5.098	5	0.04	0.1	11.88
25K	51.95	2.242	3	0.025	0.0375	11.17

Table 4.3: Accuracy and Standard deviation at different F_s, when $F_c = 1MHz$

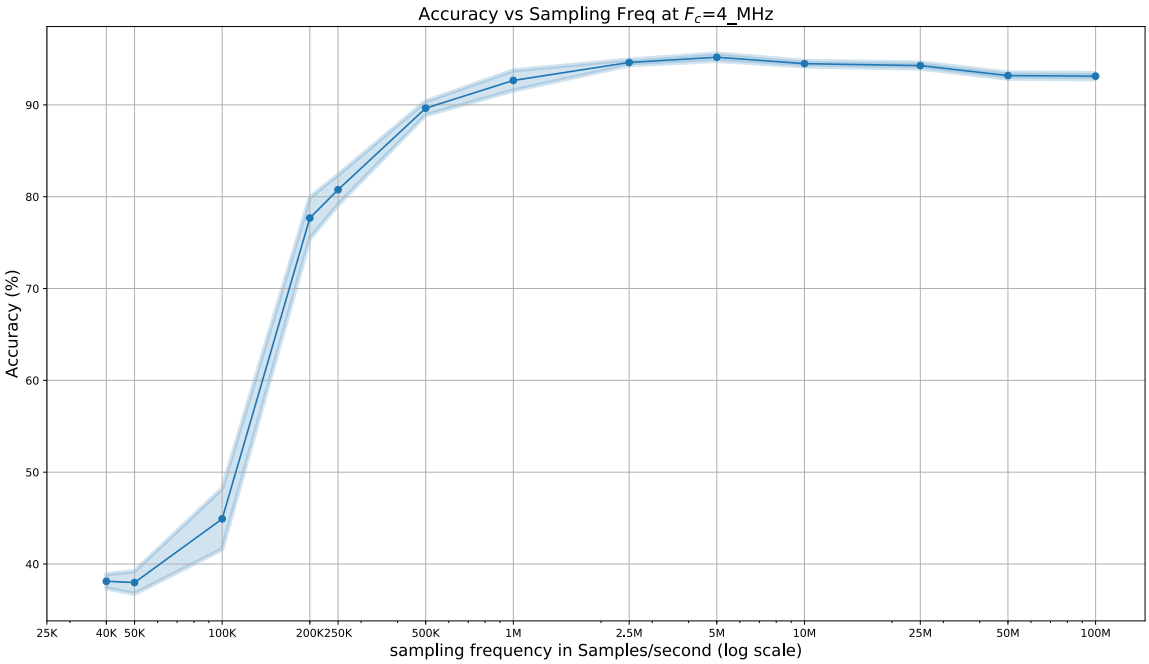


Figure 4.6: Classification accuracy at different sampling frequencies, when $F_c = 4MHz$

F_s **(Samples/s)**	**Accuracy**(%)	**STD**	$Lmin$	$\frac{F_s}{F_c}$	**m**	**CPU time(s)**
100M	93.13	0.337	2803	25	35037.5	415.75
50M	93.19	0.335	1402	12.5	8762.5	253.26
25M	94.28	0.326	701	6.25	2190.62	111.27
10M	94.49	0.297	281	2.5	351.25	50.09
5M	95.19	0.394	141	1.25	88.125	31.66
2.5M	94.62	0.304	71	0.625	22.1875	22.44
1M	92.66	1.095	29	0.25	3.625	16.31
500K	89.64	0.752	15	0.125	0.9375	13.33
250K	80.76	1.638	8	0.0625	0.25	12.78
200K	77.68	2.242	6	0.05	0.15	11.5
100K	44.92	3.313	3	0.025	0.0375	12.19
50K	37.98	1.236	2	0.0125	0.0125	11.7
40K	38.12	0.768	2	0.01	0.01	11.37
25K	NaN	NaN	NaN	NaN	NaN	NaN

Table 4.4: Accuracy and Standard deviation at different F_s, when $F_c = 4MHz$

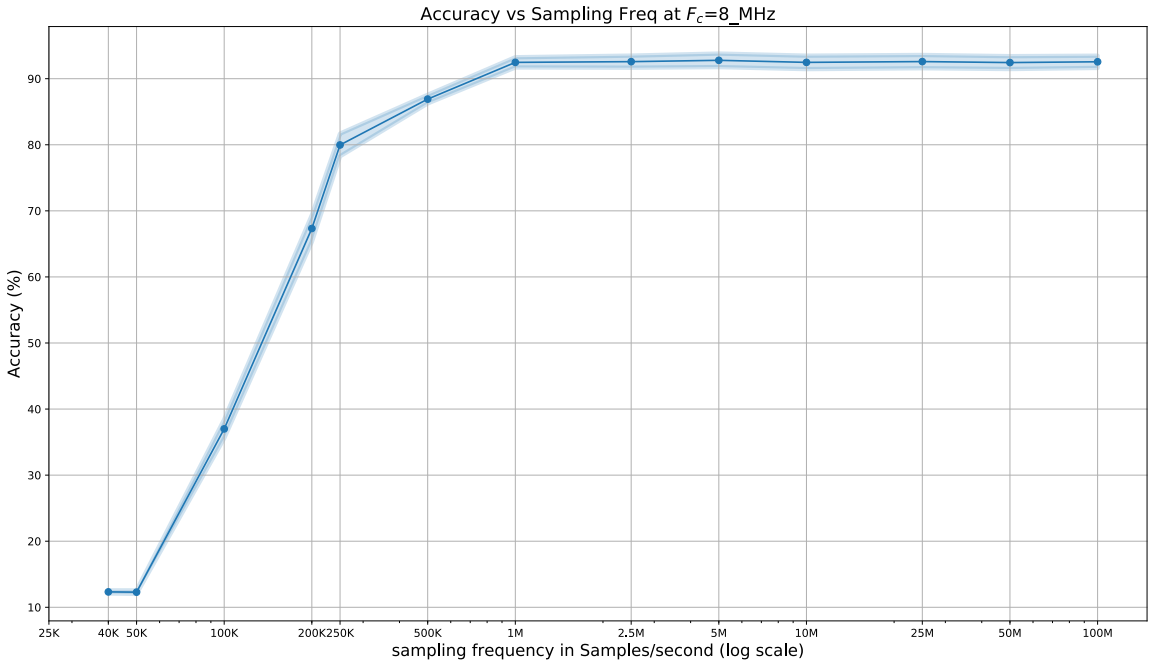


Figure 4.7: Classification accuracy at different sampling frequencies, when $F_c = 8MHz$

F_s **(Samples/s)**	**Accuracy**(%)	**STD**	$Lmin$	$\frac{F_s}{F_c}$	**m**	**CPU time(s)**
100M	92.56	0.925	1403	12.5	8768.75	234.59
50M	92.44	0.974	702	6.25	2193.75	125.97
25M	92.59	1.002	351	3.125	548.438	57.28
10M	92.47	1.016	141	1.25	88.125	35.08
5M	92.78	1.023	71	0.625	22.1875	23.09
2.5M	92.58	0.916	36	0.3125	5.625	17.12
1M	92.47	0.773	15	0.125	0.9375	12.51
500K	86.91	0.651	8	0.0625	0.25	12.22
250K	79.97	1.713	4	0.03125	0.0625	12.47
200K	67.33	2.017	3	0.025	0.0375	11.43
100K	37	1.418	2	0.0125	0.0125	12.27
50K	12.29	0.264	1	0.00625	0.003125	11.36
40K	12.33	0.237	1	0.005	0.0025	11.43
25K	NaN	NaN	NaN	NaN	NaN	NaN

Table 4.5: Accuracy and Standard deviation at different F_s, when $F_c = 8MHz$

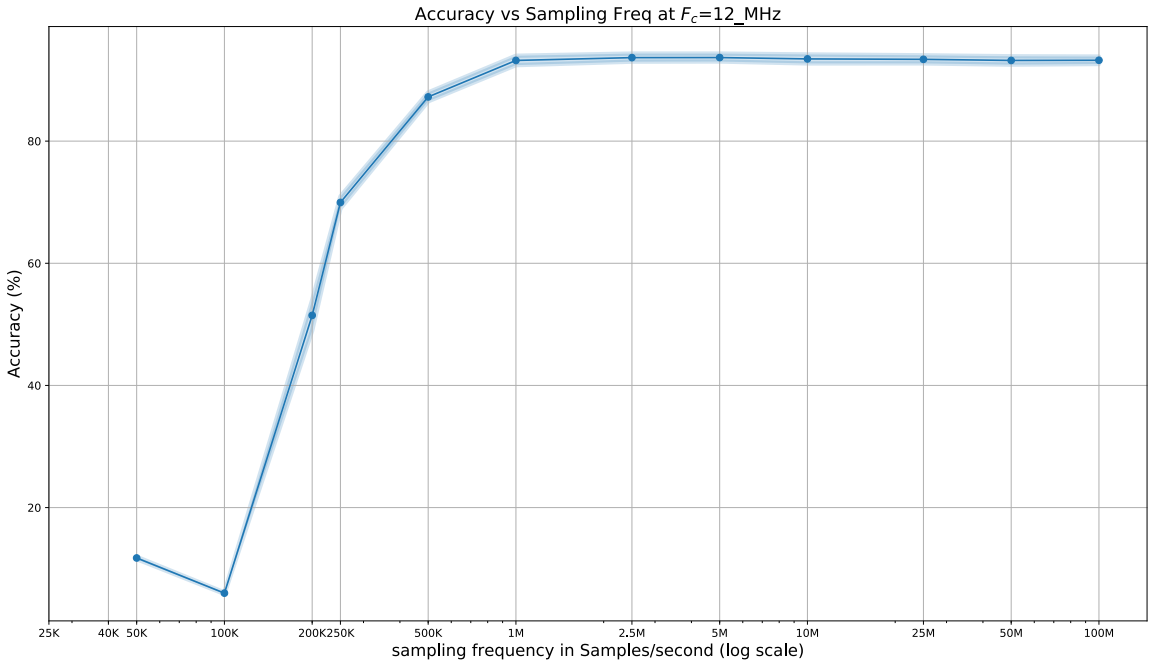


Figure 4.8: Classification accuracy at different sampling frequencies, when $F_c = 12MHz$

F_s **(Samples/s)**	**Accuracy**(%)	**STD**	$Lmin$	$\frac{F_s}{F_c}$	**m**	**CPU time(s)**
100M	93.24	0.609	937	8.33333	3904.17	156
50M	93.21	0.686	469	4.16667	977.083	77.82
25M	93.39	0.664	235	2.08333	244.792	41.7
10M	93.46	0.736	94	0.833333	39.1667	23.52
5M	93.68	0.651	47	0.416667	9.79167	17.04
2.5M	93.67	0.662	24	0.208333	2.5	14.78
1M	93.21	0.768	10	0.083333	0.416667	12.26
500K	87.24	0.724	5	0.041667	0.104167	11.65
250K	69.96	0.984	3	0.020833	0.03125	11.08
200K	51.47	2.495	2	0.016667	0.016667	11.41
100K	5.99	0.118	1	0.008333	0.004167	11.99
50K	11.75	0.183	1	0.004167	0.002083	10.98
40K	NaN	NaN	NaN	NaN	NaN	NaN
25K	NaN	NaN	NaN	NaN	NaN	NaN

Table 4.6: Accuracy and Standard deviation at different F_s, when $F_c = 12MHz$

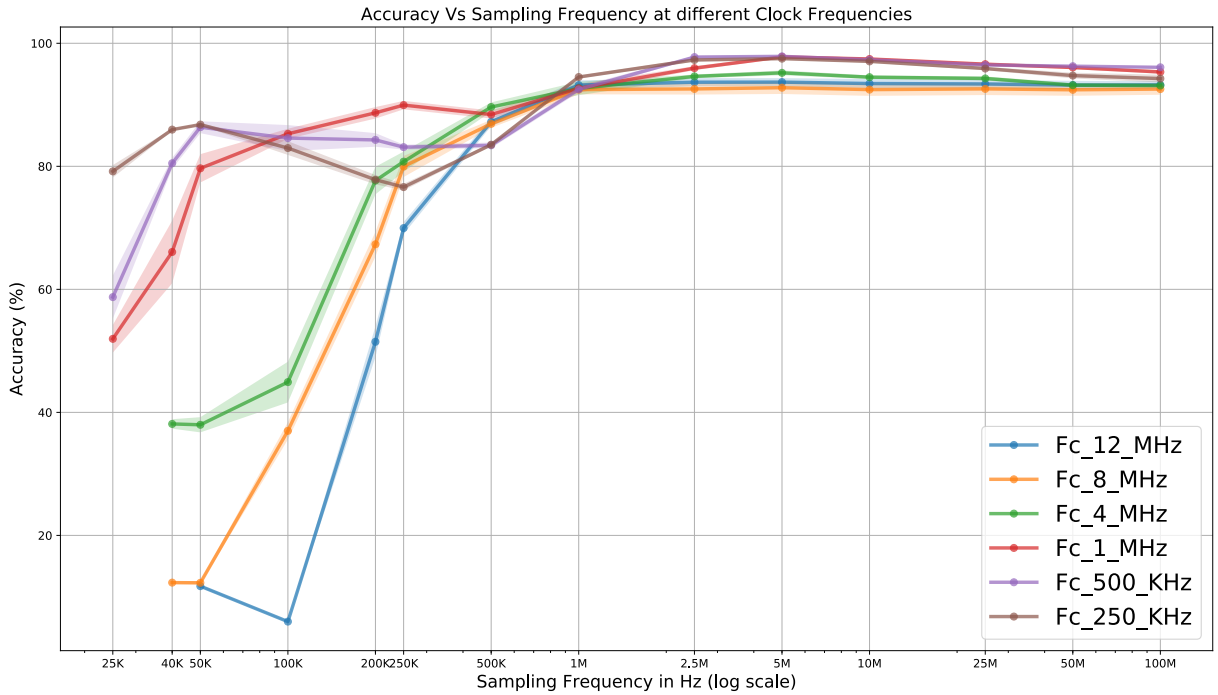


Figure 4.9: Classification Accuracy Vs Sampling Frequency at different Clock Frequencies

The plot 4.9 shows all the experimental results in one plot. The plot shows classification accuracy Vs sampling frequency at different clock frequencies. The shaded region around the curves is the standard deviation. The $F_c = 12MHz$ curve shows a corner or sudden downfall after $F_s = 500KS/s$. As, at this high clock frequency, the size of smallest segment is already very low, and if we sample it at again at a lower sampling rate, than we'll loose features and even loose the complete segment if we go ever lower. The $F_c = 8MHz$ curve shows a downfall after $F_s = 250KS/s$. The $F_c = 4MHz$ curve shows a downfall after $F_s = 200KS/s$. The $F_c = 1MHz$ curve shows a downfall after $F_s = 50KS/s$. It means that at this clock frequency, we have sufficient features in the segments or BB, that we can even go down to $F_s = 50KS/s$. This is an exciting result, as it clears the path of selecting MCUs having inbuilt ADC with lower sampling frequency, for designing the anomaly detector. The $F_c = 500KHz$ curve shows a downfall after $F_s = 40KS/s$. The $F_c = 250KHz$ curve shows a downfall after $F_s = 25KS/s$. This means that for systems running at a lower clock frequency and having smallest segment size sufficient enough for good amount of features, we can design the anomaly detector at a very low cost, as well as the size of the anomaly detector can be very small.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, we presented a study in power-trace analysis, to find the required sampling frequency, with respect to the classification accuracy, as a function of system architecture and clock frequency. We analyze the systems classification accuracy by running it at different clock speeds and varying the sampling frequency for power trace capture. We also show the effect of the sampling frequency on parameters such as classification time. For finding the sampling frequency we showed two approaches - analytical and empirical. Analytically, we found an intuitive formula which relates sampling frequency for power-trace analysis, system's clock frequency, smallest length of basic block BB and a scaling constant "m". We then presented a comprehensive empirical study of the same. We showed empirically, how the accuracy changed with sampling and clock frequency. We also showed other parameters like classification time as a function of F_s and F_c. We showed that for a good classification accuracy, the sampling frequency should be such that the smallest segment length should be large enough to have sufficient features. We also showed the possibility of selecting very low price MCU with low sampling frequency ADC for designing the anomaly detector. We strongly believe that our work provides a good starting point for more future research on non-intrusive run-time tracing of an operating system.

5.2 Future Work

The study presented in this thesis opens new areas for future research. The analytical formula given in the thesis, for finding the sampling frequency of power-traces, can be worked upon for including more variables and tuning. As mentioned in the thesis, this formula was based on the intuition which was fitting well with our dataset and results from an empirical approach. One can take advantage of the power of DTW averaging for generating the final training dataset. We used the normal averaging method for averaging all the segments from same class. DTW averaging can be an edge ahead in giving a better average of each class, because it is immune to warping in time-scale. Similarly DTW can be used for classification as it is a shape-based classification and has better results than Euclidean-distance shape-based classification. But, as DTW is computationally demanding, one can take advantage of FastDTW [44] instead of the normal DTW. The CPU and memory consumption of an anomaly-detector can also be looked upon as a function of sampling frequency and clock frequency. Any findings in this domain will help researchers and developers to develop a better and cheap anomaly-detector. If CPU and memory requirements are low, then one can use low cost MCU for developing anomaly-detector. This will be of huge impact in commercial domain. The same research can be conducted with different make and models of MCUs to find whether there is a dependence of this on sampling frequency. Experiments can be conducted for both RISC and CISC architectures. One can also add noise to the training and testing data signals and see the effect of noise on the accuracy with respect to sampling and clock frequency parameters. Feature-based classification methods can be used for classification as those can significantly decrease the classification processing time and makes the real-time processing more economical. As feature-based classification methods save only the important features from the power-trace instead of the complete trace, they can reduce CPU and memory consumption significantly.

References

[1] R. Alcock A. Nanopoulos and Y. Manolopoulos. *Information processing and technology*, chapter Feature-based Classification of Time-series Data. Nova Science Publishers, Inc., Commack, NY, USA:, 2001.

[2] Frances E. Allen. Control flow analysis. *SIGPLAN Not.*, 5(7):1–19, July 1970.

[3] D. Arora, S. Ravi, A. Raghunathan, and N. K. Jha. Secure embedded processing through hardware-assisted run-time monitoring. In *Design, Automation and Test in Europe*, pages 178–183 Vol. 1, March 2005.

[4] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31(3):606–660, May 2017.

[5] B. Bakshi and G. Stephanopoulos. Representation of process trendsiv. induction of real-time patterns from operating data for diagnosis and supervisory control. *Computers & Chemical Engineering*, 18(4):303–332, 1994.

[6] R. Bellman and R. Kalaba. On adaptive control processes. *International Journal of Advances in Computer Science and Technolog*, 4(2):1–9, 1959. IRE Transactions on Automatic Control.

[7] Nicholas Carlini, Antonio Barresi, Mathias Payer, David Wagner, and Thomas R. Gross. Control-flow bending: On the effectiveness of control-flow integrity. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 161–176, Washington, D.C., 2015. USENIX Association.

[8] Silvio Cesare and Yang Xiang. Classification of malware using structured control flow. In *Proceedings of the Eighth Australasian Symposium on Parallel and Distributed Com-*

puting - Volume 107, AusPDC '10, pages 61–70, Darlinghurst, Australia, Australia, 2010. Australian Computer Society, Inc.

[9] Noureddine Chabini and Marilyn Wolf. Reordering the assembly instructions in basic blocks to reduce switching activities on the instruction bus. *IET Computers & Digital Techniques*, 5(5):386–392, 2011.

[10] S. S. Clark, B. Ransford, A. Rahmati, S. Guineau, J. Sorber, K. Fu, and W. Xu. Wattsupdoc: Power side channels to nonintrusively discover untargeted malware on embedded medical devices. In *USENIX Workshop on Health Information Technologies*. USENIX, 2013.

[11] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, January 1967.

[12] Yuwei Cui, Chetan Surpur, Subutai Ahmad, and Jeff Hawkins. Continuous online sequence learning with an unsupervised neural network model. *CoRR*, abs/1512.05463, 2015.

[13] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn Keogh. Querying and mining of time series data: Experimental comparison of representations and distance measures. *Proc. VLDB Endow.*, 1(2):1542–1552, August 2008.

[14] Pedro Domingos. A few useful things to know about machine learning. *Commun. ACM*, 55(10):78–87, October 2012.

[15] Mouser Electronics. Choosing the optimal low power mcu, 2016.

[16] W. Euachongprasit and C. A. Ratanamahatana. Efficient multimedia time series data retrieval under uniform scaling and normalisation. In *ECIR 2008: Advances in Information Retrieval*, pages 506–513, Berlin Heidelberg, 2008. European Conference on Information Retrieval, Springer.

[17] Ben D. Fulcher, Max A. Little, and Nick S. Jones. Highly comparative time-series analysis: the empirical structure of time series and their methods. 10(83), 2013.

[18] F. Galton. Vox populi. *Nature*, 75(1949):7, 1907.

[19] Toni Giorgino. Computing and visualizing dynamic time warping alignments in r: The dtw package. *Journal of Statistical Software, Articles*, 31(7):1–24, 2009.

[20] Jianping Gou, Zhang Yi, Lan Du, and Taisong Xiong. A local mean-based k-nearest centroid neighbor classifier. *Computer Journal*, 55(9):1058–1071, 9 2012.

[21] J. Gu and X. Jin. A simple approximation for dynamic time warping search in large time series database. In *Intelligent Data Engineering and Automated Learning IDEAL 2006*, pages 841–848, Berlin Heidelberg, 2006. International Conference on Intelligent Data Engineering and Automated Learning, Springer.

[22] Texas Instruments. Cmos power consumption and cpd calculation, 1997.

[23] A. W. Moore K. Deng and M. C. Nechyba. Learning to recognize time series: combining arma models with memory-based learning. pages 246–251. Computational Intelligence in Robotics and Automation, 1997 IEEE International Symposium on, jul 1997.

[24] T. Kahveci and A. Singh. Variable length queries for time series data. In *Proceedings 17th International Conference on Data Engineering*, pages 273–282, Heidelberg, Germany, 2001. IEEE.

[25] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 388–397, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[26] S. B. Kotsiantis. Supervised machine learning: A review of classification techniques. In *Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*, pages 3–24, Amsterdam, The Netherlands, 2007. Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering, IOS Press,.

[27] Alexios Kotsifakos and Panagiotis Papapetrou. Model-based time series classification. In Hendrik Blockeel, Matthijs van Leeuwen, and Veronica Vinciotti, editors, *Advances in Intelligent Data Analysis XIII*, pages 179–191, Cham, 2014. Springer International Publishing.

[28] K. Lamichhane, C. Moreno, and S. Fischmeister. Non-intrusive program tracing of non-preemptive multitasking systems using power consumption. In *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1147–1150, Dresden, Germany, March 2018.

[29] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *In Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 2–11. ACM Press, 2003.

[30] Weiqiang Lin and et al. An overview of temporal data mining.

[31] Yannan Liu, Lingxiao Wei, Zhe Zhou, Kehuan Zhang, Wenyuan Xu, and Qiang Xu. On code execution tracking via power side-channel. page 1019. ACM Conference on Computer and Communications Security, ACM, 2016.

[32] Yannan Liu, Lingxiao Wei, Zhe Zhou, Kehuan Zhang, Wenyuan Xu, and Qiang Xu. On code execution tracking via power side-channel. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1019–1031, New York, NY, USA, 2016. ACM.

[33] K. Markantonakis M. Msgna and K. Mayes. The b-side of side channel leakage: Control flow security in embedded systems. In *International Conference on Security and Privacy in Communication Systems*, pages 288–304, The address of the publisher, 2013. Springer.

[34] C. Moreno and S. Fischmeister. Non-intrusive runtime monitoring through power consumption: A signals and system analysis approach to reconstruct the trace. In *International Conference on Runtime Verification*, page 268, 2016.

[35] Carlos Moreno and Sebastian Fischmeister. Non-intrusive runtime monitoring through power consumption to enforce safety and security properties in embedded. *Formal Methods in Software Design (FMSD)*, 2017.

[36] Carlos Moreno, Sean Kauffman, and Sebastian Fischmeister. Efficient program tracing and monitoring through power consumption – with a little help from the compiler. In *Proc. of Design, Automation, and Test (DATE)*, Dresden, Germany, 2016.

[37] Kevin P. Murphy. *Machine Learning A Probabilistic Perspective*. MIT, 2012.

[38] H. Nyquist. Certain topics in telegraph transmission theory. *Transactions of the American Institute of Electrical Engineers*, 47(2):617–644, April 1928.

[39] The organization. *Non-intrusive program tracing and debugging of deployed embedded systems through side-channel analysis*, volume 48 of *5*, Seattle, Washington, USA, 6 2013. ACM SIGPLAN Not. p. 77.

[40] François Petitjean, Germain Forestier, Geoffrey I. Webb, Ann E. Nicholson, Yanping Chen, and Eamonn Keogh. Faster and more accurate classification of time series by exploiting a novel dynamic time warping averaging algorithm. *Knowl. Inf. Syst.*, 47(1):1–26, April 2016.

[41] T. Popp, S. Mangard, and E. Oswald. Power analysis attacks and countermeasures. *IEEE Design Test of Computers*, 24(6):535–543, Nov 2007.

[42] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 262–270, New York, NY, USA, 2012. ACM.

[43] Chotirat Ann Ratanamahatana and Eamonn Keogh. Making time-series classification more accurate using learned constraints. In *In proc. of SDM Intl Conf*, pages 11–22, 2004.

[44] Stan Salvador and Philip Chan. Fastdtw: Toward accurate dynamic time warping in linear time and space. In *KDD workshop on mining temporal and sequential data*. Citeseer, 2004.

[45] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, July 1959.

[46] Otto H. Schmitt. A thermionic trigger. *Journal of Scientific Instruments*, pages 24–26, 1 1938.

[47] C. E. Shannon. Communication in the presence of noise. *Proc. Institute of Radio Engineers*, 37(1):10–21, 1949.

[48] C. Paar T. Eisenbarth and B. Weghenkel. Building a side channel based disassembler. *Transactions on Computational Science X*, pages 78–99, 7 2010. An optional note.

[49] A. Singh T. Kahveci and A. Gurel. Similarity searching for multi-attribute sequences. In *14th International Conference on Scientific and Statistical Database Management*, pages 175–184, Edinburgh, Scotland, UK, 2002. IEEE.

[50] Robert Tibshirani, Trevor Hastie, Balasubramanian Narasimhan, and Chu Gilbert. Diagnosis of multiple cancer types by shrunken centroids of gene expression. volume 99. The National Academy of Sciences, 2002.

[51] Xiaopeng Xi, Eamonn Keogh, Christian Shelton, Li Wei, and Chotirat Ann Ratanamahatana. Fast time series classification using numerosity reduction. In *In ICML06*, pages 1033–1040, 2006.

[52] Zhengzheng Xing, Jian Pei, and Eamonn Keogh. A brief survey on sequence classification. *SIGKDD Explor. Newsl.*, 12(1):40–48, November 2010.