DEVELOPING A SMART AND LOW COST DEVICE FOR MACHINING VIBRATION ANALYSIS

A Dissertation
Presented to
The Academic Faculty

By

Pierrick Rauby

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Mechanical Engineering in the
School of George W. Woodruff School of Mechanical Engineering

Georgia Institute of Technology

August 2018

DEVELOPING A SMART AND LOW COST DEVICE FOR MACHINING VIBRATION ANALYSIS

Approved by:

Dr. Kurfess, Advisor
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Saldana
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Liang
School of Mechanical Engineering
Georgia Institute of Technology

Date Approved: July 20, 2018

In a world of change, the learners shall inherit the earth, while the learned shall find themselves perfectly suited for a world that no longer exists.

Eric Hoffer

ACKNOWLEDGEMENTS

I would like to especially thank Professor Thomas Kurfess for his guidance during this work. He consistently helped me on the research with advice that allowed me to take a step back when I was going the wrong way.

I also would like to thank Professor Steven Y. Liang and Dr. Christopher J. Saldana for being part of the reading committee of this work despite their very busy schedule.

My thanks also go to the administrative staff of the Office of International Education and the School of Mechanical Engineering, especially Mrs. Glenda Johnson whose explanation tremendously helped me with paperwork.

Finally, I must express my gratitude to my parents and family for their trans-Atlantic support throughout this year. Thank you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Internet of Thing (IoT) is receiving an enormous attention especially when it comes to monitor machining operations. However, current technology must continue to evolve in order to reduce cost and to improve data analytics[1]. More importantly, IoT devices often raise security concerns, as they transfer a considerable amount of data to the cloud. Simultaneously, the computational power of embedded platforms has increased, giving the ability to process data locally; thus, edge computing is able to reduce the security problem as they minimize the quantity of information transferred to the cloud. Therefore, these problems can be addressed by developing a truly smart low-cost device that takes advantage of fog computing as opposed to cloud computing.

Frameworks have been developed to demonstrate the capability to remotely monitor machine health using cloud computing, the objective of this thesis is to associate those frameworks to the computational power of low-cost embedded platforms to process data locally and in real-time.

For this work a BeagleBone Black is used. It is powered by an AM335x ARM Cortex-A8 processor that runs at 1GHz. This computer is associated with an analog accelerometer through its Analog to Digital Converter. The system is monitoring vibrations on a bandsaw, as it is running Linux it does not have deterministic-sampling capabilities; therefore, the Industrial I/O subsystem is used to enable hardware interrupts on the Linux Kernel space. The vibrations generated by the cutting of different materials are recorded and used to train a machine learning algorithm on an external computer. Training will use a Kernel Support Vector Machine algorithm. Once the algorithms are trained they are will be implemented locally on the BeagleBone Black so that the analytics of the data are done at the ”edge”. The final goal is to be able to determine the nature of the material that is being cut by the bandsaw.

[1]McKinsey Global Institute: Unlocking the potential of the Internet of Things, June 2015

CHAPTER 1
INTRODUCTION

The 4th industrial revolution is underway for years thanks to the development of Cyber-Physical Systems (CPS). It was named Industry 4.0 by the German research union for economy and science in 2011 when it started a 400 million euro research program to maintain the German industry competitivity. Industry 4.0 includes many computer-related technologies such as additive manufacturing (AM), cloud computing (CC), machine learning (ML) or Internet of Things (IoT), aiming to connect all parts, tools and productions systems together. This allows a collection of large amount of data, to carry out analysis of the production process and to be able to improve it.

However, with the adoption of Industry 4.0 technologies, we are facing new issues especially in the area of security. For example, it is not desirable to stream all production data in some industries that are sensitive to information security, such as industries related to national defense. Moreover, streaming data from every possible source can lead to bandwidth issues. Hence, the cloud computing strategies can be opposed to the need of real-time and decentralised decision making concepts promoted by the Industry 4.0.

Some studies have shown the possibility of using computer on a local network instead of sending data to the cloud. However, there is little wirk currently few work on the use of embedded microprocessor platforms to process data at the edge. This presents the advantage of significantly reducing the amount of data transferred to the cloud, while simultaneously increasing security, reducing cloud storage space, and reducing transmission bandwidth [1]. Furthermore, there are currently few studies on the used of powerful embedded microprocessor platforms for data acquisition and processing. Typically, those two task are performed by different chips.

Based on this observation, this work tries to implement a real-time data acquisition

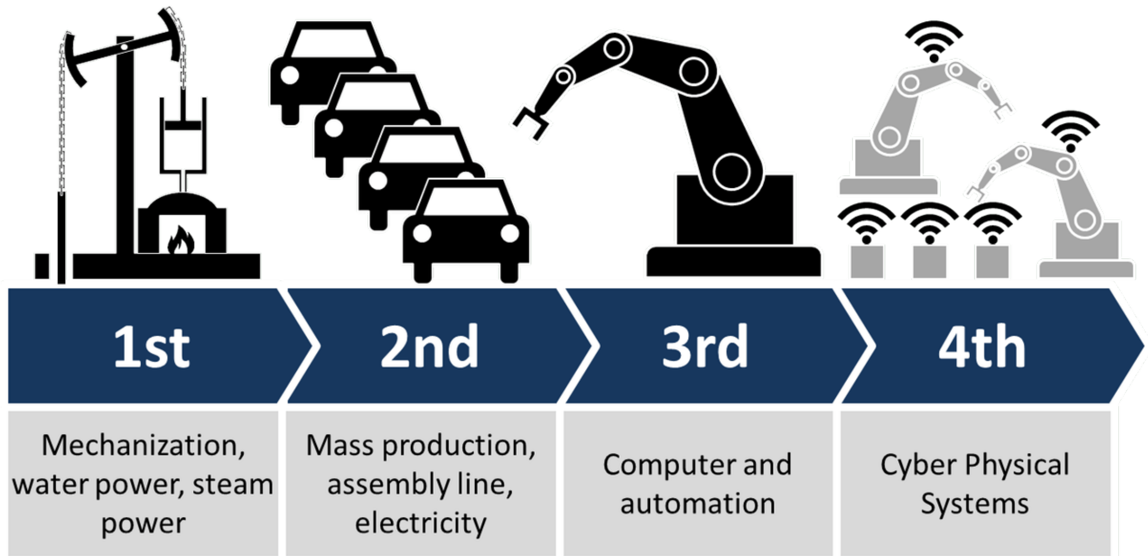


Figure 1.1: The 4 industrial revolutions. [2]

and processing solution on a BeagleBone Black micro-computer. The solution leads to a decentralized, more private, secure data management which better adresses the Industry 4.0 concerns.

First, the previous work on this topic is introduced. Then the realtime data acquisition on a linux based microprocessor is discussed. Next, the experimental setup and the trainning of a machine learning algorithm is presented. Finally, the system is tested in real conditions and the results are analyzed.

CHAPTER 2

RESEARCH BACKGROUND

When it comes to producing a mechanical part from raw material, various techniques are used; in most cases, machining is employed at some point in the process. With the development of low cost sensors and the embedded platforms, automatic machine monitoring is becoming a major axis of performance improvement for manufacturers. This chapter presents a brief review of the state of the art in terms of automatic machine monitoring. First the different sensors and data acquisition methods are presented, then a brief introduction to machine learning common algorithms is performed. Finally, the most common IoT protocols for data transfer are introduced.

2.1 Machine monitoring

In order to increase quality and productivity different sensing methods are widely used. They can be classified into direct and indirect methods [3]. Direct methods such as optical and electrical enable direct measurement of the physical characteristic that need to be accessed. This results in a high accuracy but it often requires stopping the process during the measurement which is not suitable for online production. With indirect methods, such as acoustic emission measurement, vibration or cutting force, the physical characteristic is determined through the measurement of other values such as current, force, et al. which can be acquired without interrupting the production process; thus they are more interesting for Realtime application.

2.1.1 Sensing methods used in previous studies

Direct sensing methods

Optical methods are based on different components, as in Figure 2.1: a source of illumination to enhance the quality of the image, a camera and some lens that feed the computer with data, a computer to process the data and a monitor in order to display the result of the process. Siddhpura et al. [3] states that these methods seems to be promising because of the high accuracy and flexibility, but they can only be used between production cycles which is not exactly a Realtime technique.

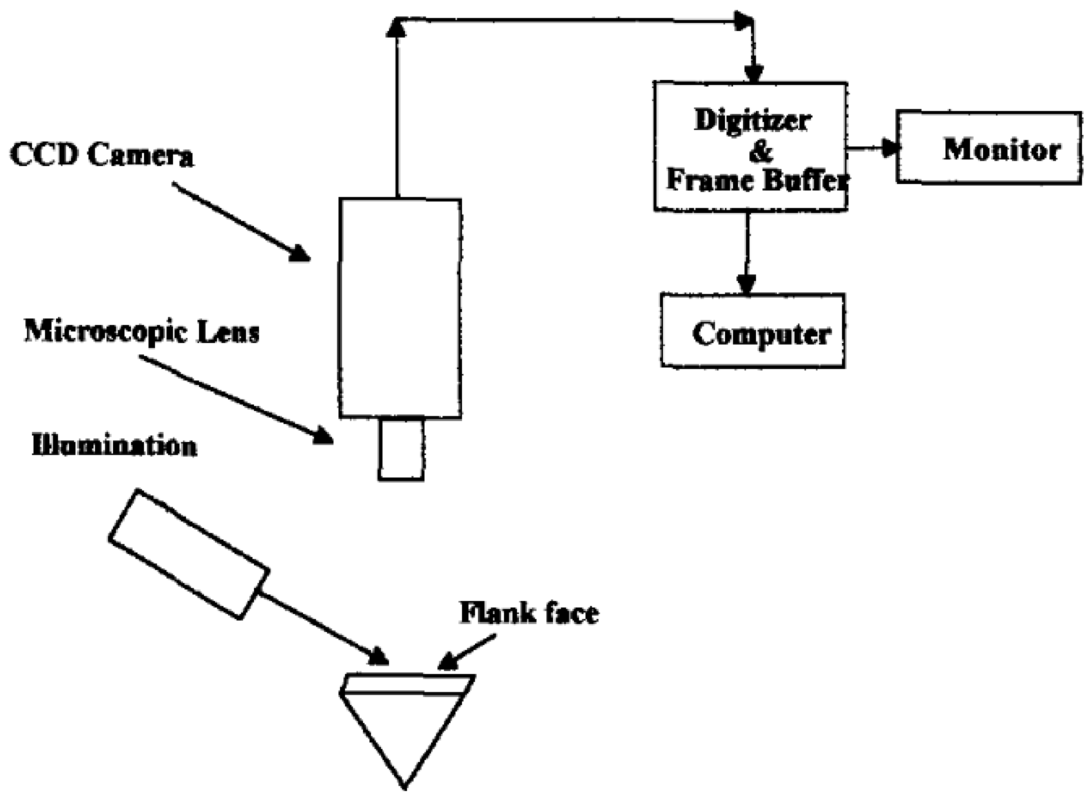


Figure 2.1: The different components for optical method in tool flank application. [4]

Electrical methods are specially used for tool wear detection. N. H. Cook [5] discussed these techniques; the electrical resistance at the contact between the tool and the part depends on the tool's wear; so it is possible to estimate the wear condition of the tool. Other electrical methods use resistor films applied to the tool. However, the variation of the cutting force can introduce bias in the resistance interpretation, these methods are not easily applicable in the industry.

Others direct methods are such as radioactive techniques or analysis of the wear particles but they are slow and not applicable to the industry.

Indirect methods

Cutting force can be measured in order to monitor the physical characteristic that needs to be determined, as an example, the force components vary as the tool wears. However, other parameters such as work harnessed and cutting parameters, also have an influence on the cutting force, which can introduce uncertitude in the measurement, in the case of tool wear prediction Dimla E. [6] discussed the importance of monitoring the static cutting force but also the dynamic cutting force in order to have an indication of the system's fluctuations. Nevertheless, this technique has been widely used by researchers, as Siddhpura et al. [7] presents in Figure 2.2.

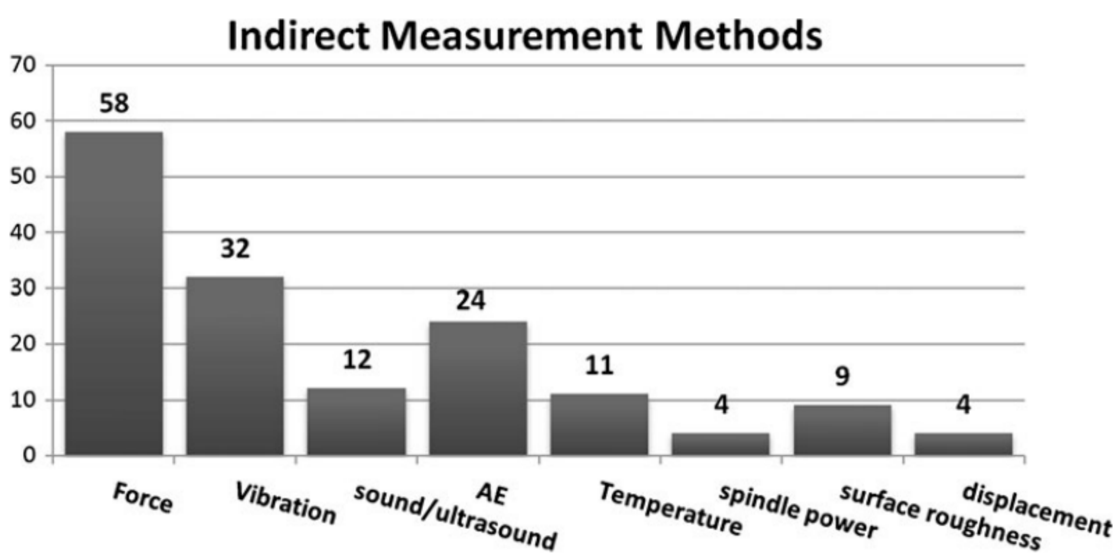


Figure 2.2: Number of publications using indirect measurement methods. [7]

Sound is recorded and the variation of low frequencies can be analyzed to have information on the cut. Again, this technique is widely used to estimate the tool wear stage; as Maropoulos, P.G. and Alamin, B. [8] explain, the sound spectra is a results of the rubbing action between the tool and the workpiece. When the flank wear enters the final stage, the sound pressure level drops off.

Variation of power input in the machine gives valuable information on the cutting process, in any machining operation electric energy is used to remove material from the workpiece. By subtracting the idle power of the machine from the measured power the power consumption for the operation can be determined. This method presents the advantage of being simple to implement; however, in some applications, it is less sensitive than other direct methods as sound or force monitoring[7].

Vibrations can be recorded using a simple accelerometer which detects the rub between the tool, the chip and the workpiece; then the signal contains information about the cut. In the case of tool wear, the amplitude of the vibration at frequencies in the range from 4 to 8 kHz increases with the cutting-edge wear. This technique has been used to implement online monitoring application by Pandit, S. M. [9]. Dan and Mathew [10] considered that, thanks to the progress in vibration measurement, this method would become more practical and cost effective.

Two categories of monitoring techniques have been discussed above; unlike direct monitoring, indirect monitoring techniques are applicable to on-line monitoring. Multiple studies have demonstrated that cutting forces, sound emissions, variation of power consumption and vibration are efficient to follow tool wear and to predict its breakage.

Whatever monitoring technique is employed, some computing power is needed after the sensor, to convert the data into human-readable information. The development of processor technology has made accessible a wide range of boards for embedded application and the most well-known are presented below.

2.1.2 Available IoT platforms

The raw data from the sensor needs to be processed before being transmitted to the user; therefore, either a microcontroller or a microprocessor can be used. Microcontrollers are usually less powerful but also less expensive than microprocessors, which can be seen as small computers.

Microcontrollers (MCU)

Microcontrollers can only run a single control loop; the absence of an operating system on those chips disables multiple threads. Since they can only achieve a single task, the relation between the input of the process and the output must perfectly understood; this enable designers to reduce the processing power of the board and the cost. The general architecture of a microcontroller, as in Figure 2.3, contains:

- In/Out interfaces
- timer
- RAM memory for data storage (volatile)
- ROM memory to store the programs
- Central Process Unit (CPU)
- Analogue to Digital Convert (ADC) is also present on most of the microcontrollers

The timer clock speed is usually in range from a few MHz to more than a hundred MHz; thus microcontrollers are not suitable for processes that require a high computational power and should only be used for simple tasks. The most well-known microcontrollers are certainly the Arduino family, but other boards such as Teensys, Particles and the ESP32 are getting interest from the developers community. In the following, the different characteristics of the Arduino Uno, the Teensy 3.2 and the Particle Photon are presented.

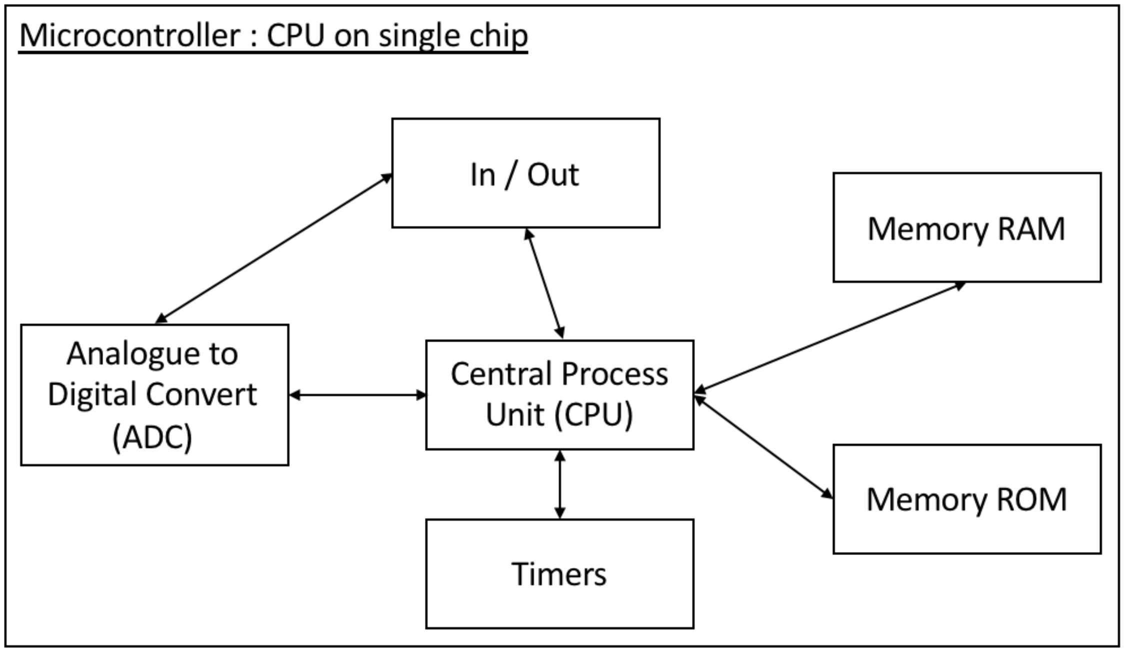


Figure 2.3: Architecture of a microcontroller.

Arduino Uno is the most famous board from Arduino®; it is an entry level microcontroller which has a very important community of users, thus it is very well documented. The Arduino Uno is based on the ATmega328P chip. It has 14 digital In/Out pins, 6 analog inputs an Inter-Integrated Circuit (I2C) bus and a Serial Peripheral Interface (SPI). The memory consists of a 2KB SRAM, 1KB EEPROM and a flash memory of 32KB, all from the ATmega328P. The clock speed is given by a 16MHz quartz crystal. The dimensions of the board are 68.6mm by 53.4mm for a 25g weight. It costs around $35 [11].

Teensy 3.2 is a USB-based microcontroller development system distributed by the SME PJRC®. As with the Arduino, the code is compiled externally and then transferred onto the board using the USB port. For this board, the code can be written either in C code or in Arduino code (.ino). It has a 64KB RAM, 2KB EEPROM which enables the use of Teensy for more advanced projects than the Arduino Uno. The flash memory is also more important, with 256KB available in the most recent version of the board. Regarding the In/Out capabilities, the Teensy 3.2 has 34 digital In/Out pins, 21 analog inputs, a SPI and a

I2C bus. The board is powered by a 32-bit ARM Cortex-M4 running at 72MHz. The board size is 35mm by 18mm (weight 15g) and it costs around $20 [12].

Particule Photon has been developed for Internet of Things projects with a Cypress BCM43362 WIFI chip. The board is powered by a 120MHz ARM Cortex-M3, it has 128KB of RAM, 16KB or 64KB of EEPROM (depending on version) and 1MB of flash memory. The connectivity with external sensors is ensured by 18 general In/Out pins, 8 analog pins, 2 SPI and 1 I2C. The dimensions of the board are 36.58mm by 20.32mm for a weight of 5g, and it costs around $20 [13].

ESP32 is commercialized by Espressif. It is powered by a Tensilica Xtensa 32-bit LX6 microprocessor with 1 or 2 cores depending on the version and running at 240Mhz. The board has also an ultra-low power co-processor that permits ADC conversions and some computing tasks while in deep-sleep mode. As for the Photon, the ESP32 provides Internet of Things capabilities with WIFI 802.11 b/g/n, Bluetooth and Bluetooth Low Energy. The memory consists in 448KB of ROM, 520KB of SRAM and the flash memory is either 2MB or 4MB depending on the versions. The connectivity is ensured by 34GPIO, 18 ADC channels, 4 SPI pins and 2 I2C pins which permits a wide range of sensors to be connected to this board. The ESP32 dimensions are 55.3mm by 28mm for a weight of 9.6g and it costs around $15 [14].

The microcontrollers presented above are not the only ones available on the market. However, their characteristics, as reprensented in table 2.1 depict well the wide range of options possible when it comes to choose a board for an application: from the first development board such as the Arduino to more advanced board such as the ESP32 it is important to specify the need before choosing the board for an application. Furthermore, choice can be made to use more powerful boards such as microprocessors; this other kind of board is introduced in the following section.

Table 2.1: Comparison between different microcontrollers

Characteristic	Arduino Uno	Tensy 3.2	Particle Photon	ESP32
Processor	ATmega328P	ARM Cortex-M4	ARM Cortex-M3	Tensilica Xtensa
Frequency	16MHz	72MHz	120MHz	240MHz
GPIOs	14	34	18	34
ADCs	6	21	8	18
SPI/I2C	yes	yes	yes	yes
WIFI/Bluetooth	on shield	No	yes/no	yes
RAM	2KB	64KB	128KB	520KB
EEPROM	1KB	2KB	16KB or 64 KB	448KB
Flash Memory	32KB	256KB	1MB	2MB or 4MB
Dimensions(mm)	68.6 by 53.4	35 by 18	36.6 by 20.3	55 by 28mm
Weight(g)	25	15	5	10
Price	35	20	20	15

Microprocessors (MPU)

Microprocessors can be seen as mini-computers, they contain most of computer's usual components:

- Central Process Units (CPU) which is the part of the chip that is responsible of all the computing tasks.
- Peripheral Interface
- Timers
- Memory such as RAM and ROM
- Inputs and Outputs chips

However, it is important to notice that those functions are not contained of a single chip, as shown in Figure 2.4, all these components can be contained in a single board but they are not contained in a the same chip. Unlike microcontrollers, microprocessors run operating systems; usually a specific version of Linux or Android is provided and sustained by the boar or chip distributor. In the following the most well-known microprocessors are presented: The Raspberry Pi 3 B+ and the BeagleBone Black (wireless version).

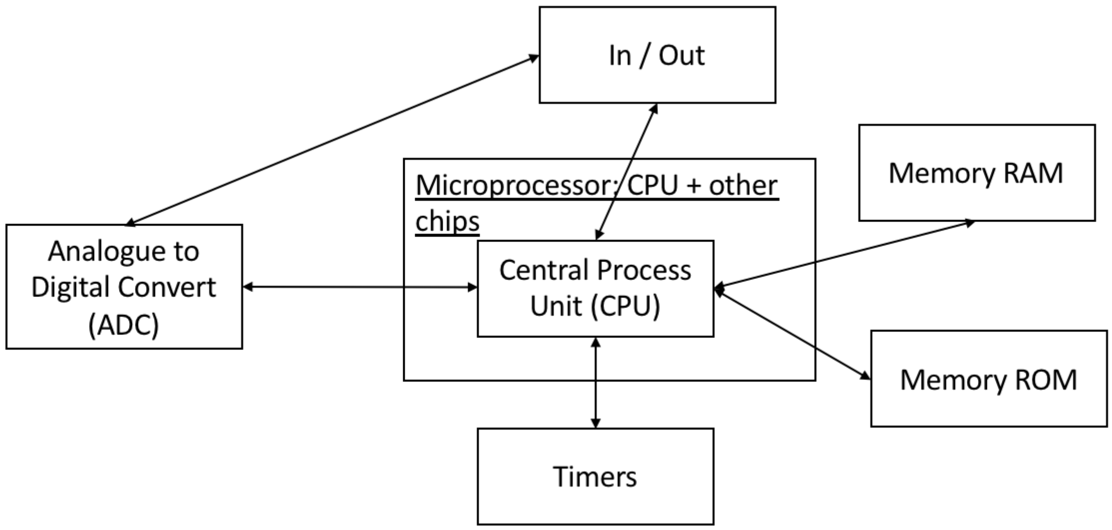


Figure 2.4: Architecture of a microprocessor.

Raspberry is a mono-board micro-computer distributed by the Raspberry foundation. It is powered by a 64-bit quad-core processor Broadcom BCM2837B0 ARM Cortex-A53 running a 1.4GHz. The memory consists in 1GB LPDDR2 SDRAM. Regarding the connectivity, in addition to 40 In/Out pins and 2 USB ports, it has, 2.4GHz and 5GHz WIFI capabilities, Bluetooth and Bluetooth Low Energy. This board has also an SPI, an I2C bus, a full-size HDMI port, and a CSI&DSI inputs to connect camera&touchscreen. However, there is no Analog to Digital Converter in the current version. Thus, the Raspberry Pi 3 needs some add-ons to be able to interact with analog sensors. The dimensions of the board are 86.9mm by 58.5mm for a weight of 41g. It costs around $35 [15].

BeagleBone Black is a low-cost community supported development platform distributed by the BeagleBoard foundation, project is totally open source, which means that all the schematics and components of the board can be found on line and bought separately. It is powered by the TI-am3358 ARM Cortex-A8 processor running at 1GHz, but it also has two Process Realtime Units (PRU) microcontrollers, each running at 200MHz whose role is to manage deterministic tasks, and which are totally integrated in the TI-am3358 chip. Connectivity is ensured by 44 In/Out pins, one high speed USB port and 8 analog inputs. The new version of the BeagleBone Black has seen its Ethernet port replaced by a 802.11 b/g/n 2.4 GHz WIFI with also Bluetooth 4.1 and Bluetooth Low Energy. The memory of the board consists in 512MB of DDR3L DRAM and 4GB flash memory, additionally the SD card port can be used to store data. The board dimensions are 86.4mm by 53.3mm for a weight of 35g. This board costs around $55 [16].

The two microprocessors presented above illustrate well two different way to use microprocessors; the BeagleBone Black, thanks to its numerous In/Out pins and its Analog to Digital Converter, is more suitable for sensor and data acquisition applications. The ti-am3358 chip also provides very interesting computing power, and the Process Realtime Units enable high speed and deterministic data acquisition. On the other hand, the Raspberry Pi interfaces, as the HDMI port, are more suited to multimedia projects; the same goes for its quad core processor, which provides more powerful graphics processing.

2.1.3 Comparison

In the two previous sections are presented the microcontrollers and the microprocessors. The first ones are less expensive but, as they do not run an Operating System, they must be dedicated to a single task, it is not possible for them to manage threads. One the other hand, their behaviors are totally deterministic. Thus, microcontrollers are very suitable for Realtime applications; however, the power of their Central Processing Unit does not permit high level computation.

In contrast, microprocessors are more powerful, which enables to run some machine learning algorithms on those boards. The use of an operating system on those chips enables the use of threads and so to have multiple applications running at the same time. However, it results in a loss of the deterministic behavior of those applications; the Operating System can ”jump” from one application to the other, which is an important drawback when it comes to acquiring data at a high and precise frequency. It is not doable by running an application on the user space of the operating system of a microprocessor.

2.2 Machine Learning (ML)

Industry 4.0 transforms the way we are producing parts. Machine Learning, as a subfield of artificial intelligence plays a very important role in this transformation. As machines are increasingly connected to sensors and the cloud, a very important amount of data is generated, it can be used to train machine learning algorithms. Those ”learning” techniques are useful, when:

- humans expertise does not exist
- humans are not able to explain their expertise
- prediction problems involve a high level of complexity

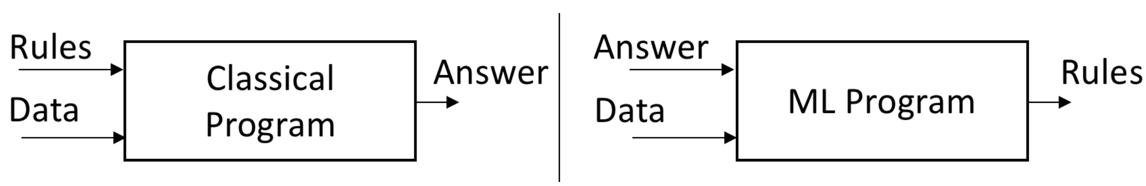


Figure 2.5: Classical programs and Machine Learning programs.

Figure 2.5 presents the difference between classical programs and machine learning problems. In the first ones, data and rules are provided as an entry, and the program gives an answer to the problem. In contrast, for machine learning programs, the entries consist in Data and already known answers; then the program establishes rules over this training set of data. Numerous studies have been conducted on the use of machine learning techniques for manufacturing prognostics.

M. Elangovan et al. [17] have discussed the effect of the Support Vector Machine (SVM) errors functions on the classification of vibration signals for single point cutting tools. The condition of a carbide tipped tool is predicted using a Kernel Support Vector Machine for two different error functions C-SVC and ν-SVC. The efficiency of these functions is then compared to other classifiers such as Decision-Trees, Naïve Bayes and Bayes net. It was found that, either for C or ν errors functions, the RBF Kernel gives higher classification efficiency. Finally, the linear Kernel can be interesting when it comes to have very fast classification. In comparison with other classification algorithms, the Kernel Support Vector Machine (KSVM) with ν-SVC has better efficiency. Then M. Elangovan et al. have shown that KSVM are promising for the prediction of the condition of a single point cutting tool.

C. Drouillet et al. [18] have used the neural network technique to predict tool life by monitoring the spindle power. End milling operations were performed on a steel work, and different MATLAB ™ learning functions were used to train a Neural Networks (NN). This method has demonstrated a good correlation between true and computed Remaining Useful Life (RUL); also it was very fast and could be used for Realtime RUL prediction.

Y. Fu et al. [19] have implemented Convolutional Neural Networks (CNN) for processing images representations of vibration signals. The vibration states have been considered to be a very promising way to real-time monitor machine states. Feeding the algorithm with an image of the signal without any preprocessing avoids possible bias introduced by the feature selection. Finally, the trained CNN showed very good results.

P. O'Donovan et al. [20] have introduced a fog computing industrial cyber-physical system for embedded low-latency machine learning application. Their research highlights that fog computing can be employed for real-time monitoring; this architecture enables a more distributed and scalable network while enhancing the privacy and the security of data.

Different machine learning algorithms have been implemented over the above-mentionned studies. A review of the different available techniques must be conducted in the following. First the difference between supervised and unsupervised machine learning is introduced, then the most well-known supervised ML methods are presented.

2.2.1 Supervised and unsupervised machine learning

As explained above, to be trained, machine learning algorithms usually expect Data and the ”answer” of the problem. However, sometimes the output is not known, and this is where the unsupervised machine learning is promising. The goal of these algorithms is to highlight the structure or the distribution of the data, thus it aims to learn a new data’s representation. The 2 major techniques of unsupervised machine learning are:

dimensional reduction: a data set of high dimension is reduced to lower dimension while keeping the ”important” characteristics. Thus, the redundancies are removed, the storage space and the computational power required to manage the dataset are reduced, finally data visualization and interpretation is improved.

clustering: the general characteristics of the data are understood, then the different object of the data set can be grouped based on those characteristics. Again, the data interpretability is improved.

However, most of the time the answers of the problems for the training sets are known; then it is called supervised learning. The aim is to make predictions rather than to enhance the data interpretability. The predictions can either be in the form of a decision function or of a classifier, that can be binary or multi-class. The mains techniques of supervised machine learning are:

- Decision Trees
- Naive Bayes classifiers
- Logist Regression
- Support Vector Machine
- Kernel Support Vector Machine
- Neural Networks

2.2.2 Supervised algorithms

Decision trees

Decision Trees can be used in other fields, but when it comes to machine learning, they are applied to predict the value or the class of an output based on given inputs; to that end these algorithms repetitively divide the working area into subs-sets, which are divided again and again: ”A decision tree is a recursive partition of the training set into smaller and smaller subsets” [21]. For data to be used in a Decision Tree model it needs to be discreet and without any ordering (e.g. classify fruit from color, shape, texture, size). Given a split variable j and a splitting point s, two regions (left and right) can be defined with:

$$R_l = x : x_j \leq s \text{ and } R_r = x : x_j > s$$

For regression problems, j and s have to be chosen in order to minimize:

$$\min_{j,s}\left(\sum_{i:x_i\in R_l(j,s)}(y_i - c_l)^2 + \sum_{i:x_i\in R_r(j,s)}(y_i - c_r)^2\right)$$

For classification problems, j and s have to be set such that the impurity is minimized:

$$\min_{j,s}\left(\frac{|R_l(j,s)|}{n}\cdot \mathrm{Imp}\left(R_l(j,s)\right) + \frac{|R_r(j,s)|}{n}\cdot \mathrm{Imp}\left(R_r(j,s)\right)\right)$$

The *impurity* $\mathrm{Imp}()$ can be either:

Classification error: the minimum probability that a point is mis-classified at the node (j, s) of the Tree:

$$\mathrm{Imp}(R_m) = 1 - \max_k \hat{p}_{mk}$$

with $\hat{p}_{mk}$ the portion of well-classified points.

Shannon’s Entropy: from information theory

$$\mathrm{Imp}\left(R_m\right) = -\sum_k \hat{p}_{mk}\log_2 \hat{p}_{mk}$$

Gini impurity: with still $\hat{p}_{mk}$ the portion of well-classified points.

$$\operatorname{Imp}(R_m) = \sum_{k=1}^{K} \hat{p}_{mk} (1 - \hat{p}_{mk})$$

Decision Trees present many advantages; they are easy to understand and to interpret, as they are a mirror to human decision making; however their predictive accuracy is not very good.

Naive Bayes classifiers

This classifier uses the posterior probabilities also called emphBayes Theorem 2.1 to make predictions.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.1}$$

For a binary classification problem, the aim is to express the probability distribution in a parametrized form. The probability of a single data point can be written as :

$$p_\theta(x, y) = p_\theta(y, x_1, ..., x_D) \tag{2.2}$$

Thanks to the Bayes Theorem 2.1 and the Naïves Bayes assumption, which states that $p(x_d|y, x_{d'}) = p(x_d|y) \; \forall \; d' \neq d$, the equality 2.2 simplifies:

$$p_\theta(x, y) = p_\theta(y) \prod_D p_\theta(x_d|y) \tag{2.3}$$

Then, depending on data type: binary, continuous... the model of $p(y|x_d)$ can be rewritten using respectively Bernoulli distribution and Gaussian distribution. Finally, the classification is the output is the class that is the more likely to be true.

Regression algorithms

Regression algorithms use the training data to fit curves and find a predictive function that maps the inputs to a continuous output $y = f(x_1, ..., x_n)$, depending on the number of features and the complexity of the relationship, different models can be used: the linear regression adjusts the coefficient b_i on the following equation $y = \sum_i b_i \cdot x_i$ in the case of n features; for more complex problems a polynomial regression can be used $y = \sum_i b_i x^i$. Finally, for some problems the logistic regression can be employed (here with the sigmoid function) $\log\left(\frac{p}{1-p}\right) = b_0 + b_1 \cdot x$

Support Vector Machine

Those algorithms are used to classify linear separable data points; as presented in Figure 2.6 (left). However, different margins can be found for the same data set and they do not split the dataset equally. Support Vector Machine (SVM) tends to find the best linear boundary between different classes by using an constrained optimization problem, which reads as:

$$\min_{\underline{w},b} \frac{1}{\gamma\left(\underline{w},b\right)} + C \cdot \sum_n \xi_n \tag{2.4}$$

with respect to : $y_n(\underline{w} \cdot x_n + b) \geq 1 - \xi_n$ and $\xi_n \geq 0$. In formula 2.4, $\gamma\left(\underline{w},b\right)$ is the value of the margin γ which depends on the weight vector $\underline{w}$ and the bias b , ξ_n is the "cost" of having a data point, which is not classified correctly as presented in Figure 2.6 (right). The

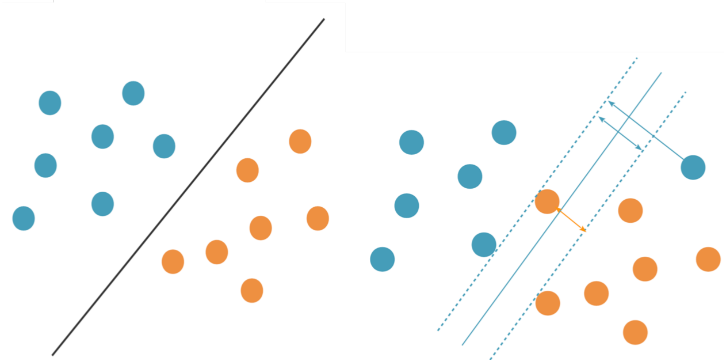


Figure 2.6: Linearly separable points (left), non-linearly separable data point (right). [22]

distance between two points x^+ and x^- at 1 unit from the margin read, as:

$$\begin{aligned} d^+ &= \frac{1}{\|\underline{w}\|} \cdot \underline{w} \cdot x^+ + b - 1 \\ d^- &= \frac{1}{\|\underline{w}\|} \cdot \underline{w} \cdot x^- - b + 1 \end{aligned} \tag{2.5}$$

So the margin γ can be expressed this way:

$$\gamma = \|d^+ - d^-\| = \frac{2}{\|\underline{w}\|} \tag{2.6}$$

and the constrained optimization problem is now to minimize the norm of the weight vector $\underline{w}$:

$$\min_{\underline{w},b} \frac{\|\underline{w}\|}{2} + C \cdot \sum_n \xi_n \tag{2.7}$$

with respect to: $y_n(\underline{w} \cdot x_n + b) \geq 1 - \xi_n$ and $\xi_n \geq 0$. As ξ_n must be positive but also minimum, it can be written that: $\xi_n = 1 - y_n(\underline{w} \cdot \underline{x_n} + b)$ (value of the classification error) if the point is not classified correctly and $\xi_n = 0$ if the point is classified correctly. Introducing $l^{(hin)}$ the hinge loss function as :

$$l^{(hin)}(a, b) = max(0, 1 - a \cdot b)$$

the term $\sum_n \xi_n = \sum_n l^{(hin)}(y_n, \underline{(}w) \cdot \underline{x_n} + b$
and equation 2.7 becomes:

$$\min_{\underline{w},b} \frac{\|\underline{w}\|}{2} + C \cdot \sum_n l^{(hin)} \left(y_n, \underline{(}w) \cdot \underline{x_n} + b \right) \tag{2.8}$$

Finding the minimum of the equation above gives information about the position of the optimum boundary. Although, this kind of algorithm is efficient for linearly separable or non-linearly separable data points with only few problematic points, sometimes, a linear boundary cannot be found between the categories (Figure 2.7) In these non-linear spaces, the use of a Kernel function is needed.

Figure 2.7: Data set where no linear boundary can be found. [22]

Kernel Support Vector Machine

Kernel functions can be used with a mapping Φ that projects the data points from the object space to a feature space where linear methods can be used, as in Figure2.8 A function

Figure 2.8: Mapping Φ from the data space $\mathcal{X}$ and the feature space $\mathcal{H}$ [23]

$K(x, x')$ defined on a set $\mathcal{X}$ is called a Kernel function if and only if there exists a Hilbert space $\mathcal{H}$ and a mapping $\Phi : \mathcal{X} \to \mathcal{H}$ such that for any x, x' in $\mathcal{X}$: $K(x, x') = \langle \Phi(x) \cdot \Phi(x') \rangle$. This enables us to use linear techniques but, more importantly the explicit computation of $\Phi(x)$ can be avoided, and $K(x, x')$ is computed instead. A Kernel Support Vector Machine (KSVM) is useful to classify data points where the data cannot be linearly separated in the data space and more importantly, in most cases Kernel methods reduce the computational power need. Thus, they are suitable for classification problems.

Finally, the most famous algorithms for machine learning are Neural Networks, section 2.2.2 presents different type of Neural Networks: Multi-Layer Perceptron (MLP), Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN).

Neural Networks

These algorithms try to replicate the way neurons work. The neuron is modeled with a perceptron, as in Figure 2.9 and its output is given by $f(x) = s\left(w_0 + \sum_{j=1}^{P} w_j \cdot x_j\right) = s\left(\underline{w}^T \underline{x}\right)$ where $s()$ is the threshold function. Other functions such as the sigmoid $\sigma = \frac{1}{1+\exp(-u)}$ can be used.

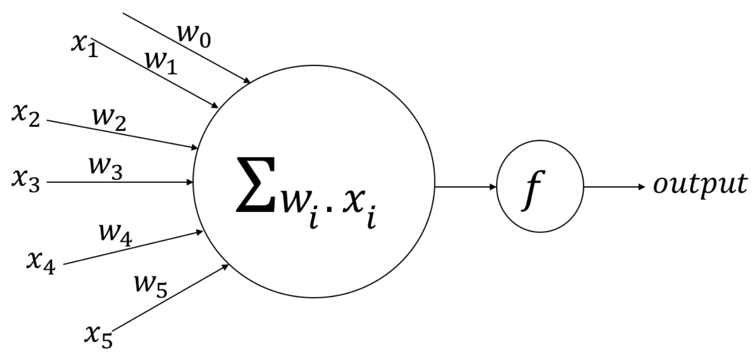


Figure 2.9: Data set where no linear boundary can be found. [24]

For binary classification (using the sigmoid function), the perceptron can be trained by adjusting all components of the weight vector $\underline{w}$ over the data set. For classification problems the cross-entropy error is generally used (η denotes the learning rate):

$$\mathcal{H}\left(f(\underline{x^i}), y^i\right) = -y^i \cdot \log(f(\underline{x^i})) - (1 - y^i) \cdot \log(1 - f(\underline{x^i})) \tag{2.9}$$

Then the weight update for every iteration reads as:

$$\Delta w_j = -\eta \frac{\partial \mathcal{H}\left(f(\underline{x}^i), y^i\right)}{\partial w_j}$$
$$\Delta w_j = \eta\left(y^i - f(x^i)\right) x_j \tag{2.10}$$

However, in the case of multiclass classification, the softmax function, equation 2.11, is used to find which class is more probable than the other. If class k is more probable than the other then $\sigma_k(x) \approx 1$ else $\sigma_k(x) \approx 0$.

$$\sigma_k(x) = \frac{\exp(\underline{w^k}^T \cdot \underline{x})}{\sum_{l=1}^{K} \exp(\underline{w^l}^T \cdot \underline{x})} \tag{2.11}$$

Then, the weight update reads as $\Delta w_j^k = \eta\left(y^i - f_k(x^i)\right) x_j^i$. Finally, for each training instance: $w_j^{t+1} = w_j^t + \Delta w_j^t$.

Adding several layers of Perceptrons as presented in Figure 2.10

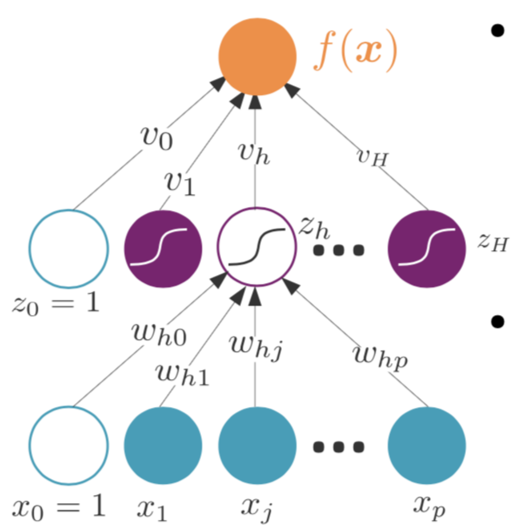


Figure 2.10: Multi-layer Perceptron structure. [25]

It is composed of 3 or more layers of Perceptrons, each layer feeding the following one. This algorithm is efficient for non-linear data classification.

Convolutional Neural Networks, on the contrary add more layers, the first operation transforms the input into feature maps that compose the convolution layer; then after one or multiple convolution maps a rectification layer is applied with functions such as ReLU, sigmoid... At the end, the last layers consist of a common Multi-layer perceptron. Convolutional neural networks are mostly used for image processing, however, Y. Fu et al. [19] have used them for Machining vibration states monitoring based on image representation. The advantage of this technique is that they were able to reduce the bias introduced by feature selection that must be performed for other machine learning methods such that Kernel Support Vector Machine.

Finally, Recursive Neural Networks (RNN) add more connections between the hidden layers of a Convolutional Neural Networks. The nodes are fed information from the previous layer but also information from their own last state. This enables them to learn from the past.

Those different machine learning algorithms can be used to classify images or preprocessed signals from sensors. The choice of the algorithm and its parameters can be made thanks to the programmers knowledge, and different setups maybe tested to find the most suitable one.

2.3 Communication Protocols for data transmission

In the following the major protocols for data transmission and Industry 4.0 are presented: MQTT, CoAP, Bluetooth, Bluetooth Low Energy (BLE), HTTP and WebSockets,

2.3.1 MQTT

MQTT stands for Message Queue Telemetry Transport. It is a lightweight data protocol that uses a publish and subscribe architecture was initially developed by Dr. Andy Standford Clark for IMB and Alan Nipper for Arcom; now the protocol is open source and maintained by the MQTT organization. This a Machine to Machine standard that uses a message broker to forward messages to clients depending on topics.

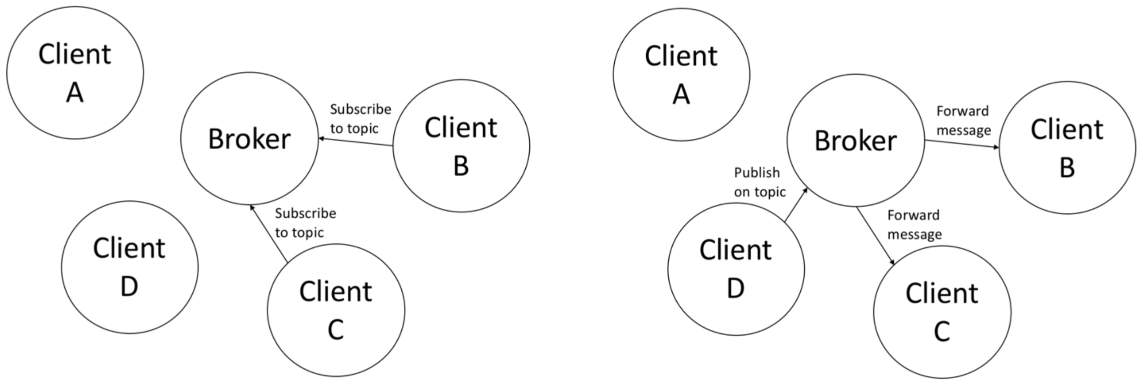


Figure 2.11: MQTT protocol, subscribtion (left) and publishing (right).

As presented in Figure 2.11, first the different clients subscribe to the topics they want to receive messages about. Then every time a client publishes a message about the corresponding topic, the broker forwards the message to the clients that have subscribed to this particular topic. This mode enables one to one, one to many and many to one communications.

2.3.2 CoAP

CoAp stands for Constrained Application Protocol, as is MQTT, it is designed for machine to machine applications. It has been optimized for peripheric and constrained networks. It is based on the REST architecture and uses a client to server model, in which clients send requests to the server in order to receive data as a response. However, the packets are lower than for other protocols. Such as HTTP, for example; the CoAP header is limited to 4 bytes (compared to the 100 bytes for HTTP). This allows the use of the CoAP protocol for small embedded devices, which makes CoAP a good protocol for Industry 4.0 and Internet of Things applications.

2.3.3 WebSockets

Usually, internet communications over a client and a server use HTTP; the client sends a request to the server in order to establish a connection, then data is transferred from the server to the client, and at the end of the transfer the connection is closed. One the other hand, WebSockets solve some issues of the HTTP protocol; the communication between the client and the server stays open, and both can send and receive data at the same time. This enables a full duplex communication that is very interesting for receiving data from sensors and to push information from the cloud.

2.3.4 Bluetooth and Bluetooth Low Energy

The above presented protocols (MQTT, CoAP ...) usually communicate using wired or wireless internet infrastructure. In contrast, Bluetooth and Bluetooth Low Energy (BLE) are wireless protocols that use radio frequency 2.4GHz. The communication is established between two devices, and even if it is very stable, the range is quite short, and communication with more that 2 devices is not possible. The protocol is widely used to connect wireless devices for Internet of Things applications.

Bluetooth Low Energy is a new version of the Bluetooth protocol that uses a low data rate in order to reduce the battery consumption of the devices.

2.3.5 LORA

LoRa from Lo(ng) Ra(nge) is a wireless protocol; it is a Low Power Wide Area Network (LPWAN). This means that it is suitable for application where the range and the autonomy are more important than the bandwidth. Lora denotes both the physical interface, which, patented in 2014, is still proprietary, and the public LoRaWAN that was developed by SEMTECH and defines the communication protocol. The aim of this protocol is to ensure the communication between gateways that are connected to the Ethernet and end-nodes that are acquiring data (Figure 2.12).

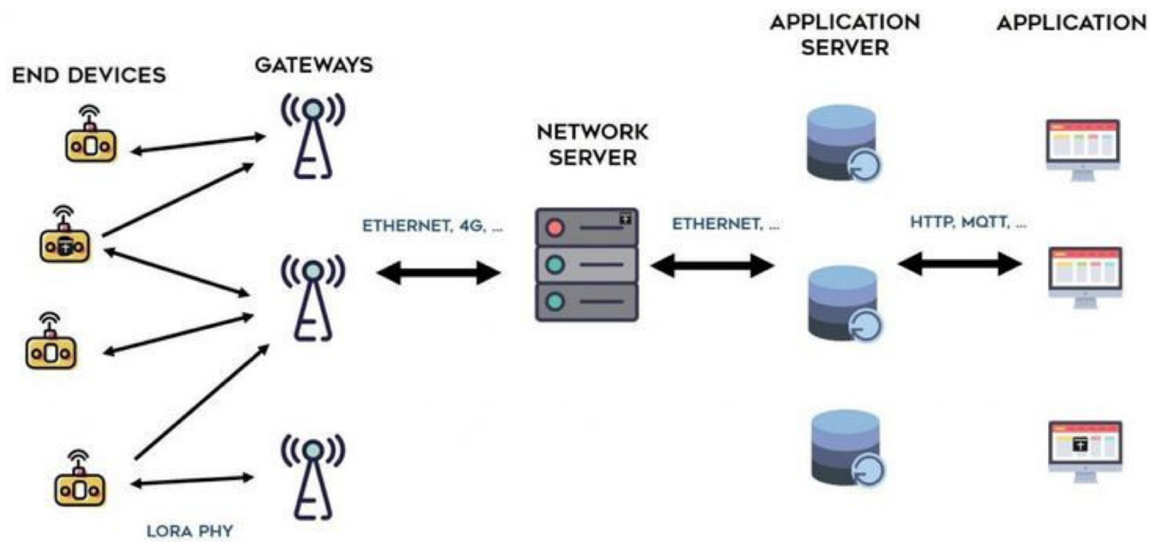


Figure 2.12: A typical LoRa Architecture. [26]

2.3.6 Zigbee

This is a 2.4GHz standard built on IEEE 802.15.4 norm. This mesh network is designed for low band width, short range communication, but those compromises come with a very low power consumption. Thanks to the mesh capability of this network, each node can act as an end-point or as a repeater that forwards the message to the next node.

2.4 Cloud computing and Edge computing

The recent improvement of communication technologies enables the use of powerful remote computers to process data. Complex architecture can be used to acquire data on the machine shop or on other industrial infrastructure. This is known as cloud computing and those architectures can be also used to store important amount of data.

R. I.S. Pereira et al. [27] used the cloud's computing power to monitor a photovoltaic plant. The data were acquired with a Raspberry Pi and sent to the cloud to be processed. S. Yang et al. [28] have presented a unified Framework and Platform for Design of Cloud-Based Machine Monitoring and Manufacturing Systems; this study was focused on the sensor development and wireless communications. C. Kan et al. [29] have introduced parallel computing and a network analytics for fast Industrial Internet of Things (IIoT) for machine condition monitoring. This network, even if it is computationally expensive, uses the embedded distributed power to follow the machine's condition.

D. Wu et al. [30] have used the computing power of the cloud to process data and to build predictive models. In contrast to other similar studies, those algorithms were then exported to a private cloud and were used to make predictions on the data. A proof of concept is used to demonstrate the architecture is presented in Figure 2.13

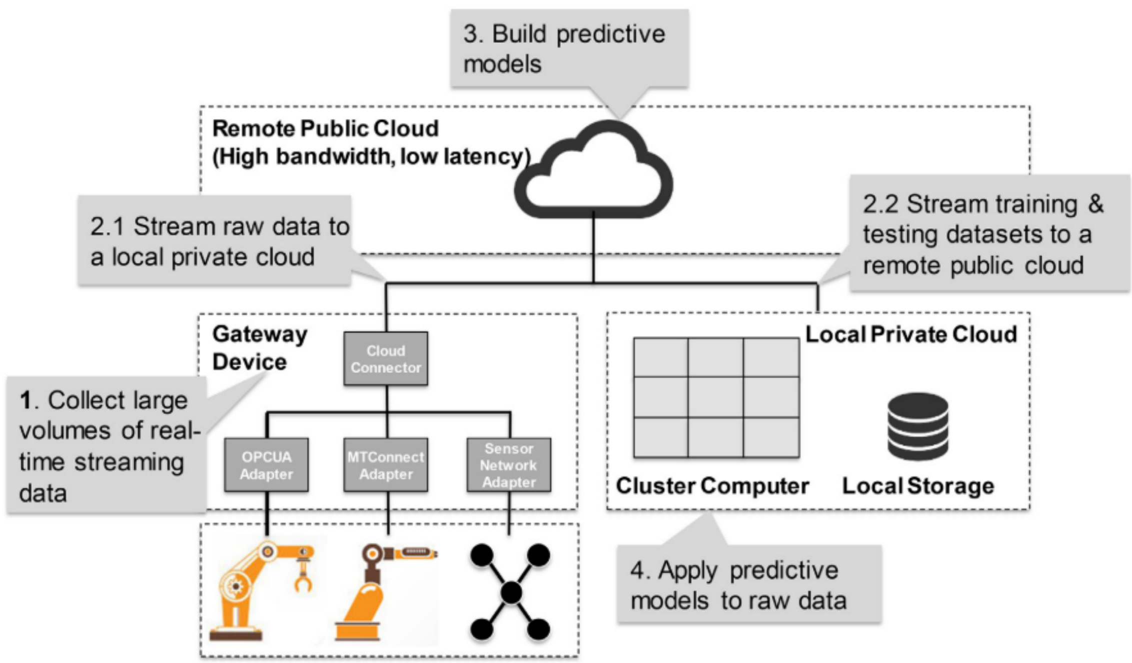


Figure 2.13: Fog based computational Network. [30]

P. O'Donovan et al. [20] uses the idea to process data with local resources. The main idea is to use computers that are not located on the cloud but are physically in the factory in order to execute a predictive model. Then this technique avoids exporting large amounts of data to the cloud. This solves some Industry 4.0 concerns, such as decentralized and autonomous decision-making management. Moreover, this approach improves security, privacy and reliability of all of the system, since the data remains on at the factory level.

Usually the policy of data management depends on the company and may not be adequate to the cloud service provider. The architecture proposed in their study is presented in Figure 2.14.

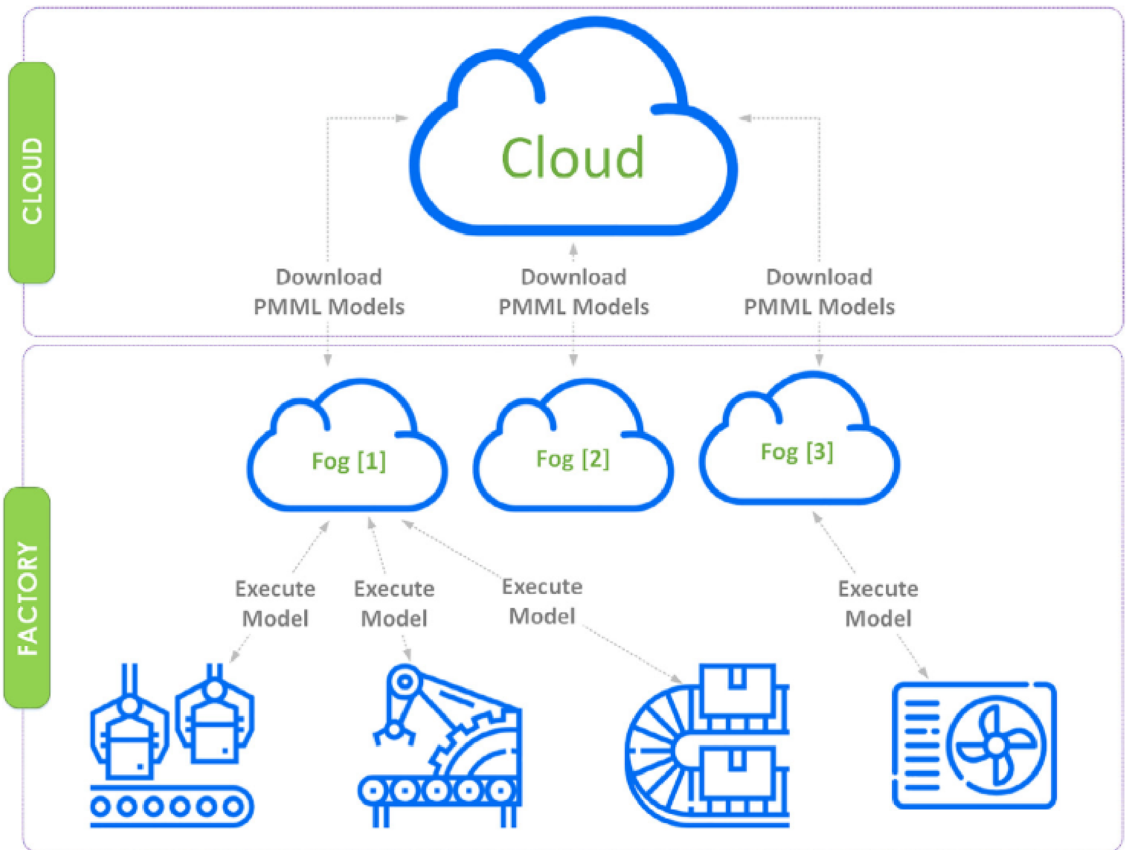


Figure 2.14: Fog computing with cyber-physical interaction. [20]

The Industry 4.0 proposes a more decentralized computing architecture. For economic reasons, companies tend to improve the machine monitoring architecture to make more accurate real-time predictions or analyse of the factory. This has been seen as a solution, but it presents problems of security, privacy and reliability. Fog computing, on the other hand seems to be a more promising solution to address those issues, as the new embedded systems enable the use of powerful algorithms on very small and low cost systems, such as the BeagleBone Black or the Raspberry Pi. Some architectures are proposed in the previous studies; however, few of them use the entire processing power of those chips and other local computers are often added to process data. In this thesis the main goal is to use the full capacities of these processors for both data acquisitions and data processing, more than fog computing this could be called Edge Computing.

The case study will be the vibration monitoring of cuts of different materials with a band saw. The final system should be able to real-time distinguish the material being cut.

CHAPTER 3
PROPOSED FRAMEWORK

The main concern of this work is to have a system which is fast enough to be considered Realtime and the processing power should be located on the board that realizes the data acquisition. The following section first presents the hardware architecture used in this thesis, then the Software architecture is introduced.

3.1 Hardware components

This part presents the hardware selected for this project. As for most machine monitoring projects the different components are:

- a sensor to transform the physical phenomenon into an electric signal
- a mechanical adaptor to mount the sensor on the machine
- a microcontroller or microprocessor to process the data
- an adaptor may be added to fit the tension between the sensor and the Board
- a cloud service or remote computer, which is only used in the training phase

Indeed, two phases have to be identified in this project. The first one consists in the data acquisition for training the machine learning algorithm; the second phase is the deployment of the board with the trained algorithm on real conditions. During the first phase, the samples are concatenated on the embedded board and sent to a remote computer. Once all the sample sets have been acquired, the machine learning algorithm is trained on the training set, then an evaluation is conducted on the test set. Typical ratios between the training set and the test set are respectively 80% and 20% of the sample set. When accuracy on the test set is good enough, the algorithm can be exported to the Embedded board and the learning phase is completed. Figure 3.1 sums-up the different steps of this phase.

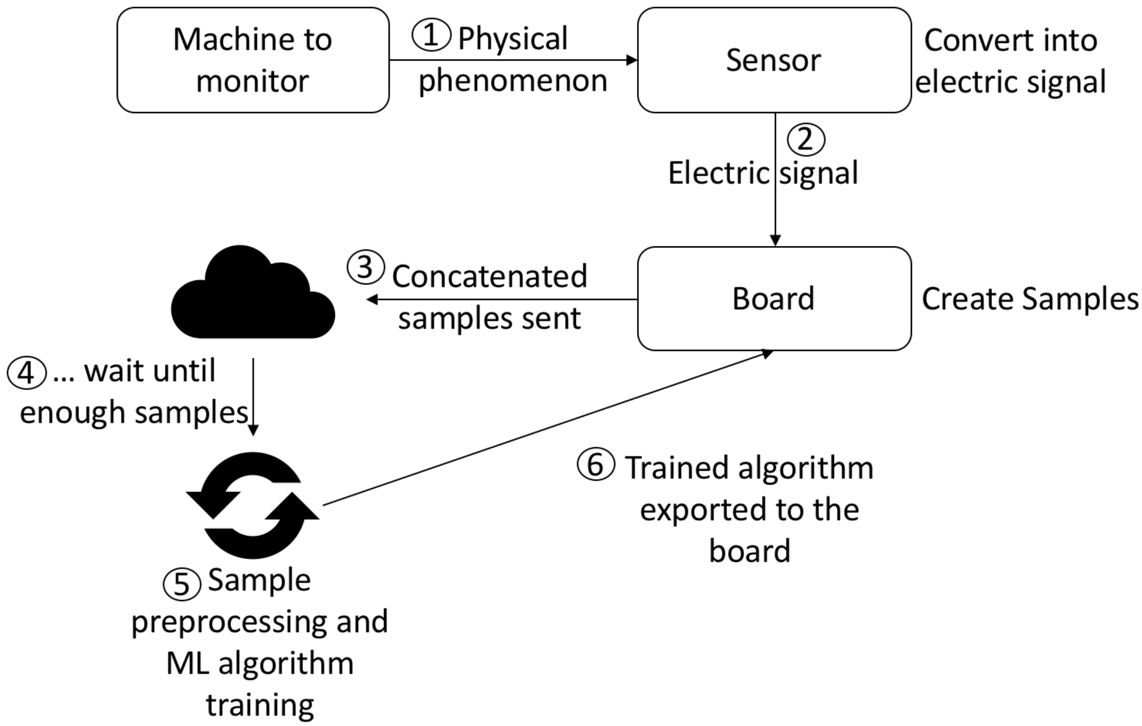


Figure 3.1: The different steps of the training phase

The second phase is the deployment phase where the trained algorithm has been exported onto the board. The sensor keeps sending data to it, so the samples, after being concatenated, are fed into the algorithm. The algorithm returns the classification results that can be sent to the cloud. Figure 3.2 presents the hardware architecture of this phase.

Type of sensor The choice of the sensor depends on the physical phenomenon that is going to be used, as presented in 2.1.1. Indirect sensing methods are more suited for machine real-time monitoring. M. Siddhpura et al. [31] stated that vibration sensing is easy to implement; moreover, vibrations can be acquired using a simple accelerometer. This kind of sensor is very cheap and is suited to this application.

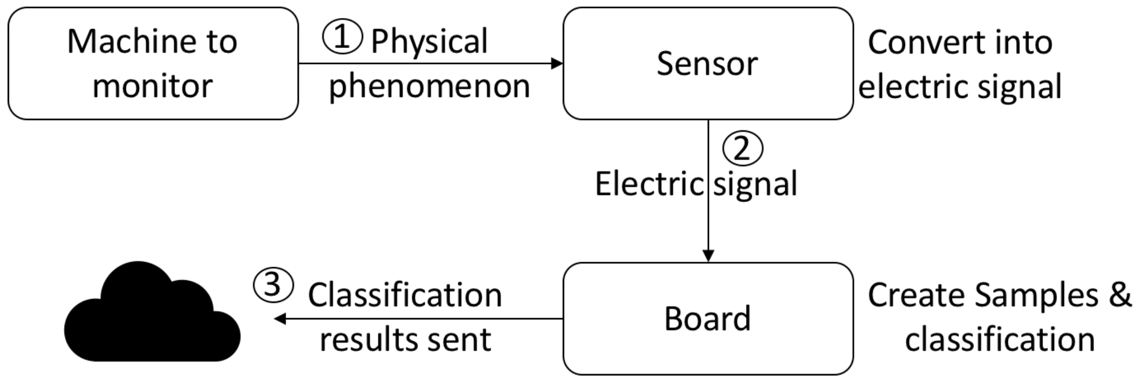


Figure 3.2: The deployed system.

Type of board The board used in this work needs to be powerful enough to process data and run the classification algorithm in real-time. Even if the deterministic capabilities of micro-controllers presented in 2.1.2 are an advantage for vibration acquisition because they provide a constant and precise sampling rate, their computing power is not important enough for the desired application. A choice is made to use a microprocessor such as the BeagleBone Black or the Raspberry Pi. The absence of deterministic properties of this type of board is being addressed in the following part 4.1.

3.2 Software architecture

Concerning the learning phase, as we want to sample vibration very fast the code should first sample N_d data points and store them in a file which will constitute a sample. N_d should be large enough to depict a representative period of the signal, but small enough to let the process be real-time. Thus, N_d depends on the sampling frequency f_s of the system, the higher f_s is, the larger N_d should be, thanks to relation 3.1:

$$f_s \cdot N_d = C \tag{3.1}$$

with C a time constant that should be far above the characteristic time of the system. This sampling operation should be repeated N_s times for the N_c category of sample that we want to be able to classify:

- band saw off.
- band saw running, setup up to cut material 1 but not cutting (no physical interaction between the saw and the workpiece).
- same case as above but setup for material 2.
- band saw cutting material 1 and setup for material 1.
- same case as above for material 2.

Once all the N_s samples have been acquired for the N_c classe, they are exported to the remote computer where the features such as dominant frequencies, mean, max, min... of the samples are extracted. For the N_c classes, tables of N_s by N_f are created and concatenated to create a data frame of size $(N_c \cdot N_s)$ by N_f, which is split into the training set and the test set. The selected machine learning algorithm is trained thanks to these sets.

During the deployment phase, the board should acquire N_d points at the frequency f_s to create a sample, then the N_f features are extracted and feed into the algorithm. The classification result is finally sent to the cloud.

The architecture implementation is discussed in the following chapter. First, the selected hardware is presented, then a solution for deterministic data acquisition is introduced, the training of the machine learning algorithm is detailed afterward, and finally the results of the implementation are considered.

CHAPTER 4
IMPLEMENTATION AND RESULTS

This section presents the implementation of the above presented architecture. Once the implementation has been validated, it is tested on real conditions.

4.1 Hardware selection

Band saw The band saw used for the work is the **8-Mark-II vertical Tilt-Frame** band saw available in the Montgomery Machining Mall (Figure 4.1). This industrial band saw is designed for metal cutting at speeds ranging from 50 to 450 fpm.

Figure 4.1: The Band Saw used for this study.

Accelerometer Different types of accelerometer can be used, depending on the type of communication that we want to be implement. The communication can be either SPI or I2C if the sensor is a digital one. On the other hand, analog sensors do not provide these communication interfaces, but they require the board to use an analog to digital converter. Most of the low-cost accelerometers that use SPI or I2C does not have high sample frequencies. As an example, the ADXL345 has SPI and I2C capabilities, but its preferred sampling rate is 1000Hz for a cost of \$17; the MMA8451 has I2C capabilities but a maximum output rate of 800Hz for a cost of \$7.50. In contrast, analog accelerometers such as the ADXL203EB are a bit more expensive, but the acceleration value can be accessed at very high frequencies; the sampling limit is generally set by the maximum sampling frequency of the board's analog to digital converter. The needed output rate of the accelerometer is directly linked to the frequency we want to observe on the band saw. This frequency is given by the impact of the teeth on the workpiece. In order to estimate the output frequency, the speed of the blade is set to the maximum possible: 450 fpm, the blade has 10 teeth per inch. Thus, the frequency of the impact is:

$$\frac{450 \cdot 120}{60} = 900 \text{ teeth per second}$$

The Nyquist-Shannon Theorem states that to observe this frequency, the sampling frequency should be at least 1800Hz. Considering the output rate of digital accelerometers, choice is made to use the analog accelerometer: the **ADXL203EB**.

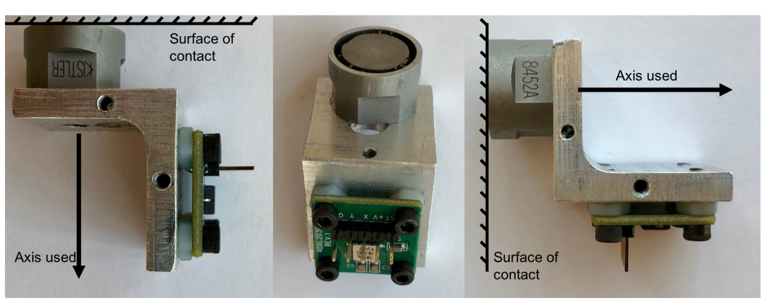


Figure 4.2: The mechanical adaptor for fix the accelerometer on the band saw.

mechanical adaptor In order to set the accelerometer on the band saw, a mechanical adaptor is designed and machined. The final part is presented in Figure 4.2; 4 screws hold the ADXL203EB on the adaptor, a magnet is added on the other side to ensure that the adaptor is securely fixed on the band saw.

Only one axis of the accelerometer is going to be used; the choice of this axis is made so that is it normal to the surface where the magnet is put. It has to be noticed that the use of another of the ADXL203EB will note provide reliable data since the magnet can only ensure that the accelerometer is oriented along the axis normal to the surface of reference, it does not prevent the adaptor from sliding on this surface.

Choice of the Board As discussed in 3.1, microprocessors seems to be the more promising solution for this work. Between the BeagleBone Black and the Raspberry Pi, the main difference is the absence of an Analog to Digital Converter on the Raspberry Pi; moreover, this board does not have Process Real-time Units; therefore, even if the version of the Raspberry Pi has a better processing power that the BeagleBone Black and the benefit of a wider community, the choice goes in favor of the **BeagleBone Black wireless**.

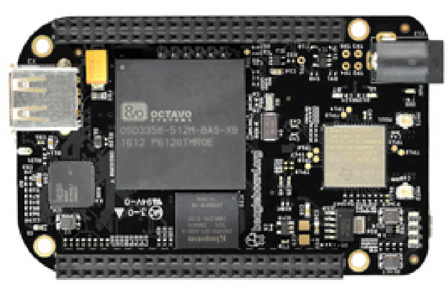


Figure 4.3: The Beaglebone Black wireless.

Beaglebone cape for electrical adaptation The output tension of the ADXL203EB ranges from 0 to 5V; however, the maximum input voltage on the Analog to Digital Converter of the BeagleBone Black is 1.8V. An adaptator cape (Figure 4.4), which acts as a tension divider, is built to avoid burning the BeagleBone Black. The two resistor values are: $R_1 =$

$6.8\text{k}\Omega$ and $R_2 = 12\text{k}\Omega$, then the tension divider gives:

$$V_{out}^{max} = \frac{R_1}{R_1 + R_2} \cdot V_{in}^{max} = \frac{6.8}{6.8 + 12} \cdot 5 \rightarrow \boxed{V_{out}^{max} = 1.8 \text{ V}} \tag{4.1}$$

The Eagle Files for the board are presented in Appendix A

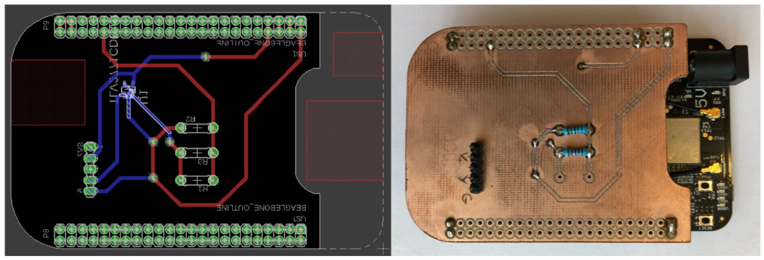


Figure 4.4: The cape for the electrical adaptator

Figure 4.5 presents the final hardware for this thesis with every component ready to be used. Now that it has been introduced, the following section discussed real-time data acquisition on the BeagleBone Black.

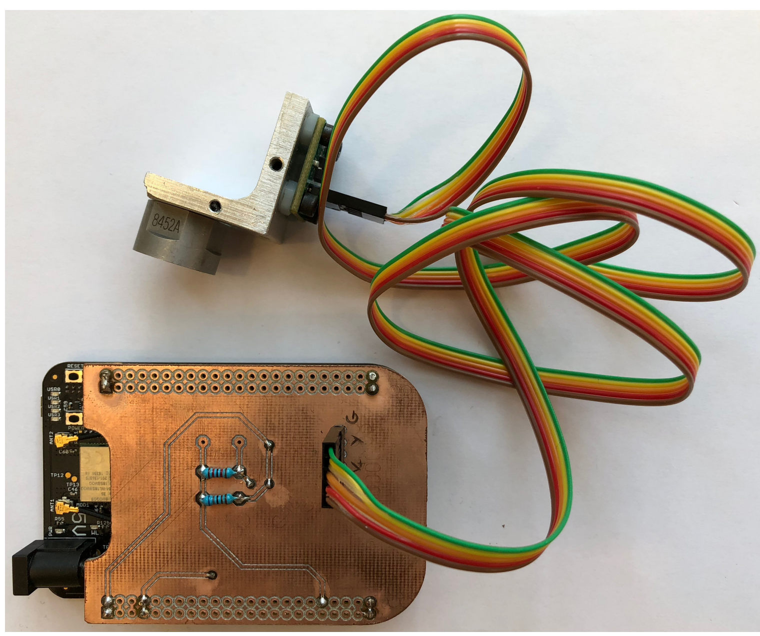


Figure 4.5: The final hardware setup for this work.

4.2 Realtime data acquisition on the ti-am335x chip

The chip running on the BeagleBone Black is the ti-am3358. Two ways of getting deterministic sampling on this chip are considered in this section. First an attempted use of the two Process Real-time Units is detailed, then the Linux Industrial In/Out subsystem capabilities are introduced.

4.2.1 Process Realtime Unit (PRU)

Presentation of the PRUs

The 2 PRUs are microcontrollers such as Arduino® and the Teensy. It means that they are able to execute real-time processing, then the *"programmable nature of the PRU, along with its access to pins, events and all system on chip (SoC) resources, provides flexibility in implementing fast real-time responses, specialized data handling operations"* [32]. Thus, those PRUs are fully integrated in the global architecture of the ti-am3358 chip (Figure 4.6), which is very useful when it comes to carry out time critical operations such as fast data acquisition.

The RPMsg framework

In order to enable communication between the two process real-time units, Texas Instruments has created a framework that is used to send and receive message between the ARM and the PRU:

> *RPMsg is a message passing mechanism that requests resources through Remoteproc and builds on top of the virtio framework. Shared buffers are requested through the resource_table and provided by the Remoteproc module during PRU firmware loading. The shared buffers are contained inside a vring data structure in DDR memory. There are two vrings provided per PRU core, one vring is used for messages passed to the ARM® and the other vring is*

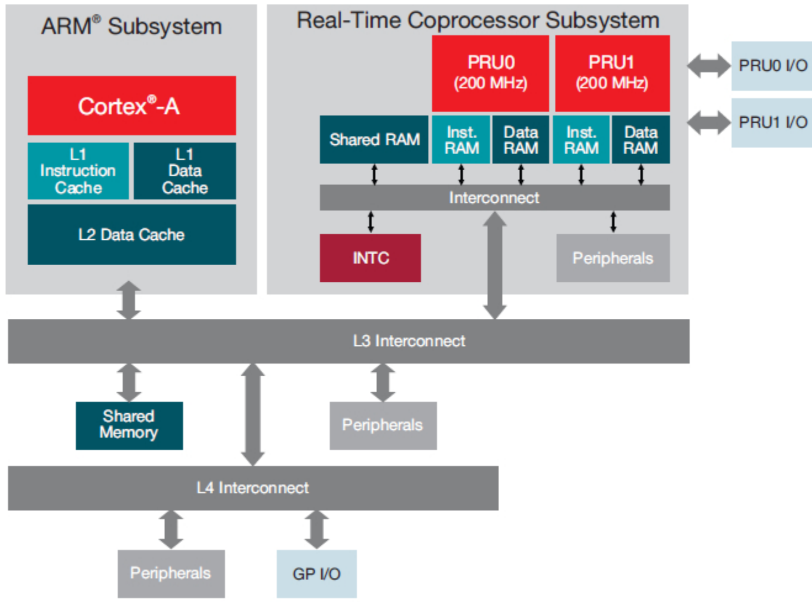


Figure 4.6: Architecture of the AM3358 with Cortex® -A8 and the 2 PRUs [33]

used for messages received from the ARM®. System level mailboxes are used to notify cores (ARM® or PRU) when new messages are waiting in the shared buffers.

Texas Instrument[34]

Figure 4.7 presents the interactions between the ARM® and the PRUs the process is quite complicated; to make it simple, the data is placed in the DDR memory of the chip and a system of mail boxes between the Linux Kernel of the ARM and the PRU subsystem delivers notifications that data are ready to be read.

This PRU subsystem seemed to be promising, as it was supposed to permit users to combine the real-time advantages of a microcontroller in a processor. However, this solution suffers of an important lack of documentation; to illustrate this point it has to be considered that 2 months of work have been necessary just to start the PRUs from the ARM. After this, a communication was established between the ARM® user space and the PRU subsystem via the Linux Kernel and RPMsg. This communication was then extended to the ADXL345 using a bit banging technique on the BeagleBone GPIO. There was no

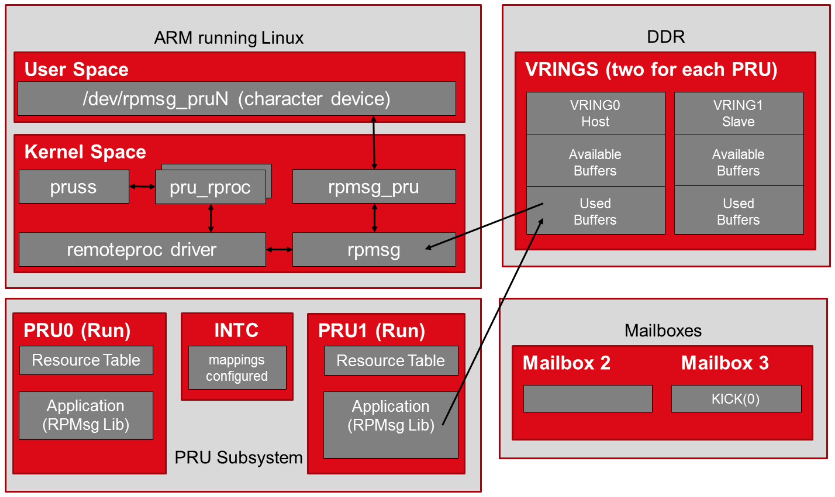


Figure 4.7: Interaction between the ARM® and the PRUs when using RPMsg [34]

success in establishing a communication between the Analog to Digital Converter of the BeagleBone and the PRUs.

The PRU subsystem approach was discontinued, and the work was documented in a tutorial presented in Appendix B for hypothetical future works using the PRU. The next solution consists in the use of the Linux Industrial In/Out system, which takes advantage of the hardware Interrupt capabilities of the Linux Kernel.

4.2.2 Linux Industrial I/O (IIO) subsystem

Linux Operating System

This section presents the function of the Industrial In/Out subsystem that is used in this work to get data from the Analog to Digital Converter of the BeagleBone Black. First, the software structure of Linux and its interaction with the hardware is introduced, then the Linux IIO subsystem is presented and the use of Linux Kernel Driver and Device trees is detailed. Finally, the code used in the user space to interact with the kernel is presented.

Linux can be seen as decomposed in two parts (Figure 4.8) a Kernel Space and a User Space. The first one has initially been developed by Linus Torvalds and is now supported by the Linux Foundation. It derives from Unix systems, was announced in August 1991 and the first version (0.02) released in October of 1991. The Kernel is the corner stone of the Linux Operating system; it is responsible for managing the interactions of the hardware components. On the other hand, the user space is the space of applications; it is where the user can interact with the operating system.

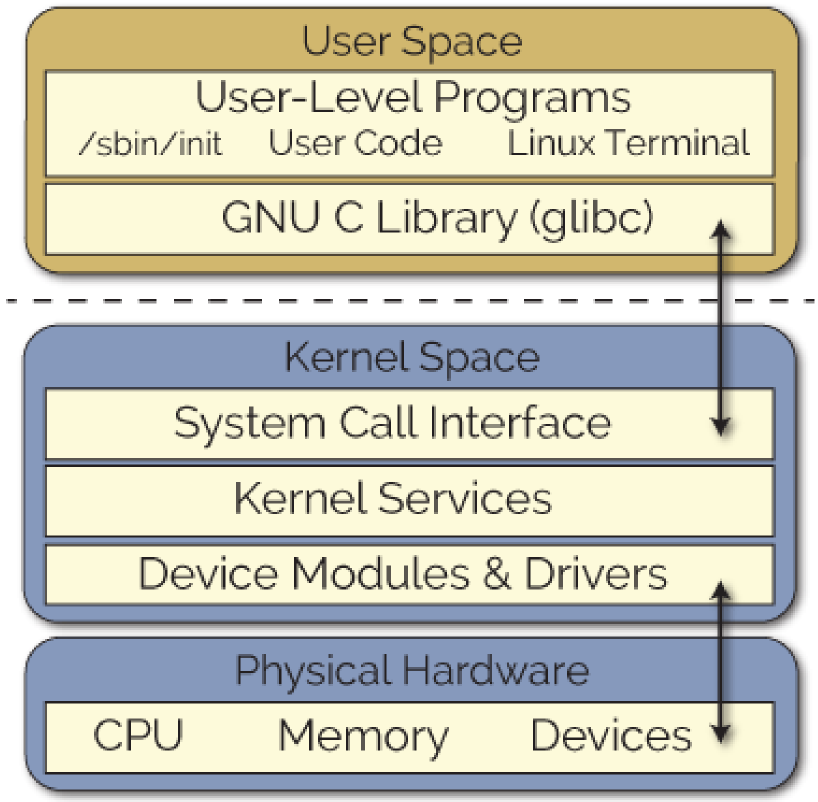


Figure 4.8: The Linux user and kernel spaces [35]

The communication between the user space and the kernel space is managed by the C library and system calls. On the other side the kernel uses device modules and drivers to interact with the hardware. The power of Linux consists in its versatility; it has to be able to manage numerous different piece of hardware. Thus, in order to reduce the size of the kernel, drivers and kernel modules are used; they can be loaded and unloaded to ”tell” the kernel how to deal with a particular piece of hardware.

Industrial In/Out subsystem

The main interest of using a Kernel code is that it has hardware interrupt capabilities, which are suitable for deterministic data sampling at high rates. This is the reason why the Industrial In/Out subsystem has been introduced: for operation where it comes to get data from sensors such as: ADC, accelerometers, DAC, gyroscopes, temperature sensors, pression sensor... in short, every sensing device that require an analog to digital conversion.

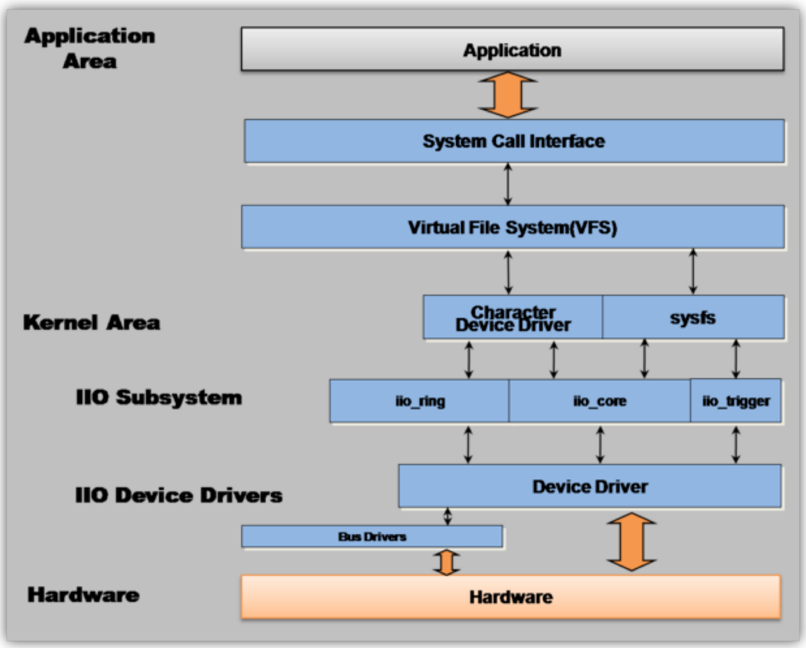


Figure 4.9: The Linux user and kernel spaces [36]

As shown in figure 4.9 this subsystem lives in the kernel space and is used to display the hardware information in the user space. The IIO ring, core and trigger receive information from the user space via sysfs interface and then returns data in a device character generally located in :

```
/sys/bus/iio/iio:deviceN/
```

The interaction between the subsystem and the hardware is managed by hardware specific drivers.

The ti_am335x_adc Linux Kernel driver

In order to use the Industrial In/Out subsystem the Linux kernel driver corresponding to the ti-am3358 chip has to be compiled and loaded in the kernel. The source code is provided by Texas Instruments on Gitorious [37]; however, in order to have a faster sampling frequency it is possible to change the clock reference of the ADC from 3MHz to 24MHz; for this the ti_am335x_tsadc.h header line 140 is changed from :

```
#define ADC_CLK 3000000
```

to,

```
#define ADC_CLK 24000000
```

The final header code is attached on Appendix C of this work.

Then the ti_am335x_adc.c can be compiled and loaded into the kernel. The other parameters required by the ADC to work are specified using device tree overlays; the BB-ADC-00A0.dts is presented in Appendix D. the different channels that can be used are specified on line 37, in our case the ADC's channel 3 is the only one connected. The sampling frequency of the ADC is set with the 3 parameters "ti, chan-step–avg", "ti,chan–step–opendelay" and "ti,chan–step–sampledelay" (lines 38–41). The number of clock cycles necessary for a conversion is then given by the formula:

$$\text{num cycles} = \text{opendelay} \cdot (\text{sample delay} + \text{convtime}) \cdot \text{averaging} \tag{4.2}$$

In our particular setup we have:

$$\text{num cycles} = 1 + (13 + 1) \cdot 8 = 112 \text{ cycles} \tag{4.3}$$

Then for a 3MHz clock the sampling frequency is 25kHz.

User space application

The data from the ADC is now ready to be displayed in the device character on the user space. To do so the acquisition has to be launched with an application. For one sample of N_d points the steps are:

- disable the iio_trigger, because we want a software Interrupt
- activate the iio_channel that we want to read on the ADC.
- create the buffer directory to set the buffer length
- in the /Results folder create a .csv file named after a timestamp to store the sample points
- read the values on the buffer and store them on the .csv file

All these steps are done with the iio_generic_buffer.c (see Appendix E), this code was adapted from Jonathan Cameron's example [38] in order to Disable the hardware trigger and to have the data stored.

For the training phase process, many samples have to be acquired. This process has been automated with a script in Appendix F. Once the number of sample N_s and the number of points per samples are specified, the script starts N_s acquisitions and stores them on the BeagleBone and exports them to the remote computer with an SSH secure copy.

During the deployment phase only one sample is needed, and the acquisition is launched thanks to a master application.

Finally, the data can be deterministically acquired via the IIO subsystem. The ADC parameters are set using a device tree overlay, then the ti_am335x_adc driver sample the ADC at the given frequency and the iio_generic_buffer application read the device character in the user space to store the data points of the sample in a .csv file. The next part will present the experimental setup for the data acquisition.

4.3 Experimental setup

This section explains the material choice for this work, then the setup on the Mark II band saw is presented. Finally the sample size and frequency are determined.

4.3.1 Coice of the Materials

This work does not aim to realize very precise classification between different materials. In Section 3.2 different classes are introduced. A choice has been made to limit the number of material to 2: Aluminum and Steel. Those materials are frequently used in the Montgomery machine shop, and data can be acquired for cuts on scraps in order to reduce the cost of the study. Consequently, the exact alloy is unfortunately unknown.

4.3.2 System setup on the band saw

The sensor is placed on the clamp of the band saw and the BeagleBone is directly connected to the computer. The final setup is presented on figure 4.10

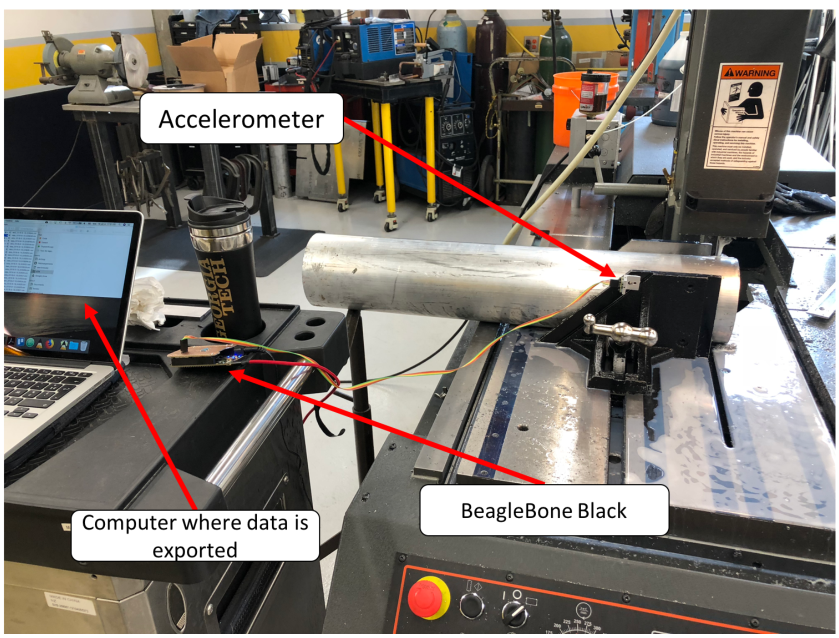


Figure 4.10: The final setup on the machine

4.3.3 Sample size and frequency

Sampling frequency

As presented in 4.1, the sampling frequency of the ADC should be at least 1800Hz. However, because of the device tree settings, it is quite complicated to find a precise sampling frequency. In order to have a reliable value, the parameters are set as presented in table 4.1.

Table 4.1: Device Tree and clock settings for the ADC

Parameter	Value
Clock frequency	3MHz
ti,adc–channels	$\langle 3 \rangle$
ti,chan–step–avg	$\langle 8 \rangle$
ti,chan–step–opendelay	$\langle 0 \rangle$
Ati,chan–step–sampledelay	$\langle 0 \rangle$

The resulting sampling frequency of the system is **25kHz**. This value is verified and validated with a wave generator of which sine waves are sampled. Given the number of points of the sample, the number of waves observed on the sample and the frequency of the signal it is possible to find the sampling frequency of the ADC with the equation:

$$\mathrm{ADC_{frequency}} = \frac{\mathrm{Number_{points}} \cdot \mathrm{f_{generator}}}{\mathrm{N_{waves}}} \tag{4.4}$$

The results are presented in table 4.2. The frequencies of the waves were chosen to find an integer number of waves with the hypothesis of a 25kHz sampling frequency:

Table 4.2: Sampling frequency validation

Number of points	$\mathrm{f_{generator}}$ (Hz)	approximated N_{waves}	$ADC_{frequency}$ kHz
1000	150	6	25
1000	25	1	25
1000	825	33	25
1000	57	2,3	24.7

Number of points per sample

In order to get a good idea of the signal it is decided to sample during at least 0.5 s. According to the sampling frequency chosen above the number of point needs to be more than 12500 points. Moreover, a Fast Fourier Transform will be performed on the sample. So it is interesting to have a number of samples which is a power of 2. Finally, the number of samples is set to 2^{14} =**16384**.

4.3.4 Data acquisition

Using the above presented setup and given parameters, 2000 samples were acquired for each of the 5 classes, presented in 3.2 according to the following steps:

- band saw off.
- band saw running, setup up to cut aluminum but not cutting (no physical interaction between the saw and the workpiece).
- same case as above but setup for steel.
- band saw cutting Aluminum and setup for material 1.
- same case as above for steel.

It has to be noticed that the steel workpiece and the aluminum workpiece did not have the same shape; the aluminum part was a rod and the steel part was a plate as shown in figure 4.11.

The setup parameters of the Mark II band saw to cut steel and aluminum are presented in table 4.3:

This data sets represent 5 folders, each of them containing 2000 .csv files of 16834 lines each. Once those data are acquired and exported to the remote computer, the machine learning algorithms have to be trained. Figure 4.12 represents 5 samples, one for each class.

Figure 4.11: The steel part (left) and aluminum part (right)

Table 4.3: Band Saw Setup

Material	Speed (fmp)	Feed (lbs)
Aluminum	300	30
Steel	150	30

4.4 Feature selection and preprocessing

This section presents some the parameters chosen for the training of the

4.4.1 choice of Kernel Support Vector Machine (KSMV)

In 2.2 different machine learning algorithms were introduced. In this work the choice of using the **Kernel Support Vector Machine** is made. Indeed, this type of machine learning algorithm is quite simple and not computationally expensive, as the classification is made thanks to distance computation which is less complexe than the numerous calculations needed for other algorithm based on Neural network approach. Moreover, Elangovan et al. [17] have shown good results in machine vibration analysis using Kernel Support Vectors.

4.4.2 Feature selection

In order to train the Kernel Support Vector Machine, so features have to be extracted from the sample. Elangovan et al. [17] have used, mean, standard error, median, standard devi-

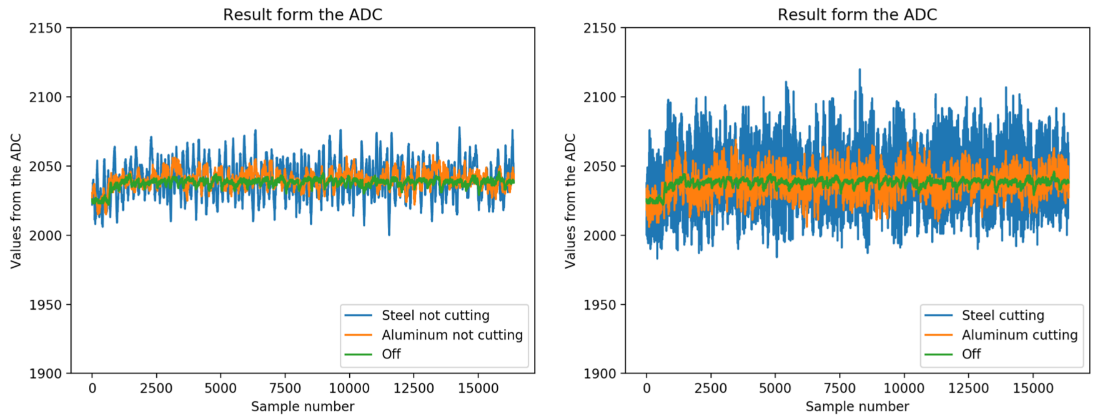


Figure 4.12: The 5 samples for the classes.

ation, sample variance, kurtosis, skewness, range, minimum and maximum. In this work the selected features are:

- the mean of the sample
- the median
- the standard deviation
- the variance
- the minimum and the maximum
- the first 3 major frequencies of the Fast Fourier Transform and their associated amplitudes

4.4.3 Preprocessing

The data sets from the different cuts have to be preprocessed in order to extract the features selected in the previous section. To that end, a python script is used in a given the data folder of one class. This script computes the features for all .csv file in this folder and returns the result in the form of a new .csv. Figure 4.13 presents the functioning, the inputs and outputs of preprocessing.py (code in Appendix G)

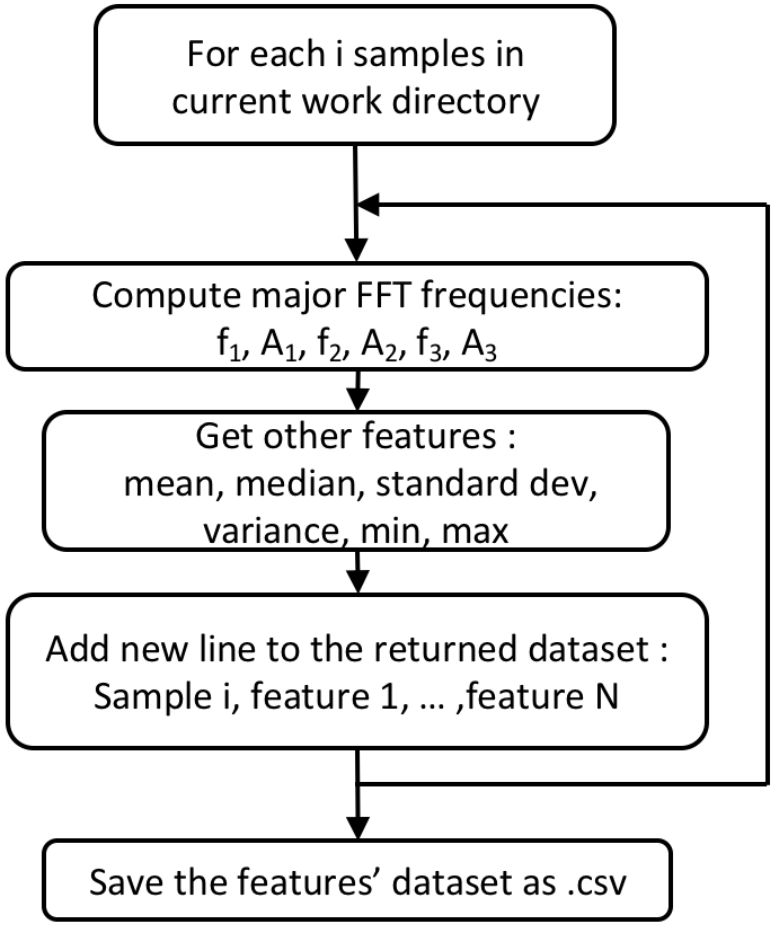


Figure 4.13: Preprocessing flow chart

Once all the data sets are created they are randomly concatenated in one single data set of size $N_s \cdot N_c$ rows by $N_f + 2$ columns (for the index and the corresponding class of the sample). This data set is going to be split into a training set and a test set to train the algorithm in the next part.

4.5 Trainning and deployment

This part presents the training of the Kernel Support Vector Machine algorithm, then it is exported to the BeagleBone Black. Finally, the main application functioning is detailed.

4.5.1 Training of the algorithm

The data set is imported and split into a 80% training set and a 20% test set; the code to train the algorithm is presented in appendix H. Different types of the kernels are used, and the prediction is evaluated on the test set. The results are presented in table 4.4. The detailed confusion matrix and statistics are presented in appendix I.

Table 4.4: Result on the test set for different kernels

kernel	training duration	avg precision
linear	4s	0.99
rbf	3s	0.92
sigmoid	3s	0.04
poly	∞	NA

The results for the linear kernel are far better than for other kernels. This maybe a sign of overtraining; nevertheless, this is the type of kernel that is chosen for the rest of the study.

4.5.2 Export classifier and deployment on the BeagleBone Black

The python classifier object trained in the previous section is converted into a binary object using the pickle method. However, the data management library Pandas and the machine learning library scikitlearn were not successfully installed on the BeagleBone Black. The found solution consists in using another installation method than the recommended one: instead of using *pip* or *apt–get install* tools, miniconda was downloaded on the BeagleBone board. This enabled the use of *conda install* command and finally, old version of the pandas and scikitlearn libraries were successfully installed on the BeagleBone Black.

4.5.3 Main Application Code

A specific code needs to be written for the deployment phase. It has to load the classifier object from the pickle binary file then perform the acquisition of the sample of 16384 points, extract the features out of this sample, feed those features into the classifier and return the result of the classification. The figure 4.14 presents the flow chart of this code (Appendix 6).

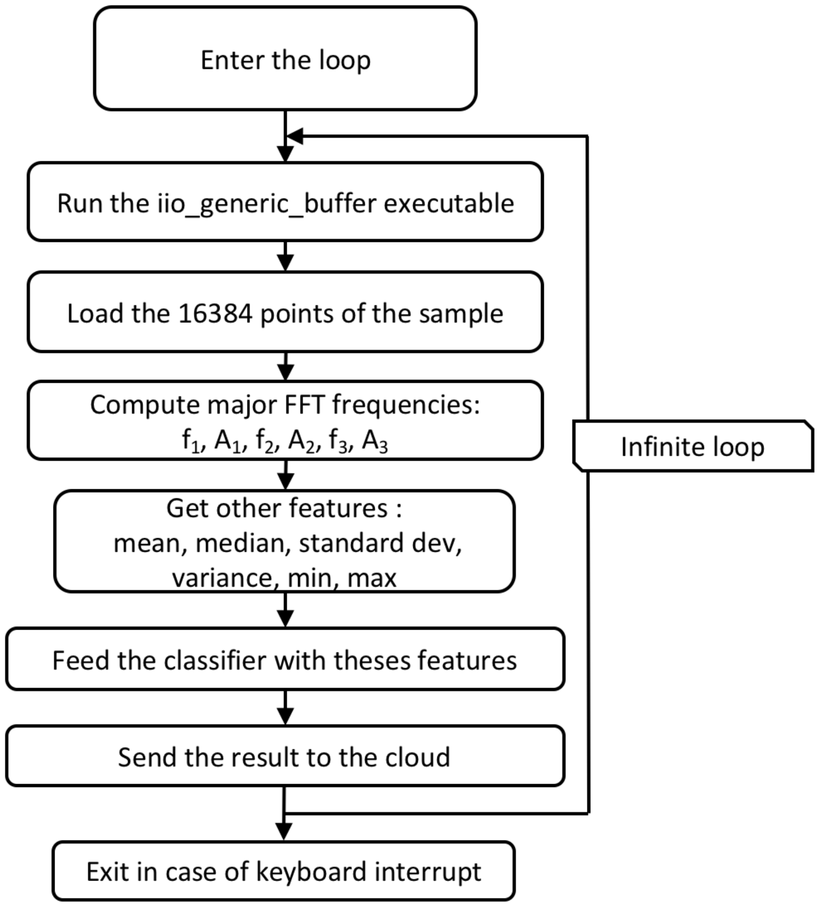


Figure 4.14: The main application flow chart

4.6 Architecture validation and Classification results

The final system is placed on the band saw and tested on an aluminum radiator. The sensor is put at the same place as for the data acquisition phase; the BeagleBone is powered and the main script is launched. The process worked correctly over more than 200 samples, and every sample was acquired and analyzed in less than 1 second. **Over thoses 200 samples, 71.5% where correctly predicted to be a cut of aluminum, 27.5% were wrongly**

predicted to be a from the vibration the band saw running for steel but not cutting, and 1% were predicted to be from the band saw running for aluminum but not cutting. Figure 4.15 presents the setup for the validation of the architecture. Other tests were conducted on an aluminum rod, an aluminum plate, a steel rod and steel plate. For the aluminum rod and the steel plate, the precision of the algorithm is around 95%; however, for the other shapes, the precision drops to 75%.

Figure 4.15: The experimental setup for testing on the radiator

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Contribution of this Thesis

This thesis contributes to 3 different topics: deterministic data acquisition, deployment of machine learning algorithms on ARM microprocessors and the development of a single board system for machine vibration monitoring, indeed:

- the use of a Linux power microprocessor for deterministic data acquisition with the Industrial In/Out subsystem is presented. This subsystem was successfully used to sample the Analog to Digital Converter of the BeagleBone Black.
- A work on the use of the Process Real-Time Units was accomplish. Even if this solution was finally discontinued, the written documentation has already been used several times by the BeagleBone community. It explains how to enable the 2 process real-time units from the Linux user space and to transfer data between the ARM-Cortex A8 and the process real-time units.
- the use of trained machine learning algorithms was demonstrated on the BeagleBone black. This work, even if imperfect, constitutes a proof of concept and opens the door to more advanced systems using machine learning techniques on embedded systems.
- Finally, this thesis presents a low-cost monitoring system that can be used for real-time vibration monitoring.

5.2 Limitations of the study and recommendations

Most of this work consisted in the development of the low cost and smart system for vibration analysis. However, the tests conducted in real conditions have demonstrated a high influence of the form of the workpiece on the predicted class. This is certainly linked to the training samples used in this work, a data set that includes different shapes of workpieces for the same material should be used to train the algorithm. Depending on the new influence of the shape, two cases can be identified:

- if the influence of the form of the workpiece appears to be less important, then the current classification method can be used.
- if the influence of the shape is still very important, then the classes should be modified in order to takes the different possible shape of the workpiece into account.

Furthermore, the study has been limited to only two materials; it could be interesting to train the model with other materials to see if the KSVM classifier with a linear kernel is still efficient. This highlights the issue of data acquisition for Machine Learning algorithm training: the first classes attributed to the training set cannot be labeled automatically; the classes of these samples were hard coded in the acquisition script. Then, an important improvement to this system could be to add a user interface where a non-expert employee can easily select a label for the cut that he is going to perform. This way, the training set could be easily generated without requiring an expert to stand by the machine during all the data acquisition.

Finally, only one sensor has been used for this study. More sensors could be used such as a microphone. Since the sound of the cut greatly changed between materials, it may be interesting to couple a microphone with the accelerometer.

5.3 Conclusion

This thesis presents the development of a low cost smart device for machine vibration analysis. The primary goal was to implement real-time data processing on an embedded system using machine learning techniques. This approach aims to meet the need of a more distributed architecture for real-time decision making in the context of the Industry 4.0. It also avoids sending significant amounts of data to the cloud, simultaneously reducing the bandwidth required and improving the safety and security of the system. First, a variety of microprocessors were evaluated in order to find the most promising board and programming technique for the project. Then, a deterministic data acquisition was performed measuring a band saw cutting different materials, using the Linux Industrial In/Out kernel subsystem. After the samples were acquired, a Linear Kernel Support Vector Machine algorithm was trained and tested. This classification algorithm was exported to the embedded system and tests in real condition were carried out showing good results. The results of the classification were found to be very sensitive to the geometry of the work piece. Finally, areas for future work and several ways to meliorate this system have been suggested.

Appendices

APPENDIX A

EAGLE FILE FOR THE BEAGLEBONE BLACK CAPE

A.1 The front side of the BeagleBone Cape

A.2 The back side of the BeagleBone Cape

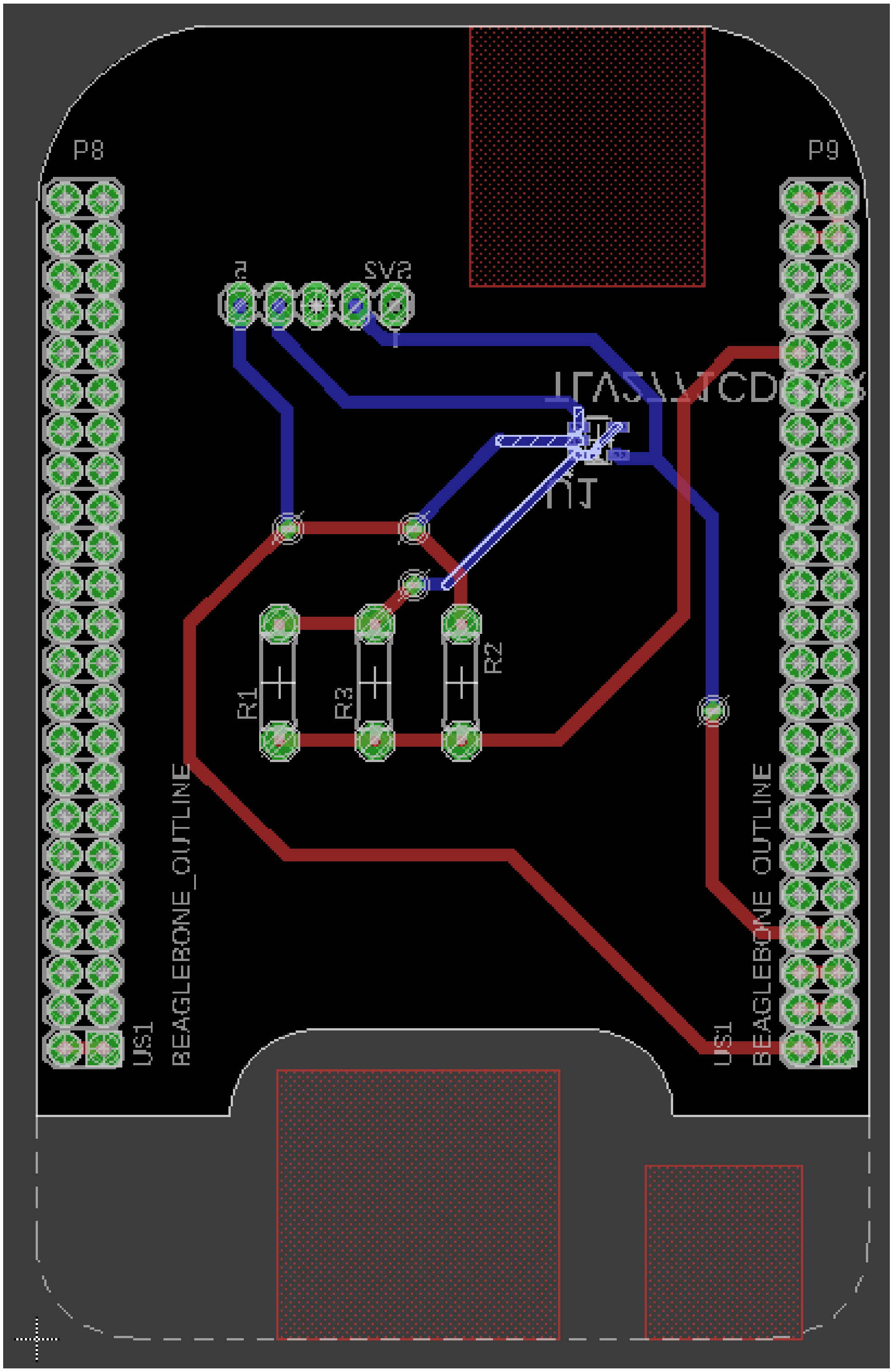


Figure A.1: The front side of the BeagleBone Cape

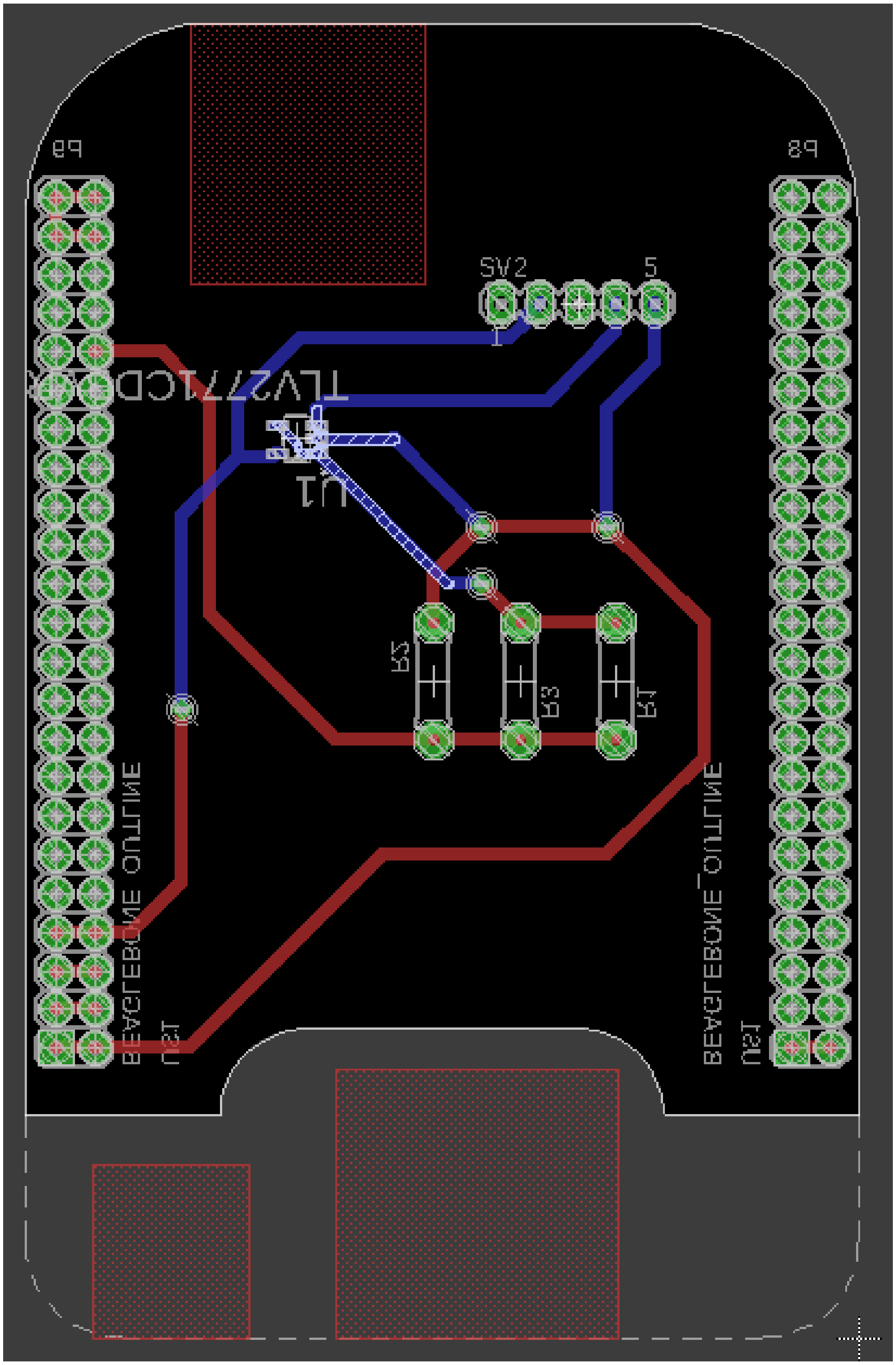


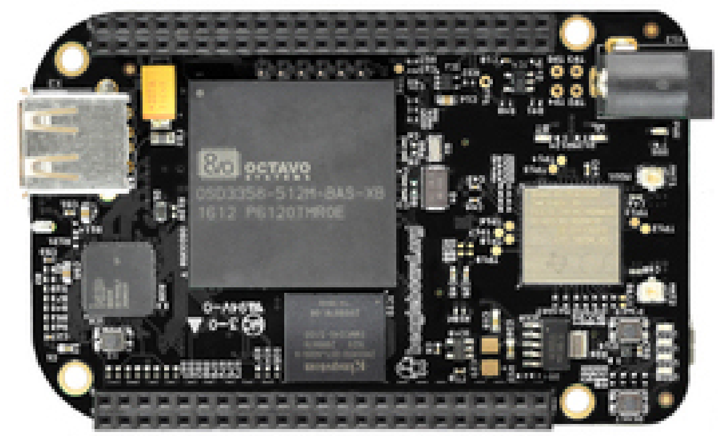
Figure A.2: The back side of the BeagleBone Cape

APPENDIX B

PRU TUTORIAL

Using the PRUs and RPMsg

BeagleBone™ Black Wireless Linux Debian 4.9.45-ti-r57



Pierrick Rauby
Master Thesis Student

Last revision: July 24, 2018

Contents

Chapter 1

Introduction

The BeagleBone™ Black is a low-cost development platform powered by an AM335x 1GHz ARM® Cortex-A8, among its different features, the AM335x presents two Process Real Time Units (PRU). For my master thesis I will need to use those two micro-controllers in order to acquire data from an accelerometer, it took some time to enable the PRU and the communication framework: RPMsg. The purpose of this document, is to explain the method followed to enable those embedded micro-controllers and the framework.

Zubeen Tolani and Gregory Raven are acknowledged for the very complete documentation they have provided about the PRUs and the RPMsg framework which can be found here:

- BeagleScope repository on GitHub from Zubeen Tolani
- PRU ADC repository on GitHub from Gregory Raven

Chapter 2

Hardware presentation

For this project the board used is a *BeagleBone™ Black wireless* powered by an *Octavo Systems OSD3358* which characteristics are :

- 512 MB DDR3 RAM
- 4GB 8-bit eMMC on board flash storage
- 3D graphic accelerator
- Neon floating-point accelerator
- *2 PRUs : 32-bit microcontrollers*

The software used is the Debian image: *Linux Beaglebone™ 4.9.45-ti-r57*

As explained is the introduction the idea of the project is to use the PRUs to acquire data from a sensor and send them to the ARM® Cortex of the BeagleBone™. But what are the PRUs and the ARM®? Basically, the Ocotovo contains the TI AM335X chip which itself contains:

- 1 ARM® Cortex®-A8: This is the part of the chip that runs the Linux operating system. This microprocessor as a "computer" processor is not able to carry out real-time operations.
- 2 Process Real-time Units (PRU) that are microcontrollers such as Arduino®/Teensy ones. It means that they are able to execute real-time processing, then the *programmable nature of the PRU, along with its access to pins, events and all system on chip (SoC) resources, provides flexibility in implementing fast real-time responses, specialized data handling operations* [TexasInstruments, 2017]. Thus, PRUs are very useful when it comes to carry out time critical operations such as fast data acquisitions.

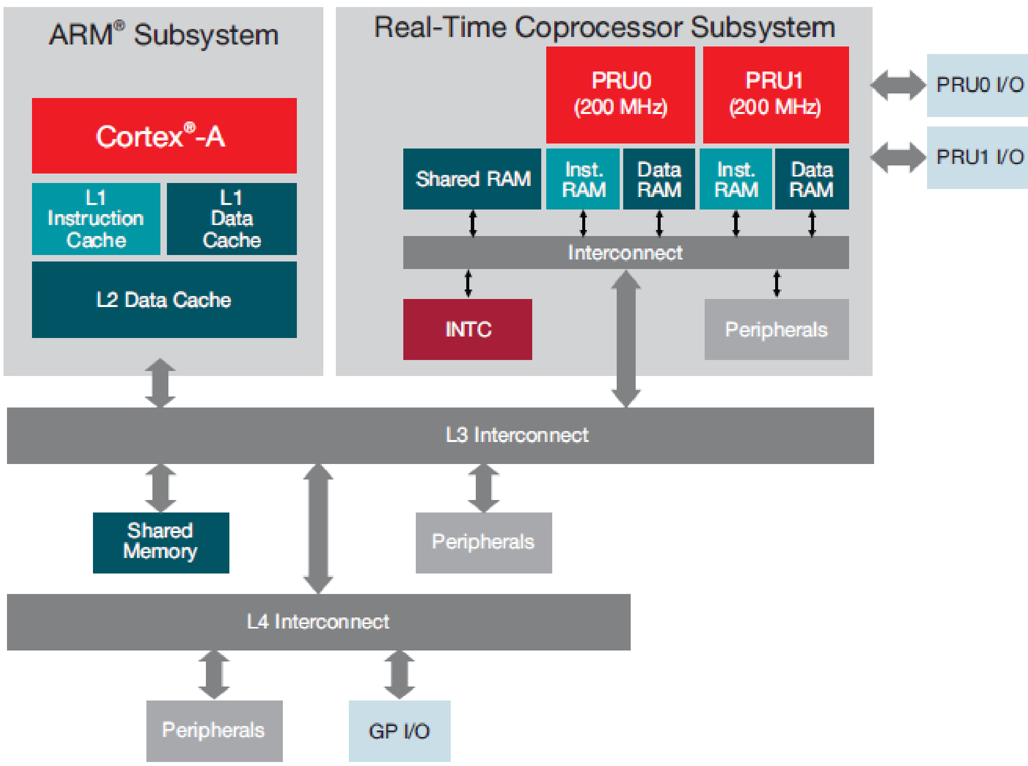


Figure 2.1: Architecture of the AM335x with Cortex®-A8 and the 2 PRUs [TexasInstrument, 2017a]

Chapter 3

Enabling the PRUs

In this chapter, the setup of the PRU is explained. The different steps are based on the work of [Tolani, 2016], who presents a very complete set of instructions in order to setup the PRUs for *Debian 4.4.12-ti-r31*, basically his work is adapted here for *Debian 4.9.45-ti-r57.*.

3.1 Setting up the PRUs

3.1.1 Disabling the HDMI cape and loading the PRU overlay

The PRUs have access to many pins on the BeagleBone™, however some pins are also used by the HDMI. Thus, the HMDI must be disabled before using the PRUs [Yoder, 2017]. In order to do so we are going to disable the loading of the device tree corresponding to the HDMI.

Remark: *The Device Tree (DT), and Device Tree Overlay are a way to describe hardware in a system. An example of this would be to describe how the UART or HDMI interacts with the system, which pins, how they should be mixed, the device to enable, and which driver to use* [Cooper, 2015].

First of all, you need to SSH into the BeagleBone™ Black as root, then navigate to the uEnv.txt file by typing in:

```
cd /boot/
nano uEnv.txt
```

Then the uEnv.txt file should appear as in figure 3.1:

```
#Docs: http://elinux.org/Beagleboard:U-boot_partitioning_layout_2.0

uname_r=4.9.45-ti-r57
#uuid=
#dtb=


###U-Boot Overlays###
###Documentation: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian#U-Boot_Overlays
###Master Enable
```

Figure 3.1: uEnv.txt

In this file, you should go down to the section,

```
###Disable auto loading of virtual capes (emmc/video/wireless/adc)
```

and uncomment the two lines as shown below, this avoids the loading of the HDMI overlays at boot time:

```
###Disable auto loading of virtual capes (emmc/video/wireless/adc)
#disable_uboot_overlay_emmc=1
disable_uboot_overlay_video=1
disable_uboot_overlay_audio=1
#disable_uboot_overlay_wireless=1
#disable_uboot_overlay_adc=1
###
```

In the same document we are going to ask for the loading of the PRUSS overlay at boot time, scroll down to the section:

```
###PRUSS OPTIONS
###pru_rproc (4.4.x-ti kernel)
```

change these lines:

```
###PRUSS OPTIONS
###pru_rproc (4.4.x-ti kernel)
#uboot_overlay_pru=/lib/firmware/AM335X-PRU-RPROC-4-4-TI-00A0.dtbo
###pru_uio (4.4.x-ti & mainline/bone kernel)
uboot_overlay_pru=/lib/firmware/AM335X-PRU-UIO-00A0.dtbo
###
```

to:

```
###PRUSS OPTIONS
###pru_rproc (4.4.x-ti kernel)
uboot_overlay_pru=/lib/firmware/AM335X-PRU-RPROC-4-9-TI-00A0.dtbo
###pru_uio (4.4.x-ti & mainline/bone kernel)
#uboot_overlay_pru=/lib/firmware/AM335X-PRU-UIO-00A0.dtbo
###
```

3 modifications: 1 uncomment, 1 comment and the 4-4-TI-00A0.dtbo becomes 4-9-TI-00A0.dtbo

Finally, just reboot the board. The HDMI capes should be disabled, so we have access to the different PINs of the board with the PRU, Figure 3.2 presents the PIN for the Header 8 (more details on appendix A and B).

3.1.2 Installing GCC compiler

Since the PRUs are based on TI's proprietary architecture [Tolani, 2016], we have to compile the C code that we want to execute with a compiler. In this project GCC is used.

```
cd
wget -c http://software-dl.ti.com/codegen/esd/cgt_public_sw/PRU/2.1.2/
                    ti_cgt_pru_2.1.2_armlinuxa8hf_busybox_installer.sh
chmod +x ti_cgt_pru_2.1.2_armlinuxa8hf_busybox_installer.sh
./ti_cgt_pru_2.1.2_armlinuxa8hf_busybox_installer.sh
cd
rm ti_cgt_pru_2.1.2_armlinuxa8hf_busybox_installer.sh
```

3.1.3 Creating the symbolic links between folders

Then, some symbolic links have to be created:

```
cd /usr/share/ti/cgt-pru/
mkdir bin
cd
ln -s /usr/bin/clpru /usr/share/ti/cgt-pru/bin/clpru
ln -s /usr/bin/lnkpru /usr/share/ti/cgt-pru/bin/lnkpru
```

P8_20	33	0x884/084	63	GPIO1_31	gpio1[31]	pr1_pru1_pru_r31_13	pr1_pru1_pru_r30_13
P8_21	32	0x880/080	62	GPIO1_30	gpio1[30]	pr1_pru1_pru_r31_12	pr1_pru1_pru_r30_12
P8_22	5	0x814/014	37	GPIO1_5	gpio1[5]		
P8_23	4	0x810/010	36	GPIO1_4	gpio1[4]		
P8_24	1	0x804/004	33	GPIO1_1	gpio1[1]		
P8_25	0	0x800/000	32	GPIO1_0	gpio1[0]		
P8_26	31	0x87c/07c	61	GPIO1_29	gpio1[29]		
P8_27	56	0x8e0/0e0	86	GPIO2_22	gpio2[22]	pr1_pru1_pru_r31_8	pr1_pru1_pru_r30_8
P8_28	58	0x8e8/0e8	88	GPIO2_24	gpio2[24]	pr1_pru1_pru_r31_10	pr1_pru1_pru_r30_10
P8_29	57	0x8e4/0e4	87	GPIO2_23	gpio2[23]	pr1_pru1_pru_r31_9	pr1_pru1_pru_r30_9
P8_30	59	0x8ec/0ec	89	GPIO2_25	gpio2[25]		
P8_31	54	0x8d8/0d8	10	UART5_CTSN	gpio0[10]	uart5_ctsn	
P8_32	55	0x8dc/0dc	11	UART5_RTSN	gpio0[11]	uart5_rtsn	
P8_33	53	0x8d4/0d4	9	UART4_RTSN	gpio0[9]	uart4_rtsn	
P8_34	51	0x8cc/0cc	81	UART3_RTSN	gpio2[17]	uart3_rtsn	
P8_35	52	0x8d0/0d0	8	UART4_CTSN	gpio0[8]	uart4_ctsn	
P8_36	50	0x8c8/0c8	80	UART3_CTSN	gpio2[16]	uart3_ctsn	
P8_37	48	0x8c0/0c0	78	UART5_TXD	gpio2[14]	uart2_ctsn	
P8_38	49	0x8c4/0c4	79	UART5_RXD	gpio2[15]	uart2_rtsn	
P8_39	46	0x8b8/0b8	76	GPIO2_12	gpio2[12]	pr1_pru1_pru_r31_6	pr1_pru1_pru_r30_6
P8_40	47	0x8bc/0bc	77	GPIO2_13	gpio2[13]	pr1_pru1_pru_r31_7	pr1_pru1_pru_r30_7
P8_41	44	0x8b0/0b0	74	GPIO2_10	gpio2[10]	pr1_pru1_pru_r31_4	pr1_pru1_pru_r30_4
P8_42	45	0x8b4/0b4	75	GPIO2_11	gpio2[11]	pr1_pru1_pru_r31_5	pr1_pru1_pru_r30_5
P8_43	42	0x8a8/0a8	72	GPIO2_8	gpio2[8]	pr1_pru1_pru_r31_2	pr1_pru1_pru_r30_2
P8_44	43	0x8ac/0ac	73	GPIO2_9	gpio2[9]	pr1_pru1_pru_r31_3	pr1_pru1_pru_r30_3
P8_45	40	0x8a0/0a0	70	GPIO2_6	gpio2[6]	pr1_pru1_pru_r31_0	pr1_pru1_pru_r30_0
P8_46	41	0x8a4/0a4	71	GPIO2_7	gpio2[7]	pr1_pru1_pru_r31_1	pr1_pru1_pru_r30_1

Figure 3.2: P8 header and corresponding PRU [Molloy, 2014]

Finally, we want that "*PRU_CGT*" to point to the "*/usr/share/ti/cgt-pru/*":

```
export PRU_CGT=/usr/share/ti/cgt-pru
```

Because we want this last command to be executed every time we boot the Beaglebone™:

```
cd
nano ~/.bashrc
```

and add this:

```
export PRU_CGT=/usr/share/ti/cgt-pru
```

Then save and quit and reboot.

3.2 Testing the PRUs

Now that everything is ready we can test the PRU with a "hello world!" example in which a small LED is triggered with the PRU. Let's create a small circuit with the LED and two resistors and copy the code testing codes on the BeagleBone™.

3.2.1 Hardware

The circuit used to test the PRU is presented in figure 3.3. Pin P8_45 is used as the output pin and pin P8_1 is connected to the ground of the circuit.

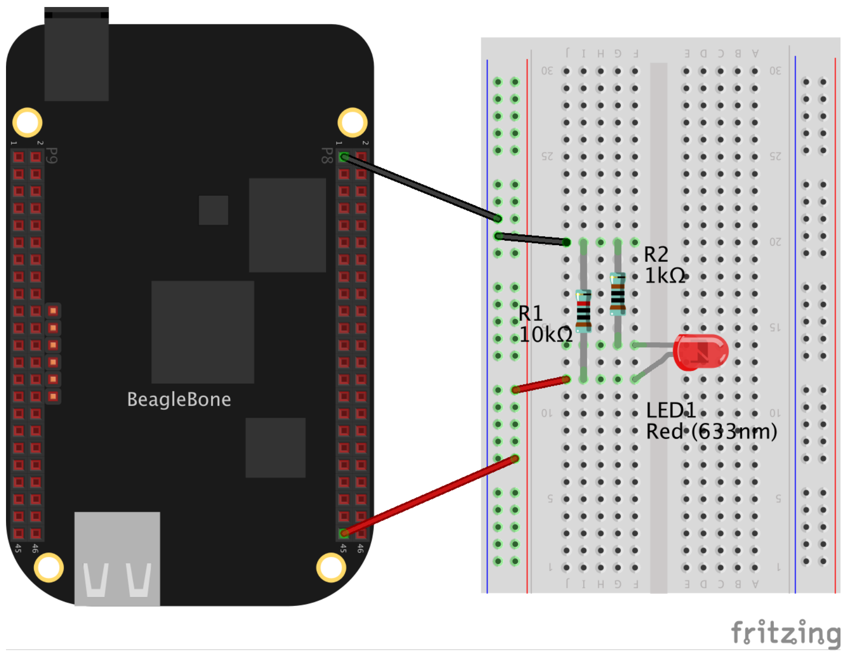


Figure 3.3: The circuit for Hello_PRU program

3.2.2 Code

Now that the hardware is ready, let's copy the code. First of all, go back to the "*/root*" folder of the BeagleBone™:

```
cd
```

And create a new folder "Hello_PRU":

```
mkdir Hello_PRU
```

In this folder we are going to put 5 files and 2 folders:

- Hello_PRU.c
- AM335x_PRU.cmd
- resource_table_empty.h
- Makefile
- deploy.sh
- lib which contains some needed libraries
- include which contains resource files for the different TI processors

3.2.2.1 Hello_PRU.c

This is the C code that is going to make our LED blink.

Lines 38 to 40 correspond to the inclusion of needed files. Lines 42 and 43 correspond to the declaration of two important registers, _R30 and _R31.
In the main loop (from line 45 to the end) , the volatile "gpio" is used to toggle the value of the _R30 between 0x000F and 0x0000, waiting between each toggling thanks to the "_delay_cycles()" function (which is an intrinsic compiler function [Tolani, 2016]).
Remark: The compiler would not allow any variable other than _R31 and _R30 to be of the "register" type, and the compiler does not allow to access any of the 29-R0 registers of the PRU [Tolani, 2016].

```c
#include <stdint.h>
#include <pru_cfg.h>
#include "resource_table_empty.h"

volatile register uint32_t __R30;
volatile register uint32_t __R31;

void main(void)
{
  volatile uint32_t gpio;

  /* Clear SYSCFG[STANDBY_INIT] to enable OCP master port */
  CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;

  /* Toggle GPO pins TODO: Figure out which to use */
  gpio = 0x000F;

  /* TODO: Create stop condition, else it will toggle indefinitely */
  while (1) {
    __R30 ^= gpio;
    __delay_cycles(100000000);
  }
}
```

Figure 3.4: Hello_PRU.c code [Tolani, 2016]

3.2.2.2 AM335x_PRU.cmd

PRUs are pretty simple processing cores, but the PRUSS system is highly integrated and provides the PRU a rich set of peripherals. All these peripherals inside the PRUSS are at different address locations and they need to be configured by the Linux kernel at the time of firmware loading onto the PRUs. The "AM335x_PRU.cmd" file provides a mapping to the linker, from different sections of code, to different memory locations inside the PRUSS. [Tolani, 2016] Thus this file is a linker command file that is used for linking PRU programs built with the C compiler and the resulting .out file on an AM335x device. Basically, you will need this file every time you create a PRU code such as the one above and compile it with GCC.

3.2.2.3 resource_table_empty.h

This empty resource table is needed by the "*AM335x_PRU.cmd*", it is used by Remoteproc, on the host-side to allocate reserved/resources. Since we do not use Remoteproc for the moment (but we will later) we just give an empty file to "*AM335x_PRU.cmd*".

3.2.2.4 Makefile

This file is going to invoke the GCC compiler, to give the location of the resources needed to compile Hello_PRU.c into the "*.out*" file.

3.2.2.5 deploy.sh

This is a bash script that is going to clean the project and to call the Makefile. Once the compilation is finish, deploy.sh copy the resulting file "*.out*" from the "*/gen/* folder to into "*/lib/firmware/am335x-pru1-fw*" folder. This last folder is very important, because the PRU1 is kicked off, it is going to execute the "*.out*" file placed in this folder (the corresponding folder for PRU0 is */lib/firmware/am335x-pru0-fw*).

3.2.3 Running the example

Now, everything is ready to test the PRU setup, you just have to go in the "*Hello_PRU*" folder and enter the command:

```
sh deploy.sh
```

The "*deploy.sh*" script is run, calls the "*MAKEFILE*", places the result of the compilation and kicks of the PRU. Finally, the LED should be blinking on PIN P8_45.

Chapter 4

RPMsg

The next step is to enable communication between the PRUs and the ARM®Cortex. This will be very useful when it comes to send data collected with the PRUs.
The different steps are based on the work of [Raven, 2016], who presents a very complete set of instructions in order to enable the RPMsg framework in his project: *Using the Beaglebone™ Green Programmable Real-Time Unit with the Remoteproc and Remote Messaging Framework to Capture and Play Data from an ADC.*

4.1 Presentation of RPMsg

TI explains it better than I do:

> *RPMsg is a message passing mechanism that requests resources through Remoteproc and builds on top of the virtio framework. Shared buffers are requested through the resource_table and provided by the Remoteproc module during PRU firmware loading. The shared buffers are contained inside a vring data structure in DDR memory. There are two vrings provided per PRU core, one vring is used for messages passed to the ARM® and the other vring is used for messages received from the ARM®. System level mailboxes are used to notify cores (ARM® or PRU) when new messages are waiting in the shared buffers.*

[TexasInstrument, 2017b]

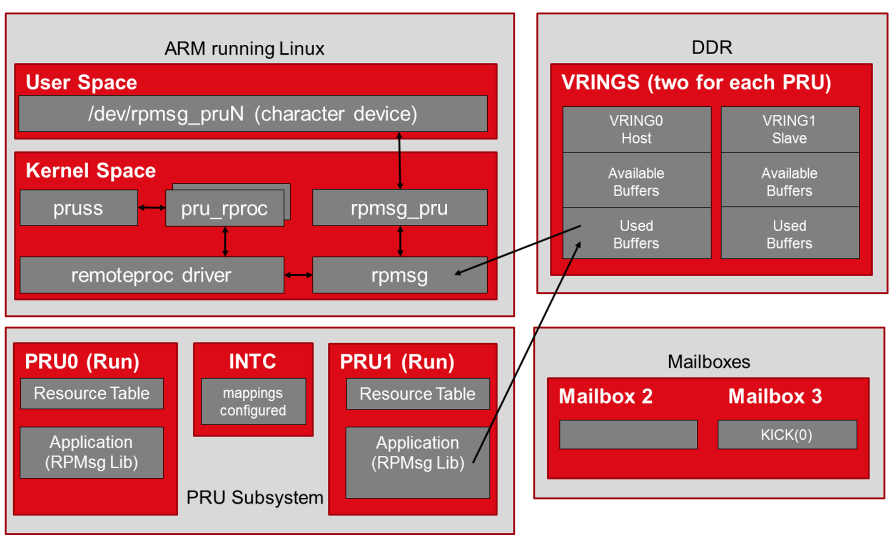


Figure 4.1: Interaction between the ARM®and the PRUs when using RPMsg [TexasInstrument, 2017b]

As explained above, RPMsg uses Remoteproc to transfer messages between the PRUs and the ARM®. Actually, Remoteproc has already been setup in the Chapter 3 when we have loaded the following device tree :

```
uboot_overlay_pru=/lib/firmware/AM335X-PRU-RPROC-4-9-TI-00A0.dtbo
```

Now we are going to enable the RPMsg mechanism.

4.2 Setup

We are going to recompile some device trees:

```
cd /opt/source/bb.org-overlays/
make
make install
```

Then a new device tree must be added when we boot the Beaglebone™:

```
cd
nano /boot/uEnv.txt
```

Go to the section:

```
###Custom Cape
```

and add the following line:

```
###Custom Cape
dtb_overlay=/lib/firmware/am335x-boneblack.dtbo
```

Then save, quit the file and reboot the BeagleBone™ Black. In order to verify that everything is ready, once the board is on and after few seconds you can go to:

```
cd /sys/bus/platform/devices
ls
```

In this folder you should be able to see:

```
4a300000.pruss
4a320000.intc
4a334000.pru0
4a338000.pru1
```

If yes, then everything is ok.

4.3 Testing

Now we are going to use the RPMsg framework with a small example in which we are going to send a "Hi PRU" message from the ARM®to the PRU, which is going to answer: "Hi Cortex-A8". Go back to the *"/root"* folder and create a new folder:

```
cd
mkdir Test_RPMsg
```

this folder will contain the code for the ARM®a nested folder: *"PRU_codes"*, let's create this last folder:

```
cd Test_RPMsg
mkdir PRU_codes
```

4.3.1 Code for the Cortex-A8

Inside the *"Test_RPMsg"* folder create these files :

- deploy_echo_ARM.sh
- rpmsg_pru_user_space_echo.c

4.3.1.1 deploy_echo_ARM.sh

It is only a bash script that is going to compile *rpmsg_pru_user_space_echo.c* and execute it.

4.3.1.2 rpmsg_pru_user_space_echo.c

This code is going to be executed by the Cortex-A8. It will open the device character for PRU1, send 10 "Hello PRU!" messages through the RPMsg channel and read the answer into the device character.

```c
#define NUM_MESSAGES    10
#define DEVICE_NAME   "/dev/rpmsg_pru31"

int main(void)
{
  struct pollfd pollfds[1];
  int i;
  int result = 0;

  /* Open the rpmsg_pru character device file */
  pollfds[0].fd = open(DEVICE_NAME, O_RDWR);

  /*
   * If the RPMsg channel doesn't exist yet the character device
   * won't either.
   * Make sure the PRU firmware is loaded and that the rpmsg_pru
   * module is inserted.
   */
  if (pollfds[0].fd < 0) {
    printf("Failed to open %s\n", DEVICE_NAME);
    return -1;
  }

  /* The RPMsg channel exists and the character device is opened */
  printf("Opened %s, sending %d messages\n\n", DEVICE_NAME, NUM_MESSAGES);

  for (i = 0; i < NUM_MESSAGES; i++) {
    /* Send 'hello world!' to the PRU through the RPMsg channel */
    result = write(pollfds[0].fd, "hello PRU!", 10);
    if (result > 0)
      printf("Message %d: Sent to PRU\n", i);

    /* Poll until we receive a message from the PRU and then print it */
    result = read(pollfds[0].fd, readBuf, 13);
    if (result > 0)
      printf("Message %d received from PRU:%s\n\n", i, readBuf);
  }

  /* Received all the messages the example is complete */
  printf("Received %d messages, closing %s\n", NUM_MESSAGES, DEVICE_NAME);

  /* Close the rpmsg_pru character device file */
  close(pollfds[0].fd);

  return 0;
}
```

Figure 4.2: Main loop of the rpmsg_pru_user_space_echo code

4.3.2 Code for the PRU

Then you will put a file and 4 folders into the *"PRU_codes"* folder, those codes are going to be executed on PRU0 and PRU1:

- deploy_echo.sh
- the *"lib"* folder which contains some needed libraries
- the *"include"* folder which contains resources files for the different TI processors
- PRU_Halt which contains all needed codes for PRU0:
  - AM335x_PRU.cmd
  - main.c
  - Makefile
  - resource_table_empty.h
- PRU_RPMsg_Echo_Interrupt1, which contains the codes for PRU1:
  - AM335x_PRU.cmd
  - main.c
  - Makefile
  - resource_table_1.h

4.3.2.1 deploy.sh

As for the Cortex-A8 folder, this is a bash script that computes the codes for both PRU and that launches them.

4.3.2.2 PRU_Halt

In order to avoid any problems we are going to stop the PRU0 as soon as we start it, this is the role of the *"__Halt()"* function in main.c provided by [TexasInstrument, 2014] in the Software Support Package.

```c

#include <stdint.h>
#include "resource_table_empty.h"

int main(void)
{
  __halt();
}

```

Figure 4.3: Main loop of the PRU_Halt code, *"__Halt()"* function stops PRU0

4.3.2.3 PRU_RPMsg_Echo_Interrupt1

This is the interesting part of the PRU codes. As we did for the section 3.4, we need the *"AM335x_PRU.cmd"* and *"resource_table_1.h"* and a *"Makefile"*. The main.c code is presented in figure 4.4.

After creating the device character *"rpmsg_pru31"* for the communication with the Cortex-A8, the PRU is going to wait for receiving a message from the Cortex. Each time it receives a message, the PRU is going to send back a message *"Hello_Cortext-A8!"* using the pru_rpmsg_send() function.

```c
#define VIRTIO_CONFIG_S_DRIVER_OK 4

uint8_t payload[RPMSG_BUF_SIZE];

/*
 * main.c
 */
void main(void)
{
  struct pru_rpmsg_transport transport;
  uint16_t src, dst, len;
  volatile uint8_t *status;

  /* Allow OCP master port access by the PRU so the PRU can read external memories */
  CT_CFG.SYSCFG_bit.STANDBY_INIT = 0;

  /* Clear the status of the PRU-ICSS system event that the ARM will use to 'kick' us */
  CT_INTC.SICR_bit.STS_CLR_IDX = FROM_ARM_HOST;

  /* Make sure the Linux drivers are ready for RPMsg communication */
  status = &resourceTable.rpmsg_vdev.status;
  while (!(*status & VIRTIO_CONFIG_S_DRIVER_OK));

  /* Initialize the RPMsg transport structure */
  pru_rpmsg_init(&transport, &resourceTable.rpmsg_vring0, &resourceTable.rpmsg_vring1, TO_ARM_HOST, FROM_ARM_HOST);

  /* Create the RPMsg channel between the PRU and ARM user space using the transport structure. */
  while (pru_rpmsg_channel(RPMSG_NS_CREATE, &transport, CHAN_NAME, CHAN_DESC, CHAN_PORT) != PRU_RPMSG_SUCCESS);
  while (1) {
    /* Check bit 30 of register R31 to see if the ARM has kicked us */
    if (__R31 & HOST_INT) {
      /* Clear the event status */
      CT_INTC.SICR_bit.STS_CLR_IDX = FROM_ARM_HOST;
      /* Receive all available messages, multiple messages can be sent per kick */
      while (pru_rpmsg_receive(&transport, &src, &dst, payload, &len) == PRU_RPMSG_SUCCESS) {
        /* Echo the message back to the same address from which we just received */
        strcpy((char *)payload, "Hello Cortex-A8!");
        pru_rpmsg_send(&transport, dst, src, payload, 16);
      }
    }
  }
}
```

Figure 4.4: Main loop of the PRU_RPMsg_Echo_Interrupt1code

4.3.3 Starting the project

Once you have placed every file in the Test_RPMsg folder you can start both PRUs and Cortex-A8. For this, go into the PRU_codes folder and execute the deploy_echo.sh script:

```
cd
cd /Test_RPMsg/PRU_codes
sh deploy_echo.sh
```

The go into the Test_RPMsg folder and execute the other bash script:

```
cd
cd /Test_RPMsg
sh deploy_echo_ARM.sh
```

You should see something like in figure 4.5 in the console.

If both examples of sections 3 and 4.1 were run successfully then you are good to go.

```
root@beaglebone:~/Test_RPMsg# sh deploy_echo_ARM.sh
---##########---Compilling C code---##########---
 ---##########---Starting...
Opened /dev/rpmsg_pru31, sending 10 messages

Message 0: Sent to PRU
Message 0 received from PRU:Hello Cortex-A8!

Message 1: Sent to PRU
Message 1 received from PRU:Hello Cortex-A8!

Message 2: Sent to PRU
Message 2 received from PRU:Hello Cortex-A8!

Message 3: Sent to PRU
Message 3 received from PRU:Hello Cortex-A8!

Message 4: Sent to PRU
Message 4 received from PRU:Hello Cortex-A8!

Message 5: Sent to PRU
Message 5 received from PRU:Hello Cortex-A8!

Message 6: Sent to PRU
Message 6 received from PRU:Hello Cortex-A8!

Message 7: Sent to PRU
Message 7 received from PRU:Hello Cortex-A8!

Message 8: Sent to PRU
Message 8 received from PRU:Hello Cortex-A8!

Message 9: Sent to PRU
Message 9 received from PRU:Hello Cortex-A8!

Received 10 messages, closing /dev/rpmsg_pru31
root@beaglebone:~/Test_RPMsg#
```

Figure 4.5: Expected result for the Test_RPMsg folder

Appendix A

PIN Header 8

Pin	$PINS	ADDR	GPIO	Name	Mode7	Mode6	Mode5	Mode4	Mode3	Mode2	Mode1	Mode0	CPU	Notes
P8_01		Offset from:		DGND										Ground
P8_02		44e10800		DGND										Ground
P8_03	6	0x818/018	38	GPIO1_6	gpio1[6]						mmc1_dat6	gpmc_ad6	R9	Allocated emmc2
P8_04	7	0x81c/01c	39	GPIO1_7	gpio1[7]						mmc1_dat7	gpmc_ad7	T9	Allocated emmc2
P8_05	2	0x808/008	34	GPIO1_2	gpio1[2]						mmc1_dat2	gpmc_ad2	R8	Allocated emmc2
P8_06	3	0x80c/00c	35	GPIO1_3	gpio1[3]						mmc1_dat3	gpmc_ad3	T8	Allocated emmc2
P8_07	36	0x890/090	66	TIMER4	gpio2[2]					timer4		gpmc_advn_ale	R7	
P8_08	37	0x894/094	67	TIMER7	gpio2[3]					timer7		gpmc_oen_ren	T7	
P8_09	39	0x89c/09c	69	TIMER5	gpio2[5]					timer5		gpmc_be0n_cle	T6	
P8_10	38	0x898/098	68	TIMER6	gpio2[4]					timer6		gpmc_wen	U6	
P8_11	13	0x834/034	45	GPIO1_13	gpio1[13]	pr1_pru0_pru_r30_15		eQEP2B_in	mmc2_dat1	mmc1_dat5	lcd_data18	gpmc_ad13	R12	
P8_12	12	0x830/030	44	GPIO1_12	gpio1[12]	pr1_pru0_pru_r30_14		EQEP2A_IN	MMC2_DAT0	MMC1_DAT4	LCD_DATA19	GPMC_AD12	T12	
P8_13	9	0x824/024	23	EHRPWM2B	gpio0[23]			ehrpwm2B	mmc2_dat5	mmc1_dat1	lcd_data22	gpmc_ad9	T10	
P8_14	10	0x828/028	26	GPIO0_26	gpio0[26]			ehrpwm2_tripzone_in	mmc2_dat6	mmc1_dat2	lcd_data21	gpmc_ad10	T11	
P8_15	15	0x83c/03c	47	GPIO1_15	gpio1[15]	pr1_pru0_pru_r31_15		eQEP2_strobe	mmc2_dat3	mmc1_dat7	lcd_data16	gpmc_ad15	U13	
P8_16	14	0x838/038	46	GPIO1_14	gpio1[14]	pr1_pru0_pru_r31_14		eQEP2_index	mmc2_dat2	mmc1_dat6	lcd_data17	gpmc_ad14	V13	
P8_17	11	0x82c/02c	27	GPIO0_27	gpio0[27]			ehrpwm0_synco	mmc2_dat7	mmc1_dat3	lcd_data20	gpmc_ad11	U12	
P8_18	35	0x88c/08c	65	GPIO2_1	gpio2[1]	mcasp0_fsr			mmc2_clk	gpmc_wait1	lcd_memory_clk	gpmc_clk_mux0	V12	
P8_19	8	0x820/020	22	EHRPWM2A	gpio0[22]			ehrpwm2A	mmc2_dat4	mmc1_dat0	lcd_data23	gpmc_ad8	U10	
P8_20	33	0x884/084	63	GPIO1_31	gpio1[31]	pr1_pru1_pru_r31_13	pr1_pru1_pru_r30_13			mmc1_cmd	gpmc_be1n	gpmc_csn2	V9	Allocated emmc2
P8_21	32	0x880/080	62	GPIO1_30	gpio1[30]	pr1_pru1_pru_r31_12	pr1_pru1_pru_r30_12			mmc1_clk	gpmc_clk	gpmc_csn1	U9	Allocated emmc2
P8_22	5	0x814/014	37	GPIO1_5	gpio1[5]						mmc1_dat5	gpmc_ad5	V8	Allocated emmc2
P8_23	4	0x810/010	36	GPIO1_4	gpio1[4]						mmc1_dat4	gpmc_ad4	U8	Allocated emmc2
P8_24	1	0x804/004	33	GPIO1_1	gpio1[1]						mmc1_dat1	gpmc_ad1	V7	Allocated emmc2
P8_25	0	0x800/000	32	GPIO1_0	gpio1[0]						mmc1_dat0	gpmc_ad0	U7	Allocated emmc2
P8_26	31	0x87c/07c	61	GPIO1_29	gpio1[29]							gpmc_csn0	V6	
P8_27	56	0x8e0/0e0	86	GPIO2_22	gpio2[22]	pr1_pru1_pru_r31_8	pr1_pru1_pru_r30_8				gpmc_a8	lcd_vsync	U5	Allocated HDMI
P8_28	58	0x8e8/0e8	88	GPIO2_24	gpio2[24]	pr1_pru1_pru_r31_10	pr1_pru1_pru_r30_10				gpmc_a10	lcd_pclk	V5	Allocated HDMI
P8_29	57	0x8e4/0e4	87	GPIO2_23	gpio2[23]	pr1_pru1_pru_r31_9	pr1_pru1_pru_r30_9				gpmc_a9	lcd_hsync	R5	Allocated HDMI
P8_30	59	0x8ec/0ec	89	GPIO2_25	gpio2[25]	pr1_pru1_pru_r31_11	pr1_pru1_pru_r30_11				gpmc_a11	lcd_ac_bias_en	R6	Allocated HDMI
P8_31	54	0x8d8/0d8	10	UART5_CTSN	gpio0[10]	uart5_ctsn		uart5_rxd	mcasp0_axr1	eQEP1_index	gpmc_a18	lcd_data14	V4	Allocated HDMI
P8_32	55	0x8dc/0dc	11	UART5_RTSN	gpio0[11]	uart5_rtsn		mcasp0_axr3	mcasp0_ahclkx	eQEP1_strobe	gpmc_a19	lcd_data15	T5	Allocated HDMI
P8_33	53	0x8d4/0d4	9	UART4_RTSN	gpio0[9]	uart4_rtsn		mcasp0_axr3	mcasp0_fsr	eQEP1B_in	gpmc_a17	lcd_data13	V3	Allocated HDMI
P8_34	51	0x8cc/0cc	81	UART3_RTSN	gpio2[17]	uart3_rtsn		mcasp0_axr2	mcasp0_ahclkr	ehrpwm1B	gpmc_a15	lcd_data11	U4	Allocated HDMI
P8_35	52	0x8d0/0d0	8	UART4_CTSN	gpio0[8]	uart4_ctsn		mcasp0_axr2	mcasp0_aclkr	eQEP1A_in	gpmc_a16	lcd_data12	V2	Allocated HDMI
P8_36	50	0x8c8/0c8	80	UART3_CTSN	gpio2[16]	uart3_ctsn			mcasp0_axr0	ehrpwm1A	gpmc_a14	lcd_data10	U3	Allocated HDMI
P8_37	48	0x8c0/0c0	78	UART5_TXD	gpio2[14]	uart2_ctsn		uart5_txd	mcasp0_aclkx	ehrpwm1_tripzone_in	gpmc_a12	lcd_data8	U1	Allocated HDMI
P8_38	49	0x8c4/0c4	79	UART5_RXD	gpio2[15]	uart2_rtsn		uart5_rxd	mcasp0_fsx	ehrpwm0_synco	gpmc_a13	lcd_data9	U2	Allocated HDMI
P8_39	46	0x8b8/0b8	76	GPIO2_12	gpio2[12]	pr1_pru1_pru_r31_6	pr1_pru1_pru_r30_6		eQEP2_index		gpmc_a6	lcd_data6	T3	Allocated HDMI
P8_40	47	0x8bc/0bc	77	GPIO2_13	gpio2[13]	pr1_pru1_pru_r31_7	pr1_pru1_pru_r30_7	pr1_edio_data_out7	eQEP2_strobe		gpmc_a7	lcd_data7	T4	Allocated HDMI
P8_41	44	0x8b0/0b0	74	GPIO2_10	gpio2[10]	pr1_pru1_pru_r31_4	pr1_pru1_pru_r30_4		eQEP2A_in		gpmc_a4	lcd_data4	T1	Allocated HDMI
P8_42	45	0x8b4/0b4	75	GPIO2_11	gpio2[11]	pr1_pru1_pru_r31_5	pr1_pru1_pru_r30_5		eQEP2B_in		gpmc_a5	lcd_data5	T2	Allocated HDMI
P8_43	42	0x8a8/0a8	72	GPIO2_8	gpio2[8]	pr1_pru1_pru_r31_2	pr1_pru1_pru_r30_2		ehrpwm2_tripzone_in		gpmc_a2	lcd_data2	R3	Allocated HDMI
P8_44	43	0x8ac/0ac	73	GPIO2_9	gpio2[9]	pr1_pru1_pru_r31_3	pr1_pru1_pru_r30_3		ehrpwm0_synco		gpmc_a3	lcd_data3	R4	Allocated HDMI
P8_45	40	0x8a0/0a0	70	GPIO2_6	gpio2[6]	pr1_pru1_pru_r31_0	pr1_pru1_pru_r30_0		ehrpwm2A		gpmc_a0	lcd_data0	R1	Allocated HDMI
P8_46	41	0x8a4/0a4	71	GPIO2_7	gpio2[7]	pr1_pru1_pru_r31_1	pr1_pru1_pru_r30_1		ehrpwm2B		gpmc_a1	lcd_data1	R2	Allocated HDMI
P9 Header	cat $PINS	ADDR +	GPIO NO.	Name	Mode 7	Mode 6	Mode 5	Mode 4	Mode 3	Mode 2	Mode 1	Mode 0	CPU	

EXPLORINGBEAGLEBONE
TOOLS AND TECHNIQUES FOR BUILDING WITH EMBEDDED LINUX

The BeagleBone Black P8 Header

www.ExploringBeagleBone.com

Appendix B

PIN Header 9

Pin	$PINS	ADDR	GPIO	Name	Mode7	Mode6	Mode5	Mode4	Mode3	Mode2	Mode1	Mode0	CPU	Notes
P9_01		44e10000		GND										Ground
P9_02		Offset from:		GND										Ground
P9_03		44e10800		DC_3.3V										250mA Max Current
P9_04				DC_3.3V										250mA Max Current
P9_05				VDD_5V										1A Max Current
P9_06				VDD_5V										1A Max Current
P9_07				SYS_5V										250mA Max Current
P9_08				SYS_5V										250mA Max Current
P9_09				PWR_BUT										5V Level (pulled up PMIC)
P9_10				SYS_RESETn								RESET_OUT	A10	
P9_11	28	0x870/070	30	UART4_RXD	gpio0[30]	uart4_rxd_mux2		mmc1_sdcd	rmii2_crs_dv	gpmc_csn4	mii2_crs	gpmc_wait0	T17	All GPIOs to 4-6mA output
P9_12	30	0x878/078	60	GPIO1_28	gpio1[28]	mcasp0_aclkr_mux3		gpmc_dir	mmc2_dat3	gpmc_csn6	mii2_col	gpmc_be1n	U18	and approx. 8mA on input.
P9_13	29	0x874/074	31	UART4_TXD	gpio0[31]	uart4_txd_mux2		mmc2_sdcd	rmii2_rxerr	gpmc_csn5	mii2_rxerr	gpmc_wpn	U17	
P9_14	18	0x848/048	50	EHRPWM1A	gpio1[18]	ehrpwm1A_mux1		gpmc_a18	mmc2_dat1	rgmii2_td3	mii2_txd3	gpmc_a2	U14	
P9_15	16	0x840/040	48	GPIO1_16	gpio1[16]	ehrpwm1_tripzone_input		gpmc_a16	mii2_txen	rmii2_tctl	gmii2_txen	gpmc_a0	R13	
P9_16	19	0x84c/04c	51	EHRPWM1B	gpio1[19]	ehrpwm1B_mux1		gpmc_a19	mmc2_dat2	rgmii2_td2	mii2_txd2	gpmc_a3	T14	
P9_17	87	0x95c/15c	5	I2C1_SCL	gpio0[5]			pr1_uart0_txd	ehrpwm0_synci	I2C1_SCL	mmc2_sdwp	spi0_cs0	A16	
P9_18	86	0x958/158	4	I2C1_SDA	gpio0[4]			pr1_uart0_rxd	ehrpwm0_tripzone	I2C1_SDA	mmc1_sdwp	spi0_d1	B16	
P9_19	95	0x97c/17c	13	I2C2_SCL	gpio0[13]		pr1_uart0_rts_n	spi1_cs1	I2C2_SCL	dcan0_rx	timer5	uart1_rtsn	D17	Allocated I2C2
P9_20	94	0x978/178	12	I2C2_SDA	gpio0[12]		pr1_uart0_cts_n	spi1_cs0	I2C2_SDA	dcan0_tx	timer6	uart1_ctsn	D18	Allocated I2C2
P9_21	85	0x954/154	3	UART2_TXD	gpio0[3]	EMU3_mux1		pr1_uart0_rts_n	ehrpwm0B	I2C2_SCL	uart2_txd	spi0_d0	B17	
P9_22	84	0x950/150	2	UART2_RXD	gpio0[2]	EMU2_mux1		pr1_uart0_cts_n	ehrpwm0A	I2C2_SDA	uart2_rxd	spi0_sclk	A17	
P9_23	17	0x844/044	49	GPIO1_17	gpio1[17]	ehrpwm0_synco		gpmc_a17	mmc2_dat0	rgmii2_rxdv	gmii2_rxdv	gpmc_a1	V14	
P9_24	97	0x984/184	15	UART1_TXD	gpio0[15]	pr1_pru0_pru_r31_16	pr1_uart0_txd		I2C1_SCL	dcan1_rx	mmc2_sdwp	uart1_txd	D15	
P9_25	107	0x9ac/1ac	117	GPIO3_21	gpio3[21]	pr1_pru0_pru_r31_7	pr1_pru0_pru_r30_7	EMU4_mux2	mcasp1_axr1	mcasp0_axr3	eQEP0_strobe	mcasp0_ahclkx	A14	Allocated mcasp0_pins
P9_26	96	0x980/180	14	UART1_RXD	gpio0[14]	pr1_pru1_pru_r31_16	pr1_uart0_rxd		I2C1_SDA	dcan1_tx	mmc1_sdwp	uart1_rxd	D16	
P9_27	105	0x9a4/1a4	115	GPIO3_19	gpio3[19]	pr1_pru0_pru_r31_5	pr1_pru0_pru_r30_5	EMU2_mux2	mcasp1_fsx	mcasp0_axr3	eQEP0B_in	mcasp0_fsr	C13	Allocated mcasp0_pins
P9_28	103	0x99c/19c	113	SPI1_CS0	gpio3[17]	pr1_pru0_pru_r31_3	pr1_pru0_pru_r30_3	eCAP2_in_PWM2_out	spi1_cs0	mcasp0_axr2	ehrpwm0_synci	mcasp0_ahclkr	C12	Allocated mcasp0_pins
P9_29	101	0x994/194	111	SPI1_D0	gpio3[15]	pr1_pru0_pru_r31_1	pr1_pru0_pru_r30_1	mmc1_sdcd_mux1	spi1_d0		ehrpwm0B	mcasp0_fsx	B13	Allocated mcasp0_pins
P9_30	102	0x998/198	112	SPI1_D1	gpio3[16]	pr1_pru0_pru_r31_2	pr1_pru0_pru_r30_2	mmc2_sdcd_mux1	spi1_d1		ehrpwm0_tripzone	mcasp0_axr0	D12	Allocated mcasp0_pins
P9_31	100	0x990/190	110	SPI1_SCLK	gpio3[14]	pr1_pru0_pru_r31_0	pr1_pru0_pru_r30_0	mmc0_sdcd_mux1	spi1_sclk		ehrpwm0A	mcasp0_aclkx	A13	Allocated mcasp0_pins
P9_32				VADC										1.8 ADC Volt. Ref.
P9_33				AIN4									C8	1.8V input
P9_34				AGND										Ground for ADC
P9_35				AIN6									A8	1.8V input
P9_36				AIN5									B8	1.8V input
P9_37				AIN2									B7	1.8V input
P9_38				AIN3									A7	1.8V input
P9_39				AIN0									B6	1.8V input
P9_40				AIN1									C7	1.8V input
P9_41A	109	0x9b4/1b4	20	CLKOUT2	gpio0[20]	EMU3_mux0	pr1_pru0_pru_r31_16	timer7_mux1	clkout2	tclkin		xdma_event_intr1	D14	Both to P21 of P11
P9_41B		0x9a8/1a8	116	GPIO3_20	gpio3[20]	pr1_pru0_pru_r31_6	pr1_pru0_pru_r30_6	emu3	Mcasp1_axr0		eQEP0_index	mcasp0_axr1	D13	Both to P21 of P11
P9_42A	89	0x964/164	7	GPIO0_7	gpio0[7]	xdma_event_intr2	mmc0_sdwp	spi1_sclk	pr1_ecap0_ecap_capin_apwm_o	spi1_cs1	uart3_txd	eCAP0_in_PWM0_out	C18	Both to P22 of P11
P9_42B		0x9a0/1a0	114	GPIO3_18	gpio3[18]	pr1_pru0_pru_r31_4	pr1_pru0_pru_r30_4		Mcasp1_aclkx	Mcaspo_axr2	eQEP0A_in	Mcasp0_aclkr	B12	Allocated mcasp0_pins
P9_43				GND										- See Pg.50 of the SRM
P9_44				GND										Ground
P9_45				GND										Ground
P9_46	cat		(Mode 7)	GND										Ground
P9	$PINS	ADDR +	GPIO NO.	Name	Mode 7						Mode 1	Mode 0	CPU	Notes

EXPLORINGBEAGLEBONE
TOOLS AND TECHNIQUES FOR BUILDING WITH EMBEDDED LINUX

The BeagleBone Black P9 Header

www.ExploringBeagleBone.com

Bibliography

Justin Cooper. Device tree overlays. https://learn.adafruit.com/introduction-to-the-beaglebone-black-device-tree/overview, 2015.

Derek Molloy. *Exploring BeagleBone: Tools and Techniques for Building with Embedded Linux.* Wiley, 2014. ISBN 1118935128. URL http://www.exploringbeaglebone.com/.

Gregory Raven. Using the beaglebone green programmable real-time unit with the remoteproc and remote messaging framework to capture and play data from an adc. https://github.com/Greg-R/pruadc1, 2016.

TexasInstrument. Pru software support package. https://git.ti.com/pru-software-support-package, 2014.

TexasInstrument. Pru read latencies. http://processors.wiki.ti.com/index.php/PRU_Read_Latencies, 2017a.

TexasInstrument. Pru-icss remoteproc and rpmsg. http://processors.wiki.ti.com/index.php/PRU-ICSS_Remoteproc_and_RPMsg, 2017b.

TexasInstruments. *AM335x and AMIC110 SitaraTM Processors*, 2017.

Zubeen Tolani. Ptp - programming the prus 1: Blinky. https://www.zeekhuge.me/post/ptp_blinky/, 2016.

Mark A. Yoder. Ebc exercise 30 pru via remoteproc and rpmsg. https://elinux.org/EBC_Exercise_30_PRU_via_remoteproc_and_RPMsg, 2017.

APPENDIX C

TI_AM335X_TSADC.H HEADER

```
#ifndef __LINUX_TI_AM335X_TSCADC_MFD_H
#define __LINUX_TI_AM335X_TSCADC_MFD_H

/*
* TI Touch Screen / ADC MFD driver
*
* Copyright (C) 2012 Texas Instruments Incorporated - http://www.ti.com/
* Source modified by Pierrick Rauby
* This program is free software; you can redistribute it and/or
* modify it under the terms of the GNU General Public License as
* published by the Free Software Foundation version 2.
*
* This program is distributed "as is" WITHOUT ANY WARRANTY of any
* kind, whether express or implied; without even the implied warranty
* of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*/

#include <linux/mfd/core.h>

#define REG_RAWIRQSTATUS        0x024
#define REG_IRQSTATUS           0x028
#define REG_IRQENABLE           0x02C
#define REG_IRQCLR              0x030
#define REG_IRQWAKEUP           0x034
#define REG_DMAENABLE_SET       0x038
#define REG_DMAENABLE_CLEAR     0x03c
#define REG_CTRL                0x040
#define REG_ADCFSM              0x044
#define REG_CLKDIV              0x04C
#define REG_SE                  0x054
#define REG_IDLECONFIG          0x058
#define REG_CHARGECONFIG        0x05C
#define REG_CHARGEDELAY         0x060
#define REG_STEPCONFIG(n)       (0x64 + ((n) * 8))
```

```
#define REG_STEPDELAY(n)          (0x68 + ((n) * 8))
#define REG_FIFO0CNT              0xE4
#define REG_FIFO0THR              0xE8
#define REG_FIFO1CNT              0xF0
#define REG_FIFO1THR              0xF4
#define REG_DMA1REQ               0xF8
#define REG_FIFO0                 0x100
#define REG_FIFO1                 0x200

/*      Register Bitfields      */
/* IRQ wakeup enable */
#define IRQWKUP_ENB               BIT(0)

/* Step Enable */
#define STEPENB_MASK              (0x1FFFF << 0)
#define STEPENB(val)              ((val) << 0)
#define ENB(val)                          (1 << (val))
#define STPENB_STEPENB            STEPENB(0x1FFFF)
#define STPENB_STEPENB_TC         STEPENB(0x1FFF)

/* IRQ enable */
#define IRQENB_HW_PEN             BIT(0)
#define IRQENB_EOS                BIT(1)
#define IRQENB_FIFO0THRES         BIT(2)
#define IRQENB_FIFO0OVRRUN        BIT(3)
#define IRQENB_FIFO0UNDRFLW       BIT(4)
#define IRQENB_FIFO1THRES         BIT(5)
#define IRQENB_FIFO1OVRRUN        BIT(6)
#define IRQENB_FIFO1UNDRFLW       BIT(7)
#define IRQENB_PENUP              BIT(9)

/* Step Configuration */
#define STEPCONFIG_MODE_MASK      (3 << 0)
#define STEPCONFIG_MODE(val)      ((val) << 0)
#define STEPCONFIG_MODE_SWCNT     STEPCONFIG_MODE(1)
#define STEPCONFIG_MODE_HWSYNC    STEPCONFIG_MODE(2)
#define STEPCONFIG_AVG_MASK       (7 << 2)
#define STEPCONFIG_AVG(val)       ((val) << 2)
#define STEPCONFIG_AVG_16         STEPCONFIG_AVG(4)
#define STEPCONFIG_XPP            BIT(5)
#define STEPCONFIG_XNN            BIT(6)
```

```
#define STEPCONFIG_YPP             BIT(7)
#define STEPCONFIG_YNN             BIT(8)
#define STEPCONFIG_XNP             BIT(9)
#define STEPCONFIG_YPN             BIT(10)
#define STEPCONFIG_INM_MASK        (0xF << 15)
#define STEPCONFIG_INM(val)        ((val) << 15)
#define STEPCONFIG_INM_ADCREFM     STEPCONFIG_INM(8)
#define STEPCONFIG_INP_MASK        (0xF << 19)
#define STEPCONFIG_INP(val)        ((val) << 19)
#define STEPCONFIG_INP_AN4         STEPCONFIG_INP(4)
#define STEPCONFIG_INP_ADCREFM     STEPCONFIG_INP(8)
#define STEPCONFIG_FIFO1           BIT(26)

/* Delay register */
#define STEPDELAY_OPEN_MASK        (0x3FFFF << 0)
#define STEPDELAY_OPEN(val)        ((val) << 0)
#define STEPCONFIG_OPENDLY         STEPDELAY_OPEN(0x098)
#define STEPDELAY_SAMPLE_MASK      (0xFF << 24)
#define STEPDELAY_SAMPLE(val)      ((val) << 24)
#define STEPCONFIG_SAMPLEDLY       STEPDELAY_SAMPLE(0)

/* Charge Config */
#define STEPCHARGE_RFP_MASK        (7 << 12)
#define STEPCHARGE_RFP(val)        ((val) << 12)
#define STEPCHARGE_RFP_XPUL        STEPCHARGE_RFP(1)
#define STEPCHARGE_INM_MASK        (0xF << 15)
#define STEPCHARGE_INM(val)        ((val) << 15)
#define STEPCHARGE_INM_AN1         STEPCHARGE_INM(1)
#define STEPCHARGE_INP_MASK        (0xF << 19)
#define STEPCHARGE_INP(val)        ((val) << 19)
#define STEPCHARGE_RFM_MASK        (3 << 23)
#define STEPCHARGE_RFM(val)        ((val) << 23)
#define STEPCHARGE_RFM_XNUR        STEPCHARGE_RFM(1)

/* Charge delay */
#define CHARGEDLY_OPEN_MASK        (0x3FFFF << 0)
#define CHARGEDLY_OPEN(val)        ((val) << 0)
#define CHARGEDLY_OPENDLY          CHARGEDLY_OPEN(0x400)

/* Control register */
#define CNTRLREG_TSCSSENB          BIT(0)
```

```
#define CNTRLREG_STEPID            BIT(1)
#define CNTRLREG_STEPCONFIGWRT     BIT(2)
#define CNTRLREG_POWERDOWN         BIT(4)
#define CNTRLREG_AFE_CTRL_MASK     (3 << 5)
#define CNTRLREG_AFE_CTRL(val)     ((val) << 5)
#define CNTRLREG_4WIRE             CNTRLREG_AFE_CTRL(1)
#define CNTRLREG_5WIRE             CNTRLREG_AFE_CTRL(2)
#define CNTRLREG_8WIRE             CNTRLREG_AFE_CTRL(3)
#define CNTRLREG_TSCENB            BIT(7)

/* FIFO READ Register */
#define FIFOREAD_DATA_MASK (0xfff << 0)
#define FIFOREAD_CHNLID_MASK (0xf << 16)

/* DMA ENABLE/CLEAR Register */
#define DMA_FIFO0                  BIT(0)
#define DMA_FIFO1                  BIT(1)

/* Sequencer Status */
#define SEQ_STATUS BIT(5)
#define CHARGE_STEP                0x11

#define ADC_CLK                    24000000
#define TOTAL_STEPS                16
#define TOTAL_CHANNELS             8
#define FIFO1_THRESHOLD            19

/*
* time in us for processing a single channel, calculated as follows:
*
* max num cycles = open delay + (sample delay + conv time) * averaging
*
* max num cycles: 262143 + (255 + 13) * 16 = 266431
*
* clock frequency: 26MHz / 8 = 3.25MHz
* clock period: 1 / 3.25MHz = 308ns
*
* max processing time: 266431 * 308ns = 83ms(approx)
*/
#define IDLE_TIMEOUT 83 /* milliseconds */

```

```
#define TSCADC_CELLS              2

struct ti_tscadc_dev {
struct device *dev;
struct regmap *regmap;
void __iomem *tscadc_base;
phys_addr_t tscadc_phys_base;
int irq;
int used_cells; /* 1-2 */
int tsc_wires;
int tsc_cell;   /* -1 if not used */
int adc_cell;   /* -1 if not used */
struct mfd_cell cells[TSCADC_CELLS];
u32 reg_se_cache;
bool adc_waiting;
bool adc_in_use;
wait_queue_head_t reg_se_wait;
spinlock_t reg_lock;
unsigned int clk_div;

/* tsc device */
struct titsc *tsc;

/* adc device */
struct adc_device *adc;
};

static inline struct ti_tscadc_dev *ti_tscadc_dev_get(struct platform_device *p)
{
struct ti_tscadc_dev **tscadc_dev = p->dev.platform_data;

return *tscadc_dev;
}

void am335x_tsc_se_set_cache(struct ti_tscadc_dev *tsadc, u32 val);
void am335x_tsc_se_set_once(struct ti_tscadc_dev *tsadc, u32 val);
void am335x_tsc_se_clr(struct ti_tscadc_dev *tsadc, u32 val);
void am335x_tsc_se_adc_done(struct ti_tscadc_dev *tsadc);

#endif
```

APPENDIX D

BB-ADC-00A0.DTS DEVICE TREE OVERLAY

```
/*
 * Copyright (C) 2012 Texas Instruments Incorporated - http://www.ti.com/
 * Source modified by Pierrick Rauby
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 */

/dts-v1/;
/plugin/;

/ {
        compatible = "ti,beaglebone", "ti,beaglebone-black", "ti,beaglebone-green";

        // identification
        part-number = "BB-ADC";
        version = "00A0";

        // resources this cape uses
        exclusive-use =
                "P9.39",                // AIN0
                "P9.40",                // AIN1
                "P9.37",                // AIN2
                "P9.38",                // AIN3
                "P9.33",                // AIN4
                "P9.36",                // AIN5
                "P9.35",                // AIN6

                "tscadc";       // hardware ip used

        fragment@0 {
                target = <&tscadc>;
                __overlay__ {

                        status = "okay";
```

```
                adc {
                    ti,adc-channels = <3>;
                    ti,chan-step-avg = <8>;//we are averaging over 8 sample before sending
    // the result to the kernel
                    ti,chan-step-opendelay = <0>;
                    ti,chan-step-sampledelay = <0>;
                };
            };
        };
};
```

APPENDIX E

THE IIO_GENERIC_BUFFER.C APPLICATION

```
/* Industrialio buffer test code.
 *
 * Copyright (c) 2008 Jonathan Cameron
 * Source modified by Pierrick Rauby
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 as published by
 * the Free Software Foundation.
 *
 * This program is primarily intended as an example application.
 * Reads the current buffer setup from sysfs and starts a short capture
 * from the specified device, pretty printing the result after appropriate
 * conversion.
 *
 * Command line parameters
 * generic_buffer -n <device_name> -t <trigger_name>
 * If trigger name is not specified the program assumes you want a dataready
 * trigger associated with the device and goes looking for it.
 *
 */

#include <unistd.h>
#include <stdlib.h>
#include <dirent.h>
#include <fcntl.h>
#include <stdio.h>
#include <errno.h>
#include <sys/stat.h>
#include <sys/dir.h>
#include <linux/types.h>
#include <string.h>
#include <poll.h>
#include <endian.h>
#include <getopt.h>
#include <inttypes.h>
#include <stdbool.h>
```

```
#include <signal.h>
#include <time.h>
#include "iio_utils.h"

/**
 * enum autochan - state for the automatic channel enabling mechanism
 */
enum autochan { AUTOCHANNELS_DISABLED,
            AUTOCHANNELS_ENABLED,
            AUTOCHANNELS_ACTIVE,
};

/**
 * size_from_channelarray() - calculate the storage size of a scan
 * @channels:           the channel info array
 * @num_channels:       number of channels
 *
 * Has the side effect of filling the channels[i].location values used
 * in processing the buffer output.
 **/
int size_from_channelarray(struct iio_channel_info *channels, int num_channels)
{
  int bytes = 0;
  int i = 0;
  while (i < num_channels) {
    if (bytes % channels[i].bytes == 0)
      channels[i].location = bytes;
    else
      channels[i].location = bytes - bytes % channels[i].bytes
        + channels[i].bytes;

  bytes = channels[i].location + channels[i].bytes;
  i++;
  }
  return bytes;
}

void print1byte(uint8_t input, struct iio_channel_info *info)
{
  /*
   * Shift before conversion to avoid sign extension
```

```
    * of left aligned data
    */
   input >>= info->shift;
   input &= info->mask;
   if (info->is_signed) {
   int8_t val = (int8_t)(input << (8 - info->bits_used)) >>
     (8 - info->bits_used);
   printf("%05f ", ((float)val + info->offset) * info->scale);
   } else {
     printf("%05f ", ((float)input + info->offset) * info->scale);
   }
}

void print2byte(uint16_t input, struct iio_channel_info *info, int j, char *myString)
{
/* First swap if incorrect endian */
   if (info->be)
     input = be16toh(input);
   else
     input = le16toh(input);
   /*
    * Shift before conversion to avoid sign extension
    * of left aligned data
    */
   input >>= info->shift;
   input &= info->mask;
   if (info->is_signed) {
     int16_t val = (int16_t)(input << (16 - info->bits_used)) >>
       (16 - info->bits_used);
   //   printf("%05f", ((float)val + info->offset)*info->scale);
     sprintf(myString,"%d,%05f\n", j,((float)val + info->offset) * info->scale);
   } else {
   sprintf(myString,"%d,%05f\n", j,((float)input + info->offset) * info->scale);
  // printf("%05f", ((float)input + info->offset)*info->scale);
   }
}

void print4byte(uint32_t input, struct iio_channel_info *info)
{
         /* First swap if incorrect endian */
         if (info->be)
```

```
                input = be32toh(input);
        else
                input = le32toh(input);

        /*
         * Shift before conversion to avoid sign extension
         * of left aligned data
         */
        input >>= info->shift;
        input &= info->mask;
        if (info->is_signed) {
                int32_t val = (int32_t)(input << (32 - info->bits_used)) >>
                              (32 - info->bits_used);
                printf("%05f ", ((float)val + info->offset) * info->scale);
        } else {
                printf("%05f ", ((float)input + info->offset) * info->scale);
        }
}

void print8byte(uint64_t input, struct iio_channel_info *info)
{
        /* First swap if incorrect endian */
        if (info->be)
                input = be64toh(input);
        else
                input = le64toh(input);

        /*
         * Shift before conversion to avoid sign extension
         * of left aligned data
         */
        input >>= info->shift;
        input &= info->mask;
        if (info->is_signed) {
                int64_t val = (int64_t)(input << (64 - info->bits_used)) >>
                              (64 - info->bits_used);
                /* special case for timestamp */
                if (info->scale == 1.0f && info->offset == 0.0f)
                        printf("%" PRId64 " ", val);
                else
                        printf("%05f ",
```

```
                                ((float)val + info->offset) * info->scale);
        } else {
                printf("%05f ", ((float)input + info->offset) * info->scale);
        }
}

/**
 * process_scan() - print out the values in SI units
 * @data: pointer to the start of the scan
 * @channels: information about the channels.
 * Note: size_from_channelarray must have been called first
 *    to fill the location offsets.
 * @num_channels: number of channels
 **/
void process_scan(char *data, struct iio_channel_info *channels, int num_channels,int j,
    char *myString)
{
  int k;
  for (k = 0; k < num_channels; k++)
    switch (channels[k].bytes) {
      /* only a few cases implemented so far */
    case 1:
      print1byte(*(uint8_t *)(data + channels[k].location),
          &channels[k]);
      break;
    case 2:
      print2byte(*(uint16_t *)(data + channels[k].location),
          &channels[k],j, myString);
      break;
    case 4:
      print4byte(*(uint32_t *)(data + channels[k].location),
          &channels[k]);
      break;
    case 8:
      print8byte(*(uint64_t *)(data + channels[k].location),
          &channels[k]);
      break;
    default:
      break;
    }
  //printf("\n");
```

```
}

static int enable_disable_all_channels(char *dev_dir_name, int enable)
{
	const struct dirent *ent;
	char scanelemdir[256];
	DIR *dp;
	int ret;

	snprintf(scanelemdir, sizeof(scanelemdir),
		 FORMAT_SCAN_ELEMENTS_DIR, dev_dir_name);
	scanelemdir[sizeof(scanelemdir)-1] = '\0';

	dp = opendir(scanelemdir);
	if (!dp) {
		fprintf(stderr, "Enabling/disabling channels: can't open %s\n",
			scanelemdir);
		return -EIO;
	}

	ret = -ENOENT;
	while (ent = readdir(dp), ent) {
		if (iioutils_check_suffix(ent->d_name, "_en")) {
			printf("%sabling: %s\n",
			       enable ? "En" : "Dis",
			       ent->d_name);
			ret = write_sysfs_int(ent->d_name, scanelemdir,
					      enable);
			if (ret < 0)
				fprintf(stderr, "Failed to enable/disable %s\n",
					ent->d_name);
		}
	}

	if (closedir(dp) == -1) {
		perror("Enabling/disabling channels: "
		       "Failed to close directory");
		return -errno;
	}
	return 0;
}
```

```
void print_usage(void)
{
        fprintf(stderr, "Usage: generic_buffer [options]...\n"
                "Capture, convert and output data from IIO device buffer\n"
                "  -a         Auto-activate all available channels\n"
                "  -A         Force-activate ALL channels\n"
                "  -c <n>     Do n conversions\n"
                "  -e         Disable wait for event (new data)\n"
                "  -g         Use trigger-less mode\n"
                "  -l <n>     Set buffer length to n samples\n"
                "  --device-name -n <name>\n"
                "  --device-num -N <num>\n"
                "        Set device by name or number (mandatory)\n"
                "  --trigger-name -t <name>\n"
                "  --trigger-num -T <num>\n"
                "        Set trigger by name or number\n"
                "  -w <n>     Set delay between reads in us (event-less mode)\n");
}

enum autochan autochannels = AUTOCHANNELS_DISABLED;
char *dev_dir_name = NULL;
char *buf_dir_name = NULL;
bool current_trigger_set = false;

void cleanup(void)
{
        int ret;

        /* Disable trigger */
        if (dev_dir_name && current_trigger_set) {
                /* Disconnect the trigger - just write a dummy name. */
                ret = write_sysfs_string("trigger/current_trigger",
                                         dev_dir_name, "NULL");
                if (ret < 0)
                        fprintf(stderr, "Failed to disable trigger: %s\n",
                                strerror(-ret));
                current_trigger_set = false;
        }

        /* Disable buffer */
```

```c
	if (buf_dir_name) {
		ret = write_sysfs_int("enable", buf_dir_name, 0);
		if (ret < 0)
			fprintf(stderr, "Failed to disable buffer: %s\n",
				strerror(-ret));
	}

	/* Disable channels if auto-enabled */
	if (dev_dir_name && autochannels == AUTOCHANNELS_ACTIVE) {
		ret = enable_disable_all_channels(dev_dir_name, 0);
		if (ret)
			fprintf(stderr, "Failed to disable all channels\n");
		autochannels = AUTOCHANNELS_DISABLED;
	}
}

void sig_handler(int signum)
{
	fprintf(stderr, "Caught signal %d\n", signum);
	cleanup();
	exit(-signum);
}

void register_cleanup(void)
{
	struct sigaction sa = { .sa_handler = sig_handler };
	const int signums[] = { SIGINT, SIGTERM, SIGABRT };
	int ret, i;

	for (i = 0; i < ARRAY_SIZE(signums); ++i) {
		ret = sigaction(signums[i], &sa, NULL);
		if (ret) {
			perror("Failed to register signal handler");
			exit(-1);
		}
	}
}

static const struct option longopts[] = {
	{ "device-name",	1, 0, 'n' },
	{ "device-num",		1, 0, 'N' },
```

```
	{ "trigger-name",	1, 0, 't' },
	{ "trigger-num",	1, 0, 'T' },
	{ },
};

int main(int argc, char **argv)
{
	unsigned long num_loops = 1; //why do I would like more than 1 loop
	unsigned long timedelay = 1000000; // wait a bit so the character
	 // device file apears
	unsigned long buf_len = 128;

	int ret, c, i, j, toread;
	int fp = -1;

	int num_channels = 0;
	char *trigger_name = NULL, *device_name = NULL;

	char *data = NULL;
	ssize_t read_size;
	int dev_num = -1, trig_num = -1;
	char *buffer_access = NULL;
	int scan_size;
	int noevents = 0;
	int notrigger = 0;
	char *dummy;
	bool force_autochannels = false;

	struct iio_channel_info *channels = NULL;

	register_cleanup();

	while ((c = getopt_long(argc, argv, "aAc:egl:n:N:t:T:w:?", longopts,
				NULL)) != -1) {
		switch (c) {
		case 'a':
			autochannels = AUTOCHANNELS_ENABLED;
			break;
		case 'A':
			autochannels = AUTOCHANNELS_ENABLED;
			force_autochannels = true;
```

```
				break;
			case 'c':
				errno = 0;
				num_loops = strtoul(optarg, &dummy, 10); //parses the number and the name of the option
				if (errno) {
					ret = -errno;
					goto error;
				}
				break;
			case 'e':
				noevents = 1;
				break;
			case 'g':
				notrigger = 1;
				break;
			case 'l':
				errno = 0;
				buf_len = strtoul(optarg, &dummy, 10);
				if (errno) {
					ret = -errno;
					goto error;
				}
				break;
			case 'n':
				device_name = strdup(optarg);
				break;
			case 'N':
				errno = 0;
				dev_num = strtoul(optarg, &dummy, 10);
				if (errno) {
					ret = -errno;
					goto error;
				}
				break;
			case 't':
				trigger_name = strdup(optarg); //duplicates the string
				break;
			case 'T':
				errno = 0;
				trig_num = strtoul(optarg, &dummy, 10);
```

```
			if (errno)
				return -errno;
			break;
		case 'w':
			errno = 0;
			timedelay = strtoul(optarg, &dummy, 10);
			if (errno) {
				ret = -errno;
				goto error;
			}
			break;
		case '?':
			print_usage();
			ret = -1;
			goto error;
		}
	}

	/* Find the device requested */
	if (dev_num < 0 && !device_name) {
		fprintf(stderr, "Device not set\n");
		print_usage();
		ret = -1;
		goto error;
	}
else if (dev_num >= 0 && device_name) {
		fprintf(stderr, "Only one of --device-num or --device-name needs to be set\n");
		print_usage();
		ret = -1;
		goto error;
	}
else if (dev_num < 0) {
		dev_num = find_type_by_name(device_name, "iio:device");
		if (dev_num < 0) {
			fprintf(stderr, "Failed to find the %s\n", device_name);
			ret = dev_num;
			goto error;
		}
	}
	printf("iio device number being used is %d\n", dev_num);
```

```
	ret = asprintf(&dev_dir_name, "%siio:device%d", iio_dir, dev_num);
	if (ret < 0)
		return -ENOMEM;
	/* Fetch device_name if specified by number */
	if (!device_name) {
		device_name = malloc(IIO_MAX_NAME_LENGTH);
		if (!device_name) {
			ret = -ENOMEM;
			goto error;
		}
		ret = read_sysfs_string("name", dev_dir_name, device_name);
		if (ret < 0) {
			fprintf(stderr, "Failed to read name of device %d\n", dev_num);
			goto error;
		}
	}
/* Trigger setup */
	if (notrigger) {
		printf("trigger-less mode selected\n");
	} else if (trig_num >= 0) {
		char *trig_dev_name;
		ret = asprintf(&trig_dev_name, "%strigger%d", iio_dir, trig_num);
		if (ret < 0) {
			return -ENOMEM;
		}
		trigger_name = malloc(IIO_MAX_NAME_LENGTH);
		ret = read_sysfs_string("name", trig_dev_name, trigger_name);
		free(trig_dev_name);
		if (ret < 0) {
			fprintf(stderr, "Failed to read trigger%d name from\n", trig_num);
			return ret;
		}
		printf("iio trigger number being used is %d\n", trig_num);
		}
	/*
	 * Parse the files in scan_elements to identify what channels are
	 * present
	 */
	ret = build_channel_array(dev_dir_name, &channels, &num_channels);
	if (ret) {
```

```
				fprintf(stderr, "Problem reading scan element information\n"
					"diag %s\n", dev_dir_name);
				goto error;
			}
			if (num_channels && autochannels == AUTOCHANNELS_ENABLED &&
			    !force_autochannels) {
				fprintf(stderr, "Auto-channels selected but some channels "
					"are already activated in sysfs\n");
				fprintf(stderr, "Proceeding without activating any channels\n");
			}

			if ((!num_channels && autochannels == AUTOCHANNELS_ENABLED) ||
			    (autochannels == AUTOCHANNELS_ENABLED && force_autochannels)) {
				fprintf(stderr, "Enabling all channels\n");

				ret = enable_disable_all_channels(dev_dir_name, 1);
				if (ret) {
					fprintf(stderr, "Failed to enable all channels\n");
					goto error;
				}

				/* This flags that we need to disable the channels again */
				autochannels = AUTOCHANNELS_ACTIVE;

				ret = build_channel_array(dev_dir_name, &channels,
							  &num_channels);
				if (ret) {
					fprintf(stderr, "Problem reading scan element "
						"information\n"
						"diag %s\n", dev_dir_name);
					goto error;
				}
				if (!num_channels) {
					fprintf(stderr, "Still no channels after "
						"auto-enabling, giving up\n");
					goto error;
				}
			}

			if (!num_channels && autochannels == AUTOCHANNELS_DISABLED) {
				fprintf(stderr,
```

```
				"No channels are enabled, we have nothing to scan.\n");
			fprintf(stderr, "Enable channels manually in "
				FORMAT_SCAN_ELEMENTS_DIR
				"/*_en or pass -a to autoenable channels and "
				"try again.\n", dev_dir_name);
			ret = -ENOENT;
			goto error;
		}

		/*
		 * Construct the directory name for the associated buffer.
		 * As we know that the lis3l02dq has only one buffer this may
		 * be built rather than found.
		 */
		ret = asprintf(&buf_dir_name,
			       "%siio:device%d/buffer", iio_dir, dev_num);
		if (ret < 0) {
			ret = -ENOMEM;
			goto error;
		}

	printf("%s\n", dev_dir_name);
		/* Setup ring buffer parameters */
		ret = write_sysfs_int("length", buf_dir_name, buf_len);
		if (ret < 0)
			goto error;

		/* Enable the buffer */
		ret = write_sysfs_int("enable", buf_dir_name, 1);
		if (ret < 0) {
			fprintf(stderr,
				"Failed to enable buffer: %s\n", strerror(-ret));
			goto error;
		}

		scan_size = size_from_channelarray(channels, num_channels);
		data = malloc(scan_size * buf_len);
		if (!data) {
			ret = -ENOMEM;
			goto error;
		}
```

```

        ret = asprintf(&buffer_access, "/dev/iio:device%d", dev_num);
        if (ret < 0) {
                ret = -ENOMEM;
                goto error;
        }

        /* Attempt to open non blocking the access dev */
        fp = open(buffer_access, O_RDONLY | O_NONBLOCK);
        if (fp == -1) {
                ret = -errno;
                fprintf(stderr, "Failed to open %s\n", buffer_access);
                goto error;
        }


//the file where we want to print the result
FILE * fa;
time_t t = time(NULL);
struct tm tm = *localtime(&t);
char fileName[20];
sprintf(fileName,"Results/data_%d-%d-%d_%d:%d:%d.csv", tm.tm_year+1900, tm.tm_mon+1, tm.
    tm_mday, tm.tm_hour, tm.tm_min,tm.tm_sec);
fa= fopen(fileName,"w+");
char firstLine[20];
fputs(firstLine,fa);
char myString[20];
//Start Flashing
removeTrigger();
flashLed();
//acquisition loop
for (j = 0; j < num_loops; j++) {
    toread=buf_len;
    usleep(timedelay); // not shure that this part has to be commented
    read_size = read(fp, data, toread * scan_size);
    if (read_size < 0) {
      if (errno == EAGAIN) {
        fprintf(stderr, "nothing available\n");
        continue;
      } else {
        break;
```

```
        }
      }

      for (i = 0; i < read_size / scan_size; i++){
        process_scan(data + scan_size * i, channels,num_channels,i, myString);
        fputs(myString,fa);
      }
  }
  //closing the file
  fclose(fa);
  // stop flahing Leds
  removeTrigger();
  error:
          cleanup();

          if (fp >= 0 && close(fp) == -1)
                  perror("Failed to close buffer");
          free(buffer_access);
          free(data);
          free(buf_dir_name);
          for (i = num_channels - 1; i >= 0; i--) {
                  free(channels[i].name);
                  free(channels[i].generic_name);
          }
          free(channels);
          free(trigger_name);
          free(device_name);
          free(dev_dir_name);

          return ret;
  }
```

APPENDIX F

THE LAUNCH.SH SCRIPT

```
#!/bin/sh
# launch.sh
# Copyright (c) 2018 Pierrick Rauby
# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License version 2 as published by
# the Free Software Foundation.
N_Samples=$1
N_Loops=$2
# i=1
echo "Cleaning 'Results' folder"
rm -rf Results
mkdir Results

echo "Deploying..."
    gcc iio_generic_buffer.c iio_utils.c -o iio_generic_buffer

echo "Here we go for ${N_Samples} repeted ${N_Loops} times"
# while [ "$i" -le $N_Loops ]; do
  ./iio_generic_buffer -a -l ${N_Samples} -L ${N_Loops} -N iio:device0
  # echo "****Loop ${i} done****"
  # i=$(( i + 1 ))
echo "########################################"
echo "Work done results are saved in /Results"
echo "########################################"
```

APPENDIX G

THE PREPROCESSING.PY CODE

```
# preprocessing.py
# Copyright (c) 2018 Pierrick Rauby
# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License version 2 as published by
# the Free Software Foundation.
# Returns a .csv from all .csv file contained in the folder where this code
# is located

#imports
import pandas as pd
import numpy as np
np.set_printoptions(threshold=np.nan)
import matplotlib.pyplot as plt
import os
import glob

#Result and Data set info:
classification=1
i=3 #Number of dominant frequencies requested
fftSize = 16383 # Number of Samples in the DataSet
samplingRate=16383 # Samples per seconds

#Gets the list of files
path=os.getcwd()
allFiles=glob.glob(path+"/*.csv")

#Final returned list
Result=pd.DataFrame()

#For loop over the all the data sets:
for file_ in allFiles:
  #Initialize the result DataFrame for this sample
  resultCash=pd.DataFrame(columns=['Name','Mean','Median','Std','Var','Min','Max','sum','f1','A1','f2','A2','f3','A3','Class'])
  #Imports the dataset
```

```
dataSet=pd.DataFrame()
fftData = []
dataSet=pd.read_csv(file_,names=["Volts"])
#FFT computation
for row_ in dataSet.values:
  fftData.append(row_[0])

fftData = np.array(fftData,dtype=float)

#Compute the FFT and the frequencies
fft = np.fft.fft(fftData) #array of xk result of the real fft
fftFreq = np.fft.fftfreq(fftSize, d=1./samplingRate) #array with corresponding
    frequencies
fftMag = np.absolute(fft)

#Find i dominant frequencies
fftMagCash=fftMag[:fftSize //2]*1 / fftSize
frequencies=[]
fftFreq=fftFreq[:fftSize //2]
for k in range(i):
  Cash=[]
  mainFreqIndex = np.argmax(fftMagCash) #get the more important term
  Cash.append(fftMagCash[mainFreqIndex]) #storing the amplitude of the max Freq
  Cash.append(fftFreq[mainFreqIndex]) #storing the max Freq
  fftMagCash=np.delete(fftMagCash,mainFreqIndex) #removing the maximum frequency
  np.delete(fftMagCash,mainFreqIndex) #removing the maximum frequency
  frequencies.append(Cash) #add this values to the result list
#print(frequencies)
#End of FFT computation

#Stores values in the resultCash list
resultCash=pd.concat([resultCash,#previous data DataFrame
                      pd.DataFrame([[ #New DataFrame
                        file_[len(path)+1:], #Name of the Sample
                        dataSet['Volts'].mean(),
                        dataSet['Volts'].median(),
                        dataSet['Volts'].std(),
                        dataSet['Volts'].var(),
                        dataSet['Volts'].min(),
                        dataSet['Volts'].max(),
                        dataSet['Volts'].sum(),
```

```
                                frequencies[0][1], #f1
                                frequencies[0][0], #A1
                                frequencies[1][1], #f2
                                frequencies[1][0], #A2
                                frequencies[2][1], #f3
                                frequencies[2][0], #A3
                                classification ]], #Class of the sample
                                columns=['Name','Mean','Median','Std','Var','Min','Max','sum','
                                    f1','A1','f2','A2','f3','A3','Class']))
  #Creates the finals list Result
  Result=pd.concat([Result,resultCash],ignore_index=True)


Result.to_csv('Cut'+str(classification)+'.csv')
print(Result)

#end of for loop over allFiles
```

APPENDIX H

THE KERNEL_SVM_TRAINNING.PY CODE

```
# kernel_SVM_trainning.py
# Copyright (c) 2018 Pierrick Rauby
# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License version 2 as published by
# the Free Software Foundation.
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd


# Assign colum names to the dataset
colnames =['Name','Mean','Median','Std','Var','Min','Max','sum','f1','A1','f2','A2','f3','A3','Class']

# Read dataset to pandas dataframe
dataSet = pd.read_csv('Data_set.csv', skiprows=[0],names=colnames)
print(dataSet.shape)
X = dataSet.drop(['Name','sum','Class'], axis=1)#.drop('Mean,axis=0)#the features
y = dataSet['Class'] #the predictions

#Splitting the dataset between trainning set and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)

#Train the algorithm
from sklearn.svm import SVC

#Uncomment for polynom kernel
# svclassifier = SVC(kernel='poly', degree=8)
# svclassifier.fit(X_train, y_train)

#Uncomment for Sigmoid Kernel
# svclassifier = SVC(kernel='sigmoid')
# svclassifier.fit(X_train, y_train)

```

```
# #Uncomment for Gaussian Kernel
svclassifier = SVC(kernel='linear') #'linear', 'poly', 'rbf', 'sigmoid',
svclassifier.fit(X_train, y_train)

#test to pickle the classifier
import pickle
classifier_pickle_path = 'classifier_pickle.pkl' #creates the name of the file
classifier_pickle = open(classifier_pickle_path,'wb') #open the file for binaryW
pickle.dump(svclassifier,classifier_pickle) #put the classifier in the file

#This makes predictions
y_pred = svclassifier.predict(X_test)

#This evaluates the algorithm
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

APPENDIX I

DETAILED RESULTS FOR LINEAR KERNEL AND RBF KERNEL ON THE TEST SET

I.1 Result for the linear kernel

I.2 Result for the rbf kernel

```
[[425   0   0   0   0]
 [  0 380   0   0   0]
 [  0   2 368  12   3]
 [  0   0   8 393   0]
 [  0   0   2   0 407]]
             precision    recall  f1-score   support

          0       1.00      1.00      1.00       425
          1       0.99      1.00      1.00       380
          2       0.97      0.96      0.96       385
          3       0.97      0.98      0.98       401
          4       0.99      1.00      0.99       409

avg / total       0.99      0.99      0.99      2000
```

Figure I.1: Confusion matrix and precision statics for the linear kernel

```
[[408   0   0   0   9]
 [  0 368   0   0  48]
 [  0   0 230   0 139]
 [  0   0   2 338  53]
 [  0   0   0   0 405]]
             precision    recall  f1-score   support

          0       1.00      0.98      0.99       417
          1       1.00      0.88      0.94       416
          2       0.99      0.62      0.77       369
          3       1.00      0.86      0.92       393
          4       0.62      1.00      0.76       405

avg / total       0.92      0.87      0.88      2000
```

Figure I.2: Confusion matrix and precision statics for the rbf kernel

CHAPTER 6

THE MAIN APPLICATION CODE

```
# main.py
# Copyright (c) 2018 Pierrick Rauby
# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License version 2 as published by
# the Free Software Foundation.

##########################Import needed libraries##############################
import os #to execute acquisition program
import pickle
import glob
import numpy as np
import pandas as pd
import datetime
from sklearn.svm import SVC # not sure if needed (maybe included in pickle)
###########################Variables declaration###############################
N_Samples = int(16384/2)
i=3 #Number of dominant frequencies a requested
fftSize = N_Samples - 1 # Number of Samples in the DataSet
samplingRate=N_Samples-1 # Samples per seconds # WARNING: check sampling frequency
################################Compilation###################################
#Uncomment the following line if you want recompile iio_generic_buffer.c
#os.system('gcc iio_generic_buffer.c iio_utils.c -o iio_generic_buffer')
##############################################################################
##########################Entering the execution Loop#########################
##############################################################################
while(1):
    # first we capture the timestamp
    timestamp_object = datetime.datetime.now()
    ###########################Cleanning Results folder###########################
    Command_Clean = "rm -rf Results"
    Process = os.system(Command_Clean)
    Command_Create = "mkdir Results"
    Process = os.system(Command_Create)
    ##########################Starting the acquisition###########################
    Command_Acquisition = "./iio_generic_buffer -a -l "+str(N_Samples)+" -N iio:device0"
```

```
    print(Command_Acquisition)
    Process = os.system(Command_Acquisition)

    # At this point data should be stored in the Result folder
    print('\n################################\n  Data stored in Result folder\n ################################')

    ########################Preprocessing the dataSet###############################
    #Final returned list
    preprocessed_dataSet=pd.DataFrame()
    #Gets the list of files
    path=os.getcwd() #The folder wh
    allFiles=glob.glob(path+"/Results/*.csv")
    #For loop over the all the data sets:
    for file_ in allFiles:
      #Initialiwe the result DataFrame for this sample
      resultCash=pd.DataFrame(columns=['Name','Mean','Median','Std','Var','Min','Max','sum','f1','A1','f2','A2','f3','A3']) #,'cj'])
      #Imports the dataset
      dataSet=pd.DataFrame()
      fftData = []
      dataSet=pd.read_csv(file_,names=["Volts"])
      #FFT computation
      for row_ in dataSet.values:
        fftData.append(row_[0])
      fftData = np.array(fftData,dtype=float)
      #Compute the FFT and the frequencies
      fft = np.fft.fft(fftData) #array of xk result of the real fft
      fftFreq = np.fft.fftfreq(fftSize, d=1./samplingRate) #array with corresponding frequencies
      fftMag = np.absolute(fft)
      #Find i dominant frequencies
      fftMagCash=fftMag[:fftSize //2]*1 / fftSize
      frequencies=[]
      fftFreq=fftFreq[:fftSize //2]
      for k in range(i):
        Cash=[]
        mainFreqIndex = np.argmax(fftMagCash) #get the more important term
        Cash.append(fftMagCash[mainFreqIndex]) #storing the amplitude of the max Freq
        Cash.append(fftFreq[mainFreqIndex]) #storing the max Freq
        fftMagCash=np.delete(fftMagCash,mainFreqIndex) #removing the maximum frequency
```

```
            np.delete(fftMagCash,mainFreqIndex) #removing the maximum frequency
            frequencies.append(Cash) #add this values to the result list
        #End of FFT computation
        #Stores values in the resultCash list
        resultCash=pd.concat([resultCash,#previous data DataFrame
                              pd.DataFrame([[ #New DataFrame
                                file_[len(path)+1:], #Name of the Sample
                                dataSet['Volts'].mean(),
                                dataSet['Volts'].median(),
                                dataSet['Volts'].std(),
                                dataSet['Volts'].var(),
                                dataSet['Volts'].min(),
                                dataSet['Volts'].max(),
                                dataSet['Volts'].sum(),
                                frequencies[0][1], #f1
                                frequencies[0][0], #A1
                                frequencies[1][1], #f2
                                frequencies[1][0], #A2
                                frequencies[2][1], #f3
                                frequencies[2][0]]], #A3
                                columns=['Name','Mean','Median','Std','Var','Min','Max','sum','f1','A1','f2','A2','f3','A3'])]) #,'class'#,''])])
        #Creates the finals list Result
        preprocessed_dataSet=pd.concat([preprocessed_dataSet,resultCash],ignore_index=True)
    #Using the trainned algorithm to predictions
    #dropping the useless features
    Xtest = preprocessed_dataSet.drop(['Name','sum'], axis=1)
    classifier_pickle_path = 'classifier_pickle.pkl'
    classifier_pickle = open(classifier_pickle_path,'rb')
    svclassifier = pickle.load(classifier_pickle)
    #converting the timestamp to string
    timestamp=str(timestamp_object.year)+"-"+str(timestamp_object.month)+"-"+str(timestamp_object.day)+"T"+str(timestamp_object.hour)+":"+str(timestamp_object.minute)+":"+str(timestamp_object.second)

    print("At time " + timestamp +" class is " + str(svclassifier.predict(Xtest)[0]))

    # TODO: send the result somewhere (MQTT)
################################################################################
##########################End of while loop and programp######################
################################################################################
```

REFERENCES

[1] A. Froehlich, *How edge computing compares with cloud computing*, `https://www.networkcomputing.com/networking/how-edge-computing-compares-cloud-computing/1264320109`, Blog, 2018.

[2] C. Roser, *Faster, better, cheaper" in the history of manufacturing : From the stone age to lean manufacturing and beyond*. Boca Raton: CRC Press, Taylor & Francis Group, 2017, ISBN: 978-1498756303.

[3] A. Siddhpura and R. Paurobally, "A review of flank wear prediction methods for tool condition monitoring in a turning process", *The international journal of advanced manufacturing technology*, vol. 65, no. 1, pp. 371–393, 2013.

[4] S. Kurada and C. Bradley, "A review of machine vision sensors for tool condition monitoring", *Computers in industry*, vol. 34, no. 1, pp. 55–72, 1997.

[5] N. Cook, "Tool wear sensors", *Wear*, vol. 62, no. 1, pp. 49–57, 1980.

[6] D. E. Dimla, "Sensor signals for tool-wear monitoring in metal cutting operationsa review of methods", *International journal of machine tools and manufacture*, vol. 40, no. 8, pp. 1073–1098, 2000.

[7] A. Siddhpura and R. Paurobally, "A review of flank wear prediction methods for tool condition monitoring in a turning process", *International journal of advanced manufacturing technology*, vol. 65, no. 1-4, pp. 371–393, 2013.

[8] P. Maropoulos and B. Alamin, "Integrated tool life prediction and management for an intelligent tool selection system", *Journal of materials processing technology*, vol. 61, no. 1-2, pp. 225–230, 1996.

[9] S. M. Pandit, "Strategy of On-line Tool Wear Sensing", vol. 104, no. August 1982, pp. 217–223, 1982.

[10] L. Dan and J. Mathew, "Tool wear and failure monitoring techniques for turningA review", *International journal of machine tools and manufacture*, vol. 30, no. 4, pp. 579–598, 1990.

[11] *Arduino uno rev 3*, `https://store.arduino.cc/arduino-uno-rev3`, Website, Accessed : 2018-06-24.

[12] *Teensy usb development board*, `https://www.pjrc.com/store/teensy32.html`, Website, Accessed : 2018-06-24.

[13] *Particule photon datasheet vo16*, `https://docs.particle.io/datasheets/photon-(wifi)/photon-datasheet/`, Website, Accessed : 2018-06-24.

[14] *The internet of things with esp32*, `http://esp32.net/`, Website, Accessed : 2018-06-24.

[15] *The raspberry pi model 3 b+*, `https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/`, Website, Accessed : 2018-06-24.

[16] *Beagleboard.org beaglebone black*, `https://beagleboard.org/black`, Website, Accessed : 2018-06-24.

[17] M. Elangovan, V. Sugumaran, K. I. Ramachandran, and S. Ravikumar, "Effect of SVM kernel functions on classification of vibration signals of a single point cutting tool", *Expert systems with applications*, vol. 38, no. 12, pp. 15 202–15 207, 2011.

[18] C. Drouillet, J. Karandikar, C. Nath, A.-C. Journeaux, M. El Mansori, and T. Kurfess, "Tool life predictions in milling using spindle power with the neural network technique", *Journal of manufacturing processes*, vol. 22, pp. 161–168, 2016.

[19] Y. Fu, Y. Zhang, Y. Gao, H. Gao, T. Mao, H. Zhou, and D. Li, "Machining vibration states monitoring based on image representation using convolutional neural networks", *Engineering applications of artificial intelligence*, vol. 65, no. July, pp. 240–251, 2017.

[20] P. O'Donovan, C. Gallagher, K. Bruton, and D. T. O'Sullivan, "A fog computing industrial cyber-physical system for embedded low-latency machine learning Industry 4.0 applications", *Manufacturing letters*, vol. 15, pp. 139–142, 2018.

[21] C.-A. Azencot, *Foundations of machine learning chapter 9: Tree-based approaches*, 2017.

[22] ——, *Foundations of machine learning chapter 10: Support vector machines*, 2017.

[23] F. Pérez-Cruz and O. Bousquet, "Kernel methods and their potential use in signal processing", *Ieee signal processing magazine*, vol. 21, no. 3, pp. 57–65, 2004.

[24] F Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in ...", *Psychological review*, vol. 65, no. 6, pp. 386–408, 1958.

[25] C.-A. Azencot, *Foundations of machine learning chapter 11: Artificial neural networks*, 2017.

[26] *Introduction a lora*, http://www.linuxembedded.fr/2017/12/introduction-a-lora/, Website, Accessed : 2018-06-24, 2018.

[27] R. I. Pereira, I. M. Dupont, P. C. Carvalho, and S. C. Jucá, "IoT embedded linux system based on Raspberry Pi applied to real-time cloud monitoring of a decentralized photovoltaic plant", *Measurement: Journal of the international measurement confederation*, vol. 114, no. January 2017, pp. 286–297, 2018.

[28] S. Yang, B. Bagheri, H.-A. Kao, and J. Lee, "A Unified Framework and Platform for Designing of Cloud-Based Machine Health Monitoring and Manufacturing Systems", *Journal of manufacturing science and engineering*, vol. 137, no. 4, p. 040 914, 2015.

[29] C. Kan, H. Yang, and S. Kumara, "Parallel computing and network analytics for fast Industrial Internet-of-Things (IIoT) machine information processing and condition monitoring", *Journal of manufacturing systems*, vol. 46, pp. 282–293, 2018.

[30] D. Wu, S. Liu, L. Zhang, J. Terpenny, R. X. Gao, T. Kurfess, and J. A. Guzzo, "A fog computing-based framework for process monitoring and prognosis in cyber-manufacturing", *Journal of manufacturing systems*, vol. 43, pp. 25–34, 2017.

[31] M. Siddhpura and R. Paurobally, "A review of chatter vibration research in turning", *International journal of machine tools and manufacture*, vol. 61, pp. 27–47, 2012.

[32] TexasInstruments, *Am335x and amic110 sitaratm processors*, 2017.

[33] TexasInstrument, *Pru read latencies*, http://processors.wiki.ti.com/index.php/PRU_Read_Latencies, 2017.

[34] ——, *Pru-icss remoteproc and rpmsg*, http://processors.wiki.ti.com/index.php/PRU-ICSS_Remoteproc_and_RPMsg, 2017.

[35] D. Molloy, *Exploring beaglebone, Tools and techniques for building with embedded linux*. Wiley, 2015, ISBN: 978-1-118-93512-5.

[36] A. Devices, *Linux industrial i/o subsystem*, https://wiki.analog.com/software/linux/docs/iio/iio, 2017.

[37] T. Instruments, *Ti_am335x_adc*, http://git.ti.com/ti-linux-kernel/ti-linux-kernel/blobs/ti-linux-3.14.y/drivers/iio/adc/ti_am335x_adc.c, 2012.

[38] J. Cameron, *Iio_generic_buffer.c*, https://github.com/torvalds/linux/blob/master/tools/iio/iio_generic_buffer.c, 2008.