Discrimination in the reading of an optical encoder

Degree Thesis
submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
by

Brenda Juliana Fernández Alayo

In partial fulfillment
of the requirements for the degree in
(Grau en Enginyeria de Tecnologies i Serveis de Telecomunicació) **ENGINEERING**

Advisor : Olga Muñoz Medina
Barcelona, January 2021

Contents

List of Figures

List of Tables

Abbreviations

ANS Automatic Neural System

BP Back Propagation

CE Chip Enable

ETSETB Barcelona School of Telecommunications Engeneering

EU European Union

HP Hewlett-Packard

MAE Matlab i les seves Aplicacions a l'Enginyeria

MLP MulriLayer Perceptron

MSE Mean Squared Error

NN Neural Network

OE Output Enable

PCB Printed Circuit Board

PEU Physical Encoder Unit

PWM Pulse Width Modulation

RAM Random Access Memory

TCL Tool Command Language

VHDL VHSIC (Very High Speed Integrated Circuits) Hardware Description Language

WE Write Enable

Abstract

The analysis of an electrical fault in an industrial machine goes through several tests to narrow down the actual problem from a mechatronic subsystem. The process itself brings along an economic and time inversion which in some cases can affect the timeline. This study aims to determine the presence of anomalies in a strip encoder in order to avoid electrical issues in future performances. Specifically, it investigates whether the data collected from an optical encoder is reliable.

To imitate the output signal from the encoder, an FPGA was used to reproduce it using VHDL language. With the help of the Cmod A7 buttons, the PCB emulates an encoder and stores the data into the RAM for its subsequent analysis. The result of the sequence is done by a neural networks process code with Python.

These results suggest whether the data's collection has any damages or it can be considered meaningful.

Revision history and approval record

Revision	Date	Purpose
0	09/12/2021	Document creation
1	22/01/2022	Document revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Brenda Juliana Fernández Alayo	brenda.fernandez@estudiantat.upc.edu
Olga Muñoz Medina	olga.munoz@upc.edu

Written by:		Reviewed and approved by:	
Date	09/12/2021	Date	23/01/2022
Name	Brenda Juliana Fernández Alayo	Name	Olga Muñoz Medina
Position	Project Author	Position	Project Supervisor

1 Introduction

The project is based on personal growth for discoveries that could solve several issues in the industrial sector.

The potential system approach is to capture all information from an optical encoder into an FPGA. The module function is to store data and pin down if there has been an anomaly in the reading.

The motivation of this topic is to avoid any troubleshooting when a workflow that includes displacement is running. The following text expose a situation where the procedure could be useful:

"An industrial machine needs to move a carriage with a certain potency from point A to B. The motor and its optical encoder allow the element to complete the order. In case the strip encoder was damaged, the firmware would indicate a wrong position, then the machine would inject more PWM to compensate the deviation. If a certain timeout was exceeded, an electrical fault would appear."

To sum up, the main goals that determine the success of this project are two. First of all, the system must solve the algorithm, that collects all data, to detect any error on the strip encoder. On another note, the product is expected to be working properly on an industrial machine.

1.1 Requirements and specifications

In order to get a successful system is crucial to establish the following requirements:

- Capture the optical encoder information into a FPGA.

  Among all the possible modules I can use, the FPGA is the one that allows to store data on its memory with a certain sample frequency. Another option would be to save it on a processor ARM, even though the code can slow down the process.

- Save data sensor in real time.

  The "real time" aspect is really significant to bring about the installation in a machine while it is running and catch all the movement whether the test fails or not.

- Make use of neural networks to analyse the encoder data

  With the help of Python and Tensorflow, the data can be analysed to define an only output that reflects how much the strip encoder is damaged.

The way forward is to rule some specifications of the system that are going to conditionate the metrics:

- System of 150 LPI (Lines Per Inch).
- Physic encoder unit 1/4/150 inches.
- Acceleration of 0.12 of the gravity value.

1.2 Work's continuation

It is fair to point out that my final project on MAE was based on this idea. The MATLAB code had encoder states and timestamps as inputs. With this information the algorithm analyses the data in pack of 4 samples to nail down the direction of the movement.

The encoder behaves different if it goes backward or forward, so each of cases follows a pattern that can be easily detected.

Once the assumed direction is fixed, some calculus must be done to obtain the position and speed. Moreover, the inputs were transitions of states, that means that between each sample, there were never the same one, so in case there were more than one loss of count the pattern determined previously would not be useful. This system worked, but taking into account the timeline of the subject, the analysis was just done for a loss of 1 sample in both directions. It entails that all the possibilities were not enclosed, that is why my final degree work want to achieve this fact and make it functional on a real machine.

1.3 Work plan, Tasks and Milestones

The project consists of three work packages that will mark the workflow and organize the tasks to be implemented.

1.3.1 Define specifications

The first work package is the most important for the FPGA implementation. The list of parameters is not trivial because that time is when all the specifications get involve in the project and influence the FPGA choice.

- Internal task 1: List of parameters.
- Internal task 2: FPGA match.
- Internal task 3: Create a MATLAB file.

1.3.2 Encoder emulation and data processing

The encoder data processing is approached in this section. An encoder emulation simulates a real output so the test can use it instead of catching traces from a running machine. Once the first internal task is accomplished, the storage data into memory takes part in the work package.

- Internal task 1: : Introduction in Xilinx.
- Internal task 2: Module designed to mark the movement transitions where the behaviour will be determined.
- Internal task 3: Generate encoder units.
- Internal task 4: Convert encoder units to signals A and B.

- Internal task 5: Contains tasks 3 and 4, its purpose is to transform the speed and position to its binary designation of A and B channels.
- Internal task 6: The tasks 3 and 4 were modified in order to take into account possible error introduced from the environment.
- Internal task 7: Generate encoder units using Ca2 to detect the backwards direction too.
- Internal task 8: Control the flow of errors introduced and behaves depending on physical encoder position.
- Internal task T9: [Matlab] Choose the first error´s appearance probability.
- Internal task T10: [Matlab] Choose the type of probability for the length of the error.
- Internal task T11: [Matlab] Choose the probability in which the error is reproduced repeatedly.
- Internal task T12: Transfer data into FPGA's memory.
- Internal task T13: Extract data from FPGA to text file.

1.3.3 Neural Network

The neural network is used to determine the state of the strip encoder. In order to create a network well structured, there must be several tests to narrow down the one that suits better for this issue.

- Internal task T1: Neural Network background.
- Internal task T2: Process dara from file.
- Internal task T3: Decide the inputs on the neuronal network.

1.3.4 Milestones

- Create a MATLAB file: Once the MATLAB file is created, the parameters for the Xilinx simulation are set and the VHDL code can be started.
- Emulatation into a text file: This task is a tipping point for the transition to the neuronal network to begin with the next issue, so the data is ready to be analysed.
- Python NN: Focused of the matter of finding out if the reading of an encoder emulation carries an error.

1.4 Gantt Diagram

Two main topics have been addressed: the emulation of an encoder that can reproduce errors when counts are lost, but at the same time the generation of random errors on the sample is studied, along with the simulation of a RAM memory.

On the other hand, the neural network analysis is correlated to the previous topic status, that implicates druging the delays accumulated from the emulation.

The work package 2 contains a gradual evolution where a lot of time is dedicated to each module and it is not until November that several tasks overlap. It is important to highlight that although the timing indicates that a module is apparently finished but it is not until time after that unforeseen situations appear and hcange the course of schedule and VHDL code needs to be redone.

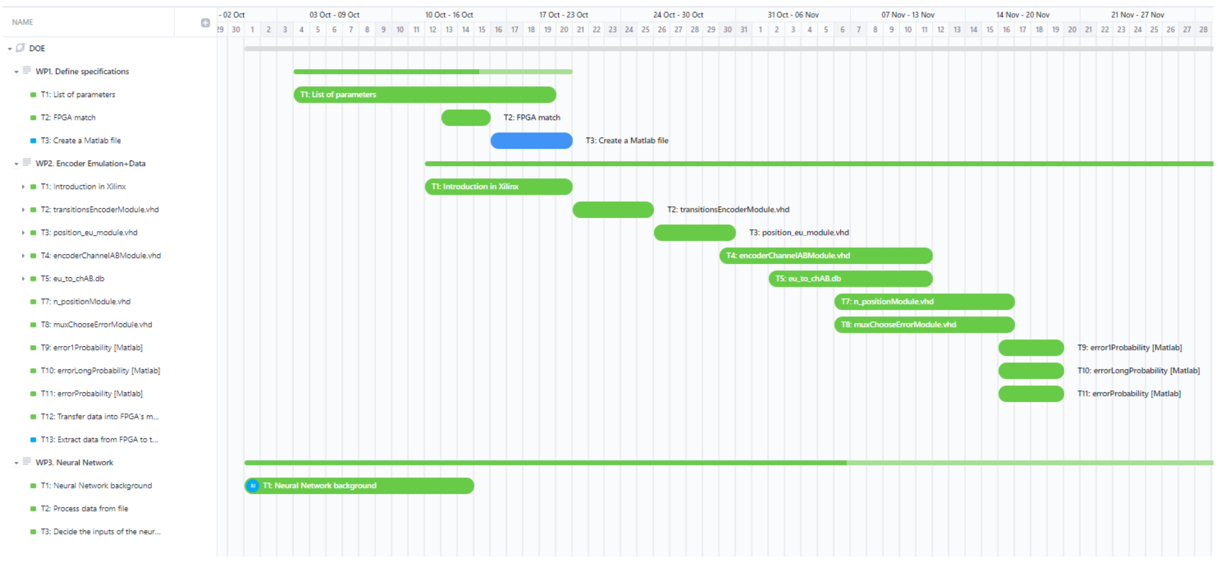


Figure 1: Gantt diagram of the project [October - November]

In December it was possible to make considerable progress on the project, but at the same time modifications were made that delayed it. Crucial decisions for the work were made on those dates, such as parameters with which the neural networks were going to measure to decide if the sample had faults or not.

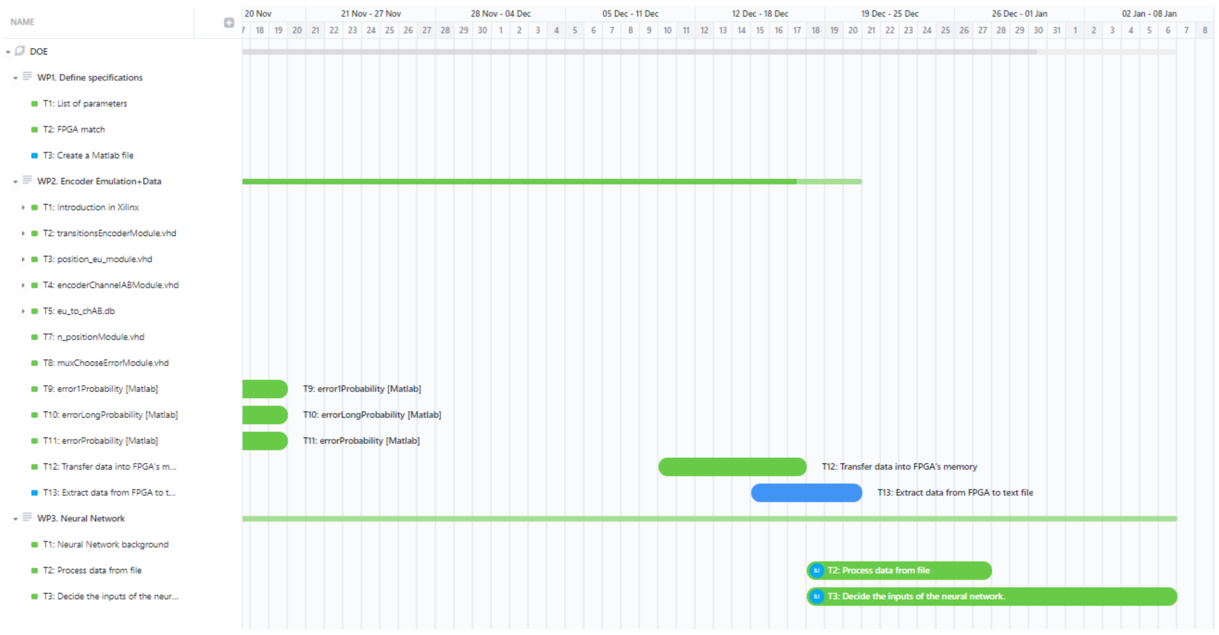


Figure 2: Gantt diagram of the project [December - January]

1.5 Deviations and incidences

Throughout the project, modifications have been made to the work plan. The fact of having a full-time schedule while studying the TFG has affected the amount of time applicable to the subject.

On the one hand, the initial delays were caused by the lack of knowledge of the tools. Vivado was used at the end of the degree but its use can not be compared to this time. On the other hand, the data analysis was to be implemented through a neural network, but this topic had not been studied before, so various tools were used, such as video tutorials on YouTube, informative web pages and the purchase of a specialized book. in "Neural networks and fuzzy systems" [6].

As soon as block-by-block simulations began, no problems were found, but when doing the final project, the execution time increased considerably, so obtaining a sample involved almost 30 minutes.

The generation of errors started in MATLAB where the necessary calculations were made and saved in a text file that was going to be read later by one of the modules in Vivado. This was changed and ended up randomly generating errors in one of the tool's modules.

In the work, two frequencies have been used, which have had to be generated starting from the crystal clock that the Cmod A7 module had. The 1.5 MHz frequency has been used for encoder emulation and the 10 MHz frequency is entered into the module's RAM for reading and writing.

Lastly, by spending a lot of time to obtain the simulation, it has been difficult to obtain data. That is why, in order to solve this inconvenience, it has been decided to reduce the length of the sample from 4 meters round trip to 1.

2 State of the art of the technology used or applied in this thesis:

Nowadays the 3D printers market has been expanded to its highest level, where the area envolves plastic and metal products. As it is known, a 3D pieces can be made with different materials that achieve the creation of a build. It is fair to point out that in my current job the principal material used in the printing workflow is powder plastic. This plastic is composed by particles that can produce errors on the printing if the sealing is not in good condition. In case of air leaks the powder could be filtered to several subsystems of the printer and affect its performance.

Printing machines used in HP has a component which paints on the bed (surface trolley) the plot demanded, this one uses a motor for its displacement and an optical encoder to accomplish a certain precision. There has been some issues related with bad performance of this element, these have inspired the thesis in question.

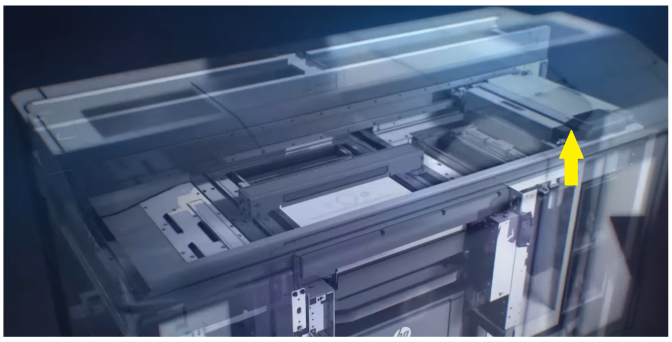


Figure 3: HP printer Multi Jet Fusion (arrow indicates the carriage element).

2.1 The use of encoders

The previous explanation introduces the source of the main objective in this analysis.

Many devices use encoders along with motors to operate with high accurary. A machine in client can behave incorrectly in case the optical encoder indicates a wrong position, the workflow tries to compensate the disruption to reach out the reference position, so the firmware demands more PWM (Pulse-width modulation) to redirect the motor. This order does not solve the issue but worsen the course, which can lead to a crash with another component or a surpass of threshold of displacement. The workflow ends with an electrical fault, so the impact was considered an obstacle which had to be surpassed by an increase of potency that ended up exceeding the maximum in a certain timeout.

In order to avoid that situation it is necessary to corroborate the state of the strip encoder but in a build where the engine needs to be ready, the error above can be easily narrowed down to an encoder issue with a system that analysies the traces of its output and inform its reliability.

2.2 Optical encoder

An optical encoder is a type of motion sensing device that uses light shone through a coded disk to track the movement of a shaft. The encoder provides feedback based on the interruption of light [1]. The light beam is picked up by a photodiode array which responds producing a sinusoidal wave form that is transformed into a square wave or pulse train. The electronic signals from the photodetector are analysed in the same manner as the quadrature signals from an interferometer [2], this second signal is used to determine more detailed motion information, like the orientation of the signals [3].

2.2.1 Fordward and backward

As forementioned, the output from an optical encoder are two signals in quadrature. The signal A follows the same performance as the strip encoder while B has an offset in phase of 90 positive degrees with forward directions and negative otherwise.

An encoder period contain four states that allow to distinguish the orientation of the movement and catch errors when the time between two transitions is not the expected. The delay among states is fixed by the speed of the test, and is called "T_state".

The signals A and B follow the next pattern {"00", "10" , "11", "01"} on a forward direction, however the backward matches with {"01", "11" , "10", "00"} . These four transitions are asociated this way:

$$S0 : "00", S1 : "10", S2 : "11", S3 : "01"$$

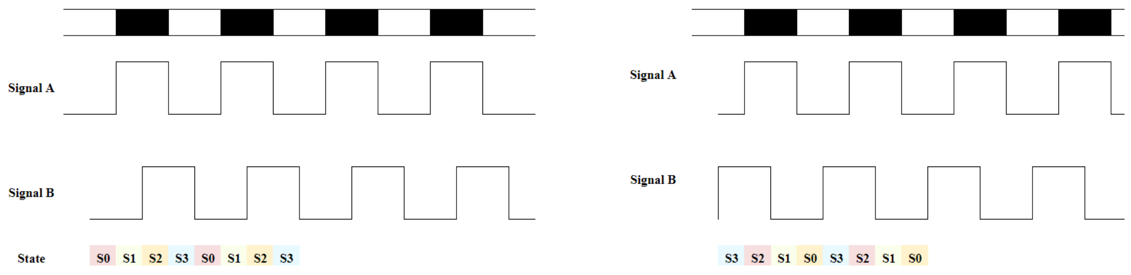


Figure 4: Signals A and B performance.

2.2.2 Performance with errors

When there is a loss of count the signals do not always match theirs patterns, moreover delays are inserted on the operation. On a forward situation, the loss of one count affects the next two transition periods. State S2 looks forward to receive a S3, but the system goes back to S1 with the double of its expected T_state, T_line. This last state remains three times the T_state and continues decreasing to S0. On top of it, the channel starts to recompose and increment to S1 with a T_line delay.

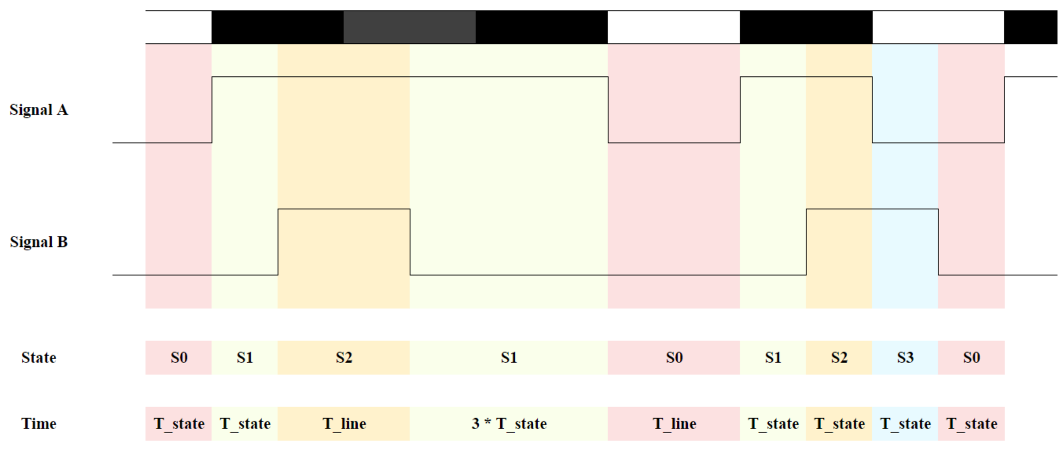


Figure 5: Appearance of errors in a positive direction.

Regarding the backward performance, when a count is missed the pattern is not affected, instead the shift from S1 to S0 carries out a retard of 5 times the T_state.

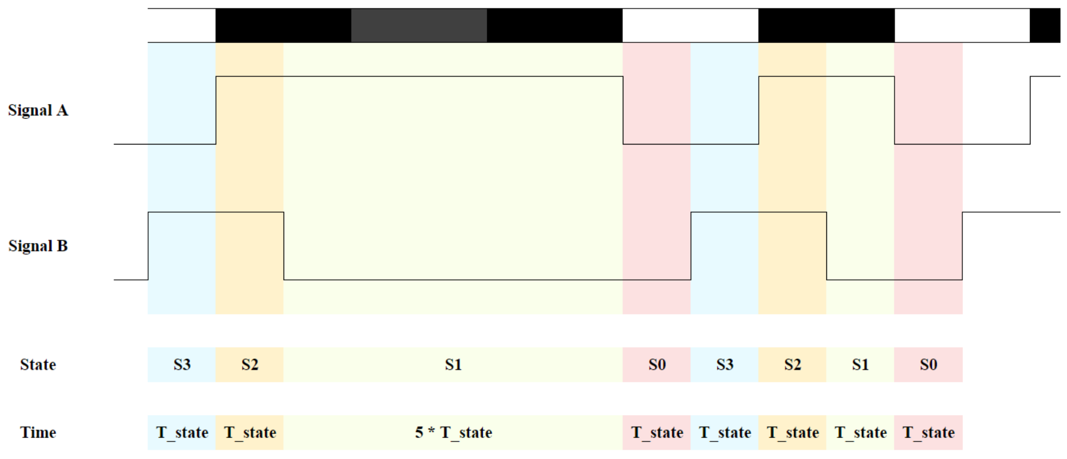


Figure 6: Appearance of errors in a negative direction.

It turns out that in forward mode, the theory says that B is activated in high level, however on the opposite direction the signal is triggered in low level.

2.3 Neural Networks

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another [8].

2.3.1 Structure of an artificial neuron

The ANS mimic the hardware structure of the nervous system, with the intention of building parallel, distributed and adaptive information processing systems, which can present a certain ”intelligent” behavior.

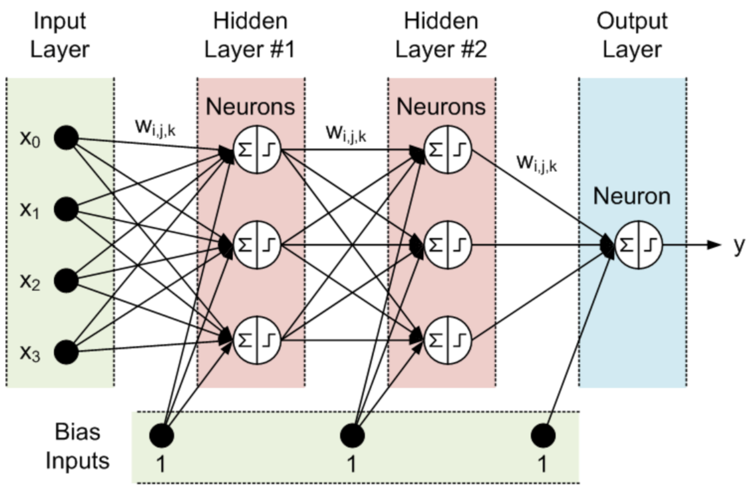


Figure 7: Artificial neuron structure.

2.3.2 Model of an artificial neuron

General model of artificial neuron: an elementary processor or neuron is a simple calculation device that, based on an input vector from the outside or from other neurons, provides a single response or output.

The following elements make up a neuron:

- Set of inputs
- Synaptic weights that represent the intensity of interaction between each presynaptic neuron and the postsynaptic neuron.
- Propagation rule, which provides the value of the postsynaptic potential of the neuron as a function of its weights and inputs.
- Activation function, provides the state of the current activation of the neuron as a function of the previous state and postsynaptic se current.

- Output function, gives the current output of the neuron based on its activation state.

The input and output variables can be binary or continuous, depending on the application model.

2.3.3 Back-propagation

A solution to the problem of training the hidden layer nodes of multilayer architectures is provided by the back-propagation algorithm or BP.

The procedure to follow to train a given MLP architecture using BP is as follows:

1. Randomly set initial weights and thresholds
2. For each pattern μ in the learning set:
   (a) Perform an execution phase to get the network response to the μ-th pattern.
   (b) Calculate the associated error signals Δ_k^μ and Δ_j^μ
   (c) Calculate the partial increase in weights and thresholds due to each pattern μ
3. Calculate the current total increment of the weights and do the same for the biases.
4. Update weights and bias
5. Compute the current error and return to point if still not successful.

2.3.4 Application list

The examples of application of neural networks that we will comment on below always correspond to commercial products.

- Telecommunications. They have been used to build linear equalizers and echo cancellers that are used in modems that work with high-speed transmission.
- Fraud with credit cards.
- Recognition of printed characters.
- Speech recognition.
- Medical applications.

2.4 Digilent Cmod A7 35T FPGA

The Digilent Cmod A7 is a small, 48-pin DIP form factor board built around a Xilinx Artix 7 FPGA. There are 44 Digital FPGA I/O signals and two FPGA Analog inputs that are routed to 100-mil-spaced through-hole pins so that users can integrate programmable logic design directly into a solderless breadboard circuit [5].

2.4.1 Characteristics

The following information predetermine the entire project, due to the capacity able to store in its memory and the length of information to analyse.

Moreover, the buttons allow the user to take part in the modality of the FPGA code.

Features

- 512 kB SRAM with an 8-bit bus and 8 ns access times
- 4 MB Quad-SPI Flash o USB-JTAG Programming Circuitry
- Powered from USB or external 3.3 - 5.5 V supply connected to DIP pins

Interaction and Sensory Devices

- 2 LEDs
- 1 RGB LED
- 2 Push Buttons

Expansion Connectors and System Connectivity

- 48-pin DIP connector with 44 Digital I/O and 2 Analog inputs (0-3.3V)
- One Pmod connector with 8 Digital I/O
- USB-UART bridge

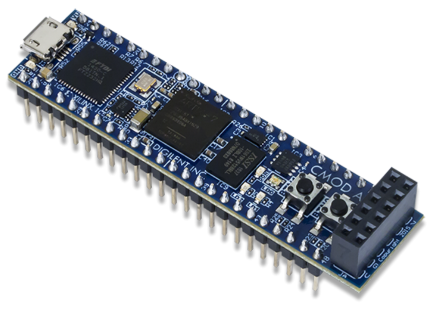


Figure 8: Cmod A7 FPGA.

2.4.2 Useful data

The capacity of the RAM determine the clock used in the encoder emulation, thus specifies the length measurable. The maximum data is 512 kB, which means that the address bus contains 19 bits.

The second and most important information fall on the 8 bit bus of data. Taking into account that two of them are reserved for the channels A and B, the remaining data is fixed further down.

2.5 TCL language

TCL is a very simple programming language. If you have programmed before, you can learn enough to write interesting TCL programs within a few hours. This page provides a quick overview of the main features of TCL. After reading this you'll probably be able to start writing simple TCL scripts on your own; however, we recommend that you consult one of the many available TCL books for more complete information [?].

TCL was designed from the outset as a flexible language with a small core, that could be adapted in ways the original authors couldn't have foreseen. Indeed, thanks to this flexibility, TCL has been used in everything from Cisco routers and Tivo set top boxes, to AOL's web servers, to the NBC television network's control center.

2.5.1 Applications

Thanks to its strong string manipulation and networking capabilities, TCL is a great match for web applications. Available software runs the gamut from client side libraries and utilities, web servers ranging from skinny embeddable libraries to full blown enterprise class servers like AOLServer, to a variety of application frameworks.

In the early years, Tk helped establish TCL's popularity as the only sane way to write GUI applications under Unix and X11. People found TCL's dynamic approach a natural fit for GUI's, making GUI development much easier and faster than with lower-level C and C++ oriented toolkits.

In modern times, Tk is still the leader when it comes to delivering a high level GUI cross-platform toolkit that naturally fits in with dynamic languages. Furthermore, theming support added in Tk 8.5 enables a native look on Unix, Windows and Mac OS X platforms.

Long before "test-driven development" had become a buzzword, TCL had been well established as a testing powerhouse, leveraging its ability to easily interface with other software and hardware.

Because TCL is a very compact language, and is easily integrated with special hardware, it is a popular choice for embedded development. You'll find TCL hidden away on many devices, including many networking products from Cisco and others, and set-top boxes including Tivo. Embedding TCL within other software projects is of course also hugely popular, and has become the dominant control language in some industries, such as in electronic design automation (EDA) and computer-aided design (CAD) applications.

3 Encoder parameters

The MATLAB tool was used in order to specify if the project could done accomplishing the specifications demanded.

The order of the process was to fix a number of meters to know if the FPGA could handle it with its capacity. The whole amount of calculations were done using the following specifications:

- An strip encoder por 150 LPI (lines per inch)
- Speed of 20 IPS (inches per second)

A pysical encoder unit, which can be black or white, must be fractioned in a number power of 2 in order to easy the problem. This subdivision is named "eu" and is going to be the resolution of the system, so a new abstract unit is created. This eu is the number of times the FPGA triggers its cristal clock to receive information from the optical encoder in one pyshical encoder unit. As it can be seen below, the number power of two chosen is 7.

$$1peu = 2^7 eu$$

It is crutial to mentioned that the data stored in the module Cmod A7 are just the transitions of change of the channels A and B, so the capacity needed follows the next rule.

$$2^{capacity-7} < 2^{19}$$

With this operation, the length of eu is fixed to 24 bits that allows negative values. The next step is to define the constants of equivalence; as it is a system of 150 LPI, one physical encoder unit is is equal to 150 multiply by 4 because each signal (A and B) generates 2 states. Once the convertions are stablished, this thesis applies all the calculations on a 2 meters lenght base. The equivalence to encoder units informs the amount of bits used for this operation and rooted in this phylosophy, the period of time for the simulation in the Vivado tool is defined as the number of total encoder units devided for the time of that traject.

With this information, the system evolve increasing the number of encoder units by one each clock's cycle, let's point out that this set is directly related with the speed of the system.

The following table shows the constants that define the project.

Table 1: Constant's table.

Parameter	Value
speed_ips	20
long_meters	2
lpi_system	150
bits_eu	24
k_signalsAB	4

The equivalences mentioned in the previous sections are summarized below.

Table 2: Equivalences's table.

$150 * 4$ peu	1 inch	$k\_peu\_inch = lpi\_system * k\_signalsAB$
2^7 eu	1 peu	$k\_eu\_peu = 2^7$
$150 * 4 * 2^7$ eu	1 inch	$k\_eu\_inch = k\_peu\_inch * k\_eu\_peu$

Finally, the results of the necessary calculations are included to determine the clock frequency that has to be applied to the Vivado tool when doing the simulation

Table 3: Calculation's table.

2 m
78.74 inches
6047244 eu
$\log_2(6047244) = 22.53$ bits
$22.53 < 24$
$round(2^{22.5}) = 5931642$ eu $= 78.86$ inches
$t_{sim} = \frac{inch_{total}}{IPS} = \frac{78.86}{20} = 3.94$ s
$T_{CLK} = \frac{tsim}{total_{eu}} = \frac{3.94}{5931642} = 664ns$ s
$f_{CLK} = \frac{1}{T_{CLK}} = 1.5$ MHz
$speed_{eu/s} = \frac{total_{eu}}{tsim} = \frac{5931642}{3.94} = 1.5$ eu/s
$speed_{eu/T} = speed_{eu/s} * T_{CLK} = 1.5 * 664 * 10^{-9} = 0.9996$ eu/T
$accel_{m/s^2} = 0.12 * g = 0.12 * 9.81 = 1.1772$ m/s^2
$accel_{eu/s^2} = accel_{m/s^2} * \frac{39.37inch}{1m} * \frac{76800eu}{1inch} = 3.5594 * 10^6$ eu/s^2
$accel_{eu/T^2} = accel_{eu/s^2} * T_{CLK}^2 = 3.5594 * 10^6 * (664 * 10^{-9})^2 = 1.57 * 10^{-6}$ eu/T^2
50 cm
19.7 inches
1511811 eu
$\log_2(1511811) = 20.53$ bits
$20.53 < 24$
$round(2^{20.5}) = 1482910$ eu $= 19.30$ inches
$_{sim} = \frac{inch_{total}}{IPS} = \frac{19.30}{20} = 0.965$ s

4 Vivado modules

Xilinx's Vivado Design Suite is an integrated design environment for embedded development. Vivado has been chosen because of the previous experience with the tool along the seventh quarter of the degree. This software suite is used to code the encoder emulation, storage data control and RAM behaviours. It is one of the most reliable tools when it comes to waveforms simulations and VHDL/Verilog code verification.

The Vivado tool implements four differents situacions. Each of them defines a role and its is crutial to be sincronized with the following so the entire simulation runs correctly. The first and most essential one is the emulation of and optical encoder, whose inputs are used to start performing and reading the values from the SRAM. The second one is in charge of sending the A and B transition changes and control the memory access to the FPGA. The next module simulates a memory RAM where the data is stored and read subsequently. The last one interacts with the user, it is so when data is read, it is send to a text file to be analysed at a later stage.

4.1 Frequency divisor

The FPGA contains a cristal clock of 100MHz that must be adapted to approach the frequency demanded in the previous calculations. In this section, the frequency from the Cmod A7 is converted into a 1.47 MHz for the encoder emulation calculation that is sligthly similar to the one needed, 1.5 MHz. A second frequency of 10 MHz is used for the RAM in order to be accurarate in data's transitions.

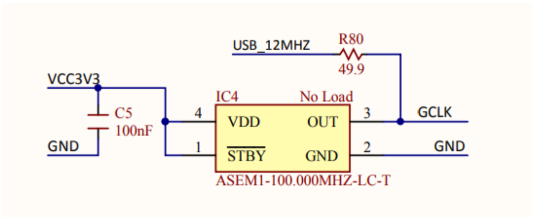


Figure 9: FPGA's Cristal clock.

4.2 Encoder emulation

In order to achieve a powerful encoder emulation, it is necessary to create different blocks where each one has a specific function.

4.2.1 Encoder operator

This module allows the user to iterate with the test and choose whether to run two types of behaviors. We present the first case, in which we want to simulate a positive and negative acceleration ramp since, at a certain point, a negative value will be indicated.

If you want a behavior that replicates a round trip MRU movement, the second button must be pressed. For this last event, velocity and initial position outputs will be fixed for the rest of the test as one and zero correspondingly; since the speed is constant, that constant output is the only one to have value one. In addition, the feedback of the current position of the encoder unit serves to change direction and thus be able to cover as many situations as possible.

Finally, the ”enable” output informs if any of the available buttons have been pressed. This variable allows to start with the generation of errors.

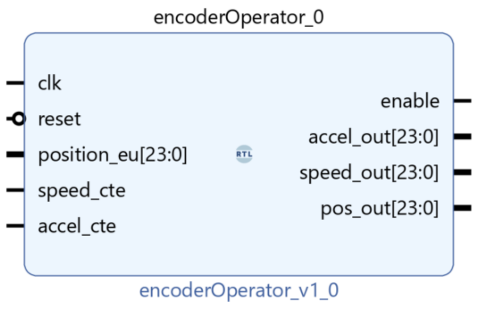


Figure 10: [Module] Encoder operator.

4.2.2 Encoder units position

The functionality of this section is to increase or decrease the variable counter taking into account the speed, acceleration and position.

The three input registers are the integral calculations of the one before.

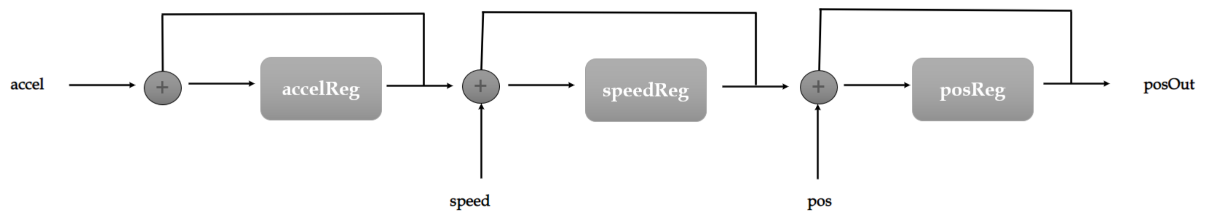


Figure 11: Integral to obtain posOut.

The euPos module modifies the output depending on the ”error” input value. Once this is ’1’ the ”e_block_minus” bits indicates if the encoder position register must stay the same or decrease its value.

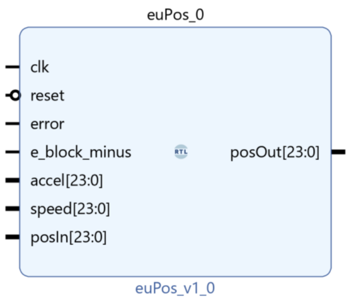


Figure 12: [Module] Encoder position.

4.2.3 Encoder units to AB signals

This code is one of the simplest within the project, it transforms positions 7 and 6 of the encoder unit vector to signals A and B.

Table 4: AB's conversion table.

pos[7,6]	AB
00	10
01	11
10	01
11	10

The fact of extracting those indexes from the encoder position is beacuase the calculations declare that the number of iterations inside a physical encoder is 2^7, 128. That means that each state or encoder transitions takes place 2^6, 64 units. So the positions 7 and 6 accomplish this demand.

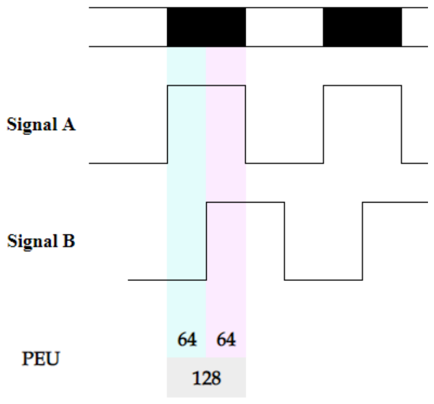


Figure 13: States in a single peu.

The fault occurs when the AB signal transfer from AB = B"11" to AB = B"10". Each pos_{eu} stays the same during T_state time, so when due to an error the value $pos_{eu}[7,6]$ remains 2 times T_state, the error market is considered to be the highest value possible of $pos_{eu}[7,6] =$ "01", which is $pos_{eu}[7:0] = B"01111111" = X"7F"$.

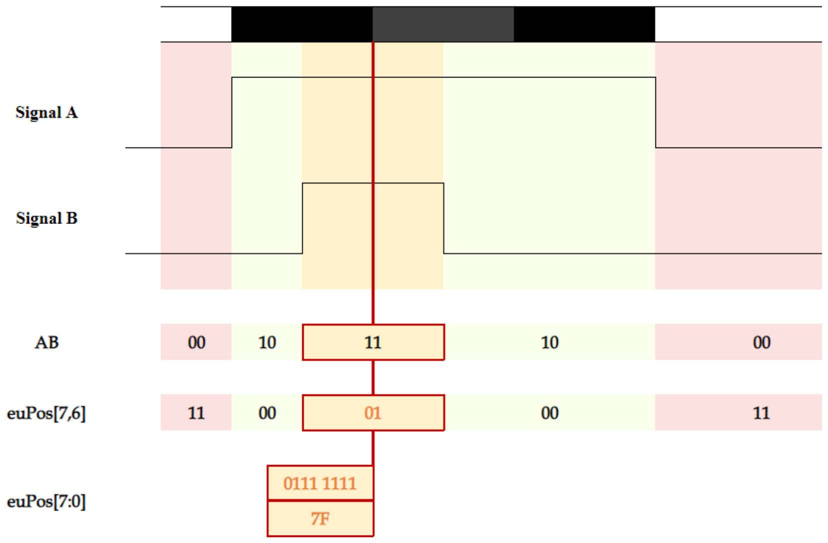


Figure 14: Error in a X"7F" euPos[7,6].

On the other hand, when going backwards, the pattern where the first anomaly is found is in the transition from AB = B"10" to AB = B"00". As mentioned before, the error is target as the maximum value of $pos_{eu}[7,6] =$ "00", in which case leads to $pos_{eu}[7:0] = B"00111111" = X"3F"$.

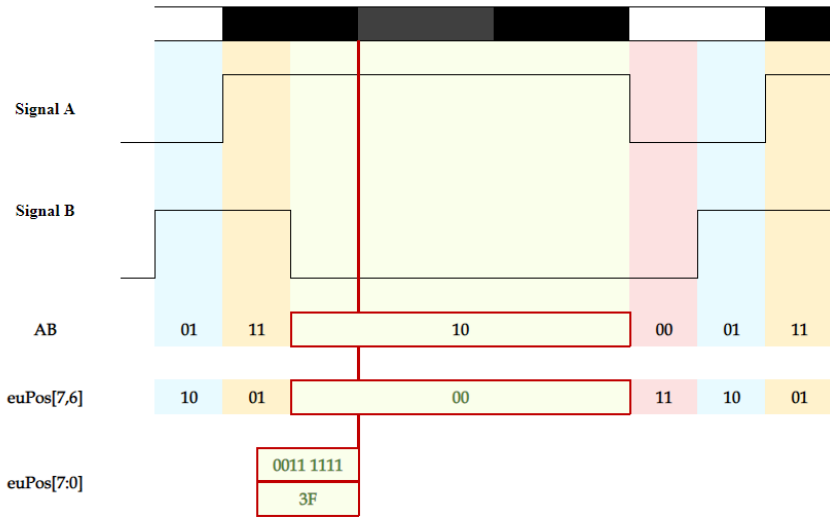


Figure 15: Error in a X"3F" euPos[7,6].

4.2.4 Error generator

The main functions are two: the generation of random errors and the management control when detecting one.

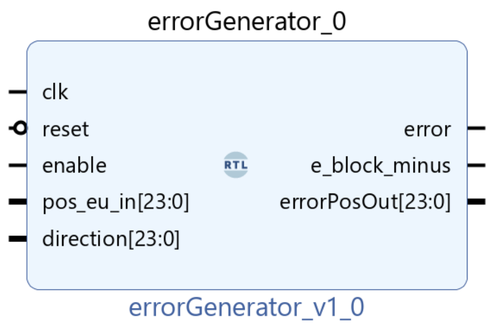


Figure 16: [Module] Error generator.

Regarding the first case, when a change is detected in the "enable" input of active low pass, an array of errors of length "qErrores" is generated. This last number is set by pressing the reset button, and its value ranges from 16 to 19 thanks to the following command executed by the $rand_{slv}(2)$ function with a uniform probability. In addition, at the same time, the appearance of the first error along the path is calculated randomly, having as maximum "maxPosError".

This maxPosError corresponds to the 16 bits with the highest weight of the error position, since the remaining 8 bits belong to the suffix that can be X"7F" if the displacement is positive or X"3F" if it is negative.

Calculations determined that FPGA capacity is able to support readings 4 meters round trip, but when simulating it with Vivado, execution time extended much more than expected. Therefore, in order to simulate efficiently, it is decided to change the length to 40 inches, making 2 displacements of 20 inches (1536000 eu) with a change of direction.

At the same time, taking into account that an error can be concentrated in 1 centimeter and a half (45354 eu = X"B12A"), the last error that can be found must be in the account 1490646 eu (X"16BED6"), being less than X "16BE7F" we set the value of maxPosError to the 16 bits of greater weight X"16BE" (5822 eu), so the first error is computed in the range $["0000", "16BE"]$ with uniform probability, with the help of $rand_{int}$.

$$firstError_{16-bits} = rand_{int}(0, maxPosError)$$

Once the reset status has been exceeded, on each clock edge the enable value is verified and when it meets the conditions, the first value found (the one above) is indexed in the created register array and the following registers are created randomly inside a for loop of range 1 to qErrores-1, the data had these 16-bit limits $[min_{val}, max_{val}]$.

$$errorsReg(i) = rand_{int}(min_{val}, max_{val})$$

These thresholds depend on the first random error calculated, since the minimum takes that value. As mentioned before, hypothetically we will find a visual fault of a length of one and a half centimeters, so its maximum value in encoder units will be the minimum plus that length. The records stored in the error array, errorsReg, only save the 16 bits with the highest weight, since they are after concatenated by a suffix depending on their orientation.

If all 24 bits of the encoder unit position error were taken into account, the following formula would apply:

$$max_{val_{24-bits}} = error_{24-bits} + distanceError_{24-bits} = error_{24-bits} + X"B12A"$$

If only taking into account the lowest 8 bits weight, then there will be a loss of 42 decimal units, which considering the resolution is insignificant. That is the reason why the maximum value is the sum between the minimum and X"B1" which in decimal is 177 units.

$$max_{val_{16-bits}} = error_{16-bits} + distanceError_{16-bits} = error_{16-bits} + X"B1"$$

$$[min_{val}, max_{val}] = [firstError, firstError + 177]$$

To detect if the current position coincides with any value within the error vector, one by one is inspected within a for loop. When the displacement is positive (bit of the speed with the greatest weight equal to '0') the concatenated register is compared with the suffix and the position increased to 2. On the other hand, when the movement is backwards, the concatenated register is compared with the suffix and current value; that is why distinctions are made within the for loop.

On the error control side, when they detect that the auxError edge has been activated, two behaviors differ depending on the address.

If a positive speed is applied, then to reproduce the loss of a count, the encoder counts must be decreased through 2 state periods (T_{state}) so that A and B are the desired ones, then the result stays the same for the time of 2 T_{state} and it decreases again T_{state} to then continue with normal operation.

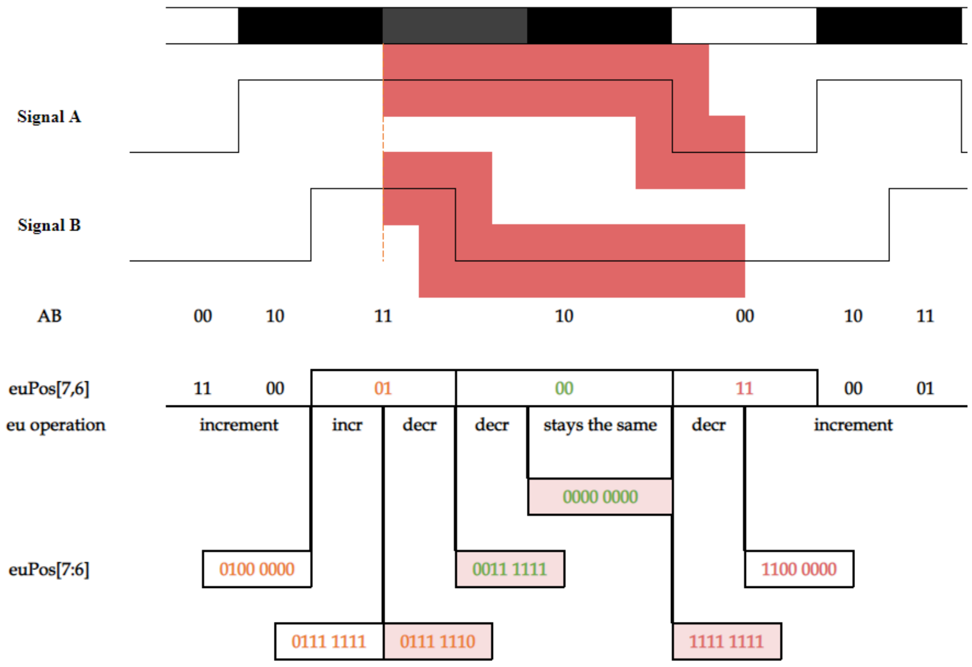


Figure 17: Analysis error in a positive direction.

In the opposite case, it is much simpler because it is only necessary to block the value for 5 periods.

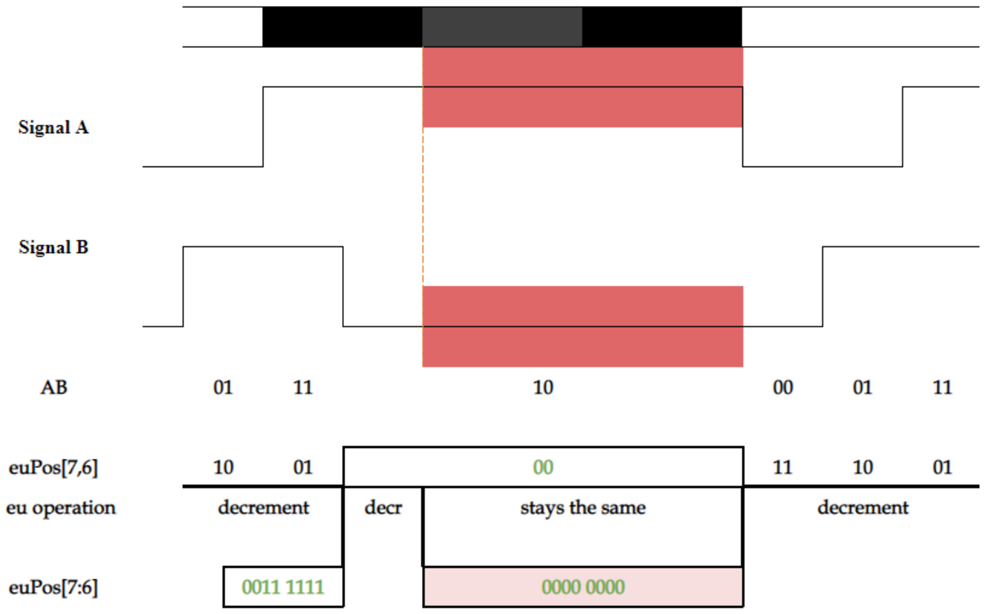


Figure 18: Analysis error in a negative direction.

A counter is used to calculate the periods, and reset to zero once it reaches the desired target. In section 2.2.2 you can see a correlation with what was explained above.

4.3 Cmod A7

The code of the following modules is the reference to be implemented in the FPGA when binary data is received from channels A and B of the optical encoder in real life.

4.3.1 AB's transition change

This module is in charge of detecting a transition change and activating the write by sending the previous data. In addition, it calculates the time difference between one transition and another and sends it as information on a 7-bit bus.

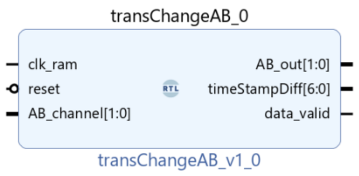


Figure 19: [Module] AB's transition change.

4.3.2 Address control

The function of this section is to control the flow of read and write commands. To do this, a 3-state machine is used: $S0_{Idle}, S1_{Write}, S2_{Read}$. The first symbolizes the inactivity of the "write_data" and "read_data" signals. On the other hand, as soon as one of them is activated, for example the first one, the state that follows is the $S1_{Write}$ that sends the data and "addrW_aux" the address. The output data is the concatenation between the bits of the physical encoder with which they mark the time difference. As the data that is sent to the FPGA can only have a length of 8 bits, it is decided to add the greater 6 bits weight of the time stamp.

The writing address is a counter that increases each time the data is requested to be written and indicates in which memory position it is desired to be stored. When the workflow finishes, it returns to the initial state in which it waits for a stimulus. In the case of wanting to read the RAM registers, it is done by imitating the behavior of a FIFO. Which means that the first in, first out. Therefore, the RamOEn output, goes low and sends the address it wants to read. This is incremented every time you want to read.

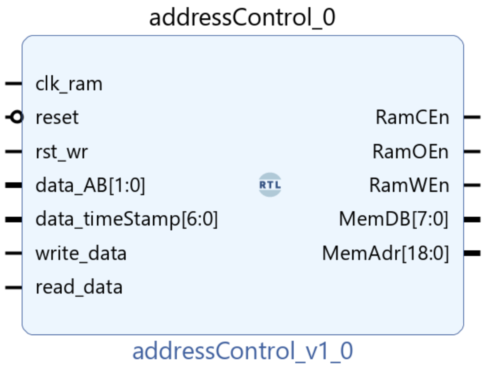


Figure 20: [Module] Address control.

4.3.3 Read data's RAM

One of the FPGA buttons is used by the user to read all dara from RAM. This brings about a pulse train signal to trigger the RamOEn and get the data.

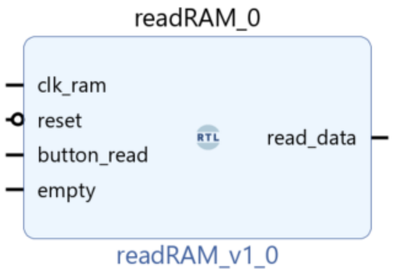


Figure 21: [Module] Read data from RAM.

4.4 RAM Memory

To verify the correct functioning of the system, the behavior of a RAM memory has been emulated; with its respective CE, OE and WE inputs that indicate reading and writing, as well as setting its capacity of 512 kB to store the 8 bits of data in the indicated address.

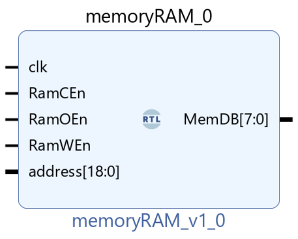


Figure 22: [Module] RAM Memory.

4.5 Store data to a file

In order to be able to analyze the samples using Python code, it is necessary to store them in a text file in an orderly manner. Regarding the code, it uses a text library that allows writing in files. First, the information is split in two: the time stamp with the 6 bits of greater weight and the corresponding AB data of the 2 bits of lesser weight. With the activation of inverse logic of the OE, the following 3 parameters are written in a single line: address, time stamp and AB data.

The data is attached without overwriting the file and in this way the base data are generated.

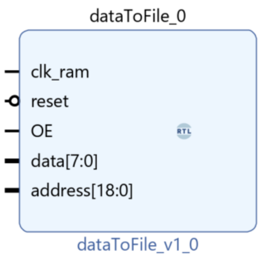


Figure 23: [Module] Data to file.

4.6 Constraints

Design constraints define the requirements that must be met by the compilation flow in order for the design to be functional on the board. Not all constraints are used by all steps in the compilation flow. For example, physical constraints are used only during the implementation steps (that is, by the placer and the router) [4].

Digilent gives the option of downloading the constraints associated with the Cmod A7 board so that only the input and output ports have to be renamed. Next, the code that orders this association will be shown along with the schematic so that it can be seen that the pins coincide.

4.6.1 CmodA7 constraints xdc file

The memory-related ports match those shown on the Vivado modules.

```
# Clock signal 100 MHz
set_property -dict { PACKAGE_PIN L17   IOSTANDARD LVCMOS33 } [get_ports { CLK }];
# Buttons
set_property -dict { PACKAGE_PIN A18   IOSTANDARD LVCMOS33 } [get_ports { BTN_EncOp }];
set_property -dict { PACKAGE_PIN B18   IOSTANDARD LVCMOS33 } [get_ports { BTN[1] }];
## Cellular RAM
set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 } [get_ports { MemAdr[0]  }];
set_property -dict { PACKAGE_PIN M19   IOSTANDARD LVCMOS33 } [get_ports { MemAdr[1]  }];
set_property -dict { PACKAGE_PIN K17   IOSTANDARD LVCMOS33 } [get_ports { MemAdr[2]  }];
set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports { MemAdr[3]  }];
set_property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33 } [get_ports { MemAdr[4]  }];
set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 } [get_ports { MemAdr[5]  }];
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { MemAdr[6]  }];
set_property -dict { PACKAGE_PIN W19  IOSTANDARD LVCMOS33 } [get_ports { MemAdr[7]  }];
set_property -dict { PACKAGE_PIN U19  IOSTANDARD LVCMOS33 } [get_ports { MemAdr[8]  }];
set_property -dict { PACKAGE_PIN V19  IOSTANDARD LVCMOS33 } [get_ports { MemAdr[9]  }];
set_property -dict { PACKAGE_PIN W18  IOSTANDARD LVCMOS33 } [get_ports { MemAdr[10] }];
set_property -dict { PACKAGE_PIN T17  IOSTANDARD LVCMOS33 } [get_ports { MemAdr[11] }];
set_property -dict { PACKAGE_PIN T18  IOSTANDARD LVCMOS33 } [get_ports { MemAdr[12] }];
set_property -dict { PACKAGE_PIN U17  IOSTANDARD LVCMOS33 } [get_ports { MemAdr[13] }];
set_property -dict { PACKAGE_PIN U18  IOSTANDARD LVCMOS33 } [get_ports { MemAdr[14] }];
set_property -dict { PACKAGE_PIN V16  IOSTANDARD LVCMOS33 } [get_ports { MemAdr[15] }];
set_property -dict { PACKAGE_PIN W16  IOSTANDARD LVCMOS33 } [get_ports { MemAdr[16] }];
set_property -dict { PACKAGE_PIN W17  IOSTANDARD LVCMOS33 } [get_ports { MemAdr[17] }];
set_property -dict { PACKAGE_PIN V15  IOSTANDARD LVCMOS33 } [get_ports { MemAdr[18] }];
set_property -dict { PACKAGE_PIN W15  IOSTANDARD LVCMOS33 } [get_ports { MemDB[0]   }];
set_property -dict { PACKAGE_PIN W13  IOSTANDARD LVCMOS33 } [get_ports { MemDB[1]   }];
set_property -dict { PACKAGE_PIN W14  IOSTANDARD LVCMOS33 } [get_ports { MemDB[2]   }];
set_property -dict { PACKAGE_PIN U15  IOSTANDARD LVCMOS33 } [get_ports { MemDB[3]   }];
set_property -dict { PACKAGE_PIN U16  IOSTANDARD LVCMOS33 } [get_ports { MemDB[4]   }];
set_property -dict { PACKAGE_PIN V13  IOSTANDARD LVCMOS33 } [get_ports { MemDB[5]   }];
set_property -dict { PACKAGE_PIN V14  IOSTANDARD LVCMOS33 } [get_ports { MemDB[6]   }];
set_property -dict { PACKAGE_PIN U14  IOSTANDARD LVCMOS33 } [get_ports { MemDB[7]   }];
set_property -dict { PACKAGE_PIN P19  IOSTANDARD LVCMOS33 } [get_ports { RamOEn     }];
set_property -dict { PACKAGE_PIN R19  IOSTANDARD LVCMOS33 } [get_ports { RamWEn     }];
set_property -dict { PACKAGE_PIN N19  IOSTANDARD LVCMOS33 } [get_ports { RamCEn     }];
```

At the same time, the PACKAGE_PIN comply with the electrical diagram of the module.

The BANK 4 collects the parameters referring to the RAM memory while the BANK 16 is in charge of its 2 physical buttons.

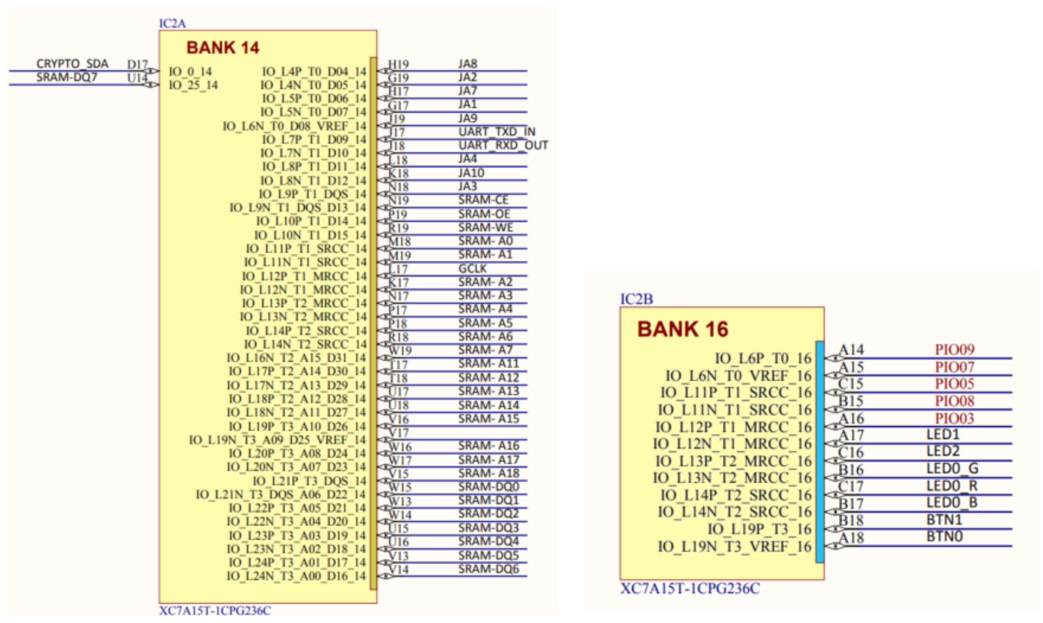


Figure 24: Cmod A7 conexions.

5 Vivado waveform results

In this section the modules from Vivado are presented along with their simulation. The interaction among modules clarify the purpose of each of them.

5.1 Encoder emulation

One of the most fundamental parts of the project is the creation of an optical encoder emulation. Because of that, a study of signal behaviour in the presence of an anomaly is needed.

In the first instance the ”encoderOperator” triggers to generate an encoder position. The BTN_EnOp input which determines the following outputs: acceleration, velocity and initial position. At the same time, the enable is activated, which is an OR of the options to simulate a path with constant speed (speed_cte) or a path with constant acceleration (accel_cte). Since we want to simulate a back and forth, we check the current position (position_eu input) to go back with negative outputs. Something that has not been mentioned is that the accel_cte option could not be implemented, which is why it has a constant value of 0.

This is because when doing the calculations it has been seen that the output's acceleration is of the order of -6, so it is considered tiny. An attempt was made to transform the desired number to a fixed-point, but that implied euPos module to use the package fixed_pkg.VHDL decodes the input accel. Unfortunately, when trying to simulate the wave, it reported that the library used did not allow that execution. With all this, it was decided not to implement this functionality since if it was not possible to visualize if the code worked, it was not worth spending more time on it.

A point and aside, the feedback between errorGenerator and euPos reflects that as euPos can modify its output value depending on ”error” and ”e_block_minus”. The error generator defines the first error based on the activation of the reset, and initializes the parameter qErrores when the ”enable” input is activated.

If the position of the encoder (pos_eu) coincides with an error created within the array, then the outputs are activated in one way or another depending on the bit with the highest weight of ”direction”, which is the speed. It is important to remember that 22.5 bits were needed, that in practice are 23 bits to reach 2 meters, but for the sake of working with negative numbers, one more bit has been introduced, this indicates the direction of the path. If it is '0' then it is considered a positive displacement and otherwise if it is '1'.

Finally, euConvAB converts the encoder positions into AB signals and activates the empty output when ”position_eu” reaches the value 0, that is when reaches the end of its run and returns to 0. Moreover, the time stamp information at the moment is sent as well.

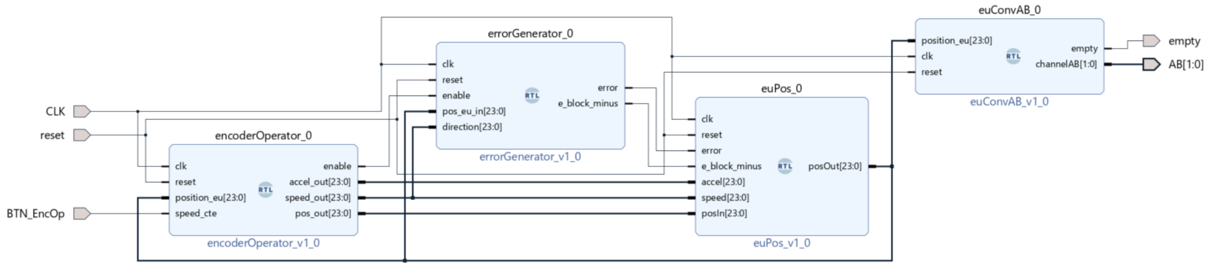


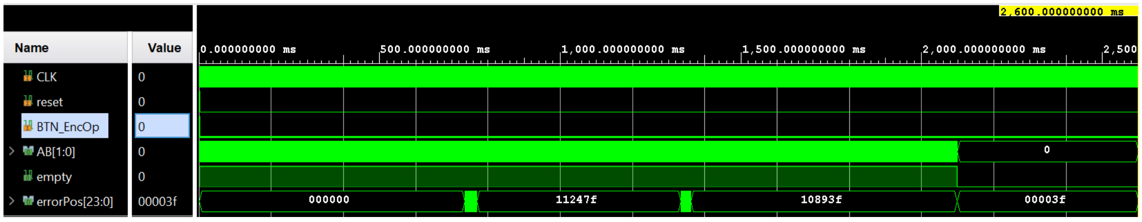
Figure 25: Encoder emulation.

When simulating this module, the inputs have been initialized with the following tcl code. The clock already takes into account that it passes through a frequency divider, so its value is not exactly 1.5 MHz, but 1.47 MHz, which is the output bounded to the desired value.

```
restart
add_force {/encoderEmulation_wrapper/CLK} -radix bin {0 0ns} {1 340ns} -repeat_every 680
    ns
add_force {/encoderEmulation_wrapper/reset} -radix bin {1 0ns} {0 2.2us}
add_force {/encoderEmulation_wrapper/BTN_EncOp} -radix bin {0 0ns} {1 11.6us} {0 12.5us}
run 2 s
```

In the next figure the reset and BTN_EncOp signals are activated at the beginning of the wave to trigger the emulate constant course option.The AB output shows that once it finishes traveling 20 inches out and 20 back it remains at 0. Empty is set to 0 when the "encoder operation" order ends, what implies that all the samples have been stored and the reading can proceed.

It is important to highlight that errors are replicated, because the errors found in the positive direction must appear on the return, although it has already been explained before that a loss of count manifests itself in a different way depending on the direction.



Name	Value
CLK	0
reset	0
BTN_EncOp	0
AB[1:0]	0
empty	0
errorPos[23:0]	00003f

Figure 26: Encoder emulation wave 1.

The figure below is a zoom of the first section where random errors are generated and a fairly uniform distribution can be seen. The affected piece occupies 33 ms, which at a speed of 20 IPS is equivalent to having traveled the expected distance, 1.67 cm.

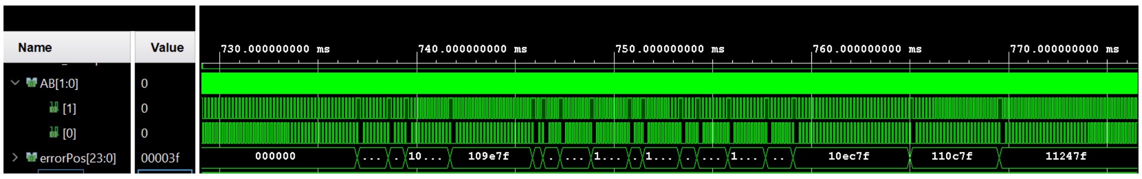


Figure 27: Encoder emulation wave 2.

In order to verify that it really behaves like an encoder when losing an account, it has been zoomed in both positive and negative.

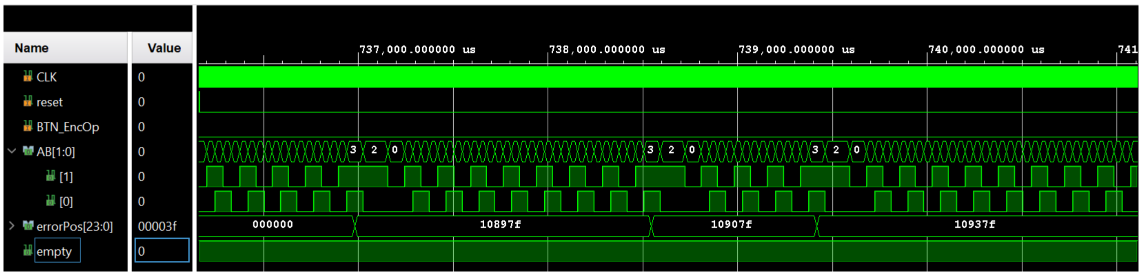


Figure 28: Encoder emulation wave 3.

With respect to a forward direction, it is seen that activation predominates when A is 1, instead it is the other way around in the opposite direction.

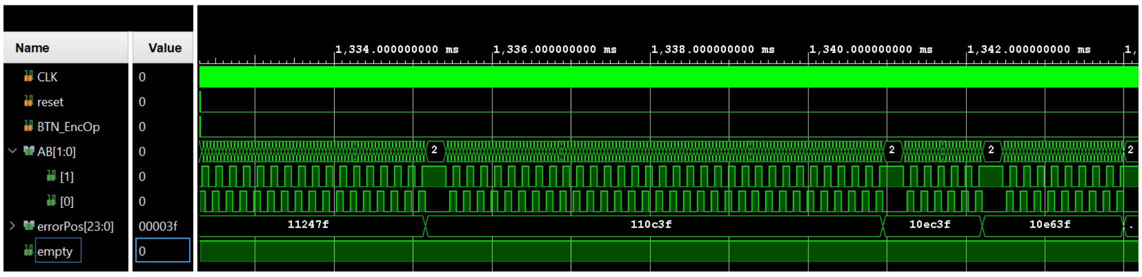


Figure 29: Encoder emulation wave 4.

5.2 Cmod A7

This set includes the code that would be implemented in the FPGA if it had 2 buttons and signals A and B of the encoder as inputs.

To begin with, signals A and B transitions are stored into the RAM memory along with the time stamp difference, at the same time data_valid is activated in each changeover.

This parameter is passed as write_data to the next module, addressControl, and activates the write command by generating control signals from RAM. Add that the writing reset activates when the button BTN_EncOp is pressed.

On the other hand, you have the option of reading the RAM memory with the BTN_Read button, which does not allow reading if empty is 0 since it considers it empty if it has not finished writing yet.

The MemDB input is the output of RAM when the RamOEn command is set low. It is from the set of read_data, MemAdr and MemDB that the data is written to a text file, all implemented in dataToFile.

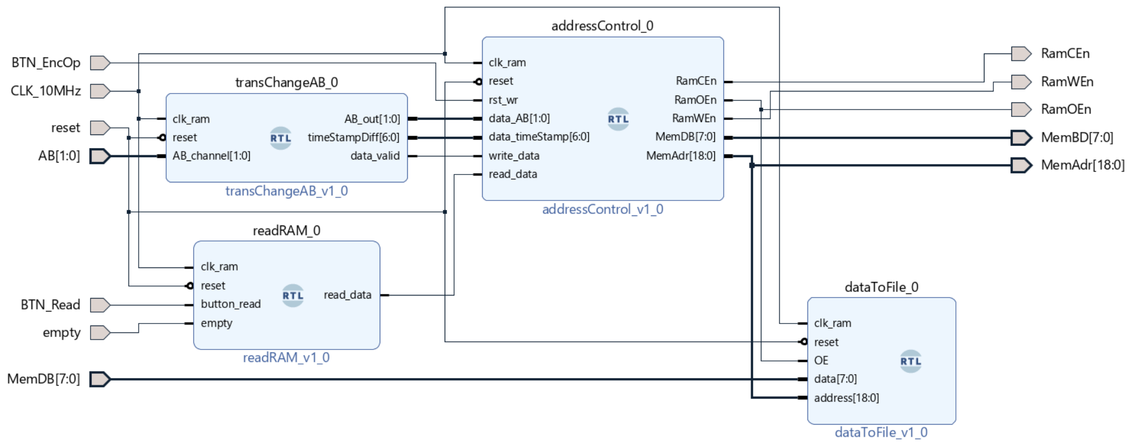


Figure 30: [Module] Cmod A7.

The clock that executes the read order is 10 MHz and that is why it is specified that way in the tcl code. In addition, since we only want to teach the operation of capturing transitions and how this affects the RAM signals, the program is run for 300 μs.

```
restart
add_force {/CmodA7_wrapper/CLK_10MHz} -radix bin {0 0ns} {1 50ns} -repeat_every 100ns
add_force {/CmodA7_wrapper/reset} -radix bin {1 0ns} {0 50ns}
add_force {/CmodA7_wrapper/BTN_EncOp} -radix bin {1 0ns} {0 70ns}
add_force {/CmodA7_wrapper/AB} -radix bin {01 0ns} {11 150us} {10 195us} {01 240us}
add_force {/CmodA7_wrapper/BTN_Read} -radix bin {0 0ns}
run 300 us
```

In this image you can see two entries that are not defined, this is intentional since this part of the block does not want to check that subsystem.

The RAM control ports work with reverse logic and also when this happens the MemAddr counter is increased in the reading process.

Clarify that the data inserted in AB has been random, we only wanted to simulate that the operation was as expected.

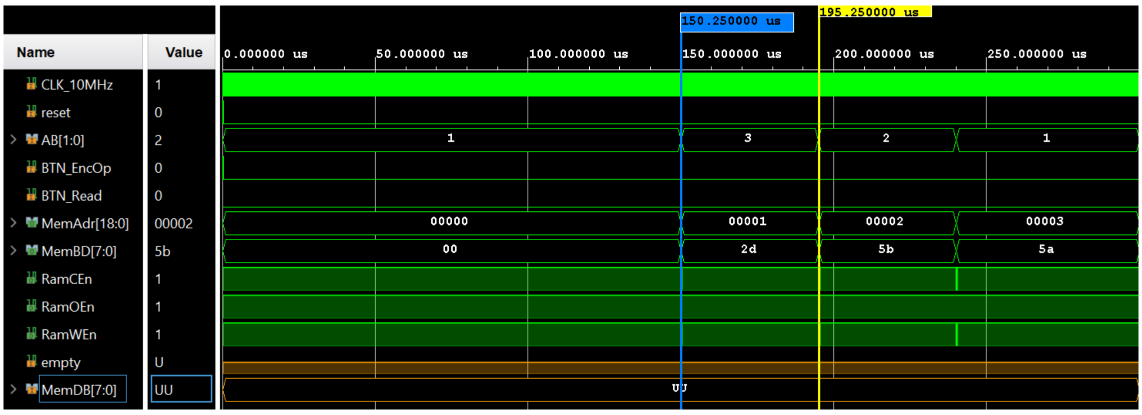


Figure 31: Cmod A7 wave 1.

The figure above has been zoomed in order to focus on the marker placed. At that time, the RAM detects a writing command and does as so in the address zero the data ”00101101”.

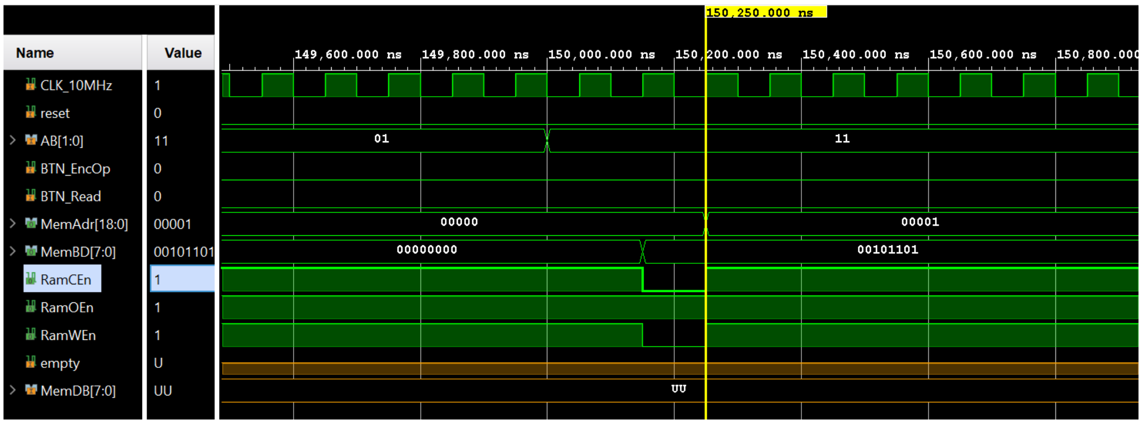


Figure 32: Cmod A7 wave 2.

5.3 Final block diagram

It is in the final project where the frequency divider is implemented, the one calculated in Matlab is the one that should serve as input in the encoder emulation, 1.5 MHz, and the other that has been created is 10 MHz in order to be used in the RAM and as a point of execution in the Cmod A7 module.

It’s outstanding, but there is no need of outputs so the data has to be written to a text file. The information that is transferred from the emulation to the Cmod are the AB data and the empty that indicates that the operation has already been completed. The input BTN_EncOp of Cmod A7 works as a write reset since when the operations button is pressed the counter is reset to 0 to do not overflow the RAM memory.

Finally, although the memory data bus is bidirectional, it has been preferred to fix it by MemDB_in and MemDB_out for simplicity of interpretation.

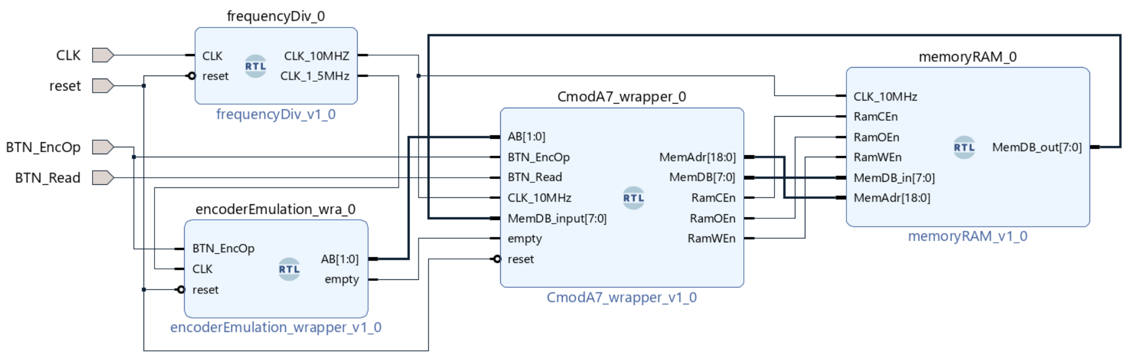


Figure 33: Project.

6 Python code

The data stored in the text file must be analyzed and specified if any indication of error has been found in the sample.

In order to achieve this, two Python documents have been created. The first one collects all data from Vivado's simulation, the other one relies on analysing through neural networks the prediction of an anomaly in the sample.

6.1 Data process code

In the first executable, each Vivado text file is processed inside a while loop, within operates four functions:

- **obtain_data(i)**

  This function locates the path of the file that you want to read from index "i" and copies it to the newArrayData variable that splits the data to be able to separate them by rows.

  Once done, it goes through a for loop to obtain the information for each of the columns and create 3 new vectors: direction, time and AB.

  While doing that, must rememeber that the time information was a 7-bit array that had to lose the lower order bit information since the ram data bus only allowed 8 bits($data_{8-bits} = time_{6-bits}$ & ab_{2-bits}). That is why 1 bit is added (Python code below).

  $$time_{7-bits} = time_{6-bits} +' 1'$$

- **peu_performance()**

  Regarding to performance, units of PEU are increased or decreased depending on the calculated direction that is determined taking into account the current and next AB position.

  In addition, an array of ones and zeros is made that is activated when the time between samples exceeds the established limits, indicating an anomaly (probTime). At the same time, calculates a time trace that can be useful to graph the position through time.

  This funtion returns the PEU and time vectors, and two more vectors:

  – PEU vector: cumulative vector of peu values.

  – time vector: cumulative vector of time values.

  – direction vector: binary vector that indicates 1 when forward and 0 otherwise.

  – probTime vector: binary vector that indicates when time exceeds the thresholds appending value 1.

- **data_NN(peu, probTime)**

  In this section, the maximum value of PEU vector determines the orientation change. The index asociated to that top is store in a new variable named "maxPosPeu".

  The time vector "probTime" splits in two, from 0 to maxPosPeu (probTimeFordward) and the rest(probTimeBackward). It is necessary to remember that when an anomaly is found in the delay, an append of value 1 is added to probTime, so if we add the items of proTimeFordward and return the sum, the result is the number of times that anomalies have been found, the next step would be to relativized. Exactly the same is done with respect to the lap time. The "norm" parameter is the division of the maxPospeu and 0.508, which is the length this test is taking into account, but in case the length pretends to be extended, this normalization would be needed.

$$percentTimeFordward = \frac{sum(probTimeFordward)}{len(probTimeFordward)} * 100 * norm$$
$$percentTimeBackward = \frac{sum(probTimeBackward)}{len(probTimeBackward)} * 100 * norm$$

  The direction vector "dir" splits in two as well, from 0 to maxPosPeu (dirForw) and the rest(dirBack). Another useful data for the neural network is the amount of negative direction there is in a fordward stretch, and the number of positive direction contains a backward propagation. Both datas are coherent to be asked. To achieve this, it must be specified that a positive direction is indicated with a 1 and a negative with 0. Then, if we apply the following lines, we obtain the percentage.

$$percentDirNegFordward = (1 - \frac{sum(dirForw)}{len(dirForw)}) * 100 * norm$$

$$percentDirPosBackward = (1 - \frac{len(dirBack) - sum(dirBack)}{len(dirBack)}) * 100 * norm$$

- **write_NN_Info(dataToFile)**

  A file called dataBase.txt is opened in writing mode to store the following data by columns: percentTimeFordward, percentTimeBackward, percenDirNegFordward, percentDirPosBackward, long_meters.

  - percentTimeFordward: Relative number of times a time interval is detected that exceeds the bounds in the positive direction.
  - percentTimeBackward: Relative number of times a time interval is detected that goes out of bounds in the negative direction.
  - percentDirNegFordward: Relative number of times a negative forward direction is detected.
  - percentDirPosBackward: Relative number of times a positive direction is detected on the lap.

6.2 Neural Network code

The first thing to do is extract the data mentioned in the previous section and copy them into a data matrix. It is necessary to indicate if the samples to be analyzed have errors and for this reason the "targets" array of 1s is created with a concatenation of 0 since the last sample does not contain errors.

Once the information matrix has been copied, the neural network is trained with an iteration of 80,000 units. This value has been increased until finding a stabilized error plot.

In the following lines, each created function will be explained in detail.

- **NeuralNetworkTraining(total_iterations)**

  This is a main function that takes care of optimizing biases and weights within a total_iterations loop. In each repetition a random index of matrixDataBase is chosen to choose a row and result (target) of this matrix.

  Once these data are obtained, the prediction begins, which the "prediction_layer2" function returns, a value between 0 and 1.

- **prediction_layer2(vectorDataBase)**

  In this section the signals of the first and second layers are generated. In the first case, the dot product of the weights and a vector of the data matrix added with the bias is made. We want to create a linear function from the input values and the generation of some weights and bias.

  $$layer1 = v_0 * w_0 + v_1 * w_1 + v_2 * w_2 + v_3 * w_3 + b$$

  For the second layer, various functions can be applied, such as the identity, the step, the piecewise linear, the Gaussian, and others. In this case, in order to simplify, the sigmoid has been applied.

  $$layer2 = \frac{1}{1 + e^{-x}}$$

- **deriv_prediction(vectorDataBase, target, prediction)**

  In this section, some derivatives must be done. The first derivative that is taken into account is that of the MSE which, being a square root, multiplies its set by 2, replacing its interior with the indicated value.

  $$MSE = \sqrt{predict - target} \longrightarrow \partial MSE = 2 * (predict - target)$$

  On the other hand, layer2 is derived, which is the sigmoid, and is replaced by the value of layer1.

  $$sigmoid = \frac{1}{1 + e^{-x}} \longrightarrow \partial sigmoid = \frac{1}{1 + e^{-x}} * (1 - \frac{1}{1 + e^{-x}})$$

Next, we have layer 1, and 2 derivatives are made, one with respect to the bias that gives 1 and the other with respect to the weights, whose result is the same database vectors.

$$\partial w_{layer1} = v_0 + v_1 + v_2 + v_3$$

$$\partial b_{layer1} = 1$$

Finally, the first two derivatives are multiplied by bias and weights to obtain the total derivatives of bias and weights.

$$\partial w = \partial MSE * \partial sigmoid * \partial w_{layer1}$$

$$\partial b = \partial MSE * \partial sigmoid * \partial b_{layer1}$$

- **update_bias_weights(derivError_bias, derivError_weights)**

  Once the errors of the bias derivatives and the weights have been calculated, the values are updated by subtracting their derivatives multiplied by alpha, learning_rate.

  $$bias = bias - (\partial b * \alpha)$$

  $$weights = weights - (\partial w * \alpha)$$

  The parameter learning rate defines de updating weights, so if the value is decreased, then the steps are smoller, and if increasing otherwise.

After executing the aforementioned, all this returns the accumulation of MSE errors and the weight and bias parameters are defined.

At this time, any Vivado sample can be passed through this prediction through this network and, once the parameters that had to be set are known, it is predicted if the data from an optical encoder contains errors.

6.2.1 Neural Network training process

In this section, screenshots and relevant code are attached to an example that generates the appropriate weights and bias. To begin with, an alpha has to be defined that, through several tests, has been decided to set at 0.1 and then the training begins with 80 million iterations.

```
learning_rate = 0.1
neural_network = NeuralNetworkTraining(80000)
```

The pertinent piece has not been added to the extraction of the data from a file, but at this point we already have the values in matrixDataBase as well as their targets. Once finished, the values to be used are displayed on the screen, as well as the learning curve.

```
########## BIAS & WEIGHTS ##########
The bias is: 3.395994781505414
The weights are: [2.17210496 1.2164467  1.73133092 1.03850637]
```

In this graph it is seen that around 80 thousand iterations the error stabilizes at 0

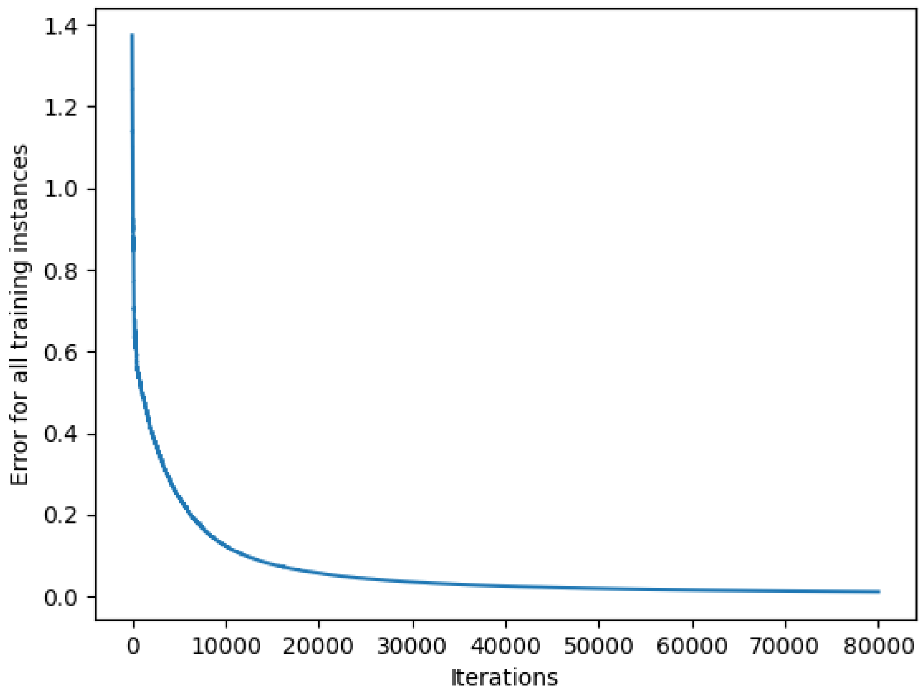


Figure 34: Cumulative error.

To verify that the calculations are correct, a test array is put. Finally, it is conditioned if the prediction value is better than 0.5, then there is no error, but if the opposite happens, unfortunately the strip encoder may be damaged.

```
############################################
#                  EXAMPLE                 #
############################################
vector_example = [0.240903804618707420 ,0.138998709892213580,  0.157833527163973030,
    40.011319561200544]
prediction = prediction_layer2(vector_example)
print(f"\n######## PREDICTION EXAMPLE ########")
if prediction < 0.5:
    phrase = "There is any error detected."
else:
    phrase = "There is an error detected!!!"
print(f"Prediction from the neural network: {prediction}\n"+phrase)
```

By viewing the terminal output, you have successfully detected an error.

```
######## PREDICTION EXAMPLE ########
Prediction from the neural network: 1.0
There is an error detected!!!
```

7 Budget

The budget consumed in this project covers the purchase of a single component since it has been worked with simulations using Vivado and analyzed using Python.

The FPGA used is the Cmod A7 module which costs \$75.

8 Conclusions

In this thesis, I addressed the problem of discrimination in the reading of an optical encoder using tools as Vivado, Matlab and Python. One of the main contributions of this work is to use neural networks analysis to accomplish detection of an error.

An encoder emulation representation has been provided, but a previous evaluation had to be done such as how the signals A and B manage to determine the orientations of the displacement and its behaviour on a loss of several physical encoders readings.

A tipping point in the project was the errors generation so the Vivado's libraries that implements random values were not considered flexible enougth due to the fact of centering in each running simulation in the same section. It is a good job that this fact did not affect on the neural network process because of its inputs. The parameters calculated in the section 6.1 do not depend on the first appearance error, but on the amount of contradicted orientation for an specific course and the number of trasitions surpass the limits.

The encoder emulations turned out to be one of the most fundamental elements in this project, even though at the beginning the task was underestimated but the process and study of the signals A and B became essential to the case.

Thanks to this thesis I had the opportunity to deepen into the Vivado tool, which was not completly familiar. On another note, Python language has been introduced for the first time, along with neural network topic, both were entirely uncharted.

From an experimental point of view, the signals A and B from the encoder emulation behave exactly like a real sample. Even so, the system works with a concrete error length of 1.5 cm, this condition does not affect to Python code study. If the error had a different prolongation, the 4 inputs inside NN (Neural Network) were changed its values increasing or decreasing each of them the same way, so at the end this would bring modifications on the bias and weigths but without affecting the neural network detection result.

The results confirm that the detection of an encoder strip failure can be detected with the help of a test of 20 IPS that stores all values from the optical encoder and is analysed further more by Python's neural network code that has been previously trained to obtain the suitable bias and weights.

9 Future Work

Many different test and experiments have been left for the future, due to lack of time and unexpected disadvantages. Future work could include topics such as implementations with differents methods of analysis, insert more than one error on a simple sample or carry out the detection within the Vivado's code.

The following lines mention some of the possible ideas that could be used in the future to extend the investigation. The purpose of this thesis is to detect possible faults in the encoder strip, and in order to do it the emulation on the encoder had to come across. These are some improvements that can apply to this thesis.

1. The generation of error could have been more flexible, so the insertion of more than a single error may have given the project.
2. The neural network used the sigmoid as a transfer function, but it would be interested to have a list of test to try with several other activation functions.
3. By having two independent programs that manage the emulation together with the storage of the data, and another one for the analysis of these, the task is doubled. However, if all the code would have been in VHDL, the result could be different. It is pending to do.

The simulation of the project have not been complicated, even though the time required on them exceeded the Vivado's capacity and in some occasions the app shuted down without warning. That is one of the reasons why the length was decreased.

The fact of not considering the MRUA situacion set aside that study. In case I had to redo the algorithm, I would raise the question of perform the emulation with 2 different clocks depending on the option: constant acceleration or constant speed simulation.

Last but not least, the change on the width of A and B pulses could differ and lead to a bad reading.That circumstance would have provided diversity of opportunities for analysis.

References

[1] What Is an Optical Encoder?
`https://www.encoder.com/article-what-is-an-optical-encoder`

[2] Optical Encoders.
`https://www.sciencedirect.com/topics/engineering/optical-encoders`, Fieldbus Systems and Their Applications 2005, 2006

[3] The Driving Force: Magnetic versus Optical Encoder Engines.
`https://www.dynapar.com/technology/optical-encoders/`

[4] Vivado Design Suite User Guide, Using Constraints.
`https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_1/ug903-vivado-using-constraints.pdf`, Xilinx April 4, 2018.

[5] Cmod A7 Reference Manual.
`https://digilent.com/reference/programmable-logic/cmod-a7/reference-manual`

[6] Bonifacio Martin del Brío y Alfredo Sanz Molina.*Redes Neuronales y Sistemas Borrosos*, tercera edición revisada y amplificada, Ra-Ma 2006.

[7] About tcl/tk
`https://www.tcl.tk/about/language.html`

[8] What are neural networks?
`https://www.ibm.com/cloud/learn/neural-networks`

Appendices

A Encoder parameters

```
clear all;
%%%%% SPECIFICATIONS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
speed_ips = 20;
long_meters = 2;
lpi_system = 150;

%%%%% CAPACITY %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
bits_memory = 24; % minimum 17 bits
bits_eu_peu = 7;
eu_capacity = 2^bits_memory; % total eu

%%%%% CONSTANTS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
k_signalsAB = 4;
k_meter_inch = 0.0254; % 1 in = 0.0254 m
k_eu_peu = 2^bits_eu_peu;
k_peu_eu = 1/k_eu_peu;
k_peu_inch = lpi_system*k_signalsAB;
k_eu_inch = round(k_peu_inch / k_peu_eu);

%%%%% LONGITUDE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

long_inches = 2 / k_meter_inch;
long_eu =   round(long_inches * k_eu_inch); % 6047244 eu, 78.74 in, 2 m
long_eu_bits = round(log2(long_eu),2); % 22.53 bits

long_max_inches = 2.^(long_eu_bits)/k_eu_inch;
long_max_eu = 2.^(long_eu_bits);
maxVal_meters = long_max_inches * k_meter_inch;
max_bitVal_meters = 2.^(23)/k_eu_inch * k_meter_inch;

%%%%% TIME + SPEED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
t_long = long_max_inches / speed_ips;
f_clk = 2^long_eu_bits / t_long;
T_clk = 1/ f_clk; % s
speed_eu_s = speed_ips * k_eu_inch ;% 1536000 eu/seconds 23 bits min
speed_peu_s = speed_eu_s /k_eu_peu;
long_eu_clk = T_clk * speed_eu_s; %1 eu

t_peu_s = 1/ (k_peu_inch*speed_ips);
t_peu_us = t_peu_s * 1e6; % 83.33 us SI CUADRA (vivado 85,42 us)!!!!!

%%%%% VIVADO SPECS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T_clk_ns = round(T_clk * 1e9,4); %ns
T_clk_ps_Vivado = round(T_clk_ns * 1e3);

%%%%% MARGIN SPECS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
t_total_capacity = 2^long_eu_bits / speed_eu_s;  % seconds to fill memory

g = 9.81;
accel_m_s = 0.12*g ;
accel_eu_s_2  = round(accel_m_s * 1/k_meter_inch*k_eu_inch);
accel_eu_T_2  = round(accel_m_s * 1/k_meter_inch*k_eu_inch)*T_clk^2;
```

B Frequency divisor

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity frequencyDiv is
    Port ( CLK, reset : in STD_LOGIC;
           CLK_10MHZ : out STD_LOGIC;
           CLK_1_5MHz : out STD_LOGIC);
end frequencyDiv;

architecture divArch of frequencyDiv is
    signal new_clk_1_5MHz : std_logic;
    signal new_clk_10MHz : std_logic;
    signal cont_1_5MHz : integer range 0 to 34 := 0;
    signal cont_10MHz : integer range 0 to 5 := 0;

begin
frequency_div: process (reset, CLK) begin

    if (reset = '1') then
        new_clk_1_5MHz <= '0';
        new_clk_10MHz <= '0';
        cont_1_5MHz <= 0;
        cont_10MHz <= 0;

    elsif rising_edge(CLK) then
        if (cont_1_5MHz = 33) then
            new_clk_1_5MHz <= not(new_clk_1_5MHz);
            cont_1_5MHz <= 0;

        else
            cont_1_5MHz<= cont_1_5MHz +1;
        end if;

        if (cont_10MHz = 5) then
            new_clk_10MHz <= not(new_clk_10MHz);
            cont_10MHz <= 0;

        else
            cont_10MHz<= cont_10MHz +1;
        end if;
    end if;
    end process;
    CLK_1_5MHz <= new_clk_1_5MHz;
    CLK_10MHZ <= new_clk_10MHz;

end divArch;
```

C Encoder Operator

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.std_logic_arith.ALL;

entity encoderOperator is
    PORT (
        clk, reset : in STD_LOGIC;
        position_eu : IN std_logic_vector(23 DOWNTO 0);
        speed_cte : IN std_logic;
        --accel_cte : IN std_logic;
        enable : out STD_LOGIC;
        accel_out, speed_out, pos_out: OUT std_logic_vector(23 DOWNTO 0)
        );
end encoderOperator;

architecture encOpArch of encoderOperator is
    signal pos, accel: std_logic_vector(23 downto 0);
    signal speed: integer range -8388608 to 8388608;
    signal option : integer range 0 to 3;
    signal back, accel_cte : std_logic := '0';
    signal cont : integer range 0 to 1325668;
    CONSTANT maxPos : INTEGER := 1536000;--5931642;

begin
    pSeq : PROCESS (clk, reset) IS
    BEGIN
        IF reset = '1' THEN
            accel <= (OTHERS => '0');
            speed <= 0;
            pos <= (OTHERS => '0');
            option <= 0;

        ELSIF clk'event AND clk = '1' THEN
            enable <= speed_cte OR accel_cte ;
            if speed_cte = '1' then
                option <= 1;
            elsif accel_cte = '1' then
                option <= 2;
                speed <= -2;
            end if;

            if option = 0 then
                accel <= (OTHERS => '0');
                speed <= 0;
                pos <= (OTHERS => '0');
            end if;

            if option = 1 then
                accel <= (OTHERS => '0');
                pos <= (OTHERS => '0');
                if position_eu = conv_std_logic_vector(maxPos,24) then
                    back <= '1';
                end if;

                if back = '1' then
                    speed <= -1;
                else
                    speed <= 1;
                end if;

                if position_eu = conv_std_logic_vector(0,24) and back = '1' then
                    option <= 0;
                    back <= '0';
                end if;
            end if;
```

```
                if option = 2 then
                    speed <= 0;
                    accel <= "000000000000000000000011";
                    pos <= (OTHERS => '0');
                end if;

            END IF;
        END PROCESS;

        accel_out <= accel;
        pos_out <= pos;
        speed_out <= conv_std_logic_vector(speed,24);


end encOpArch;
```

D Encoder position

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_arith.ALL;
USE ieee.std_logic_1164.ALL;
USE IEEE.MATH_REAL.ALL;
USE ieee.fixed_float_types.ALL;

ENTITY euPos IS
    PORT (
        clk, reset, error, e_block_minus : IN STD_LOGIC;
        accel, speed, posIn : IN STD_LOGIC_VECTOR (23 DOWNTO 0);
        posOut : OUT STD_LOGIC_VECTOR (23 DOWNTO 0));
END euPos;

ARCHITECTURE euPosArch OF euPos IS
    SIGNAL accelReg, speedReg : real := 0.0;
    SIGNAL posReg : signed (23 DOWNTO 0);
BEGIN
    pSeq : PROCESS (clk, reset) IS
    BEGIN
        IF reset = '1' THEN
            accelReg <= 0.0;
            speedReg <= 0.0;
            posReg <= (OTHERS => '0');
        ELSIF clk'event AND clk = '1' THEN
            IF error = '0' THEN --positive
                accelReg <= real(conv_integer(signed(accel))) + accelReg;
                speedReg <= real(conv_integer(signed(speed))) + speedReg + accelReg;
                posReg <= conv_signed(INTEGER(speedReg), 24) + signed(posIn);
            ELSIF error = '1' THEN --ERROR
                IF e_block_minus = '0' THEN
                    accelReg <= real(conv_integer(signed(accel))) + accelReg;
                    speedReg <= - real(conv_integer(signed(speed))) + speedReg + accelReg
                        ;
                    posReg <= conv_signed(INTEGER(speedReg), 24) + signed(posIn);
                END IF;
            END IF;
        END IF;
    END PROCESS;
    posOut <= conv_std_logic_vector(posReg, 24);

END euPosArch;
```

E Error generator and control

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE ieee.std_logic_arith.ALL;
USE ieee.math_real.ALL;

ENTITY errorGenerator IS
    PORT (
        clk, reset, enable : IN STD_LOGIC;
        pos_eu_in, direction : IN STD_LOGIC_VECTOR (23 DOWNTO 0);
        error, e_block_minus : OUT STD_LOGIC--;
        --errorPosOut : OUT STD_LOGIC_VECTOR (23 DOWNTO 0)
    );
END errorGenerator;

ARCHITECTURE errorArch OF errorGenerator IS
    CONSTANT len : INTEGER := 16;
    CONSTANT posLen : INTEGER := 24;
    CONSTANT maxPosError : INTEGER := 5822;--22992;
    SIGNAL cont : INTEGER RANGE 0 TO 320;
    SIGNAL min_val, max_val : INTEGER RANGE 0 TO 23169;
    SIGNAL pos_eu_plus : unsigned(posLen - 1 DOWNTO 0);
    SIGNAL auxError, auxDir, enable_aux : STD_LOGIC;
    SIGNAL qErrors, posError : STD_LOGIC_VECTOR(4 DOWNTO 0);
    SIGNAL valError, firstError : STD_LOGIC_VECTOR(len - 1 DOWNTO 0);
    SIGNAL errorPos : STD_LOGIC_VECTOR(posLen - 1 DOWNTO 0);
    SIGNAL sufix : STD_LOGIC_VECTOR(7 DOWNTO 0);
    TYPE error_type IS ARRAY (0 TO 25) OF STD_LOGIC_VECTOR(len - 1 DOWNTO 0);
    SIGNAL errorsReg : error_type;

BEGIN
    pSeq : PROCESS (clk, reset) IS
        VARIABLE seed1, seed2 : INTEGER := 999;
        IMPURE FUNCTION rand_slv(len : INTEGER) RETURN STD_LOGIC_VECTOR IS
            VARIABLE r : real;
            VARIABLE slv : STD_LOGIC_VECTOR(len - 1 DOWNTO 0);
        BEGIN
            FOR i IN slv'RANGE LOOP
                uniform(seed1, seed2, r);
                IF r > 0.5 THEN
                        slv(i) := '1';
                ELSE
                        slv(i) := '0';
                END IF;
            END LOOP;
            RETURN slv;
        END FUNCTION;
        IMPURE FUNCTION rand_int(min_val, max_val : INTEGER) RETURN INTEGER IS
            VARIABLE r : real;
        BEGIN
            uniform(seed1, seed2, r);
            RETURN INTEGER(
            round(r * real(max_val - min_val + 1) + real(min_val) - 0.5));
        END FUNCTION;

    BEGIN
        IF reset = '1' THEN
            auxError <= '0';
            e_block_minus <= '0';
            auxDir <= '1';
            qErrors <= CONV_STD_LOGIC_VECTOR (rand_int(10,25),5);
            valError <= CONV_STD_LOGIC_VECTOR (rand_int(0, maxPosError), len);
            errorPos <= (others => '0');
        ELSIF clk'event AND clk = '1' THEN
            enable_aux <= enable;
            IF enable_aux = '1' AND enable = '0' THEN
                posError <= (OTHERS => '0');
            errorsReg(0) <= valError;
            firstError <= valError;
           FOR i IN 1 TO conv_integer(unsigned(qErrors)) - 1 LOOP
                errorsReg(i) <= conv_std_logic_vector(rand_int(min_val, max_val), len
                    );
           END LOOP;
       END IF;

       FOR i IN 0 TO conv_integer(unsigned(qErrors)) - 1 LOOP
            IF errorsReg(i) & sufix = CONV_STD_LOGIC_VECTOR(pos_eu_plus, posLen) AND
                NOT(direction(posLen - 1)) = '1' THEN
                auxError <= '1';
                auxDir <= NOT(direction(posLen - 1));
                errorPos <= errorsReg(i) & sufix;
                errorsReg(conv_integer(unsigned(posError))) <= valError;
                valError <= errorsReg(i);
                posError <= conv_std_logic_vector(i, 5);
                errorsReg(i) <= (OTHERS => '0');
            ELSIF errorsReg(i) & sufix = pos_eu_in AND NOT(direction(posLen - 1)) =
                '0' THEN
                auxError <= '1';
                auxDir <= NOT(direction(posLen - 1));
                errorPos <= errorsReg(i) & sufix;
                errorsReg(conv_integer(unsigned(posError))) <= valError;
                valError <= errorsReg(i);
                posError <= conv_std_logic_vector(i, 5);
                errorsReg(i) <= (OTHERS => '0');

           END IF;
       END LOOP;


       IF auxError = '1' AND auxDir = '1' THEN
            IF cont < 127 THEN
                e_block_minus <= '0'; -- MINUS 128
            ELSIF cont >= 127 AND cont < 256 THEN
                e_block_minus <= '1'; -- BLOCKED 128
            ELSIF cont >= 256 AND cont < 319 THEN
                e_block_minus <= '0'; -- MINUS 64
           END IF;

            IF cont = 319 THEN
                cont <= 0;
                auxError <= '0';
           ELSE
                cont <= cont + 1;
           END IF;

       ELSIF auxError = '1' AND auxDir = '0' THEN
            IF cont < 319 THEN
                e_block_minus <= '1'; -- BLOCKED 319
           END IF;

            IF cont = 319 THEN
                cont <= 0;
                auxError <= '0';
           ELSE
                cont <= cont + 1;
           END IF;

       END IF;
   END IF;
END PROCESS;

error <= auxError;
min_val <= conv_integer(unsigned(valError));
max_val <= conv_integer(unsigned(valError)) + 177;
pos_eu_plus <= UNSIGNED(pos_eu_in) + 2;
--errorPosOut <= errorPos;
```

```
    sufix <= X"7F" WHEN direction(posLen - 1) = '0' ELSE
        X"3F" WHEN direction(posLen - 1) = '1';

END errorArch;
```

F Encoder position to AB signal

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY euConvAB IS
    PORT (
        position_eu : IN STD_LOGIC_VECTOR(23 DOWNTO 0);
        clk, reset : IN STD_LOGIC;
        empty : OUT STD_LOGIC;
        channelAB : OUT STD_LOGIC_VECTOR(1 DOWNTO 0));
END euConvAB ;

ARCHITECTURE channelsArch OF euConvAB IS
    SIGNAL pos_eu : STD_LOGIC_VECTOR(1 DOWNTO 0);

BEGIN
    pSeq : PROCESS (clk, reset) IS
    BEGIN
        IF reset = '1' THEN
            pos_eu <= "00";
        ELSE
            pos_eu <= position_eu(7) & position_eu(6);
        END IF;
    END PROCESS;

    WITH pos_eu SELECT
        channelAB <= "10" WHEN "00",
        "11" WHEN "01",
        "01" WHEN "10",
        "00" WHEN OTHERS;
    WITH position_eu SELECT
        empty <= '1' WHEN (OTHERS => '0'),
        '0' WHEN OTHERS;
END channelsArch;
```

G Transition's AB capture

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY transChangeAB IS
    PORT (
        clk_ram  : IN STD_LOGIC;
        reset  : IN STD_LOGIC;
        AB_channel  : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
        AB_out  : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
        timeStampDiff  : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
        data_valid  : OUT STD_LOGIC);
END transChangeAB;

ARCHITECTURE tranArch OF transChangeAB IS
    SIGNAL AB_aux1  :  unsigned (1 DOWNTO 0);
    SIGNAL cont  : INTEGER RANGE 0 TO 1;
    SIGNAL data_v_aux  : STD_LOGIC;
    SIGNAL intTime  : INTEGER RANGE 0 TO 2147483640 := 0;
    SIGNAL diffTS  : INTEGER RANGE 0 TO 2147483640 := 0;

BEGIN
    qSeq  : PROCESS (clk_ram , reset) IS
        VARIABLE time_v  : TIME := 0 us;
    BEGIN
        IF reset = '1' THEN
            data_v_aux <= '0';
            cont <= 0;
        ELSIF clk_ram'event AND clk_ram = '1' THEN
            time_v := now;
            IF cont = 0 THEN
                AB_aux1 <= unsigned(AB_channel);
                data_v_aux <= '0';
                cont <= cont + 1;
            ELSE
                IF AB_aux1 /= unsigned(AB_channel) THEN
                    data_v_aux <= '1';
                    diffTS <= time_v / 1 us - intTime;
                    intTime <= time_v / 1 us;
                    cont <= 0;
                END IF;
            END IF;
        END IF;
    END PROCESS;

    data_valid <= data_v_aux;
    AB_out <= CONV_STD_LOGIC_VECTOR(AB_aux1, 2);
    timeStampDiff <= conv_std_logic_vector(diffTS, 7);

END tranArch;
```

H Memory controller

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE ieee.std_logic_arith.ALL;
--use ieee.numeric_std.all;
ENTITY addressControl IS
    PORT (
        clk_ram, reset : IN STD_LOGIC;
        rst_wr : IN STD_LOGIC;
        data_AB : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
        data_timeStamp : IN STD_LOGIC_VECTOR (6 DOWNTO 0);
        write_data : IN STD_LOGIC;
        read_data : IN STD_LOGIC;
        RamCEn : OUT STD_LOGIC;
        RamOEn : OUT STD_LOGIC;
        RamWEn : OUT STD_LOGIC;
        MemDB : out STD_LOGIC_VECTOR (7 DOWNTO 0);
        MemAdr : OUT STD_LOGIC_VECTOR (18 DOWNTO 0)
        );
END addressControl;

ARCHITECTURE addrArch OF addressControl IS
    SIGNAL addrW_aux, addrR_aux : unsigned (18 DOWNTO 0) := (others => '0');
    TYPE stateMachine IS (s0_Idle, s1_Write, s2_Read);
    SIGNAL stateM : stateMachine;
    signal CE_aux, WE_aux, OE_aux, oneOption : std_logic := '1'; --activo a nivel bajo
    signal data : std_logic_vector (7 downto 0) := (others => '0');

BEGIN
    qSeq : PROCESS (clk_ram, reset) IS
    BEGIN
        IF (reset = '1' OR rst_wr = '1') THEN
            stateM <= s0_Idle;
            addrW_aux <= (others => '0');
            addrR_aux <= (others => '0');
            MemAdr <= (others => '0');
        ELSIF clk_ram'event AND clk_ram = '1' THEN

            CASE stateM IS
                WHEN s0_Idle =>

                    if write_data = '1' and oneOption = '1'  then
                        CE_aux <= '0';
                        WE_aux <= '0';
                        data <= data_timeStamp(6) & data_timeStamp(5) & data_timeStamp(4)
                            & data_timeStamp(3) & data_timeStamp(2) & data_timeStamp(1)
                           & data_AB;
                        MemAdr <= std_logic_vector(addrW_aux);
                        stateM <= s1_Write;
                    elsif read_data = '1' and oneOption = '1' then
                        CE_aux <= '0';
                        OE_aux <= '0';
                        MemAdr <= std_logic_vector(addrR_aux);
                        addrR_aux <= addrR_aux + 1;
                        stateM <= s2_Read;
                    else
                        CE_aux <= '1';
                        OE_aux <= '1';
                        WE_aux <= '1';
                        stateM <= s0_Idle;
                    END IF;
                WHEN s1_Write =>
                    addrW_aux <= addrW_aux + 1;
                    MemAdr <= addrW_aux + 1;
                    IF write_data = '1' and oneOption = '1' THEN
                        CE_aux <= '0';
                        WE_aux <= '0';
                        data <= data_timeStamp(6) & data_timeStamp(5) & data_timeStamp(4)
                            & data_timeStamp(3) & data_timeStamp(2) & data_timeStamp(1)
                           & data_AB;
                        stateM <= s1_Write;
                    elsif read_data = '1' and oneOption = '1' then
                        CE_aux <= '0';
                        OE_aux <= '0';
                        MemAdr <= std_logic_vector(addrR_aux);
                        addrR_aux <= addrR_aux + 1;
                        stateM <= s2_Read;
                    else
                        CE_aux <= '1';
                        OE_aux <= '1';
                        WE_aux <= '1';
                        stateM <= s0_Idle;
                    END IF;
                WHEN s2_Read =>
                    IF write_data = '1' and oneOption = '1' THEN
                        CE_aux <= '0';
                        WE_aux <= '0';
                        data <= data_timeStamp(6) & data_timeStamp(5) & data_timeStamp(4)
                            & data_timeStamp(3) & data_timeStamp(2) & data_timeStamp(1)
                           & data_AB;
                        stateM <= s1_Write;
                    elsif read_data = '1' and oneOption = '1' then
                        CE_aux <= '0';
                        OE_aux <= '0';
                        MemAdr <= std_logic_vector(addrR_aux);
                        addrR_aux <= addrR_aux + 1;
                        stateM <= s2_Read;
                    else
                        CE_aux <= '1';
                        OE_aux <= '1';
                        WE_aux <= '1';
                        stateM <= s0_Idle;
                    END IF;
            END CASE;
        END IF;
    END PROCESS;

    MemDB <= data;
    RamCEn <= CE_aux;
    RamOEn <= OE_aux;
    RamWEn <= WE_aux;
    oneOption <= write_data XOR read_data ;
END addrArch;
```

I RAM Emulation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.Numeric_Std.all;

entity memoryRAM is
    Port ( CLK_10MHz : in STD_LOGIC;
         RamCEn : in STD_LOGIC;
         RamOEn : in STD_LOGIC;
         RamWEn : in STD_LOGIC;
         MemDB_in : in STD_LOGIC_VECTOR (7 downto 0);
         MemDB_out : out STD_LOGIC_VECTOR (7 downto 0);
         MemAdr : in STD_LOGIC_VECTOR (18 downto 0)
        );
end memoryRAM;

architecture ramArch of memoryRAM is
    type ram_type is array (0 to 5242287) of std_logic_vector(7 downto 0);
    signal ram : ram_type;
    signal addr : std_logic_vector(18 downto 0) := (others => '0');

begin

    qSeq: PROCESS (CLK_10MHz) is
    begin
        if CLK_10MHz'event AND CLK_10MHz='1' then
            if RamCEn = '0' then
                if RamWEn = '0' then
                    ram(to_integer(unsigned(MemAdr))) <= MemDB_in;
                end if;

                if RamOEn = '0' then
                    MemDB_out <= ram(to_integer(unsigned(MemAdr)));
                    addr <= MemAdr;
                end if;

            end if;

        end if;
    end process;


end ramArch;
```

J Pulse train to read RAM

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.numeric_std.ALL;

entity readRAM is
    Port ( clk_ram, reset : in STD_LOGIC;
           button_read : in STD_LOGIC;
           empty : in STD_LOGIC;
           read_data : out STD_LOGIC);
end readRAM;

architecture readRamArch of readRAM is
    TYPE stateMachine IS (s0_Idle, s1_ContPlus, s2_Read);
    SIGNAL stateM : stateMachine;
    signal cont : unsigned (7 downto 0);
begin
    pSeq : PROCESS (clk_ram, reset) IS

    begin
        IF reset = '1' then
            cont <= (others => '0');
            read_data <= '0';
        ELSIF clk_ram'event AND clk_ram = '1' then
            CASE stateM IS
                WHEN s0_Idle =>
                    read_data <= '0';
                    cont <= (others => '0');
                    if empty = '1' then
                        stateM <= s0_Idle ;
                    elsif button_read = '1' then
                        stateM <= s1_ContPlus ;
                    end if;
                WHEN s1_ContPlus =>
                    cont <= cont + 1;
                    read_data <= '0';
                    if empty = '1' then
                        stateM <= s0_Idle ;
                    elsif cont < 200  then
                        stateM <= s1_ContPlus ;
                    elsif cont = 200 then
                        read_data <= '1';
                        stateM <= s2_Read ;
                    end if;
                WHEN s2_Read =>
                    if empty = '1' then
                        stateM <= s0_Idle ;
                    else
                        read_data <= '0';
                        cont <= (others => '0');
                        stateM <= s1_ContPlus ;
                    end if;
            END CASE;
        end if;
    END PROCESS;

end readRamArch;
```

K Store data to text file

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE STD.textio.ALL;
USE ieee.std_logic_textio.ALL;
use ieee.numeric_std.all;

entity dataToFile is
    Port (
              clk_ram  : IN STD_LOGIC;
              reset  : IN STD_LOGIC;
             OE  :  in STD_LOGIC;
            data  :  in STD_LOGIC_VECTOR (7 downto 0);
            address  :  in STD_LOGIC_VECTOR (18 downto 0)
            --LED  :  out STD_LOGIC
            );
end dataToFile;

architecture fileArch of dataToFile is
    FILE file_RESULTS  :  text OPEN write_mode IS "C:\Users\brend\OneDrive\Escritorio\TFG\
        outputWriteAB.txt";
    signal data_aux  :  std_logic_vector(7 downto 0);
    signal timeStamp  :  std_logic_vector(5 downto 0);
    signal data_AB  :  std_logic_vector(1 downto 0);

begin


    sync_stuff  :  PROCESS (clk_ram)
        VARIABLE out_line  :  line;
        VARIABLE v_TIME  :  TIME := 0 us;

    BEGIN
        if reset = '1' then
            data_aux <= (others => '0');
        ELSIF rising_edge(clk_ram) THEN
            IF data_AB(0) = '0' OR data_AB(0) = '1'  then
                IF OE = '0' THEN
                    data_aux <= data;
                    write(out_line, to_integer(unsigned(address)));
                    write(out_line, timeStamp, right, 15);
                    write(out_line, data_AB, right, 15);
                    writeline(file_RESULTS, out_line);
                END IF;
            END IF;


        END IF;
    END PROCESS;
    data_AB <= data(1) & data(0);
    timeStamp <= data(7) & data(6) & data(5) & data(4) & data(3) & data(2) ;


end fileArch;
```

L Python data process

```
import numpy as np
import csv


addressList = []
timeStampList = []
abList = []

#############################################
#          ORIENTATION CALCULATIONS          #
#############################################
def calc_direction(posNow, posAft):
    condition = (posNow == '00' and posAft == '10') or (posNow == '10' and posAft == '11'
        ) or (
        posNow == '11' and posAft == '01') or (posNow == '01' and posAft == '00')
    return condition


def obtain_data(i):
    # Open file and store data to newArrayData
    path = "C:/Users/brend/OneDrive/Escritorio/TFG/Python/outputWriteAB" + str(i)+".txt"
    f = open(path, "r")
    data = f.read()
    listData = data.split()
    #newArrayData = []
    newArrayData = np.array_split(listData, len(listData)/3)

    # Split data in address, time stamp and AB info
    for array in newArrayData:
        address = list(array)[0]
        timeStampBin = list(array)[1] + '1'
        timeStampInt = int(timeStampBin, 2)
        ab = list(array)[2]
        addressList.append(address)
        timeStampList.append(timeStampInt)
        abList.append(ab)
    return addressList, timeStampList, abList

def peu_performance():
    peu = []
    time = []
    peu.append(0)
    peuCont = 0
    time.append(0)
    timeCont = 0
    us = 1000000
    limMaxTime = 46
    limMinTime = 30
    probTime = []
    dir = []
    dir.append(1)
    for i in range(0, len(abList)-1):
        posNow = abList[i]
        posAft = abList[i+1]
        direction = calc_direction(posNow, posAft)

        if direction:
            peuCont = peuCont + 1
            dir.append(1)
        else:
            peuCont = peuCont - 1
            dir.append(0)
        peu.append(peuCont)
        timeAft = timeStampList[i+1]
        timeCont = timeCont + timeAft
        time.append(timeCont/us)
        if timeAft > limMaxTime or timeAft < limMinTime:
            probTime.append(1)
        else:
            probTime.append(0)
    return peu, probTime, time, dir


def data_NN(peu, probTime, dir):
    maxPeu = max(peu)
    maxPosPeu = peu.index(max(peu))
    k_meter_inch = 0.0254;  # 1 in = 0.0254 m
    k_signalsAB = 4
    lpi_system = 150
    k_peu_inch = lpi_system*k_signalsAB
    long_meters = maxPeu / k_peu_inch * k_meter_inch / 2
    norm = long_meters / 0.508

    dirForw = dir[0:maxPosPeu]
    dirBack = dir[maxPosPeu+1:len(dir)]
    probTimeFordward = probTime[0:maxPosPeu]
    probTimeBackward = probTime[maxPosPeu+1:len(dir)]

    percentTimeFordward = sum(probTimeFordward) / len(probTimeFordward) * 100 * norm
    percentTimeBackward = sum(probTimeBackward) / len(probTimeBackward) * 100 * norm


    percentDirNegFordward = (1 - sum(dirForw) / len(dirForw)) * 100 * norm
    percentDirPosBackward = (1 - (len(dirBack) - sum(dirBack)) / len(dirBack)) * 100 *
        norm

    return [percentTimeFordward, percentTimeBackward, percentDirNegFordward,
        percentDirPosBackward, long_meters]


def write_NN_Info(dataToFile):
    f = open("C:/Users/brend/OneDrive/Escritorio/TFG/Python/dataBase.txt", "a")
    writer = csv.writer(f, delimiter='\t')
    writer.writerow([dataToFile[0], dataToFile[1], dataToFile[2], dataToFile[3]])
    print("DONE! " + str([dataToFile[0], dataToFile[1], dataToFile[2], dataToFile[3],
        dataToFile[4]]))



i = 0
while(i < 6):
    addressList, timeStampList, abList = obtain_data(i)
    peu, probTime, time, dir = peu_performance()
    dataToFile = data_NN(peu, probTime, dir)
    write_NN_Info(dataToFile)
    i = i + 1
```

M Python Neural Network Analysis

```
from re import I
import matplotlib.pyplot as plt
import numpy as np

###########################################
#          PARAMETERS DEFINITION          #
###########################################
weights = np.array([np.random.randn(), np.random.randn(), np.random.randn(), np.random.
    randn()])
bias = np.random.randn()
layer1 = 0
layer2 = 0


###########################################
#             NEURAL NETWORK              #
###########################################
def sigmoid(x):
   return 1 / (1 + np.exp(-x))

def deriv_sigmoid(x):
    return sigmoid(x) * (1 - sigmoid(x))

def update_bias_weights(derivError_bias, derivError_weights):
    global bias, weights
    bias = bias - (derivError_bias * learning_rate)
    weights = weights - (derivError_weights * learning_rate)

def prediction_layer2(vectorDataBase):
    global layer1, layer2
    layer1 = np.dot(vectorDataBase, weights) + bias
    layer2 = sigmoid(layer1)
    return layer2

def deriv_prediction(vectorDataBase, target, prediction):
    derivError_Prediction = 2 * (prediction - target)
    deriv_Layer1 = deriv_sigmoid(layer1)
    deriv_layer1_bias = 1
    deriv_layer1_weights = vectorDataBase

    derivError_bias = derivError_Prediction * deriv_Layer1 * deriv_layer1_bias
    derivError_weights = derivError_Prediction * deriv_Layer1 * deriv_layer1_weights
    return derivError_bias, derivError_weights

def NeuralNetworkTraining(total_iterations):
    global matrixDataBase, targets
    cumulErrors = []
    for iter in range(total_iterations):
        matrix_rand_index = np.random.randint(len(matrixDataBase))

        vectorDataBase = matrixDataBase[matrix_rand_index]
        target = targets[matrix_rand_index]
        prediction = prediction_layer2(vectorDataBase)
        derivError_bias, derivError_weights = deriv_prediction(vectorDataBase, target,
            prediction)
        update_bias_weights(derivError_bias, derivError_weights)

        cumulative_error = 0
        for index_data in range(len(matrixDataBase)):
            data_instant = matrixDataBase[index_data]
            target = targets[index_data]

            predict = prediction_layer2(data_instant)
            mse = np.square(predict - target)

            cumulative_error = cumulative_error + mse
        cumulErrors.append(cumulErrors)
    return cumulErrors


############################################
#          DATA PROCESS FROM FILE          #
############################################
path = "C:/Users/brend/OneDrive/Escritorio/TFG/Python/dataBase.txt"
f = open(path, "r")
dataBase = f.read()
listDataBase = dataBase.split()
newArrayDataBase = np.array_split(listDataBase, len(listDataBase)/4)
matrixDataBase = []
row = 0
for array in newArrayDataBase:
    x = np.array([list(array)[0], list(array)[1],
                  list(array)[2], list(array)[3]])
    x = x.astype(np.float)
    matrixDataBase.append(x)
    row = row + 1

targets = np.append(np.ones(len(matrixDataBase)-1),0)


############################################
#          NN PROCESS TRAINING             #
############################################
learning_rate = 0.1
neural_network = NeuralNetworkTraining(80000)
print(f"\n##########␣BIAS␣&␣WEIGHTS␣##########")
print(f"The␣bias␣is:␣{bias}")
print(f"The␣weights␣are:␣{weights}")


############################################
#                  EXAMPLE                 #
############################################
vector_example = [0.2300380461870742    ,0.35899870989221358,    0.7083352716397303,
    48.011319561200544]
prediction = prediction_layer2(vector_example)
print(f"\n########␣PREDICTION␣EXAMPLE␣########")
if prediction < 0.5:
    phrase = "There␣is␣any␣error␣detected."
else:
    phrase = "There␣is␣an␣error␣detected!!!"
print(f"Prediction␣from␣the␣neural␣network:␣{prediction}\n"+phrase)

plt.plot(neural_network)
plt.xlabel("Iterations")
plt.ylabel("Error␣for␣all␣training␣instances")
plt.savefig("Cumulative␣count␣errors.png")
plt.show()
```