2A FISE Project Report

Development of a DRO and DIVIDER System

Selim FARCI, Abdelkader CHAMBI

June-August 2024

ENSEA
Ecole Nationale
Supérieure
de l'Electronique
et de ses Applications

Interns: Selim Farci, AbdelKader CHAMBI
Company Supervisor : Mr. Michaël EYRAUD
University Supervisor : Mr. CHIPI

Table of Contents

1 Introduction

1.1 Company Presentation

Mécaniquement vôtre, based in Switzerland, is a company managed by EYRAUD Michael. The company specialises in the overhaul of old industrial watchmaking systems, systems that are now coveted by various watch enthusiasts, such as collectors and watchmakers. EYRAUD Michael is a mechanic, which is why the company is first and foremost a workshop. During this project, we worked alongside other trainees, who themselves specialised in mechanics. The company does not have a CSR management system. It does, however, recycle as much as possible of its parcel boxes.

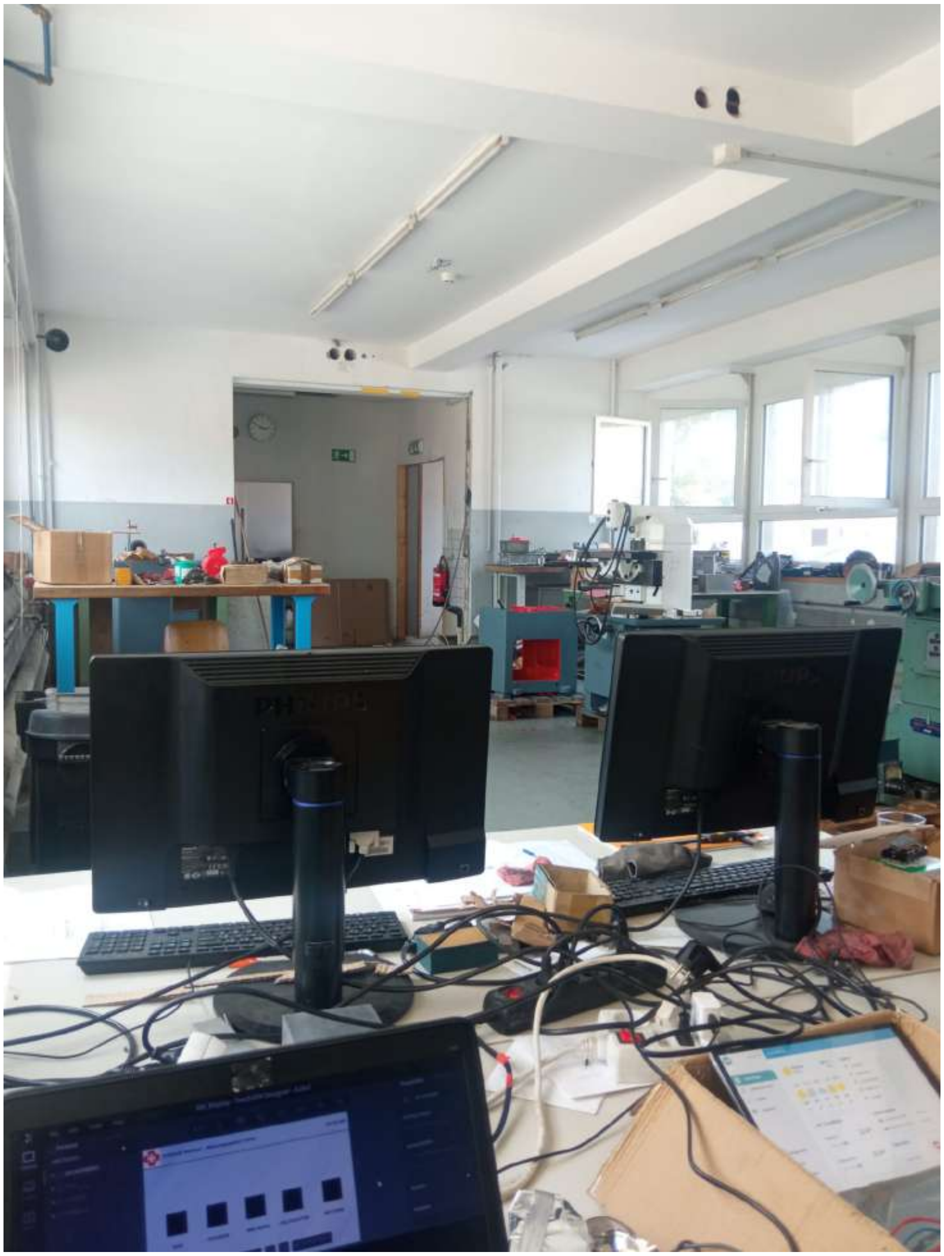


Figure 2: Increment sensors overview

The project assigned to us was part of this modernization effort, involving the development of a DRO system and a DIVIDER system. Although the entrepreneur was not an expert in electronics, he played a key role in financing the project and defining the requirements, which required a deep understanding of his expectations to translate these ideas into technical solutions.

1.2 Context and objective of the project

The objective of the project is to develop a "DRO system and a "DIVIDER system".
The DRO system have to display and modificate values given by sensors and linear scales from a minimum 7" touch screen controller.
The DIVIDER system is a complementary system of the DRO system, he have to control 2 stepper motors to perform various functions.

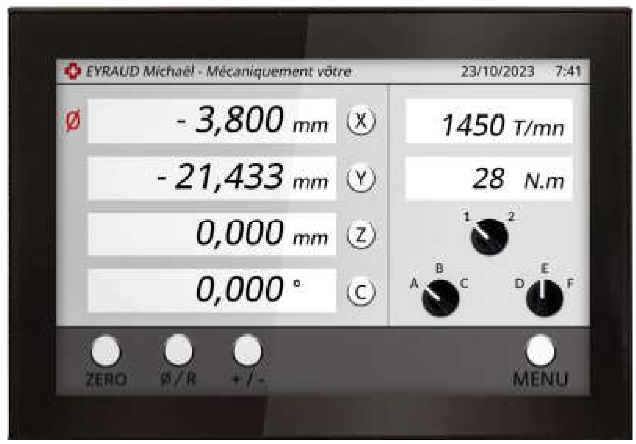
This DRO system aims to facilitate the use of industrial watchmaking systems while preserving old industrial machines. These machines, which are highly sought after by watchmaking artisans, retain their vintage appearance as they are valuable collector's items. Keep in mind that the price and industrial credibility are important factors for the development of this project.

2 DRO and DIVIDER Systems

2.1 The DRO system

The DRO system will be the control center and user interface panel with screen, control and switches, it have to run independently.
The system must be ready to control all other systems so the functions of all others systems must be programmed on the microprocessor.
The system must display with a graphical interface:
- Number of axes (values from RL2IC sensors)
– Function buttons - Spindle motor information.
Functions:
- ZERO: Set value to 0.
- Ø / R: Double value (Ø = Diameter) or reset to original (R = radius).
- + / -: Add or subtract a value (Show keyboard to give a value, apply, select axis).
- Spindle speed:
- Spindle torque:
- menu back

(a) Example of the graphical user interface of the MENU provided by the specifications document



(b) Example of the graphical user interface of the DRO provided by the specifications document

2.2 The DIVIDER system

The DIVIDER system will be a complementary system to the DRO system, Enabling control of 2 stepper motors to perform various functions.

Functioning : From the angular zero bridge and the current linear zero point. 1: Stepper motor No. 1 turns to drive a precision screw/nut system to perform a linear advance cycle including advance of a given value then return to the initial position. 2: Stepper motor No. 2 turns to drive a screw and endless wheel system, to perform angular indexing defined by a given number of divisions, applied to a given angular sector, from its current position to the next position. The angular position is controlled by an sensor (same as the DRO system) and a circular scale and must be adjusted corrected as necessary before starting a new cycle of linear motion motor 1.

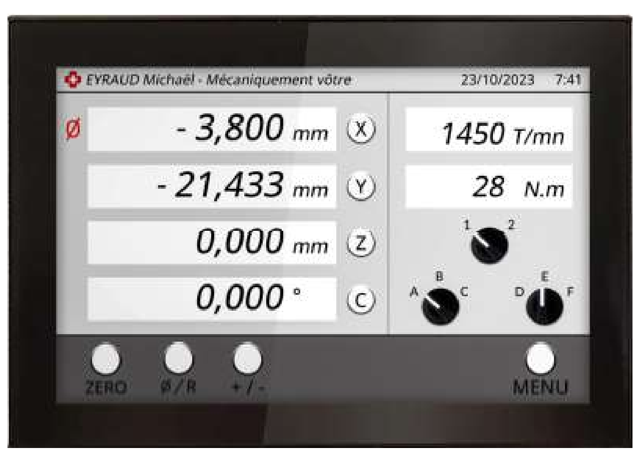


Figure 4: Example of the graphical user interface of the DIVIDER provided by the specifications document

3 SYSML and GANTT Diagrams

3.1 GANTT Diagram

3.2 Usecase diagram

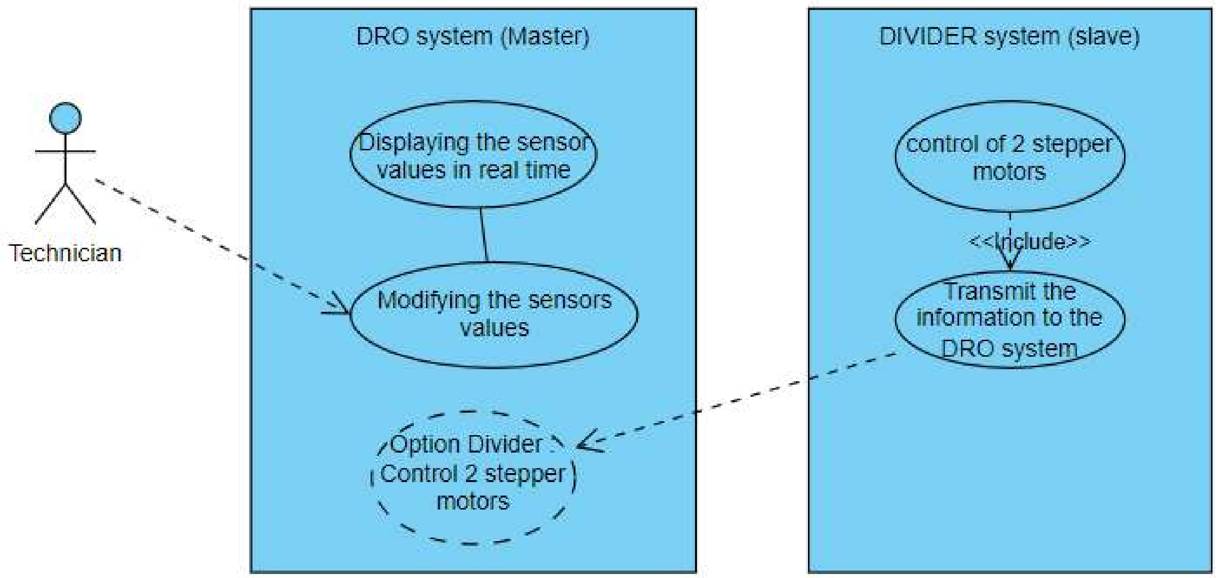


Figure 5: Usecase diagram

3.3 Requirements diagram

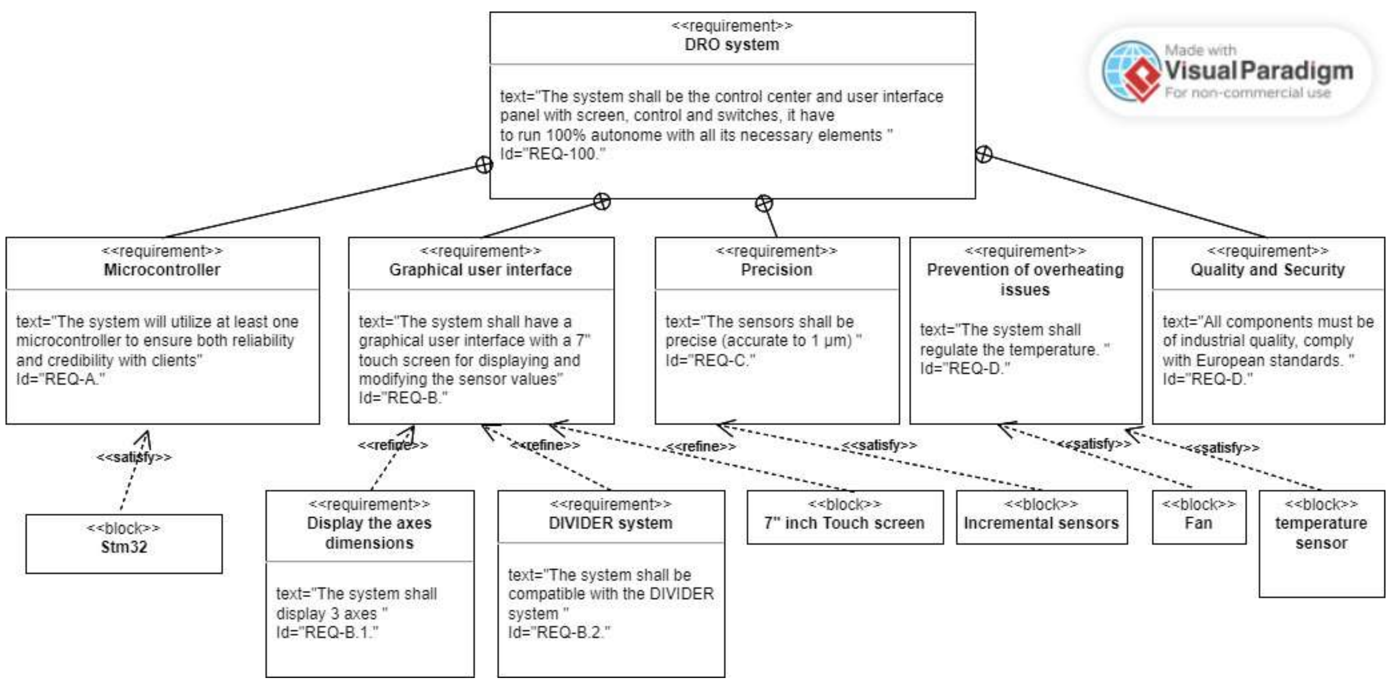


Figure 6: Requirements diagram of the DRO system

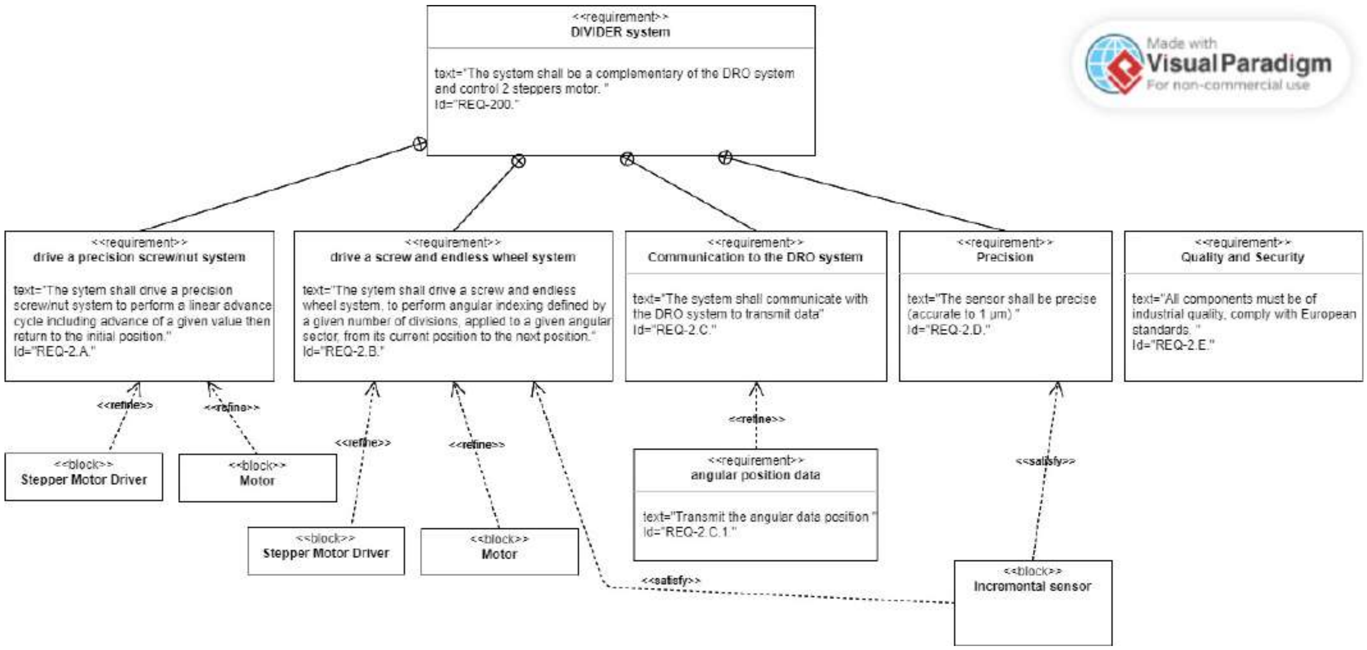


Figure 7: Requirements diagram of the DIVIDER system

3.4 Gantt diagram

4 Component selection

4.1 Choice of the micro-controller

It is important to choose a micro-controller with display capabilities in mind, because in this project the graphical user interface (GUI) is an important part of the project. We compare different technologies and screen display technologies to choose the best solution for our project while respecting the requirements. note : We didn't compared the Arduino series because its industrial credibility wasn't enough for the requirements.

Critères	**PLC**	**STMicroelectronics**	**Rasberry Pi**
Microcontroller	Siemens S7, Allen Bradley, etc.	STM32 Series	Raspberry Pi series
Microcontroller's software	TIA Portal, RSLogix	STM32CubeIDE	Raspberry Pi OS (Raspbian)
Graphical user interface (GUI) software	WinCC, FactoryTalk, GT Designer	TouchGFX	PyQt, Tkinter, Electron
programming language	Ladder Logic, FBD, ST	C, C++	Python, C++, JavaScrip
suitable environment	Industriel, Automatisation lourde	Industriel, Automatisation	Éducation, Prototypage, Domotique
type of screen	HMI Panels, Touch Screens	TFT LCD, Capacitive Touch Screens	HDMI, Capacitive Touch Screens
Price	High	Moderate	Moderate

Table 1: comparison between the different existing technologies with enough industrial credibility

We didn't select the PLC technology because its price is too high and the system shall be easily integrable with other systems. Using a programmable logic controller (PLC) is not the right solution for our project. The Raspberry Pi has an advantage in graphical user interface because it is created for this aspect, but we don't have the same adaptability in the hardware part as an STM32. Furthermore, we don't even need an OS for our GUI.

The STM32 technology seems to be the best solution for our project because this technology respect all our requirements :
- the REQ-A and the REQ-D are respected because the STM32 series is famous for its reliability and durability in industrial environments. STMicroelectronics offers a wide range of microcontrollers suitable for various industrial applications, from simple process control to complex embedded systems.
- We can connect screens for programming GUIs respecting the REQ-B
-We can more easily transmit information from the DIVIDER system to the DRO system.

4.2 Display selection

Solutions	**Screen**	**Product Details and comment**	**Price**
Sol 1	Waveshare 7" Capacitive Touch	1024x600, HDMI/USB, Cortex-M7, Moderate quality	70 dollars
Sol 2	Riverdi 7" (avec STM32H757 included)	High quality, TouchGFX is suitable because it is software from STMicroelectronics, and the STM32 is integrated into the screen	200 dollars
Sol 3	Generic 7" TFT (ILI9488)	800x480, SPI, Cortex-M4, Low quality	35 dollars

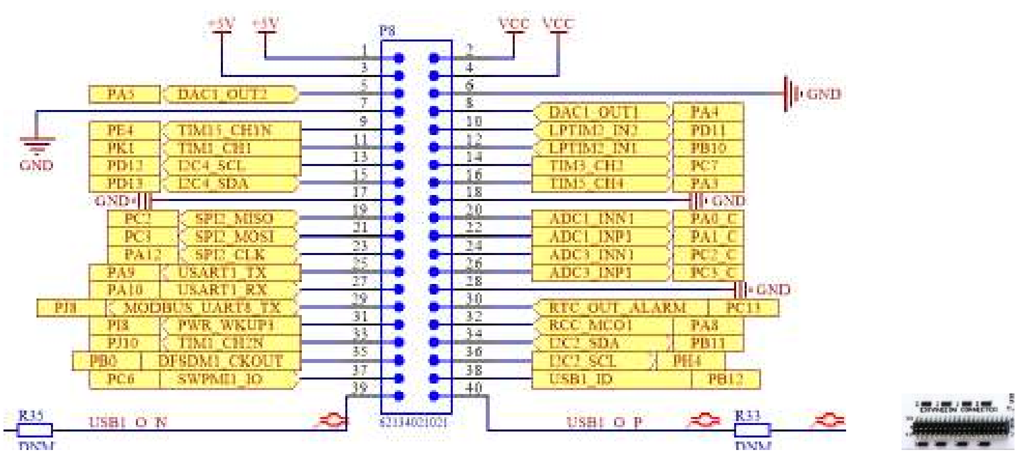
Table 2: comparison between the different existing screens

The Riverdi solution seems to be the best for our project because its software environment is more adaptable to our microcontroller since it is a component from STMicroelectronics. By using all ST components, we will simplify the programming because the components will communicate more easily with each other. With this solution, the quality of the screen is guaranteed and the STM32 is already included in the screen, making the wiring easier.

Figure 8: Riverdi screen (ref : RVT70HSSNWC00-B)

The Riverdi screen has 40 pins that can be used. These pins come from the STM32H7 (the STM included in the Riverdi). These pins have the following interfaces available:
- 2 x I2C
- 1 x UART
- 1 x USART
- 1 x SPI
- 1 x USB
- 7 x PWMs
- 2 x DACs (Digital-to-analog)
- 2 x ADCs (Analog-to-digital)

Each of the GPIO pins can be configured by software as output (push-pull or open-drain, with or without pull-up or pull-down), as input (floating, with or without pull-up or pull-down) or as peripheral alternate function. Most of the GPIO pins are shared with digital or analog alternate functions.



(a) Expansion connector pins schematic

(b) Expansion connector

Figure 9: Expansions connector

The power supply for this screen is between 8 and 48V.

4.3 Increment Sensor Selection

We need 4 sensors very precise, (accurate to 1 um). The sensor shall be indicate 3 positions (X,Y,Z) for the DRO system and an angular position for the DIVIDER system. We need so 4 sensors for respect this requirements. We selected Increment sensors because of its precision.

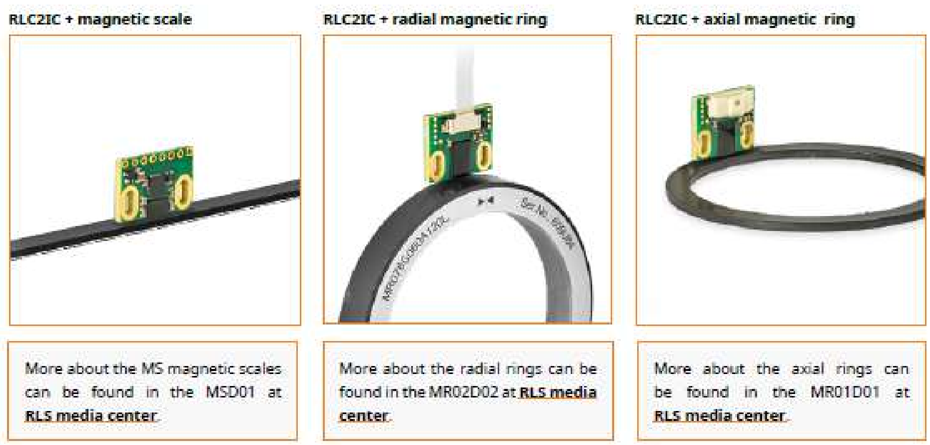
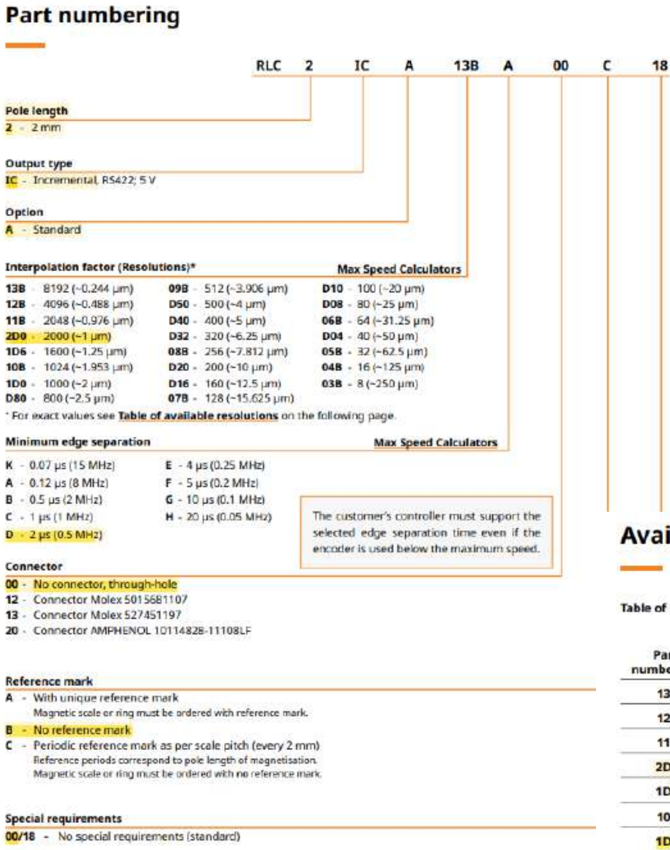


Figure 10: Increment sensors overview

We chose the reference below : RLC2ICA2D0DD0B00



Part numbering

RLC 2 IC A 13B A 00 C 18

Pole length
2 - 2 mm

Output type
IC - Incremental, RS422; 5 V

Option
A - Standard

Interpolation factor (Resolutions)* — **Max Speed Calculators**

13B - 8192 (~0.244 µm)	**09B** - 512 (~3.906 µm)	**D10** - 100 (~20 µm)
12B - 4096 (~0.488 µm)	**D50** - 500 (~4 µm)	**D08** - 80 (~25 µm)
11B - 2048 (~0.976 µm)	**D40** - 400 (~5 µm)	**06B** - 64 (~31.25 µm)
2D0 - 2000 (~1 µm)	**D32** - 320 (~6.25 µm)	**D04** - 40 (~50 µm)
1D6 - 1600 (~1.25 µm)	**08B** - 256 (~7.812 µm)	**05B** - 32 (~62.5 µm)
10B - 1024 (~1.953 µm)	**D20** - 200 (~10 µm)	**04B** - 16 (~125 µm)
1D0 - 1000 (~2 µm)	**D16** - 160 (~12.5 µm)	**03B** - 8 (~250 µm)
D80 - 800 (~2.5 µm)	**07B** - 128 (~15.625 µm)	

* For exact values see **Table of available resolutions** on the following page.

Minimum edge separation — **Max Speed Calculators**

K - 0.07 µs (15 MHz)	E - 4 µs (0.25 MHz)
A - 0.12 µs (8 MHz)	F - 5 µs (0.2 MHz)
B - 0.5 µs (2 MHz)	G - 10 µs (0.1 MHz)
C - 1 µs (1 MHz)	H - 20 µs (0.05 MHz)
D - 2 µs (0.5 MHz)	

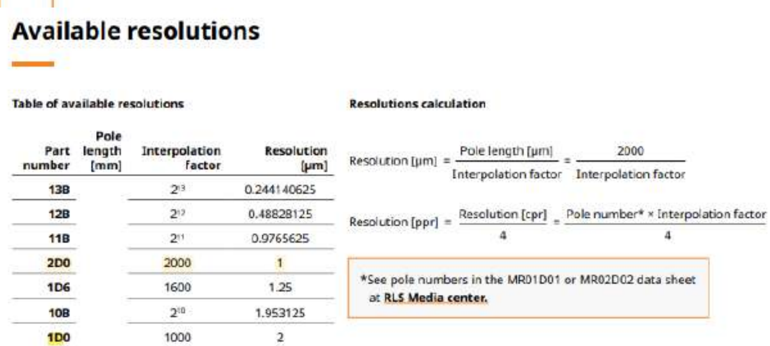
The customer's controller must support the selected edge separation time even if the encoder is used below the maximum speed.

Connector
00 - No connector, through-hole
12 - Connector Molex 5015681107
13 - Connector Molex 527451197
20 - Connector AMPHENOL 10114828-11108LF

Reference mark
A - With unique reference mark
Magnetic scale or ring must be ordered with reference mark.
B - No reference mark
C - Periodic reference mark as per scale pitch (every 2 mm)
Reference periods correspond to pole length of magnetisation.
Magnetic scale or ring must be ordered with no reference mark.

Special requirements
00/18 - No special requirements (standard)

(a) Increment sensors part numbering



Available resolutions

Table of available resolutions

Part number	Pole length [mm]	Interpolation factor	Resolution [µm]
13B		2^{13}	0.244140625
12B		2^{12}	0.48828125
11B		2^{11}	0.9765625
2D0		2000	1
1D6		1600	1.25
10B		2^{10}	1.953125
1D0		1000	2

Resolutions calculation

$$\text{Resolution [µm]} = \frac{\text{Pole length [µm]}}{\text{Interpolation factor}} = \frac{2000}{\text{Interpolation factor}}$$

$$\text{Resolution [ppr]} = \frac{\text{Resolution [cpr]}}{4} = \frac{\text{Pole number* × Interpolation factor}}{4}$$

*See pole numbers in the MR01D01 or MR02D02 data sheet at **RLS Media center.**

(b) Increment sensors pins connections

Figure 11: Increment sensors available resolution

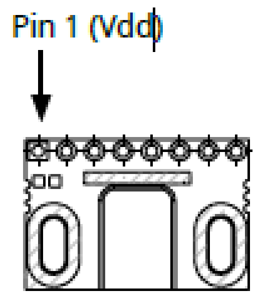
According to the Datasheet, the resolution is : 2um.

We have 8 outputs from an increment sensor : +5V, GND and 3 square-wave signals : A, B, Z and their inverted signals A–, B–, Z–.

Electrical connections

Function	Signal
Power	Vdd
	GND
Incremental signals	A+
	A-
	B+
	B-
Reference signals	Z+
	Z-

(a) Increment sensors electrical connections



(b) Increment sensors pins connections

Figure 12: Increment sensors outputs

Signal A+ : Each signal has its complementary signal to reduce Electromagnetic Interference (EMI) and improve signal robustness, especially over long cables.

Signal A: The A signal is a square wave that changes state (high/low) at regular intervals when the encoder rotates.

Signal B: The B signal is a square wave that is phase-shifted by 90 degrees relative to the A+ signal. The 90-degree phase shift between A+ and B+ allows for the determination of the direction of rotation. If A+ leads B+, the rotation is in one direction, and if B+ leads A+, the rotation is in the opposite direction.

Signal Z: The Z+ signal is a square wave that generates a single pulse per complete rotation of the encoder. The index signal (Z+) is used to provide an absolute reference for the position, allowing for re calibration or synchronization of the position to a known point.

Output type

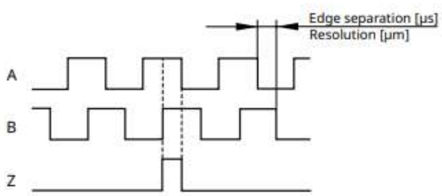
Incremental, RS422

RLC2IC

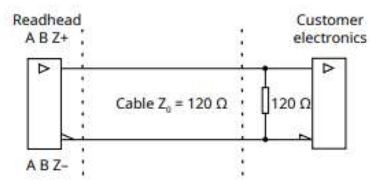
Specifications	
Output signals	3 square-wave signals A, B, Z and their inverted signals A–, B–, Z–
Reference signal	1 or more square-wave pulse Z and its complementary pulse Z–
Signal level	Differential line driver according to EIA standard RS422
Permissible load	$Z_0 \geq 120\ \Omega$ between associated outputs

Timing diagram

Complementary signals not shown



Recommended signal termination



Positive direction

Digital output signals – A leads B

For more information see the MSD01, MR02D02 or MR01D01 data sheets at **RLS Media center**.

Figure 13: Increment sensors overview

The output type is the RS422 interface. We will process the signal A, B and Z on a microcontroller so we need to convert the RS422 interface to a micrcontroller communication (as UART, I2C, SPI, TTL, ect..). For that, we can use a RS422 Converter or an Incremental encoder.

4.4 Transeiver RS485/RS422 communication

The incremental sensor output signals are RS422 type signals.

The RS-422 protocol is designed for unidirectional serial communications, typically point-to-point or point-to-multipoint. An RS-422 transceiver can be either a transmitter or a receiver, but not both on the same channel. Typically, an RS-422 transmitter sends a differential signal on one pair of wires (A+ and A-), and multiple receivers can receive this signal by reading the voltage difference between A+ and A-. This configuration allows reliable transmission over long distances, up to 1200 metres, while being resistant to noise and interference. However, communication is strictly unidirectional between the transmitter and receivers.

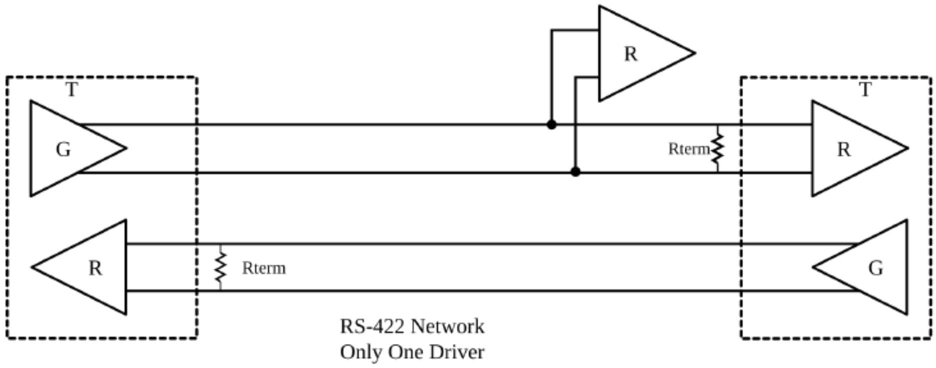


Figure 14: Rs422 Drivers/receivers

The RS-485 protocol is a differential serial communication standard used in multidrop networks, enabling several devices to share the same communication bus. It can operate in two modes: half-duplex and full-duplex.
In half-duplex mode, communication is bidirectional but not simultaneous. This means that a device can either send or receive data at a given time, but not both at the same time. This mode uses a single pair of wires for transmission and reception, simplifying cabling and reducing costs, making it ideal for systems where simultaneity is not critical.
In full-duplex mode, communication is bidirectional and simultaneous. Each device can send and receive data at the same time, requiring two pairs of wires (one for transmission and one for reception). This mode is used in applications where fast, efficient communications are essential.

For an RS-485 transceiver to be compatible with an RS-422 protocol, it must be configured in half-duplex mode. This ensures that communication remains unidirectional and that the system behaves as expected by RS-422.

In our case, the direction of data transmission is from the sensor to the microcontroller as shown below:

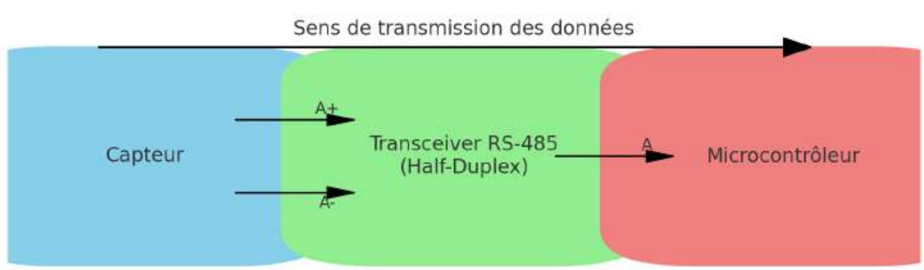


Figure 15: RS422 - Transmission data

We choose the RS485 transceiver : RS485 7 click from the manufacturer MIKROE.

Figure 16: Rs485 7 click pins

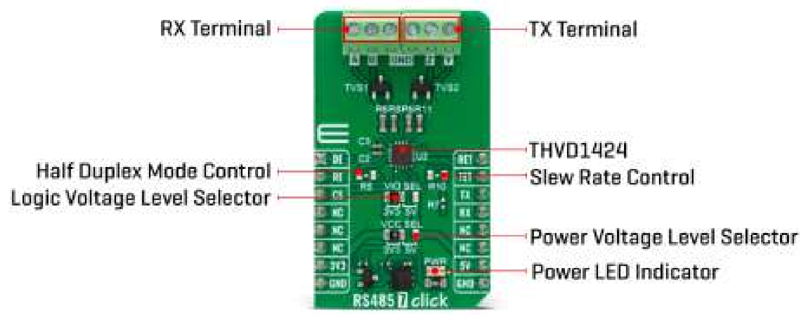


Figure 17: Rs485 7 click pins descriptions

Notes	Pin	mikroBUS				Pin	Notes
Driver Enable	**DE**	1	AN	PWM	16	**RET**	RX Termination Control
Receiver Enable	**RE**	2	RST	INT	15	**TET**	TX Termination Control
	NC	3	CS	RX	14	**TX**	UART TX
	NC	4	SCK	TX	13	**RX**	UART RX
	NC	5	MISO	SCL	12	NC	
	NC	6	MOSI	SDA	11	NC	
Power Supply	**3.3V**	7	3.3V	5V	10	**5V**	Power Supply
Ground	**GND**	8	GND	GND	9	**GND**	Ground

Figure 18: Rs485 7 click pins table

To activate the RS485 transeiver in half-duplex mode, we will use the RX terminal to receive the data from the sensor and set the driver enable (DE) to 0. The RE will be set to 1.

4.5 UART Multiplexer

We have a problem with this way, because we have just one uart interface available on the riverdi expansion pins. So we need a multiplexer to receive the data. So we need a multiplexer to transmit all data on just one bus.

So for that we will use 2 boards of this component : UART MUX CLICK from the manufacturer MIKROE.

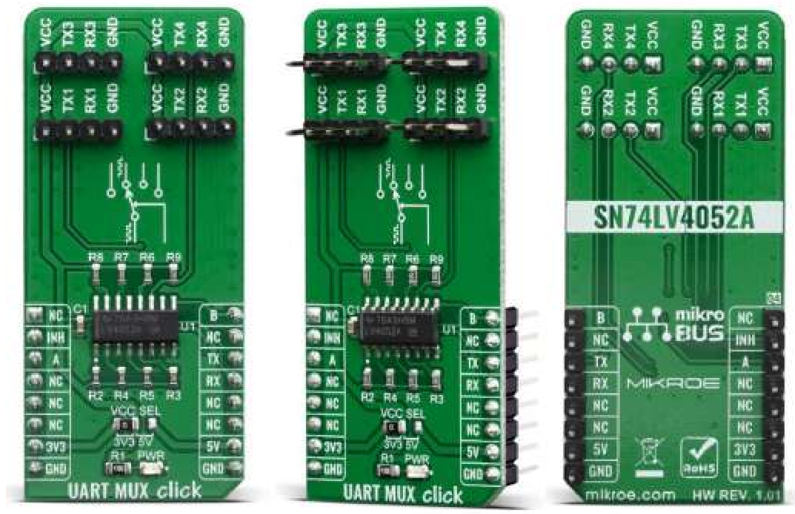


Figure 19: Multiplexer UART MUX CLICK

The UART Mux click is a Click board™ that switches the UART pins (RX and TX) from the mikroBUS™ to one of the four available outputs. It employs the SN74LV4052A, a Dual 4-Channel Multiplexer and Demultiplexer from Texas Instruments.

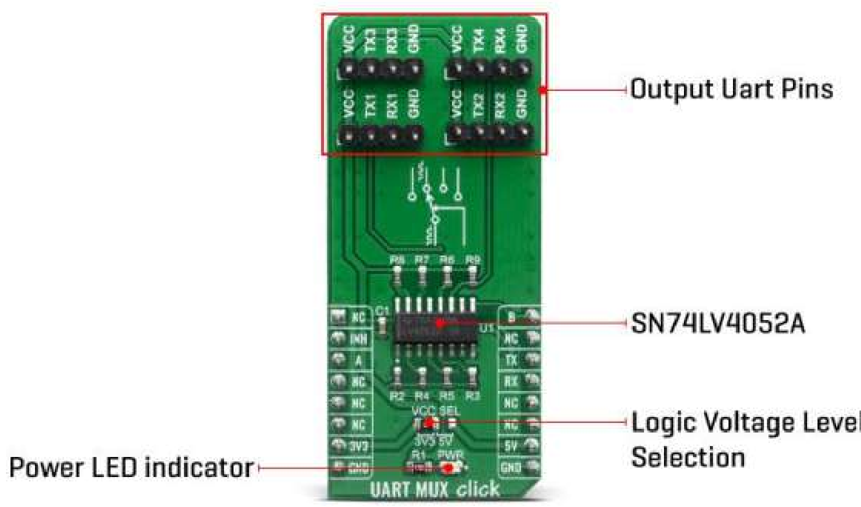


Figure 20: Multiplexer uart pins descriptions

HOW DOES IT WORK? labeled as A and B can be operated by 5V MCUs. The fourth control pin is labeled as EN pin, and it is used to enable the The active low Inhibit (INH) tri-state all the channels when high and when low, depending on the A and B inputs, one of the four independent input/outputs is connected to the UART communication pins.

4.6 Logic Analyzer

In this project we will be working with communication signals, so we need a logic analyser to simplify the data process on the prototype part. For that, we choose the logic analyzer shown below :

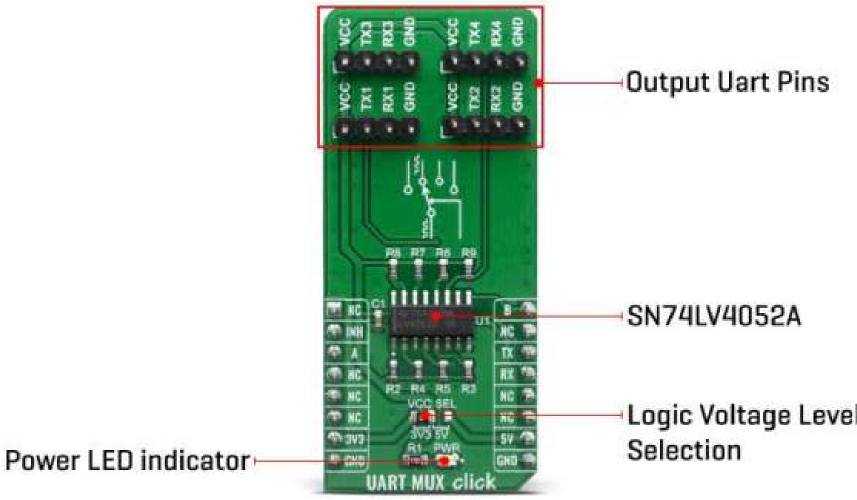


Figure 21: Logic Analyzer

We will use the software "Logic 2". This software is needed. With this Logic analyzer, we will can process the UART communication for the sensors and also the commnucation between the two boards (Riverdi and stm32F4).

4.7 Motor selection

According to the specifications, the stepper motor must be a Nema 17 type with a torque of 0.44 N.m and a good step angle accuracy (we will use 0.9°). We selected the WO-4209L-01P Lin Engineering motor respecting the requirements. The output current is 1.7A/phase and its supply voltage is 24V. (we will need this information for the driver).

Figure 22: Appearance of the Motor

Part Number	**WO-4209L-01P**
Step Angle	0.9°
Frame Size	NEMA 17
Body Length (Dim. A)	1.89 in (48 mm)
Current	1.7 Amps/Phase
Holding Torque	62 oz-in (0.44 Nm)
Resistance	1.9 Ohms/Phase
Rotor Inertia	0.37 oz-in^2
Number of Leads	4
Connection	Bipolar
Weight	0.7 lbs (0.32 kg)

(a) Motor specification

COLOR	FUNCTION
Red	A+ Phase
Blue	A- Phase
Green	B + Phase
Black	B- Phase

(b) Motor outputs

Figure 23: Motor details

4.8 Driver selection

After choosing the motor, we select the driver in accordance with the motor choice

R701P

MICROSTEPPING DRIVER

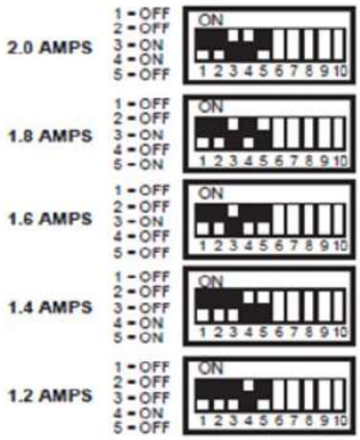
FEATURES & BENEFITS

REPLACES R701 MICROSTEPPING DRIVE

- 10 microstepping driver
- Common Ground or Common + 5 Volts Input Option Available
- Optically isolated Step, Direction, and Disable/Enable inputs
- Automatic Current Reduction
- Adjustable trimpot for noise and vibration reduction
- Operates from 18 to 80 VDC
- Selectable Driver Peak Current Ranges: 0 to 7 Amps
- Low Power Dissipation from 1 to 12 Watts (1 to 7 Amps)
- Excellent sinusoidal current waveform for smooth operation
- Low current ripple for low noise
- Low Cost
- High Efficiency

Figure 24: R701P Driver

We can regulate the output current with set switches integrated in the Driver from 0 to 7A. We select 1.6A because the current of the motor is 1.7A.



(a) Driver option set switches for regulate the output Current

Pin #	Color
1	Power Ground
2	+18 to 80 VDC
3	A Phase
4	A Bar Phase
5	B Phase
6	B Bar Phase
7	Disable Input
8	Direction Input
9	Step Input
10	+5 VDC
11	Current Set
12	Current Set

(b) Driver outputs

Figure 25: Driver details

4.9 STM32F411xE

We decided to use an STM32 NUCLEO microcontroller (specifically the STM32F411xE) because, after calculating the number of inputs/outputs used by the various components and the interface needed, we realised that the Riverdi's expansion connector did not have enough pins and interfaces to process our components.. So we decided to use an STM32 NUCLEO because of its small size and it's enough for our application.

Figure 26: Appearance of the STM32F4 NUCLEO

This microcontroller has I2C interface pins to communicate with the STM32H7 (included on the Riverdi board). Pins PB6/PB10/PA8 are for the SCL signals and PB7/PB9/PB4 are for the SDA signals.

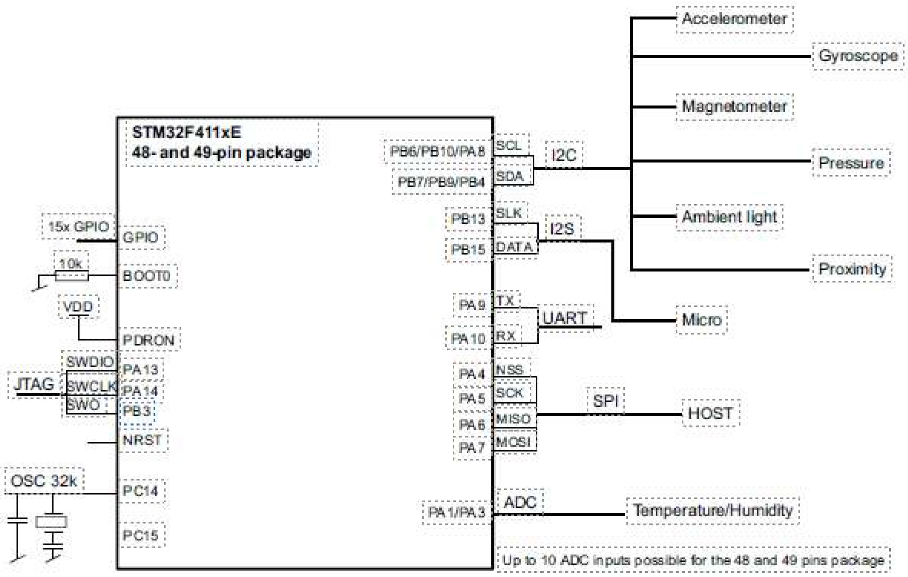


Figure 27: STM32F4 Datasheet I2C Interface Pins

4.10 Power supply selection

The power supply will supply the DRO and the DIVIDER with 24V. We need to identify the power and current consumption of each component from their datasheets.

We have get this information in our excel file (on the "Electrical characteristics" page), which you can download here: Download Excel File

According to our calculations from the Excel table, the maximum power consumption of our system (if everything is operating simultaneously at full power) is 105W with a current of 5.1A. Therefore, we need an oversized system compared to these values.

We selected the EDR-150-24 Mean Well power supply, which takes an input of 230VAC and provides an output voltage of 24VDC. The maximum power that this power supply can provides is 156W, which is 1.5 times more than necessary, and it can deliver up to 6.5A (providing a margin of 1.5A).

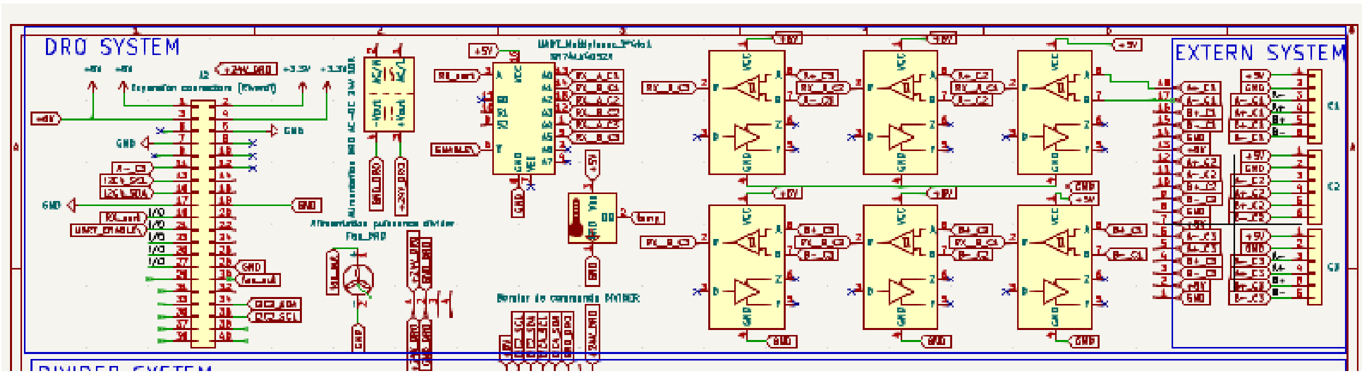
Figure 28: Appearance of the power supply

MODEL		EDR-150-24
OUTPUT	DC VOLTAGE	24V
	RATED CURRENT	6.5A / 230VAC 5.2A / 115VAC
	CURRENT RANGE	0 ~ 6.5A / 230VAC 0 ~ 5.2A / 115VAC
	RATED POWER	156W / 230VAC 125W / 115VAC
	RIPPLE & NOISE (max.) Note.2	150mVp-p
	VOLTAGE ADJ. RANGE	24 ~ 28V
	VOLTAGE TOLERANCE Note.3	±1.0%
	LINE REGULATION	±0.5%
	LOAD REGULATION	±1.0%
	SETUP, RISE TIME	1500ms, 60ms/230VAC 3000ms, 60ms/115VAC at full load
	HOLD UP TIME (Typ.)	16ms/230VAC 10ms/115VAC at full load
INPUT	VOLTAGE RANGE Note.6	90 ~ 264VAC 127 ~ 370VDC [DC input operation possible by connecting AC/L(+), AC/N(-)]
	FREQUENCY RANGE	47 ~ 63Hz
	EFFICIENCY (Typ.)	87%
	AC CURRENT (Typ.)	2.6A/115VAC 1.7A/230VAC
	INRUSH CURRENT (Typ.)	20A/115VAC 35A/230VAC
	LEAKAGE CURRENT	<1mA / 240VAC

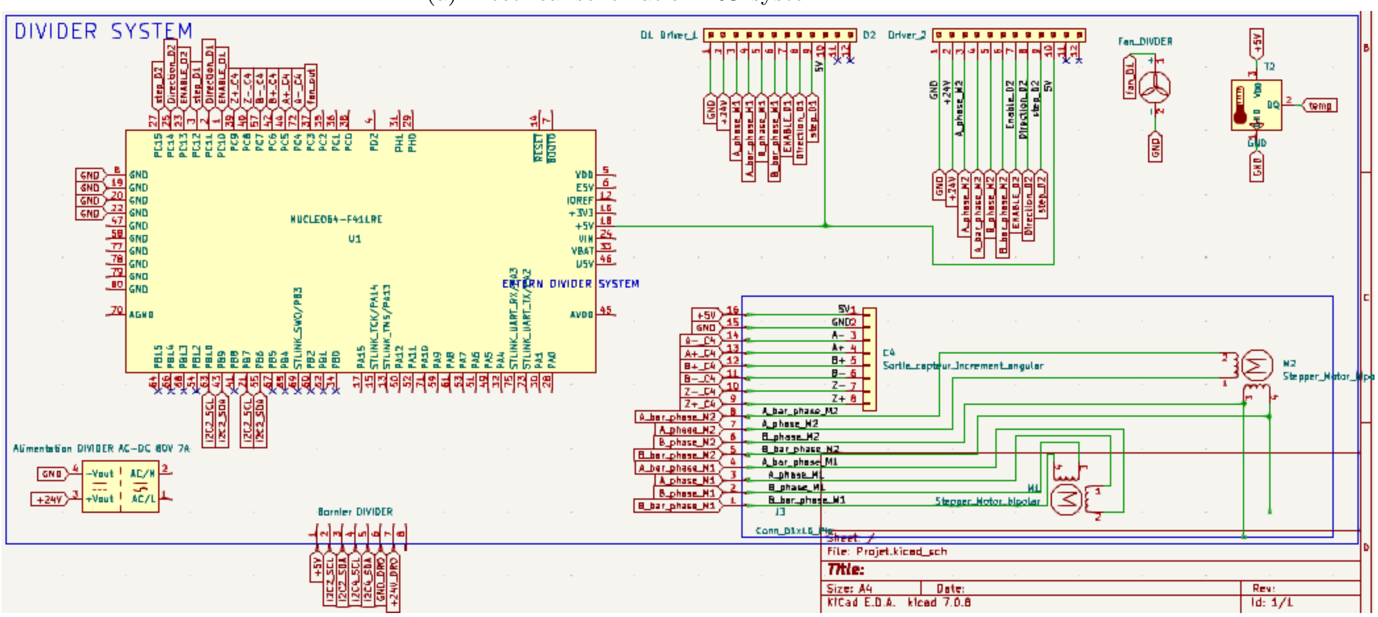
Figure 29: Specification of the power supply

5 Electrical schematic

The electrical schematic has been made in Kicad schematic.
Please Note : This electrical schematic is just for modeling the electrical design. We will not be creating a PCB afterward, so we do not need to have completely accurate symbols and footprints.



(a) Electrical schematic DRO system



(b) Electrical schematic DIVIDER system

The DRO and DIVIDER system will communicate each other by I2C interface.

6 Graphical Interface Development

6.1 User Interface Design

The graphical interface of the project was developed using TouchGFX, a powerful tool for creating embedded user interfaces on STM32 microcontrollers. The primary goal of the graphical interface was to allow smooth and intuitive interaction with the DRO (Digital Readout) system, enabling real-time display and modification of X, Y, and Z axis values.

6.1.1 MENU Screen

The user interface was designed with the end users in mind, who will operate the DRO and Divider systems. The main interface includes several screens dedicated to different functions:

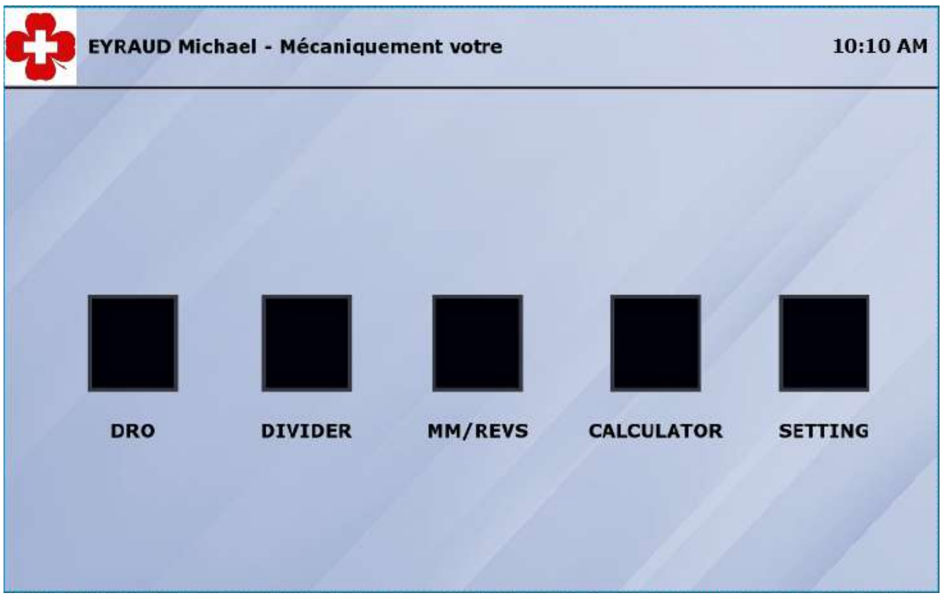


Figure 31: SPLASHMENU Graphical Interface

6.1.2 DRO Screen

Displays axis values, spindle speed, torque, and functions such as axis zeroing, value doubling, and value addition or subtraction.

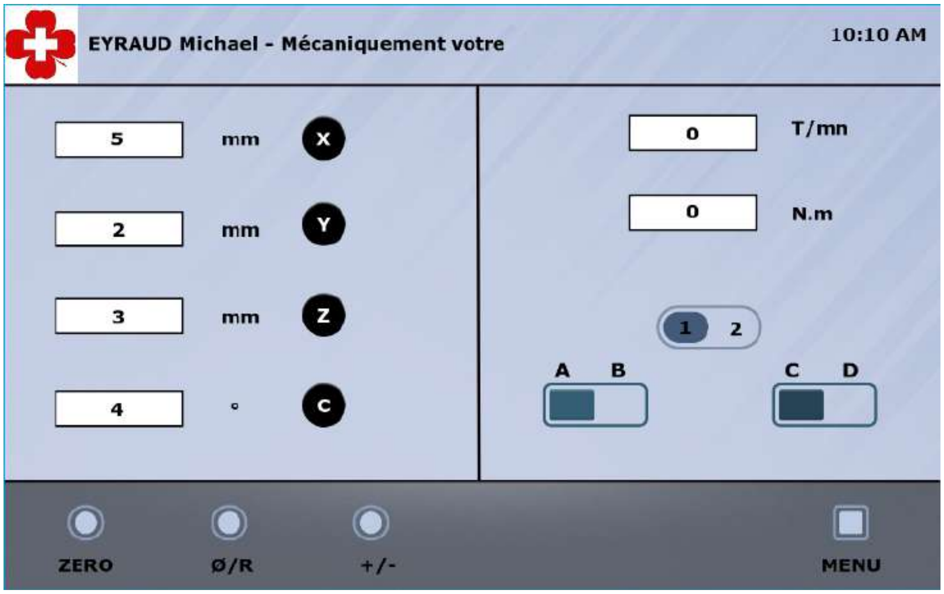


Figure 32: DRO Screen

6.1.3 Divider Screen

Allows control of Nema stepper motors, setting of angular divisions, movement distances, and associated speeds.

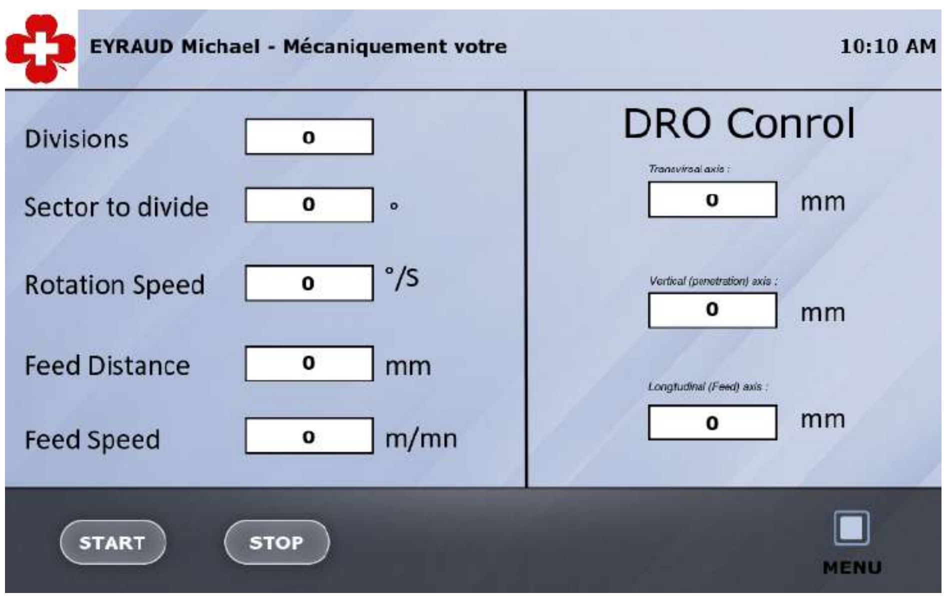


Figure 33: Divider Screen

6.1.4 MM/REVS Screen

Controls the revolutions per minute (RPM) of motors and displays spindle information, similar to the DRO screen.

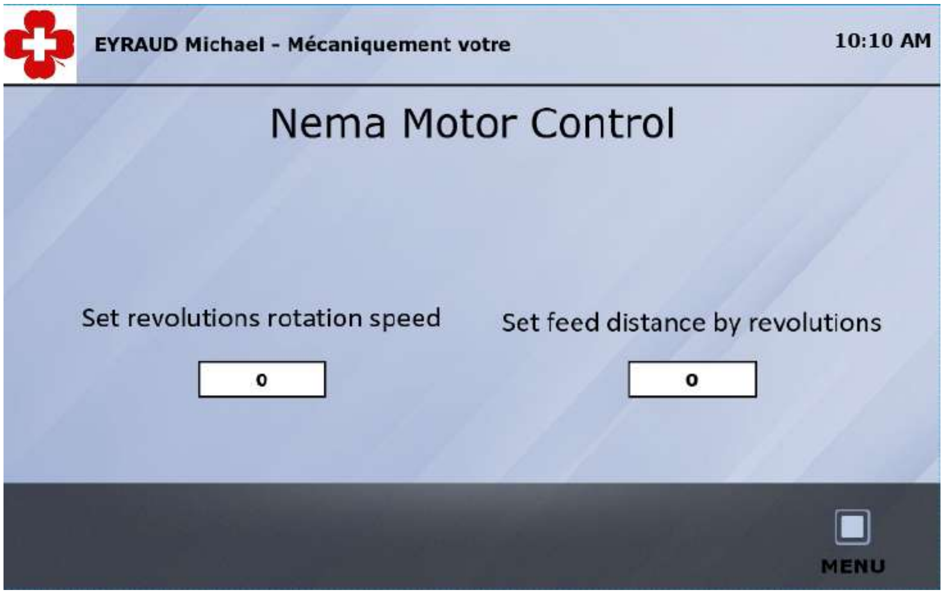


Figure 34: MM/REVS Screen

6.2 Implémentation avec TouchGFX

The graphical interface was implemented using TouchGFX within the STM32CubeIDE development environment. Key steps in the development included:

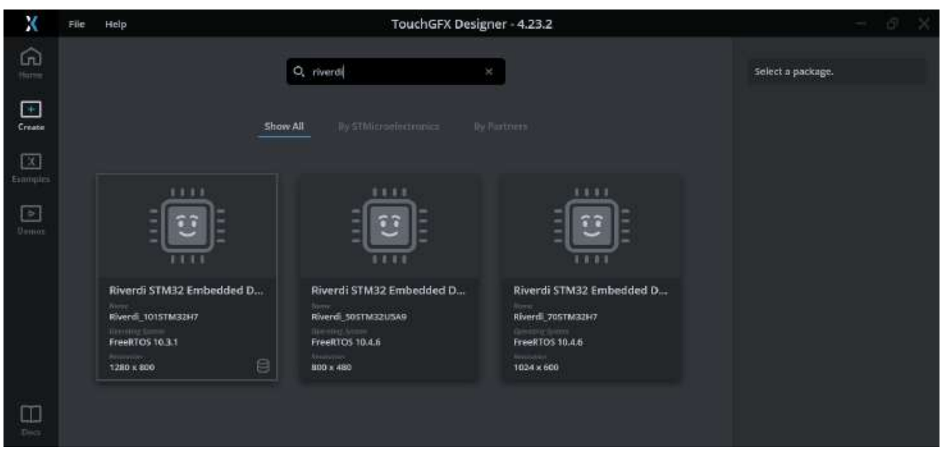


Figure 35: choice of interface

- **Initial Configuration:** The TouchGFX project was successfully configured in STM32CubeIDE, including the integration of the TouchGFX library, configuration of the STM32F7 board, and the TFT display.

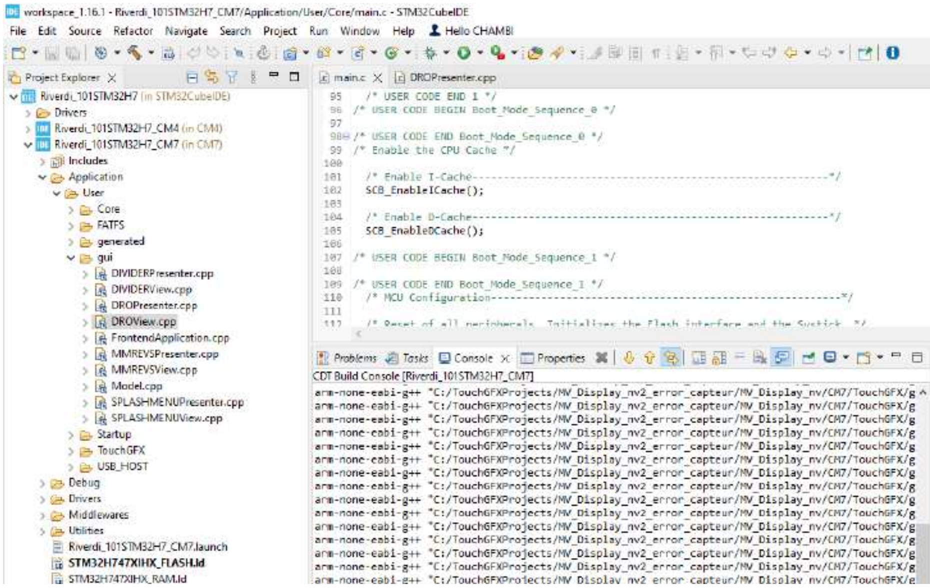


Figure 36: configuration in STM32CubeIDE

- **Screen Development:** The main screens of the interface, including the DRO screen, were created in TouchGFX Designer. Graphic elements such as TextAreas, buttons, and progress bars were added and configured.

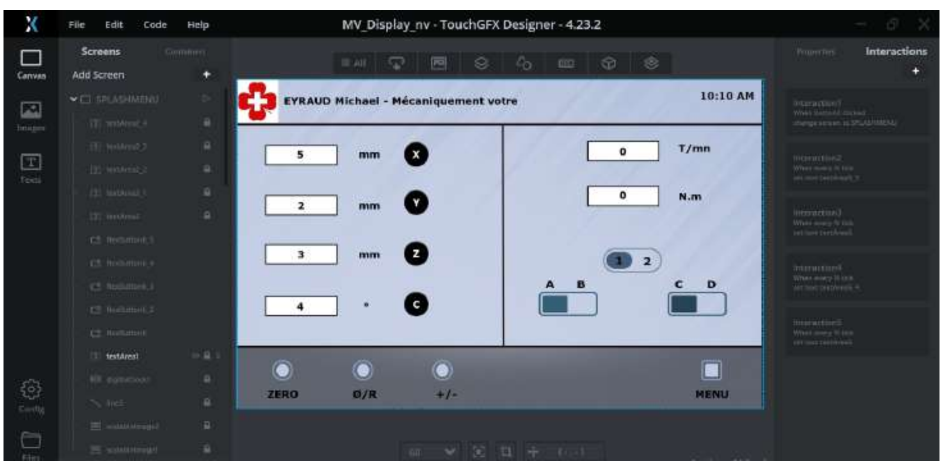


Figure 37: Graphic elements added and configured

- **Logic Implementation:** The code to update the X, Y, and Z axis values was partially implemented in the DROPresenter.cpp and DROView.cpp files.

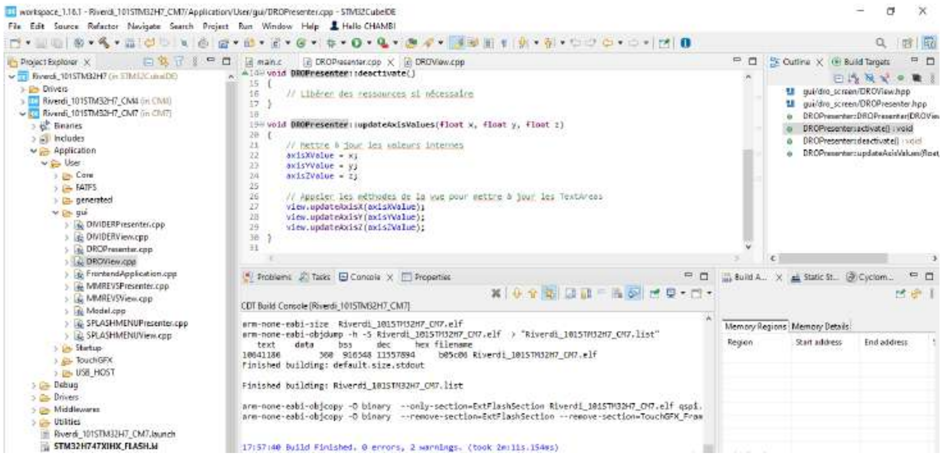


Figure 38: Logic Implementation

- **Simulation:** Simulation attempts were made successfully.

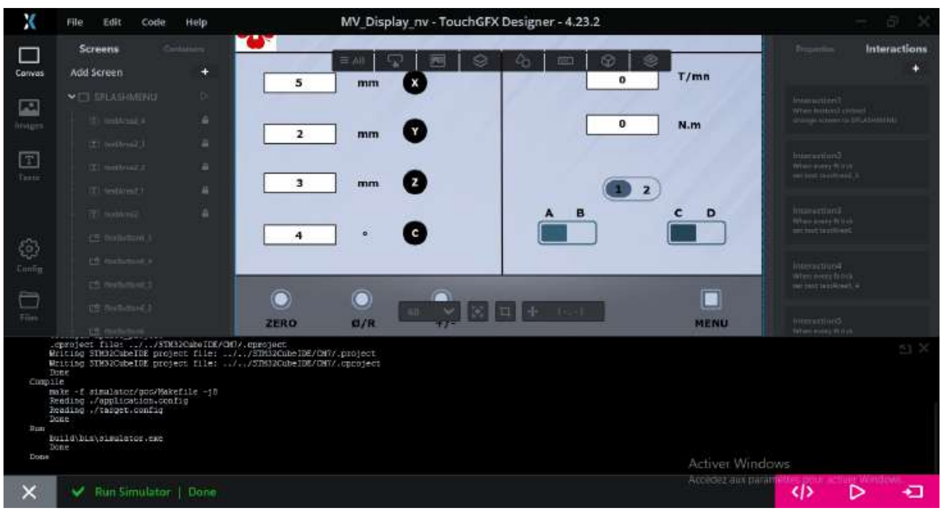


Figure 39: Simulation

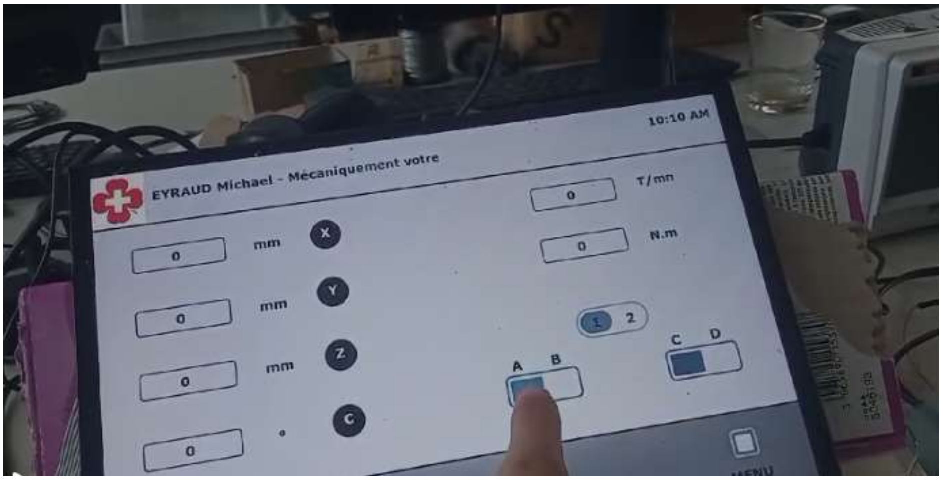


Figure 40: Implementation of the graphical interface

6.3 Button Implementation

During the development of the graphical interface, one of the key features implemented was the functionality of various buttons, such as the "Zero" button, which resets the values of the axes to zero. This feature is essential for the operation of the DRO system as it allows the user to recalibrate the machine quickly. During the development of the graphical interface, one of the key features implemented was the functionality of various buttons, such as the "Zero" button, which resets the values of the axes to zero. This feature is essential for the operation of the DRO system as it allows the user to recalibrate the machine quickly.

- **Zero Button:** The "Zero" button was successfully implemented, enabling users to reset the X, Y, and Z axis values to zero with a single click. This required careful handling of the underlying code to ensure that the reset operation was synchronized with the hardware and reflected immediately on the display.

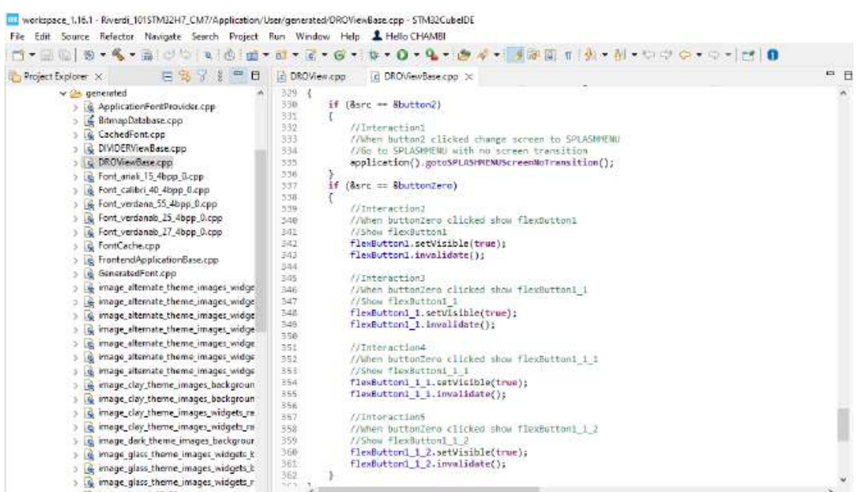


Figure 41: Implementation of Zero Button

- **Other Buttons:** In addition to the "Zero" button, other critical buttons, such as those for doubling the values or adding/subtracting specific measurements, were also implemented. Each button's function was mapped to specific operations in the code, ensuring that they performed as expected when pressed.

6.3.1 Challenges in Code Implementation

While implementing these buttons, several challenges were encountered, particularly in getting the buttons to interact correctly with the underlying system:

Complexity in Event Handling: The main challenge was handling the events triggered by these buttons, especially in ensuring that each button's function was executed without causing delays or conflicts in the system.

Code Debugging: Debugging the code associated with these buttons was also challenging. There were instances where the buttons did not respond as expected, often due to issues with how the events were being managed in the code. This required thorough testing and debugging to resolve.

7 Prototypes

7.1 Communication between the boards Riverdi and stm32F4

It exist differents way to communicate between two microcontrollers: - UART (USART) - SPI (Serial Peripheral Interface) - I2C (Inter - Integreted Circuit)

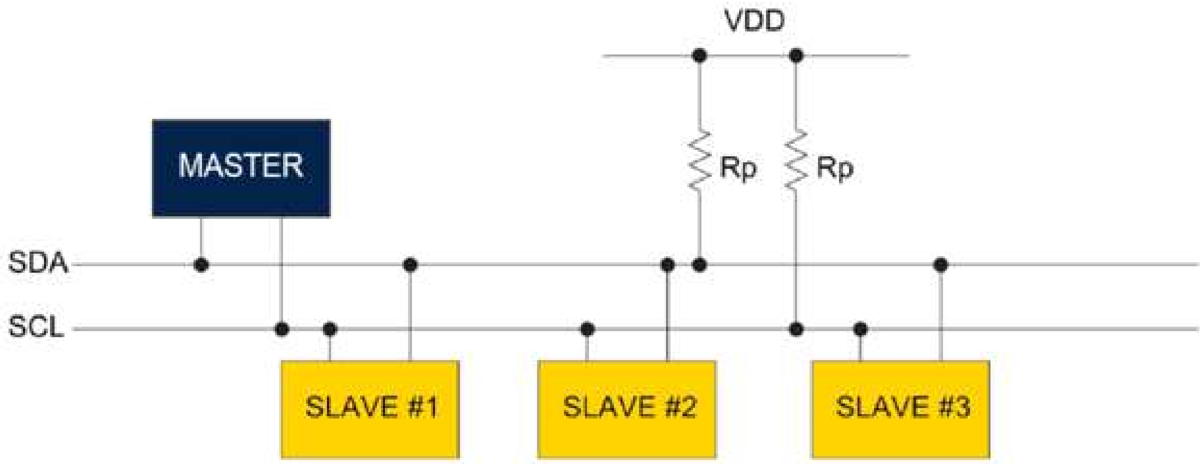
We compare the different way to communicate with the both boards

Criteria	**UART**	**SPI**	**I2C**
Advantages	Simplicity of coding, commonly used, fewer pins required than SPI. Both directions data transmit.	Duplex communication, speed, flexibility, no collision, practical for displaying sensor information.	Uses only 2 wires, supports multiple masters, ACK/NACK bits, widely used.
Disadvantages	No acknowledgment bits, slower than SPI.	Requires more pins, limited to short distances, no acknowledgment protocols, often a single master, compatibility issues with different operation modes.	Slower than SPI, protocol overhead.

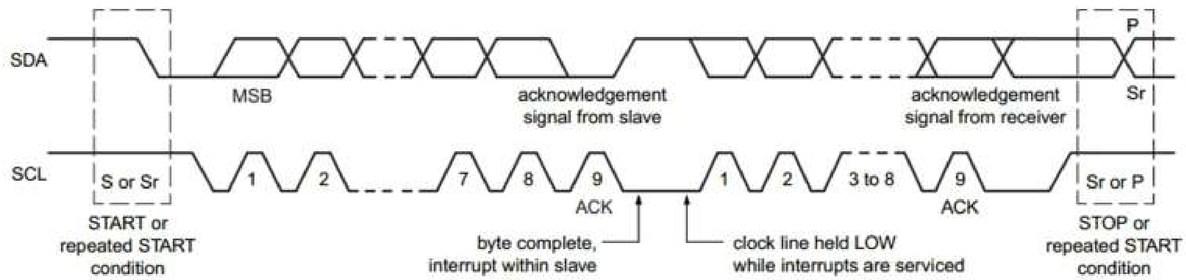
Table 3: Comparison between the different existing technologies with sufficient industrial credibility

We will develop the inter-integrated circuit (I2C) . For this, we will create a buffer to transmit the data to the board.

I2C is a two-wire serial communication system used between integrated circuits which was originally created by Philips Semiconductors back in 1982. The I2C is a multi-master, multi-slave, synchronous, bidirectional, half-duplex serial communication bus. SDA (Serial Data) is the line on which master and slave send or receive the information (sequence of bits). SCL (Serial Clock) is the clock-dedicated line for data flow synchronization.



(a) I2C Master/Slave



(b) I2C SDA SCL signals

Let's create the code for construct the I2C connexion. In the first time, we need to configure the I2C communication on the Master and Slave board.

```
' USER CODE END I2C1_Init 1 */
.2c1.Instance = I2C1;
.2c1.Init.Timing = 0x00707CBB;
.2c1.Init.OwnAddress1 = 0;
.2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
.2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
.2c1.Init.OwnAddress2 = 0;
.2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
.2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
.2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
: (HAL_I2C_Init(&hi2c1) != HAL_OK)
```

Figure 43: Master configurations

We have to define a slave address to communicate with the slave board from the master board. For that, we chose the address "0x24" so 36 in Decimal.

```
hi2c1.Instance = I2C1;
hi2c1.Init.ClockSpeed = 100000;
hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
hi2c1.Init.OwnAddress1 = 36;
hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
hi2c1.Init.OwnAddress2 = 0;
hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
if (HAL_I2C_Init(&hi2c1) != HAL_OK)
{
  Error_Handler();
}
```

Figure 44: Master configurations

For that, we need just to send a buffer with the code shown bellow :

```
uint8_t data[] = "Hello Slave!";
```

(a) Electrical schematic DRO system

```
/* USER CODE BEGIN 3 */
  HAL_I2C_Master_Transmit(&hi2c1, SLAVE_ADDRESS << 1, data, sizeof(data), HAL_MAX_DELAY);
  HAL_Delay(1000);
```

(b) Electrical schematic DIVIDER system

Start Condition (S) Stop Condition (P) Repeated Start (Restart) Condition (Sr) Acknowledge ACK (A) Not Acknowledge NACK (/A)

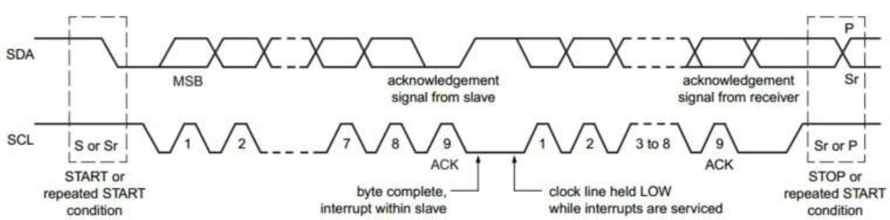
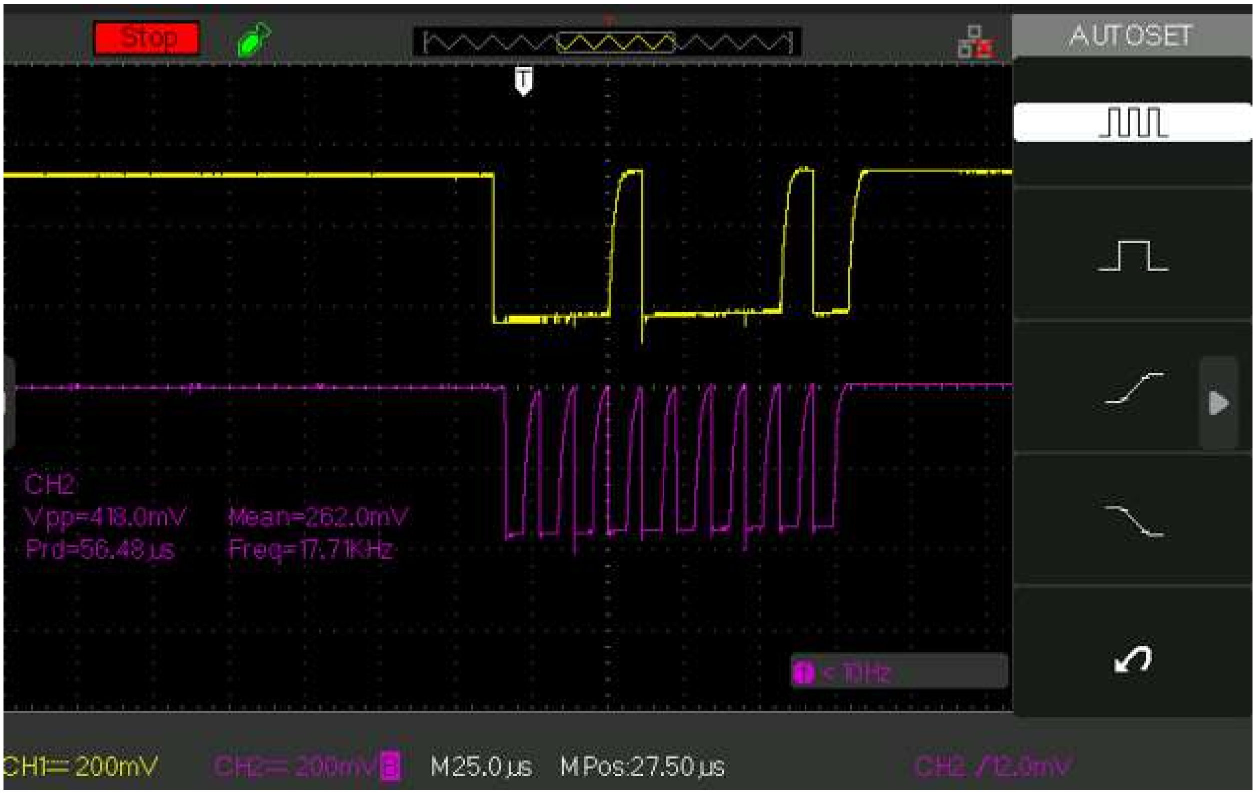


Figure 46: I2C SDA and SCL chronogram



(a) Oscilloscope : SDA (yellow) and SCL signals (rose)

```
/* USER CODE BEGIN 3 */
  HAL_I2C_Master_Transmit(&hi2c1, SLAVE_ADDRESS << 1, data, sizeof(data), HAL_MAX_DELAY);
  HAL_Delay(1000);
```

(b) Electrical schematic DIVIDER system

7.2 UART communication

We will program a uart receiver for process the output signals of the Incremental sensors. In the first time, we will just create a simple UART communication for understand how it is works. For that, we will also send a buffer from the Riverdi and receive it in the stm32F4 board. We chose pins for the UART master communication.

Figure 48: UART pins

For that, I create in the Master board a file "uart.c" :

```c
#include "usart.h"

/* USER CODE BEGIN 0 */
#define UART_RX_BUFFER_SIZE  40
uint8_t UART1_RxBuffer[UART_RX_BUFFER_SIZE] = {0};
uint16_t RxDataLen = 0;
/* USER CODE END 0 */

UART_HandleTypeDef huart1;
DMA_HandleTypeDef hdma_usart1_rx;
```

Figure 49: UART sending buffer

We configure the UART master configurations.

```c
/* USER CODE END USART1_Init 1 */
huart1.Instance = USART1;
huart1.Init.BaudRate = 115200;
huart1.Init.WordLength = UART_WORDLENGTH_8B;
huart1.Init.StopBits = UART_STOPBITS_1;
huart1.Init.Parity = UART_PARITY_NONE;
huart1.Init.Mode = UART_MODE_TX_RX;
huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
huart1.Init.OverSampling = UART_OVERSAMPLING_16;
huart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
huart1.Init.ClockPrescaler = UART_PRESCALER_DIV1;
huart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
if (HAL_UART_Init(&huart1) != HAL_OK)
{
  Error_Handler();
}
if (HAL_UARTEx_SetTxFifoThreshold(&huart1, UART_TXFIFO_THRESHOLD_1_8) != HAL_OK)
{
  Error_Handler();
}
if (HAL_UARTEx_SetRxFifoThreshold(&huart1, UART_RXFIFO_THRESHOLD_1_8) != HAL_OK)
{
  Error_Handler();
}
if (HAL_UARTEx_DisableFifoMode(&huart1) != HAL_OK)
{
  Error_Handler();
}
```

Figure 50: UART configurations

We do the same thing but with the slave side.

Figure 51: UART pins

For that, I create in the Master board a file "uart.c" :

```c
#include "usart.h"

/* USER CODE BEGIN 0 */
#define UART_RX_BUFFER_SIZE  40
uint8_t UART1_RxBuffer[UART_RX_BUFFER_SIZE] = {0};
char UART1_TxBuffer[UART_RX_BUFFER_SIZE] = {0};
uint16_t RxDataLen = 0;
/* USER CODE END 0 */

UART_HandleTypeDef huart1;
DMA_HandleTypeDef hdma_usart1_rx;
```

Figure 52: UART sending buffer

We configure the UART master configurations.

```c
/* USER CODE END USART1_Init 1 */
huart1.Instance = USART1;
huart1.Init.BaudRate = 115200;
huart1.Init.WordLength = UART_WORDLENGTH_8B;
huart1.Init.StopBits = UART_STOPBITS_1;
huart1.Init.Parity = UART_PARITY_NONE;
huart1.Init.Mode = UART_MODE_TX_RX;
huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
huart1.Init.OverSampling = UART_OVERSAMPLING_16;
if (HAL_UART_Init(&huart1) != HAL_OK)
{
  Error_Handler();
}
```

Figure 53: UART configurations

We will now watch on the logic analyzer if it is works. For that, we do the schematic shown below :

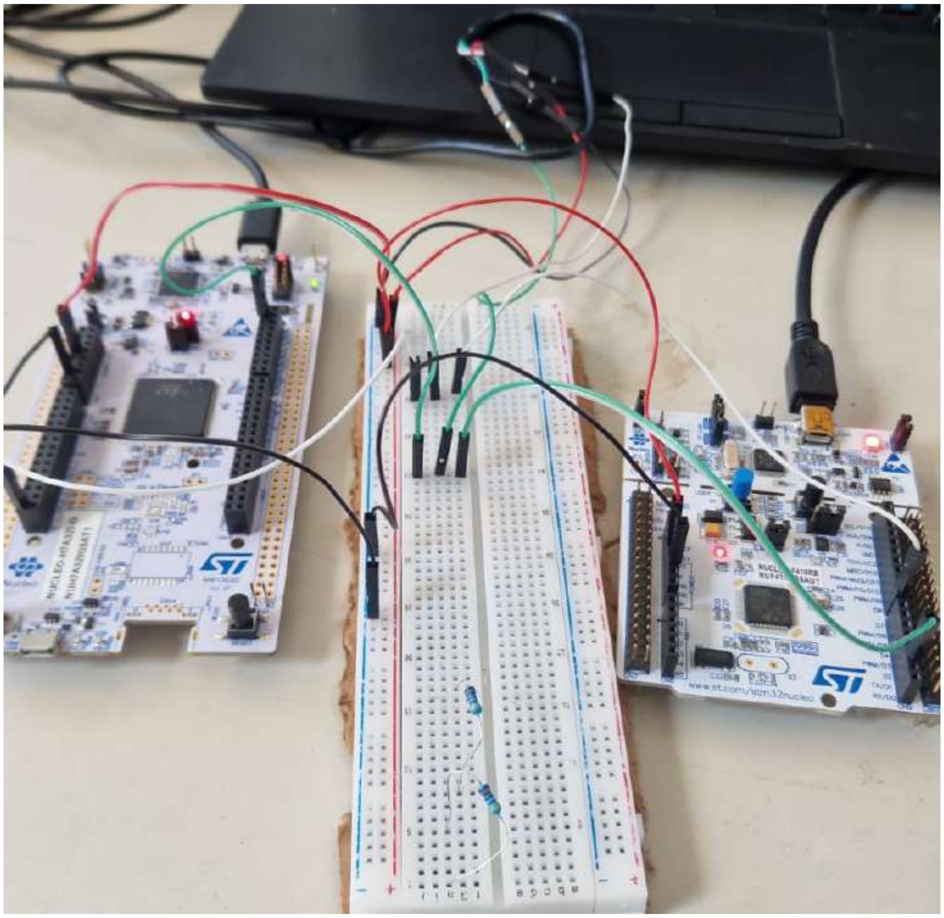


Figure 54: UART wiring

For receive the buffer data, we had a problem with the logic anlayzer, so we used a uart ttl to usb device with the Putty software :

We write on the sending buffer "Hello from the Master" and we will check if we receive correctly the information.

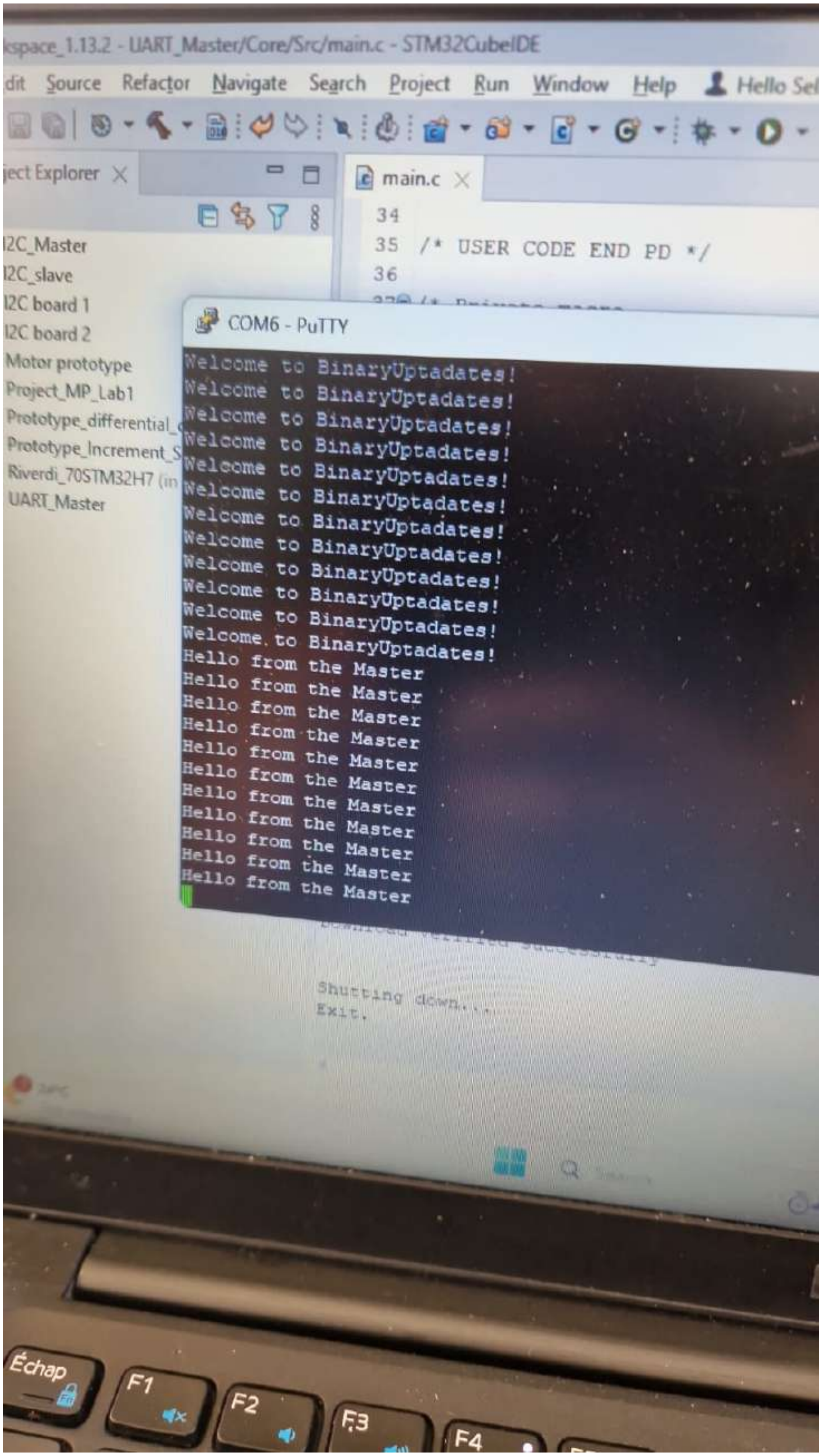


Figure 55: UART to usb device

We receive correctly the receive information so the prototype works correctly.

7.3 Motor prorotype

We generate code to control the drivers. To do this, we're going to generate the 'STEP' (PWM signal), 'ENABLE' and 'DIRECTION' control signals. We use the register block to create the PWM signal.

First we need to know some formulae.

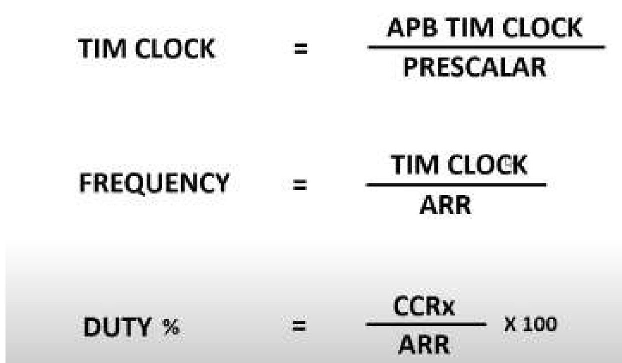


Figure 56: Formules des timers

TIM1 est lié à APB2. On identifie : APB TIM clock = 100 MHz.

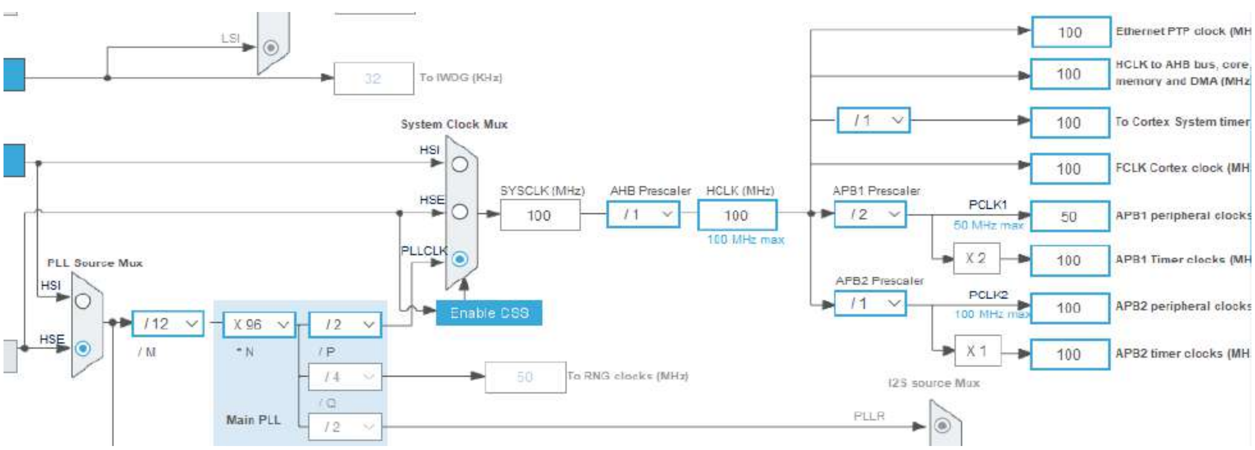
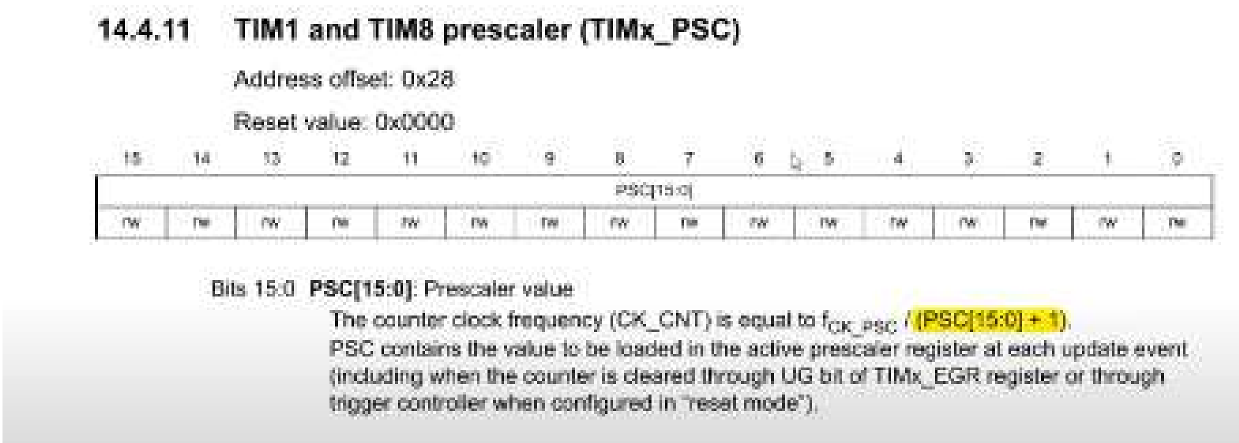


Figure 57: Configuration de l'horloge

TIM clock = APB TIM clock / PRESCALER

TIM clock = 100 Mhz / 100 = 1 Mhz Donc on paramètre PRESCALER à : 100-1 (= 99) Frequency = 1 Mhz / ARR Frequency = 1Mhz / 100 = 10 kHz

Figure 58: Configuration du TIMER 1



14.4.11 TIM1 and TIM8 prescaler (TIMx_PSC)

Address offset: 0x28

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency (CK_CNT) is equal to f_{CK_PSC} / (PSC[15:0] + 1).

PSC contains the value to be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIMx_EGR register or through trigger controller when configured in "reset mode").

Figure 59: Formules des timers

On génère également les signaux "ENABLE" et "DIRECTION" qui sont des signaux binaires.

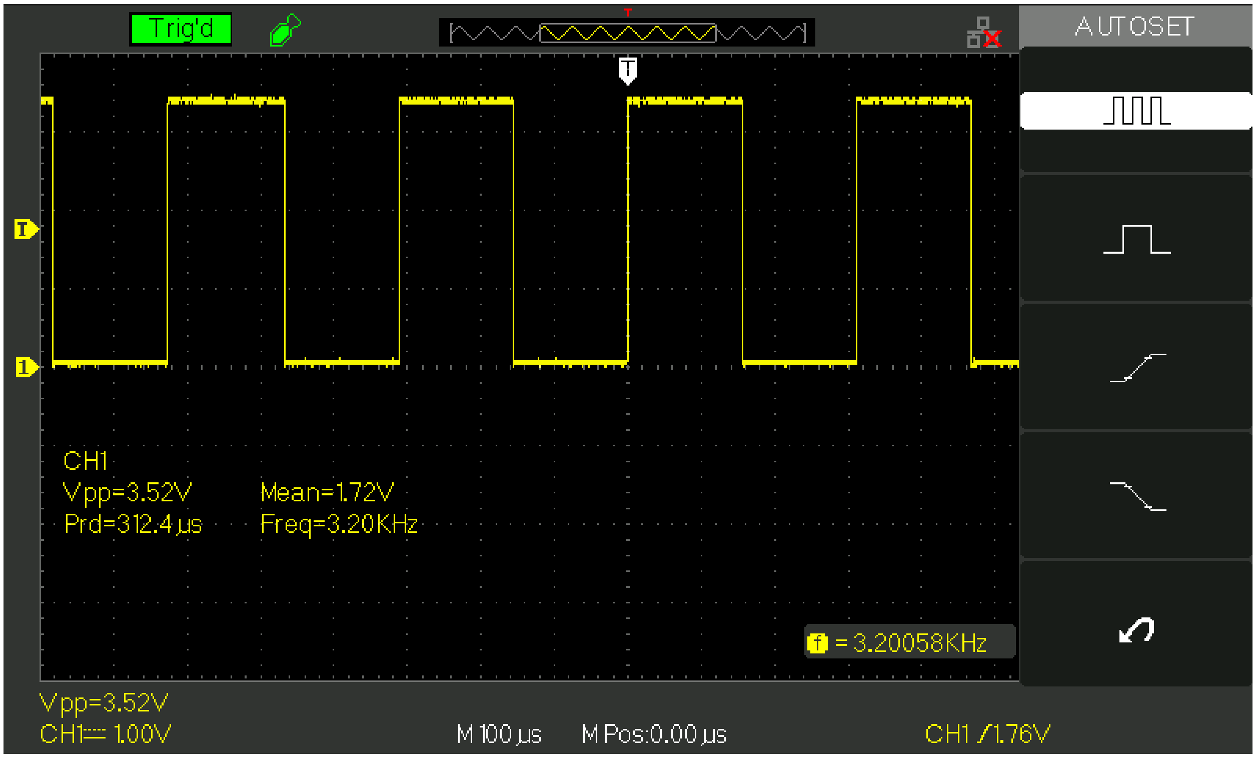


Figure 60: Visualisation de "STEP"

We generate a code that will control the drivers. For that, we will, generate signals from the microcontroller for the Direction Input, and stop input. We use block from the register for create PWM signals.

We create a motor prototype for testing the control motors part

We need to know this formulas:

Le timer 1 is conntected à APB2 TIM clock = APB TIM clock / PRESCALER TIM clock = 100 Mhz / 100 = 1 Mhz Frequency = 1 Mhz / ARR Frequency = 1Mhz / 100 = 10 kHz

The prototype system works well.

7.4 Increment sensor prototype

We received a sensor without an exact reference, so the first step was to identify each of the signals in order to know their functions and then use them.

Signal A+ : gray wire

Each signal has its complementary signal to reduce Electromagnetic Interference (EMI) and improve signal robustness, especially over long cables.

Signal A-: yellow wire The A+ signal is a square wave that changes state (high/low) at regular intervals when the encoder rotates.

Signal B+: Rose wire

The B+ signal is a square wave that is phase-shifted by 90 degrees relative to the A+ signal. The 90-degree phase shift between A+ and B+ allows for the determination of the direction of rotation. If A+ leads B+, the rotation is in one direction, and if B+ leads A+, the rotation is in the opposite direction. Signal B- : brown wire. The B- signals is the opposition of B+ wire.

The first time we use a data logic analyser with Logic 2 software (from salae). We connect A+;A-,B+,B- to the data logic analyser to check the signals. We check the signals without moving the sensor.

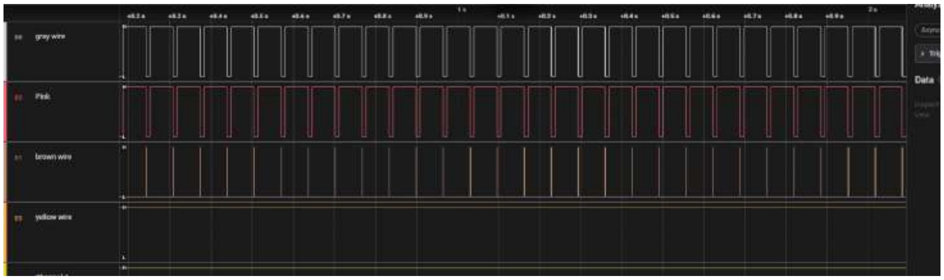


Figure 61: A+;A-;B+;B- inputs

Now, we move the sensor and along the magnetic scale.

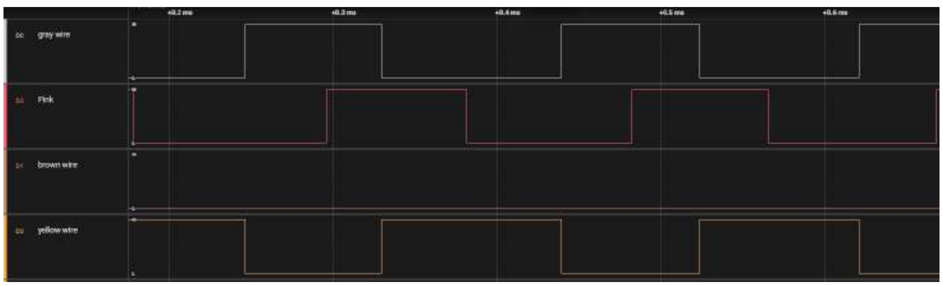


Figure 62: A+;A-;B+;B- inputs moving the sensor along the magnetic scale

All signals are correct.

We can now connect the signals to the rs485 transeiver. We had a problem in the receiving component and did't have time enough for done this prototype.

1 : Green +5V 2 : X 3 : X 4 : White 5 : red 6 : Yellow 7 : gray 8 : Brown

Jaune et blanc même signal si brown = GND Rose et gris : même signal

La communication A+, A-, B+ et B- s'appel une communication RS422.

Nous utilisons un encoder afin de traiter les signaux.

8 Challenges and Problem Solving

Several challenges were identified during development:

- **Compilation Errors:** Compilation errors were mainly due to missing text identifiers in DROView.cpp and the use of outdated or unsupported methods.
- **Text Mapping Issues:** Mapped texts were not correctly recognized, leading to additional errors during compilation.
- **Simulation Issues:** Due to the aforementioned errors, simulation could not be successfully executed, making visual validation of the interface difficult.
- **Error Correction:** It is necessary to correct the current errors by replacing incorrect text identifiers and using appropriate methods for the TextArea class.
- **Interface Finalization:** Complete the implementation of the axis value update functions in DROView and DROPresenter.
- **Testing and Validation:** After correcting the errors, the simulation must be executed to validate the interface and its functionality.
- **Optimization and Debugging:** Resolve any remaining errors or functional issues and optimize the code for smooth performance on the target hardware.

9 Results and Discussions

The final results demonstrate that the graphical interface is capable of effectively handling the assigned tasks, despite the challenges encountered during development. The user interface is intuitive, responsive, and integrates well with other systems, providing a complete solution for machine control.

10 Conclusion

10.1 astonishment report

10.1.1 Insufficient supervision and non-compliance with the requirements of the engineering assistant placement

During my work placement as an assistant engineer, I was surprised by the lack of adequate technical supervision. My tutor, Michael Eyraud, is a mechanic and didn't have the necessary skills to supervise a project requiring expertise in electronics. Abdel and I, as trainees specialising in this field, had to carry out our project completely independently, without the support of a qualified engineer. Not only is this an unusual situation, but it does not comply with the requirements for assistant engineer traineeship.

10.1.2 Inappropriate management of trainees and overloading of the tutor

As the company's only employee, the placement tutor had to manage 8 trainees at the same time, all of whom were assistant engineers. This overload resulted in ineffective management and inadequate supervision, which is particularly serious in the context of an engineering assistant placement, where specific and rigorous technical supervision is required.

10.1.3 Harassment and blackmail of trainees

During this placement, we observed various forms of harassment and blackmail by the tutor towards several trainees. We could cite dozens of examples of moral harassment, all of which have been documented, but that's not the purpose of this report today.

10.2 Conclusion

This internship allowed us to develop a user interface for a DRO system using differents technicals softwares such as stm32cubeIDe, TouchGFX, Logic 2 and more. Through this experience, we was able to enhance my skills in system engineering and software development while addressing technical challenges related to real-time sensor integration.

In summary, this experience has been enriching and has prepared me for the technical and professional challenges we will face in our careers.