Hes·so VALAIS WALLIS
School of Engineering

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts Western Switzerland

Degree Programme

Systems Engineering

Major Infotronics

BACHELOR'S THESIS

DIPLOMA 2020

Gregory Geraci

HYPNOSIA Controller

Professor
Dr. Silvan Zahno

Expert
Clivaz Romain

Submission date of the report
21.08.2020

Valais excellence

HES-SO Valais

SYND	ETE	TEVI
X	X	X

Données du travail de diplôme
Daten der Diplomarbeit

FO 1.2.02.07.CB
haj/11/01/2016

Filière / Studiengang **SYND**	Année académique / *Studienjahr* **2019/20**	No TD / *Nr. DA* **it/2020/53**
Mandant / *Auftraggeber* ☐ HES—SO Valais ☐ Industrie ☒ Etablissement partenaire *Partnerinstitution*	Etudiant / *Student* **Gregory Geraci** Professeur / *Dozent* **Silvan Zahno**	Lieu d'exécution / *Ausführungsort* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire *Partnerinstitution*
Travail confidentiel / *vertrauliche Arbeit* ☐ oui / ja [1] ☒ non / nein	Expert / *Experte (données complètes)* **Romain Clivaz** Soprod SA, Rue de la Blancherie 63, 1950 Sion romain.clivaz@soprod.ch	

Titre / *Titel*

Hypnosia - Contrôleur

Description / *Beschreibung*

Hypnosia est une jeune startup, issue du programme Business eXperience (BeX), qui est née de l'union entre ingénieurs et économistes. Le projet consiste à développer et mettre en œuvre un prototype d'un afficheur matriciel composé de mouvements de montre (biaxes et tri-axes), issue de la société SOPROD SA. Alliant à la fois les domaines de la mécatronique, de l'électronique, de l'informatique et de la mécanique, le projet consistera principalement au développement d'un contrôleur central permettant le contrôle et la gestion de tous les mouvements des moteurs. Le contrôleur principal permettra aussi de se connecter via Bluetooth à un smartphone pour une communication à distance et sans fil.

Objectifs / *Ziele*

- Développement d'un prototype fonctionnel de 6 lignes et 14 colonnes. Cela fait au total 84 moteurs
- Développement de la carte électronique universelle slave responsable du fonctionnement de plusieurs moteurs (2x3) (compatible avec ou sans master et avec des moteurs biaxes et tri-axes)
- Développement d'un protocole de communication entre le master et tous les modules d'horloge (slave)
- Développement d'un master pouvant gérer toutes les animations mises à disposition
- Développement au minimum de deux algorithmes d'animation
- Développement de la communication Bluetooth entre le smartphone (Android) et le master
- Mise en boîtier du système (optionnel)
- Minimiser les coûts de production <400CHF.

Signature ou visa / *Unterschrift oder Visum*

Responsable de l'orientation / filière
Leiter der Vertiefungsrichtung / Studiengang:

.....

[1] Etudiant / *Student* :

Geraci Gregory

Délais / *Termine*

Attribution du thème / *Ausgabe des Auftrags:*
25.05.2020

Présentation intermédiaire / *Zwischenpräsentation*
Semaine / Woche 26 (22.06 – 26.06.2020)

Remise du rapport / *Abgabe des Schlussberichts:*
21.08.2020, 12:00

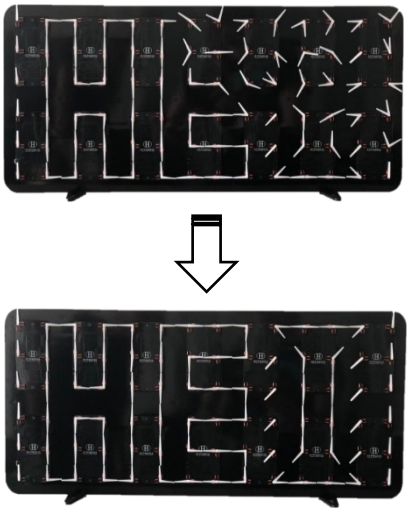
Exposition / *Ausstellung der Diplomarbeiten:*
28.08.2020 (si autorisé / *falls genehmigt*)

Défense orale / *Mündliche Verfechtung:*
Semaine / Woche 36 (31.08 – 04.09.2020)

[1] *Par sa signature, l'étudiant-e s'engage à respecter strictement la directive DI.1.2.02.07 liée au travail de diplôme.*
Durch seine Unterschrift verpflichtet sich der/die Student/in, sich an die Richtlinie DI.1.2.02.07 der Diplomarbeit zu halten.

Rapport reçu le / *Schlussbericht erhalten am* ..... Visa du secrétariat / *Visum des Sekretariats* .....

HYPNOSIA Controller



Graduate Gregory Geraci

Objectives

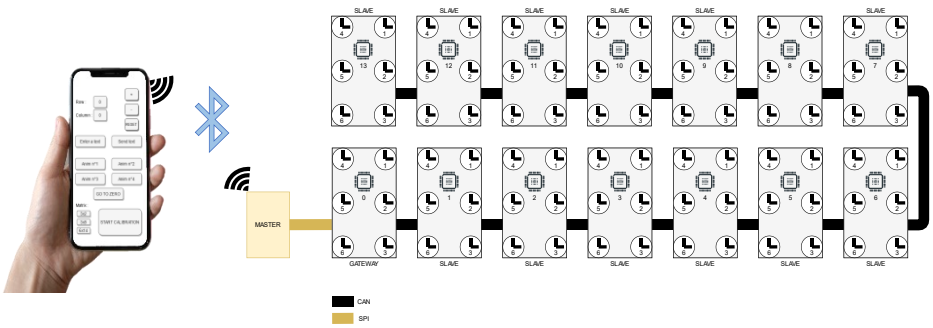
The project consists in developing and implementing a prototype of a bi-axes movements matrix display that allows the creation of the most hypnotizing animations.

Methods | Experiences | Results

The matrix display consists of 84 bi-axes movements produced by SOPROD SA. The ultimate goal is to control animations via Bluetooth according to the user's wishes.

The system is composed of a MASTER (Raspberry Pi), a GATEWAY (processor) and 13 SLAVES (processor). The system works as follows: the Raspberry Pi (MASTER) manages the Bluetooth connection and transmits the data required to control the movement matrix to the processor (GATEWAY) via SPI. This processor then processes the data received via SPI and transmits the data to all other processors (SLAVES) via a CAN bus. Each processor controls 6 bi-axes movements.

The first prototype developed gives conclusive results.



The diagram above shows the communications:

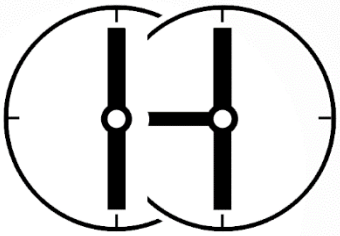
- Smartphone - MASTER via Bluetooth
- MASTER - GATEWAY via SPI
- GATEWAY - SLAVES via **CAN**

Bachelor's Thesis
| 2020 |

Degree programme
Industrial Systems

Field of application
Infotronics

Supervising professor
Dr Silvan Zahno
silvan.zahno@hevs.ch



HYPNOSIA

L'ÉLÉGANCE DÉPASSE LE TEMPS

TABLE OF CONTENTS

1 INTRODUCTION

Hypnosia[1] is a young startup from the Business eXperience (BeX) program, which was born from the union between two engineers and two economists.

Figure 1 : HYPNOSIA logo

Our start-up offers a brand new dynamic board that will take you to the rhythm of the watch pointers. Elegance, purity and simplicity best characterize our creation. This "dancing display" is a matrix of clocks that reveals various hypnotising animations and guarantees a feeling of calm.

The realization of our product will be broken down into two diploma works (see part 1.1).

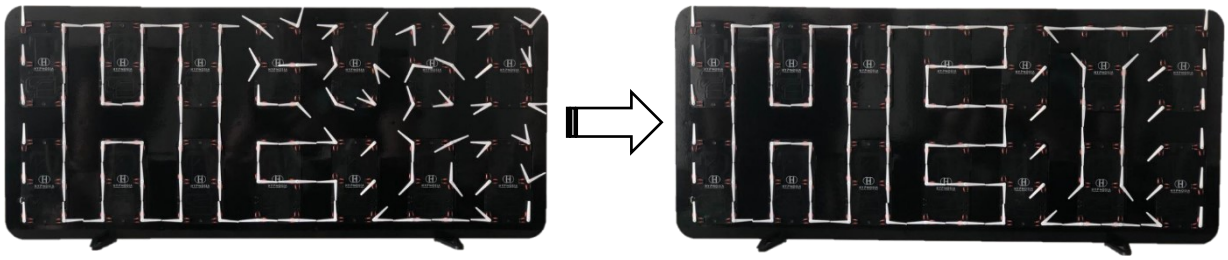


Figure 2 : Clock matrix. Product offer by HYPNOSIA

More information:

- Website: https://www.hypnosia.ch/
- Promotional video: https://www.youtube.com/watch?v=gUSDaGv5s2o
- GitHub: https://github.com/GeraciGregory/HypnosiaController

1.1 Global project description

The final objective is to create a clock matrix that can be remotely controlled with a smartphone in order to create animations at the customer's request.

In order to achieve this, the project will be broken down into two diploma works.

1. The first part, carried out by Geraci Gregory, consists in the creation of the first prototype of the clock matrix. That is to say set up a system that can communicate via Bluetooth and control all the movements in order to create animations. (*HYPNOSIA Controller* project)

2. The second part, carried out by De Campos Ruben, consists of the detection of the position of the watch pointers via image processing, an image taken by the user of the product. This information will then be sent via Bluetooth to the clock matrix to calibrate all the watch pointers. (*HYPNOSIA Calibration* project)

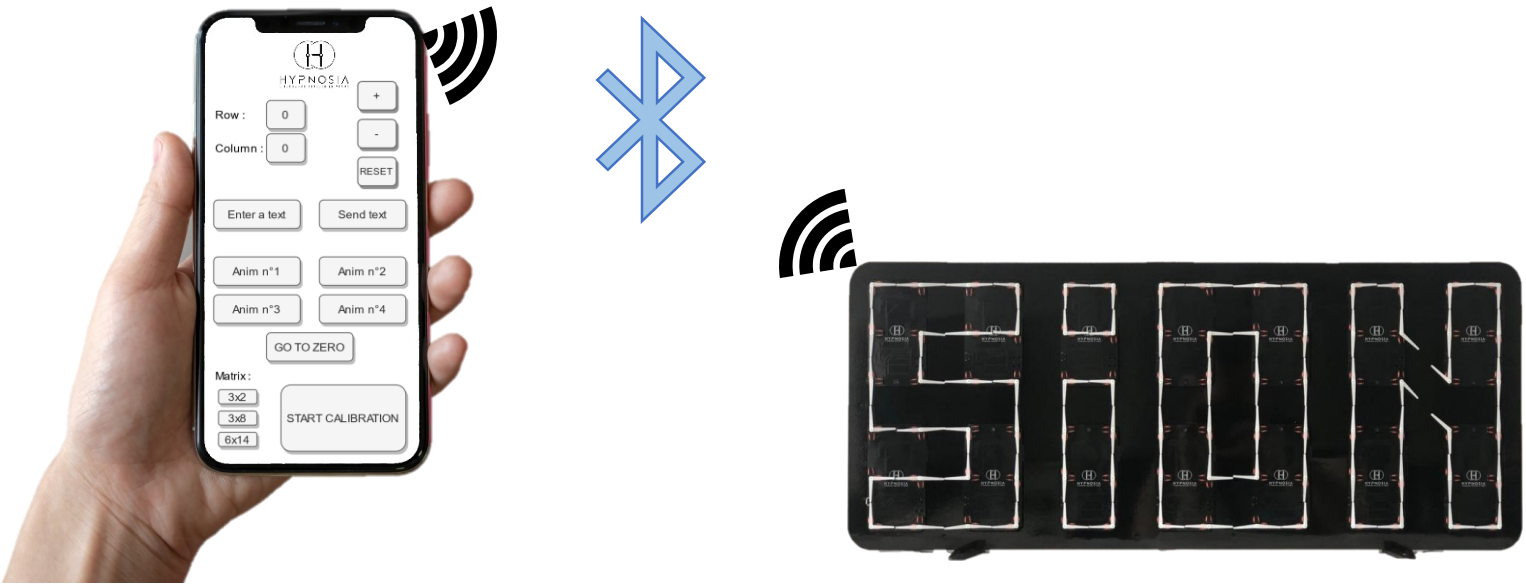


Figure 3 : Block diagram of the global project

1.2 HYPNOSIA Controller project description

As said before, the project consists of developing and implementing a prototype of a matrix display composed of movements (bi-axes and tri-axes), from the company SOPROD SA.

Combining both the fields of mechatronics, electronics, computer science and mechanics, the project will consist in the development of a system to control and manage all of the movements. The system will also allow wireless remote control via a Bluetooth link.

In order to do this, the project must be divided into several distinct parts and give them a well-defined role.
Here are the main sections:

1. **MASTER**: Its role is to manage the Bluetooth communication with the user as well as to give the right information to the GATEWAY to turn the right watch pointer of the right engine at the right time.
2. **GATEWAY**: He will have to translate the information received by the MASTER and inform all SLAVES of the actions to be taken. It will also rotate its own watch pointers.
3. **SLAVE**: Executes the actions provided by the GATEWAY. It will rotate its own watch pointers.

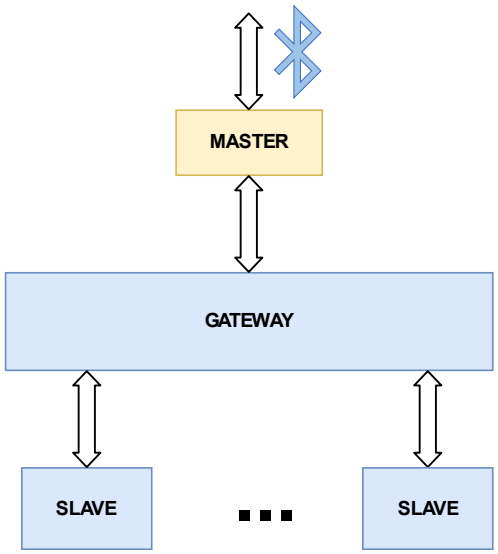


Figure 4 : MASTER – GATEWAY – SLAVE concept

Here is a block diagram with the concepts of MASTER – GATEWAY - SLAVE to give a better understanding of the final product.

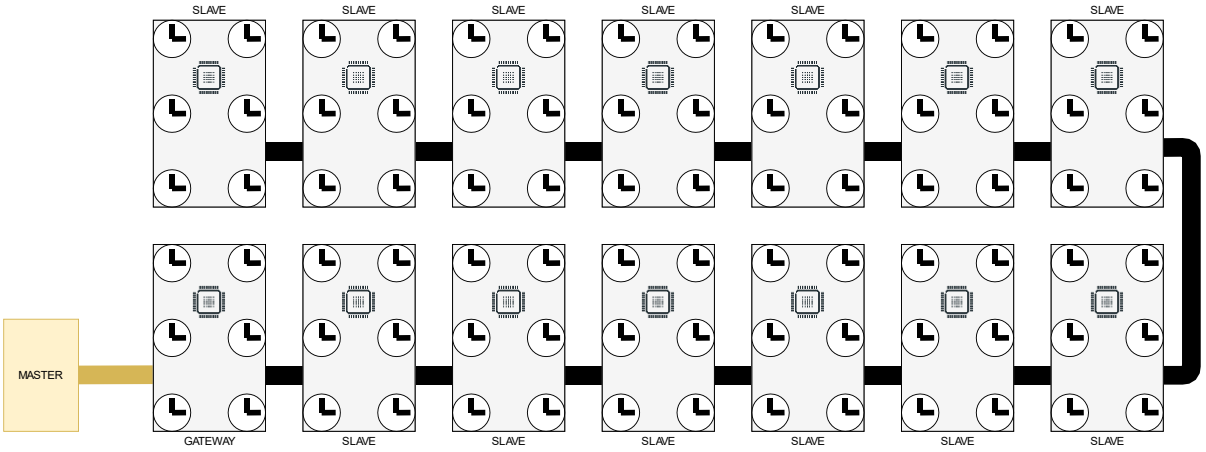


Figure 5 : MASTER – GATEWAY – SLAVE block diagram

Here is what I need to define first:

- Define a MASTER module
- Define a communication protocol between the MASTER and the GATEWAY
- Define a processor for GATEWAY and SLAVE. The goal is to build a PCB that can be a GATEWAY or a SLAVE. This also implies the choice of the electronic components.
- Define a communication protocol between the GATEWAY and the SLAVES

1.2.1 V1.0 vs V2.0

Two versions will be produced. I chose to separate in two versions, because of the second version, a larger quantity of PCBs and components will have to be ordered to make the final matrix. So we have to avoid ordering unnecessary components.

V1.0 objectives	V2.0 objectives
Test and validate the MASTER - GATEWAY communication	Remove unnecessary components
Test and validate GATEWAY - SLAVE communication	Reduce manufacturing costs
Testing and validating movements driving	Build the final matrix (see figure 70 to 72)

Table 1: Versioning objectives

1.3 Planning

A planning of the different tasks is imperative to maintain a fixed deadline.

I had 13 weeks to complete this work.

Here are the most important parts of my schedule:

- Hardware
  - Study the movement datasheet and look for the electronic components necessary to ensure proper operation
  - Schematic design, routing design and place orders
  - Develop and test the PCB V1.0
  - Develop and test the PCB V2.0
- Software
  - Develop code MASTER
  - Develop code GATEWAY/SLAVE
  - Develop code to drive movements
  - Develop code for Bluetooth communication
  - Develop code for animations
- Mechanic
  - Design and manufacturing the box
  - Design and manufacturing the watch pointers

The entire Gantt chart is attached.

2 RESEARCH & DEVELOPMENT

2.1 MASTER selection

The MASTER must give the necessary information to control the entire watch pointers matrix. It will have to:

- Communicate via Bluetooth with the user's smartphone.
- Communicate with the GATEWAY.

To carry out these tasks, I turned directly to the Raspberry Pi because I've worked with this module before. It's known at school and it allows you to work with different communication protocols like I2C, SPI or UART and has a Bluetooth chip. It also offers us an OS and a powerful libraries like Qt or a complete Java Python and whatever language/library ecosystem.

There are several models of raspberry [2]. The most interesting are:

- Raspberry Pi 4 B 4GB because it's the most powerful and it use the Bluetooth 5.0. (60$)
- Raspberry Pi Zero HW because it's the cheapest. But it use the Bluetooth 4.1 (15$)

For the development of V1.0, which uses a GUI application, so I am using the Raspberry Pi 4 to reduce compilation time and be more efficient. We will finally use the **Raspberry Pi Zero** [3] because its price is cheaper and its power corresponds perfectly to our needs. In case we want to market the product, the cheapest option must be necessary. For example, we can make the creation of an electronic board specific to our needs.

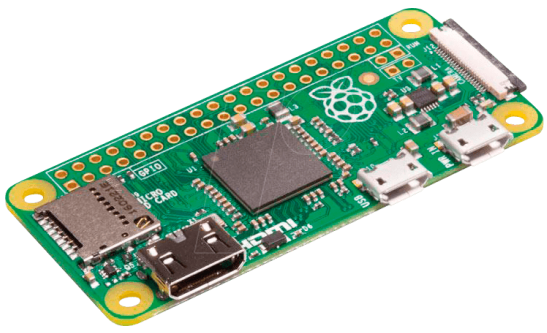


Figure 6 : Raspberry Pi Zero

Figure 7 : Raspberry Pi 4

2.2 Communication protocol for MASTER – GATEWAY

As said before, the Raspberry Pi gives us the possibility to communicate with three well-known protocols:

- SPI
- I2C
- UART

I have no constraints in the choice of protocols.

Here is a very clear description of each protocol [4].

I made the choice to use the **SPI communication protocol** between the Raspberry Pi and the GATEWAY because:

- I've already used it on other projects and never had a problem.
- It's faster than the other protocols. (UART: 115200 baud, I2C: 100kHz, SPI: 500kHz to 32MHz).

2.3 SPI protocol description

In order to achieve clear and reliable communication between the MASTER and the GATEWAY, its protocol must be defined beforehand.

Note that with the SPI protocol, there is no specific byte for the length of data to be sent. Therefore, this byte must not be forgotten.

For SPI communication, according to my needs, I decided to define with four types of frame:

1. Data frame: used for sending data.
2. Configuration frame: used to configure the settings of one processor.
3. Broadcast configuration frame: used to configure the settings of all processors.
4. Reset position zero frame: used to update the zero position.

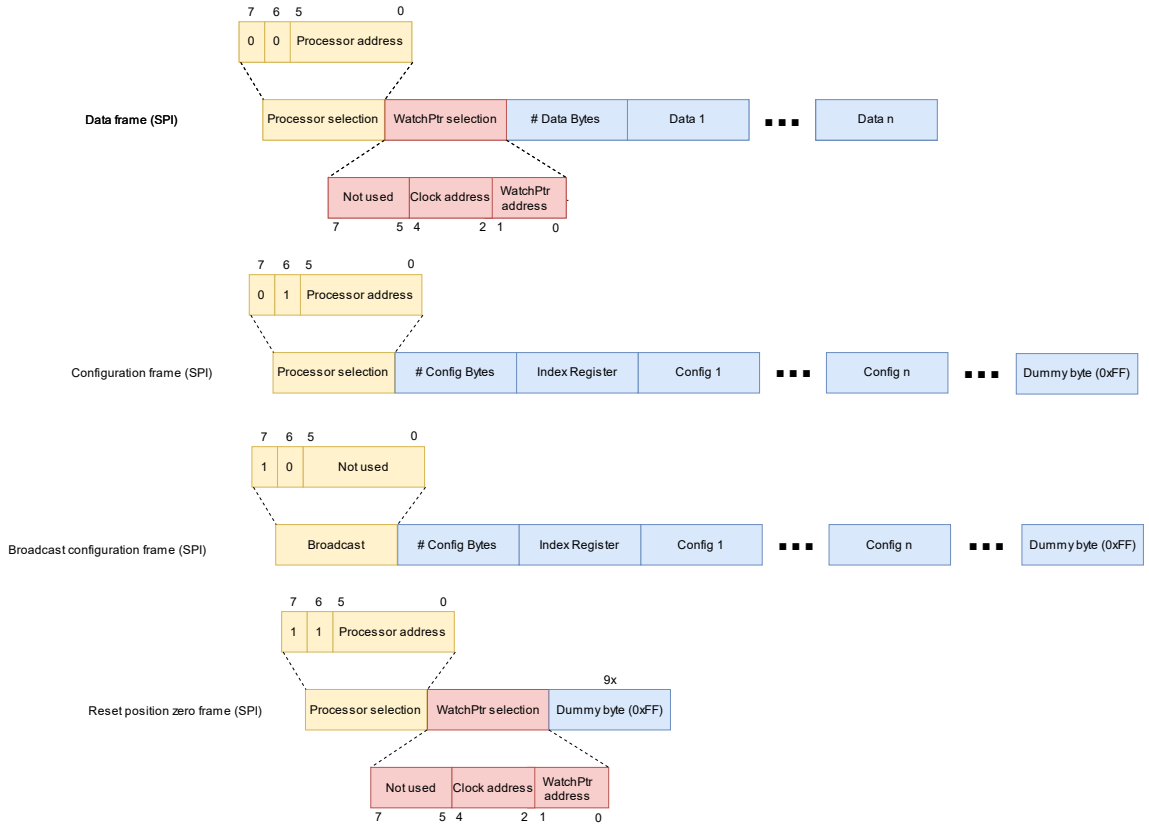


Figure 8 : SPI frames

Maximum frame calculation: 1 byte, processor selection + 1 byte, watch pointer selection + 1 byte, number of useful bytes + 8 bytes, limited by the CAN protocol, explained at point 2.5. This makes a total of **11** bytes.

In order to have frames of well-defined lengths (11 bytes), I use "dummy bytes" (0xFF).
We also need to define a registry table to properly structure the information to be sent and/or received.

Below (see table 2) is the registry table I have defined. Note that some parts have been defined but due to lack of time have not been implemented.

Comments:

- I limited myself to 8 bytes for the global or clock configuration, because the CAN protocol can send a maximum of 8 bytes per frame. This way the communication will be optimized.
- There are 8 bytes of configuration per processor and 8 bytes of information for each movement.
- There is two way to start the animation
    1) With the trigger bit
    2) With the start time (Not implemented)
- The information implemented for the movements is the “New Position” and the “Clockwise”. The rest is not yet implemented.

The free registers will allow me to adjust the useful registers if needed for an improved version.

GROUP	ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	DESCRIPTION
Configuration	0x00			reset time	trigger	connected	in progress	waiting	ready	Status	Status register used to communicate with the smartphone via Bluetooth
	0x01	Time [s]								Start time	Start timer of animation for all hands of each clocks
	0x02	Time [s]								Stop time	Stop timer of animation for all hands of each clocks (relative time)
	0x03			clock 6	clock 5	clock 4	clock 3	clock 2	clock 1	# of watch pointer	Number of watch pointer per clock (0 = 2 watch pointers, 1 = 3 watch pointers)
	0x04									free	
	0x05									free	
	0x06									free	
	0x07									free	
	0x08	/	/	/	/	/	/	/	/	Do not use	/
	0x09	/	/	/	/	/	/	/	/	Do not use	/
Clock n	0xn0	Clockwise 2	# turns 2			Clockwise 1	# turns 1			Config. 1 & 2	Configuration for watch pointer n°1 & n°2
	0xn1	New position 1								Position 1	Set new position for watch pointer n° 1 (# of steps for final position)
	0xn2	Movement Duration Time 1 [s]				Time Offset Start Time 1 [s]				Timing 1	Set time of the movement & offset time for the begining of the animation for watch pointer n° 1
	0xn3	New position 2								Position 2	Set new position for watch pointer n° 2 (# of steps for final position)
	0xn4	Movement Duration Time 2 [s]				Time Offset Start Time 2 [s]				Timing 2	Set time of the movement & offset time for the begining of the animation for watch pointer n° 2
	0xn5	free				Clockwise 3	# turns 3			Config. 3	Configuration for watch pointer n°3
	0xn6	New position 3								Position 3	Set new position for watch pointer n° 3 (# of steps for final position)
	0xn7	Movement Duration Time 3 [s]				Time Offset Start Time 3 [s]				Timing 3	Set time of the movement & offset time for the begining of the animation for watch pointer n° 3
	0xn8	/	/	/	/	/	/	/	/	Do not use	/
	0xn9	/	/	/	/	/	/	/	/	Do not use	/
	0xnA	/	/	/	/	/	/	/	/	Do not use	/
	0xnB	/	/	/	/	/	/	/	/	Do not use	/
	0xnC	/	/	/	/	/	/	/	/	Do not use	/
	0xnD	/	/	/	/	/	/	/	/	Do not use	/
	0xnE	/	/	/	/	/	/	/	/	Do not use	/
	0xnF	/	/	/	/	/	/	/	/	Do not use	/

Table 2: Registry

2.4 Communication protocol for GATEWAY – SLAVE

The main challenge for communication between all SLAVES is to ensure a communication protocol that is reliable and robust regardless of the size of the matrix.

The most commonly used communication protocols for embedded systems are I2C, SPI, UART and CAN. The following is a description of each communication protocol.

I will therefore look at the advantages and disadvantages of each in order to make the best choice.

2.4.1 *I2C*

The I2C (Inter-Integrated Circuit) communication we're going to use is very standard.

✓	✖
Relatively easy to set up communication	The length of the tracks may cause problems
Addressing to a specific processor possible	Reliable in theory but not in practice

Table 3: I2C advantages/disadvantages

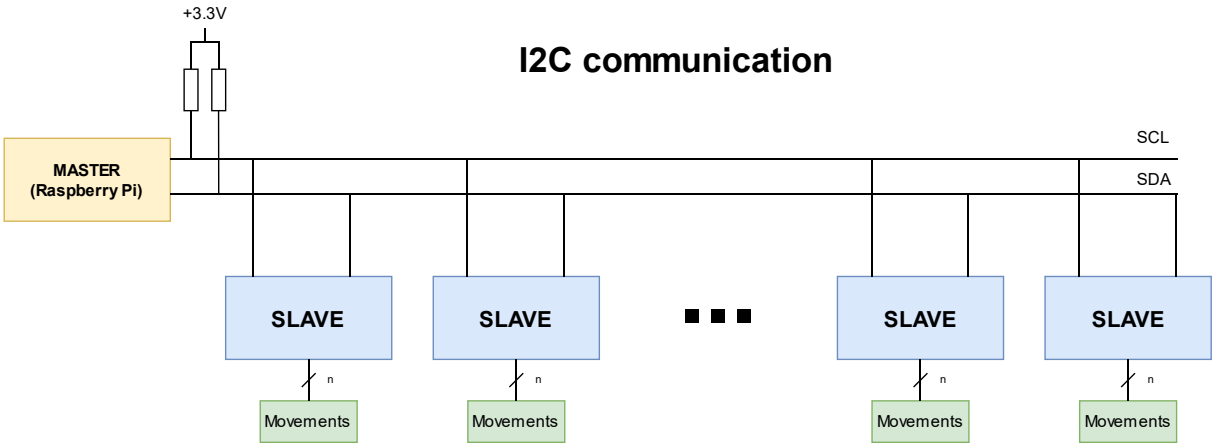


Figure 9 : I2C communication schema bloc

2.4.2 SPI

The SPI (Serial Peripheral Interface) communication will be different from the original version. Here each processor is the MASTER of the processor and the slave of the previous processor.

✓	✘
Same code for all processors	Processor specific addressing not possible
High communication speed	Complexity of the code he tracks may cause problems
No risk of track length, because the signal is generated again by the processor	

Table 4: SPI advantages/disadvantages

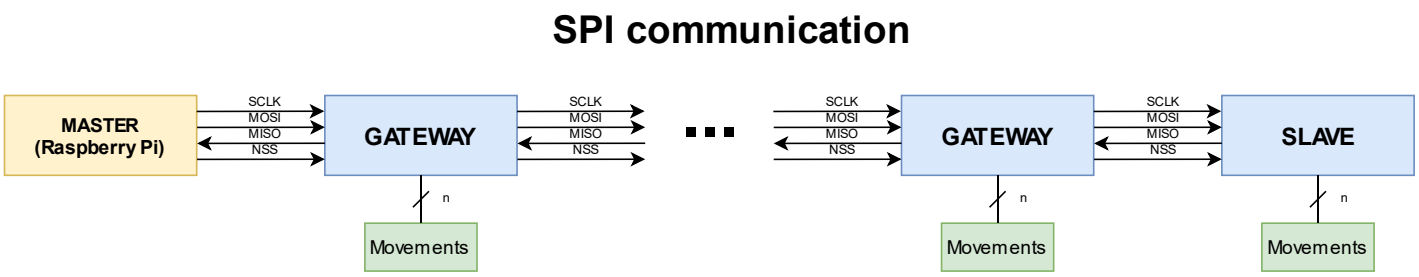


Figure 10 : SPI communication schema bloc

2.4.3 UART

The UART (Universal Asynchronous Receiver Transmitter) communication is mainly used for point-to-point communication between two devices.

Now in our use we have master and a multitude of SLAVES. We have to make an electrical arrangement to network via UART[5]. This way a communication similar to I2C is possible.

✓	✘
Better supported track lengths than for I2C	Unreliable
Relatively easy to set up communication	Slave-master communication more complicated if several slaves speak at the same time.
Addressing to a specific processor possible	Electrical arrangement

Table 5: UART advantages/disadvantages

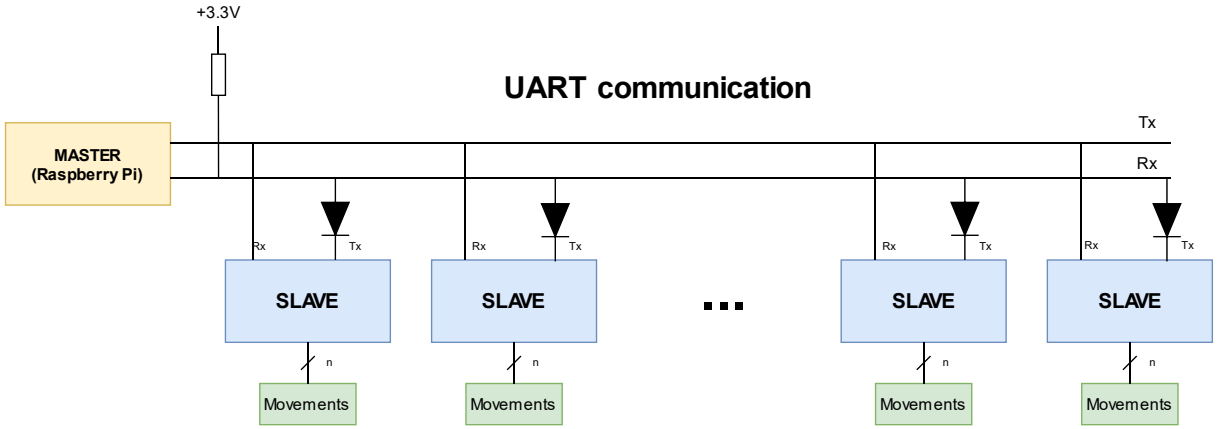


Figure 11 : UART communication schema bloc

2.4.4 CAN

CAN (Controller Area Network) is used for a network of microcontrollers. It is a serial communication protocol that supports real-time embedded systems with a high level of reliability. That's exactly our case!

✓	✘
Robust	Additional module required for electrical conversion → increase the price
Flexible	
Efficient	
Two wires	
Master less	
Collision free	
Use for long distances	
Addressing to a specific processor possible	

Table 6: CAN advantages/disadvantages

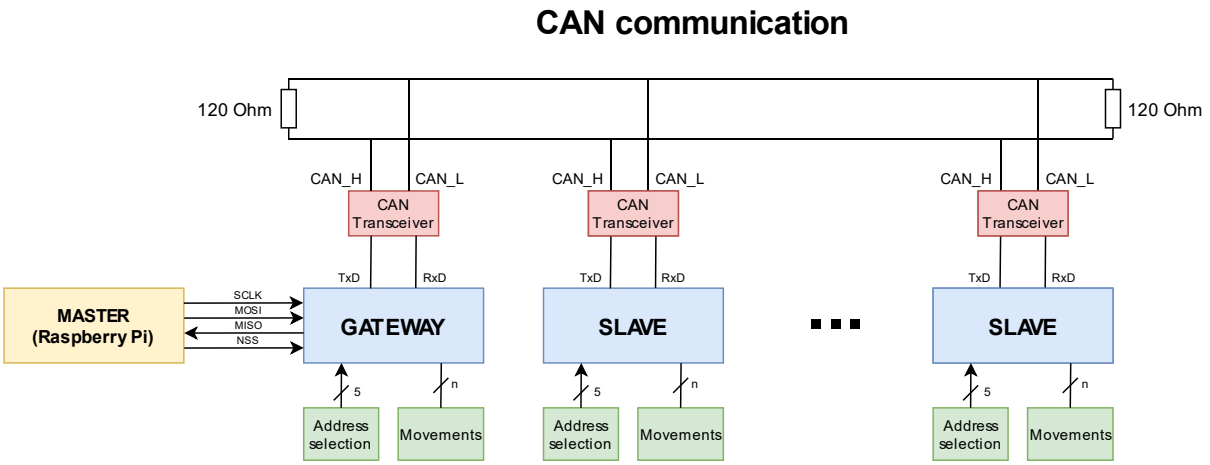


Figure 12 : CAN communication schema bloc

2.4.5 Communication protocol choice?

We have seen the advantages and disadvantages of each communication protocol.

We can clearly see that the CAN protocol has most of the advantages. The main advantages are the robustness, the use for long distances, the possibility to address a specific processor and the ease of installation.

In conclusion, the **CAN communication protocol** corresponds best to our needs.

2.5 CAN protocol description

The CAN protocol is a protocol with a well-defined frame format that must be strictly adhered to.

Here is the CAN frame format:

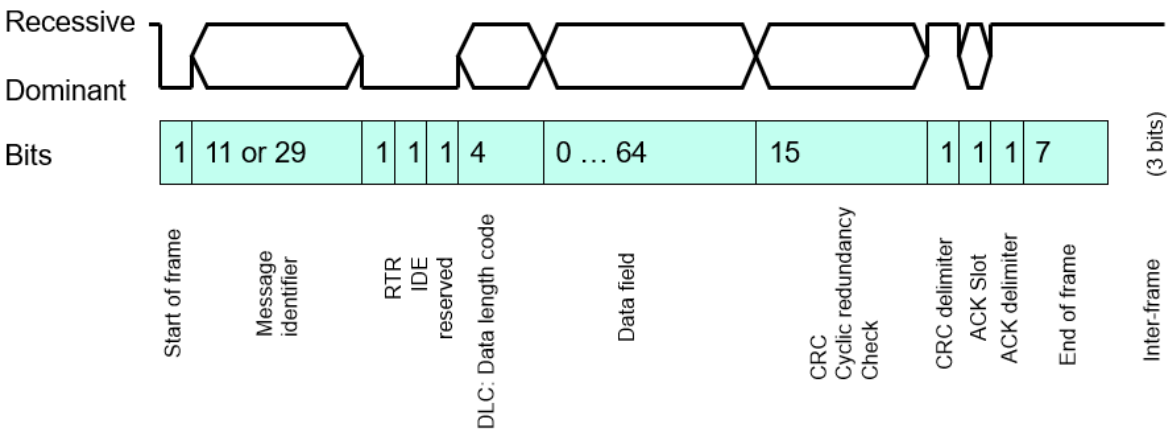


Figure 13 : CAN frame format

Source: Dominique Gabioud – Sin 221/231 CAN Theory

1. Start of Frame (SOF)
    - Single bit, always dominant state ("0"), Hard Sync on SOF edge
2. Message Identifier
    - 11 bits → CAN 2.0-A
    - 29 bits → CAN 2.0-B
    - Lowest ID has the highest priority
3. Remote Transmission Request (RTR)
    - Single bit, "1" for a request in Client-Server Model
4. Identifier Extension (IDE)
    - Single bit, "0" in CAN 2.0-A (11 bits), "1" in CAN 2.0-B (29 bits)
5. Data Length Code (DLC)
    - 4 bits, but 0 to 8 allowed.
    - Specifies the length of the data field. Max 8 can be sent
6. Data Field
    - Content freely managed by application designer
    - All actively nodes must be capable to interpret its content

7. Cyclic Redundancy Check (CRC)
    - 15 bits, all receiving nodes will use it to perform error detection
8. CRC Delimiter
    - Single bit, always recessive state ("1"), give times for receiver to perform CRC check
9. Acknowledge Slot (ACK)
    - Set to dominant ("0") by each receiver detecting a valid frame
    - Transmitter "receives" an ACK, if at least one receivers had a positive CRC check
10. ACK Delimiter
    - 1 bit, always recessive ("1")
11. End of Frame
    - 7 bits, all recessive ("1")
12. Inter Frame
    - Recessive ("1") for at least 3 bits

We have seen that the CAN frame limits to 8 bytes the number of useful data that can be transmitted.
That's why I've limited to 8 bytes in the registers, both for the configuration registers and the registers specific to each clock.

I'm using an 11-bit identifier message (see figure 14). This is enough to perfectly determine the frame used as well as the processor, clock and watch pointer used.

- Bit 0-1: watch pointer address (max 3 watch pointers)
- Bit 2-4: clock address (max 6 clocks)
- Bit 5-8: processor address (max 14 processor)
- Bit 9-10: frame type (4 different frame)

In this way, the identifier and useful data part is well broken down.

It should not be forgotten that the role of the GATEWAY is to translate the information received via SPI to send it to the SLAVES via CAN. As the frames are defined on the same principle as for the SPI protocol (see figure 8), translation is greatly simplified.

Here are the defined CAN frames:

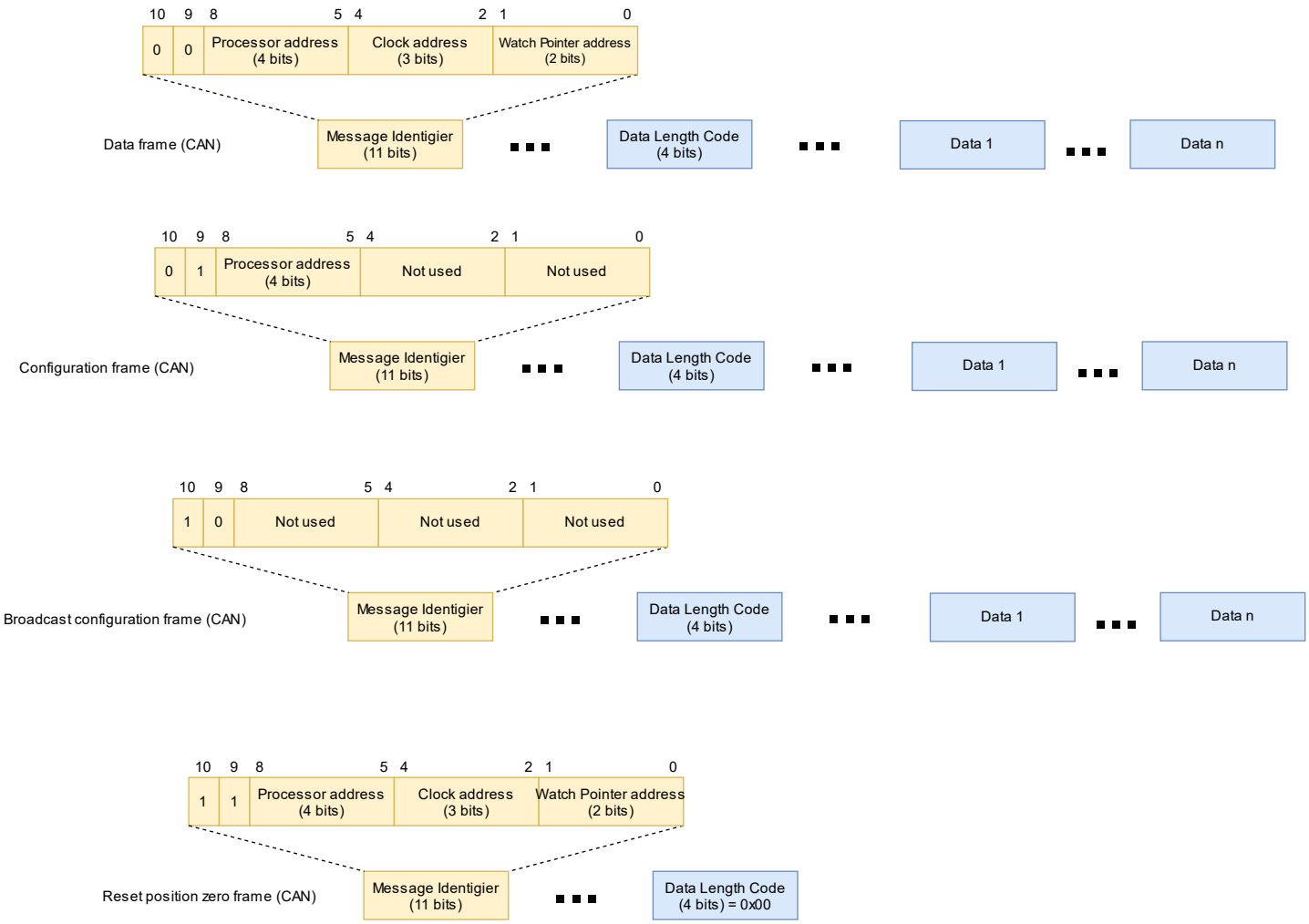


Figure 14 : CAN frames

2.6 GATEWAY & SLAVE processor selection

Regarding the choice of processor, we must ask ourselves one question.
Choosing a processor to drive only one movement? Or choose a processor to drive multiple movements?

The choice is very quickly made because here are all the advantages of working with a processor used to drive multiple movements.

One processor to drive **multiple** movements advantages
More powerful processor
More developed development environment (CubeMX)
Less processor to be programmed for the final product
Processor used in laboratory
Better prices
Memory size available in larger format

Table 7: One processor to drive multiple movements advantages

All these advantages are the disadvantages of using a processor to drive a movement.

I've set the number of movements per processor at six. Because it's a good compromise between the number of GPIOs, the power of the processor and its price. I will focus on the STM32 family because it is known at school and used in the laboratory.

2.6.1 Processor – requirements

Requirements	Comments
SPI protocol	Used to communicate with the MASTER
CAN protocol	Used to communicate with the GATEWAY or SLAVES
54 GPIOs	Used to drive the motors. We have a maximum of six motors with three watch pointers each. Each watch pointer need 3 GPIOs.

Table 8: Processor requirements

2.6.2 STM32 families – all categories

There are several categories of STM32[6]:

- High Performance
- Mainstream
- Ultra-low-power
- Wireless

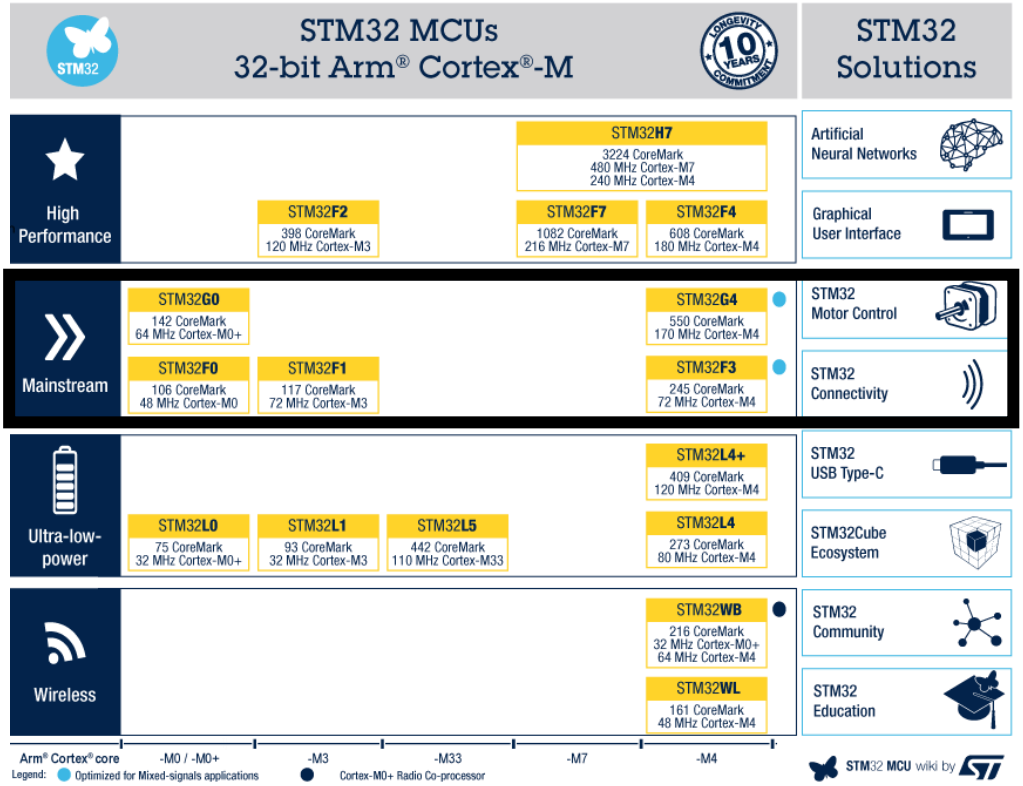


Figure 15 : STM32 families

For our project, the "**mainstream**" category is more than sufficient.

2.6.3 *STM32 families – mainstream category*

In the STM32 families - mainstream category[7] we can find several different series:

- STM32F0 Series
- STM32G0 Series
- STM32F1 Series
- STM32F3 Series
- STM32G4 Series

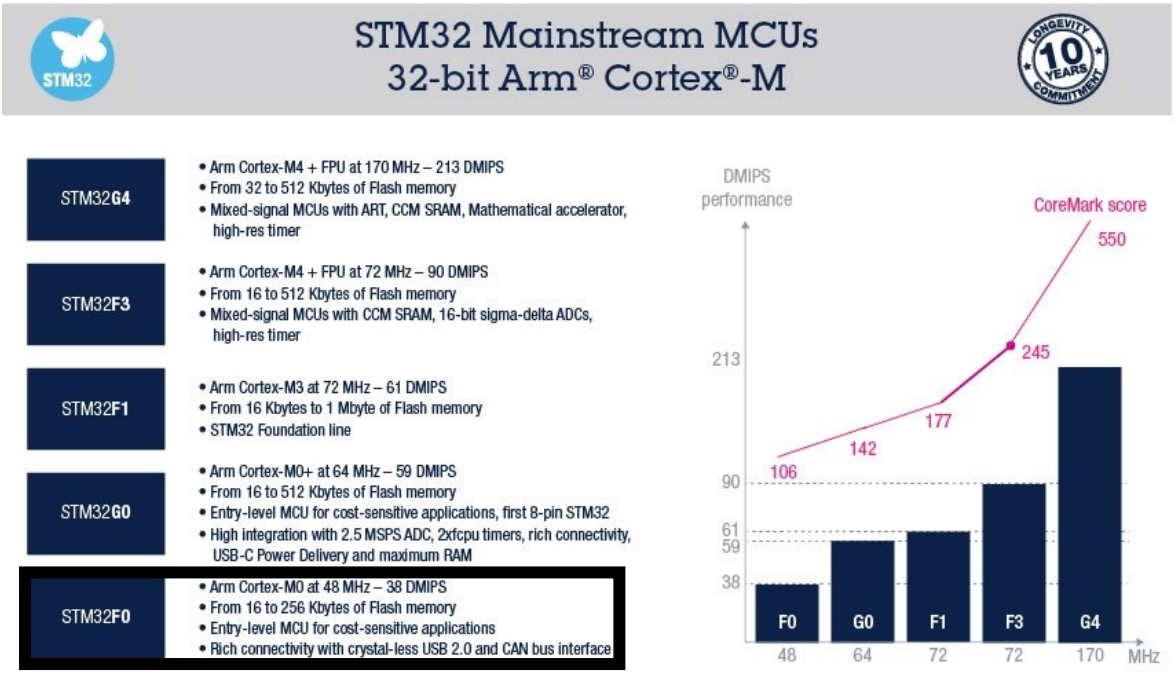


Figure 16 : STM32 families – mainstream category

(DMIPS = Dhrystone Million Instructions Per Second)

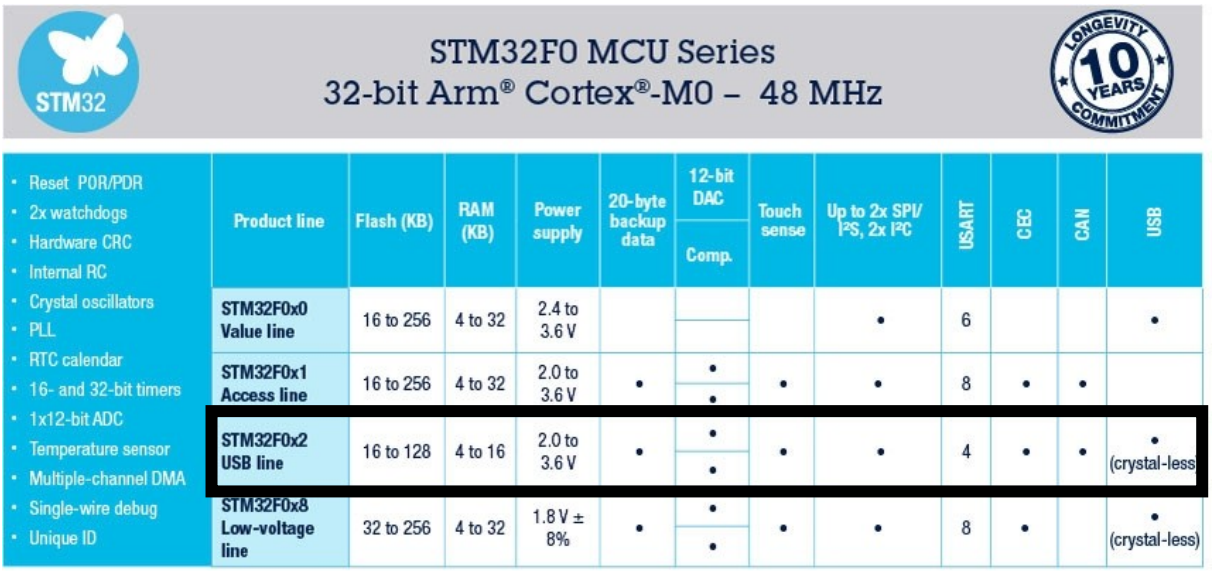
Only the STM32F0 Series provides a CAN bus interface. The other series are too powerful processors for our needs, and therefore too expensive.

So I'm going to focus on STM32F0 Series.

2.6.4 STM32F0 Series

The STM32F0 series[8] contains:

- STM32F0x0 Value line
- STM32F0x1 line
- STM32F0x2 line
- STM32F0x8-line



STM32F0 MCU Series
32-bit Arm® Cortex®-M0 – 48 MHz

- Reset POR/PDR
- 2x watchdogs
- Hardware CRC
- Internal RC
- Crystal oscillators
- PLL
- RTC calendar
- 16- and 32-bit timers
- 1x12-bit ADC
- Temperature sensor
- Multiple-channel DMA
- Single-wire debug
- Unique ID

Product line	Flash (KB)	RAM (KB)	Power supply	20-byte backup data	12-bit DAC / Comp.	Touch sense	Up to 2x SPI/ I²S, 2x I²C	USART	CEC	CAN	USB
STM32F0x0 Value line	16 to 256	4 to 32	2.4 to 3.6 V				•	6			•
STM32F0x1 Access line	16 to 256	4 to 32	2.0 to 3.6 V	•	• •	•	•	8	•	•	
STM32F0x2 USB line	16 to 128	4 to 16	2.0 to 3.6 V	•	• •	•	•	4	•	•	• (crystal-less)
STM32F0x8 Low-voltage line	32 to 256	4 to 32	1.8 V ± 8%	•	• •	•	•	8	•		• (crystal-less)

Figure 17 : STM32F0 series

Only the STM32F0x2 line[9] provides a CAN bus interface (ideal choice for communication gateway). So this line is perfectly suited for our use because we need a CAN bus interface.

PS: The CAN part shown in the table does not correspond to the CAN bus interface but it means analog-to-digital converter.

As we have to take a processor with more than 64 pins, to have a sufficient number of GPIOs, only the STM32F072Vx corresponds to our needs.

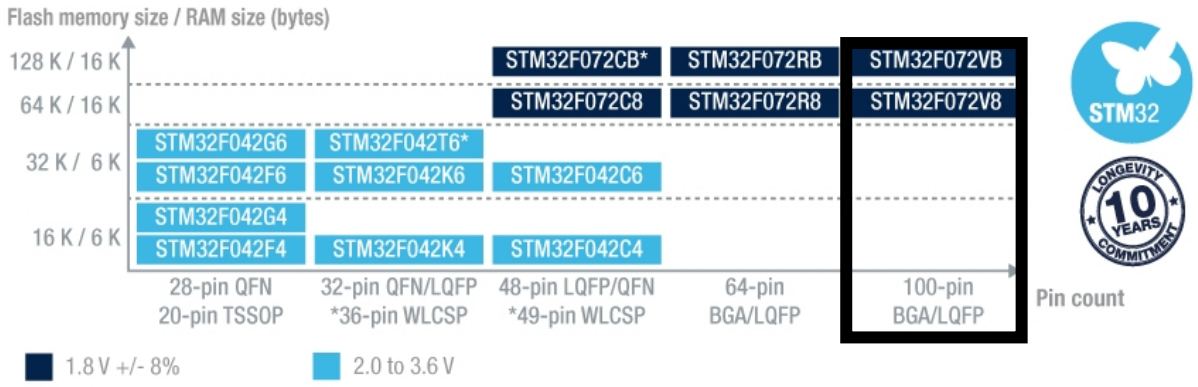


Figure 18 : STM32F0x2 line

So I found this processor: **STM32F072V8T6**[10]:

- Price : 3,82CHF / 1 unit
- Price : 3,24CHF / 10 units
- Package : LQFP-100
- Clock frequency : 48 MHz
- Memory size : 64 kB
- RAM size : 16 kB
- Number of I/O : 87
- I2C/SPI/UART/CAN
- Supply voltage: 2V to 3.6V

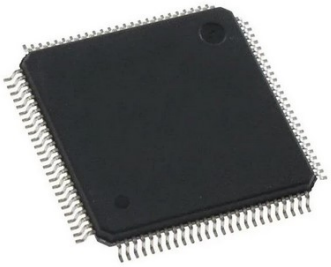


Figure 19 : STM32F072V8T6

2.6.5 Requirements summary

Requirements	Available ?
SPI protocol	✓
CAN protocol	✓
54 GPIOs	✓

Table 9: Processor requirements summary

This processor fully meets the specifications.

2.7 Altium schematic

Concerning the electronic part, I have to do the schematic in order to:

Requirements
Be able to debug with the STLINK V3SET
Have the possibility to power the matrix either through the Raspberry Pi or through an external power supply
Assign an address to each processor
Be able to connect the raspberry pi with the appropriate connector
Use the processor correctly, i.e. think about the decoupling capacitor, the reset, the external oscillator
Use the electrical converter used for CAN communication
Use connectors to connect PCBs to each other

Table 10: Schematic requirements

Here is a block diagram that includes all the parts present in the schematic:

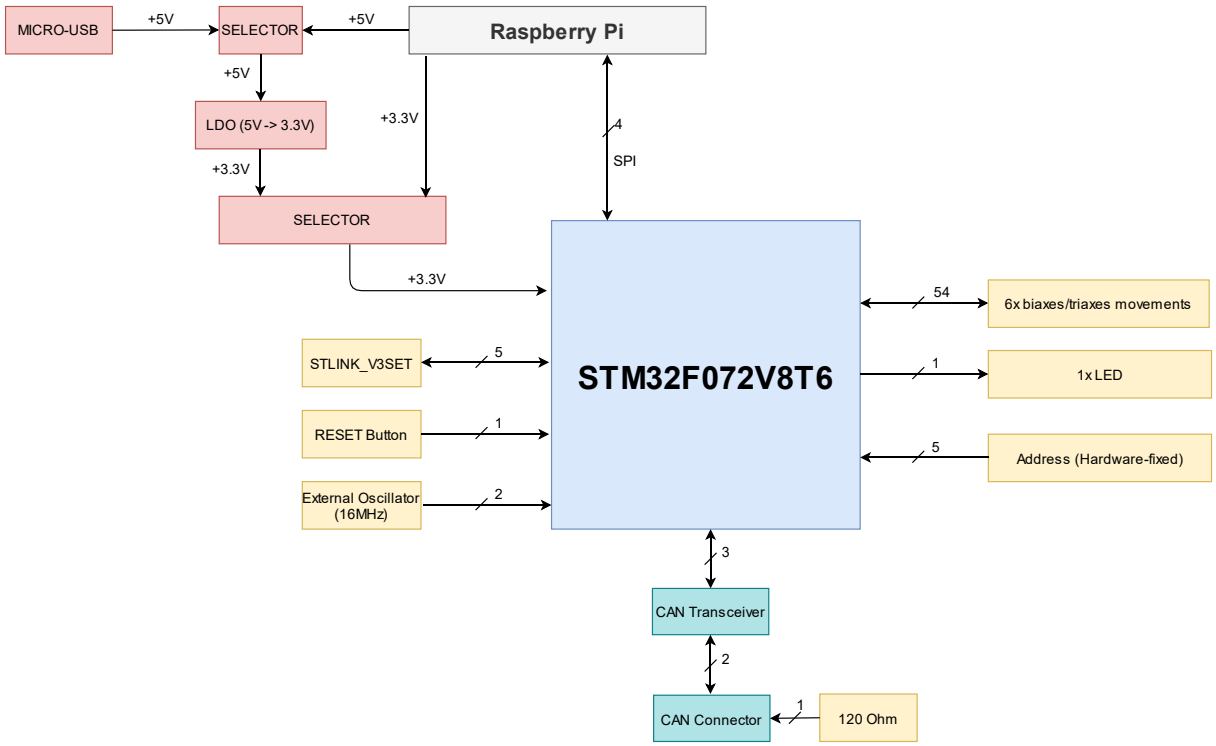


Figure 20 : Schema bloc of the schematic

The entire schematic is attached.

2.7.1 STLINK-V3SET debugger

The STLINK-V3SET is a stand-alone modular debugging and programming probe for the STM8 and STM32 microcontrollers.

It is used for programming and debugging via SWD protocol. Two UART signals are used to communicate with the target microcontroller via the Virtual Come port.

Used signals:

- T_SWDIO: data for debug/program
- T_SWCLK: clock for debug/program
- T_NRST: reset
- T_VCP_RX: UART
- T_VCP_TX : UART

Table 6. STDC14 connector pinout CN1

Pin No.	Description	Pin No.	Description
1	Reserved(1)	2	Reserved(1)
3	T_VCC(2)	4	T_JTMS/T_SWDIO
5	GND	6	T_JCLK/T_SWCLK
7	GND	8	T_JTDO/T_SWO(3)
9	T_JRCLK(4)/NC(5)	10	T_JTDI/NC(5)
11	GNDDetect(6)	12	T_NRST
13	T_VCP_RX(7)	14	T_VCP_TX(2)

1. Do not connect on target.
2. Input for STLINK-V3SET.
3. SWO is optional, required only for Serial Wire Viewer (SWV) trace.
4. Optional loopback of T_JCLK on the target side, required if loopback removed on the STLINK-V3SET side.
5. NC means not required for the SWD connection.
6. Connect to GND on target, may be used by STLINK-V3SET for detection of connection.
7. Output for STLINK-V3SET

Figure 21 : STLINK-V3SET pinout

2.7.2 LDO power management

In order to ensure sufficient power for the entire system, we have the choice to supply it either via the raspberry or an external power supply.

In case the required power cannot be delivered by the Raspberry Pi Zero, we use an LDO (Low-DropOut) to maintain a reliable power supply.

The LDO used (LD39200PU33R) delivers a 3.3V/2A, which is sufficient in our case.

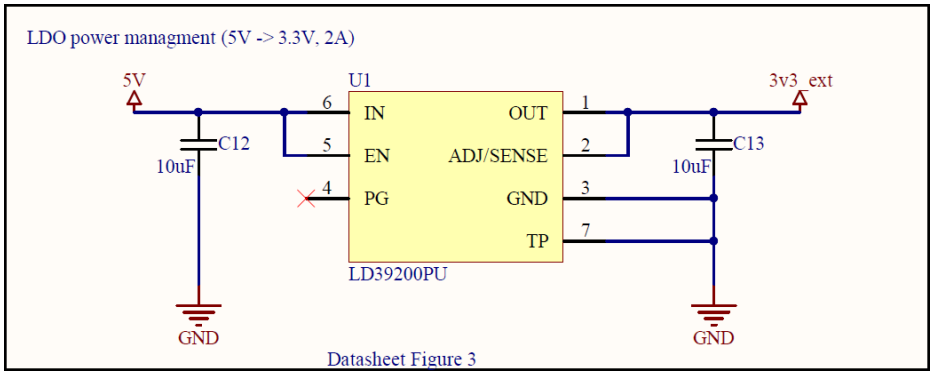


Figure 22 : LDO typical application Fixed version schematic

2.7.3 *Raspberry Pi Zero*

The raspberry pi gives us the possibility to work with:

- I2C (GPIO 2 = SDA, GPIO 3 = SCL)
- **SPI (GPIO 8 = CE0, GPIO 9 = MISO, GPIO 10 = MOSI, GPIO 11 = SCLK)**
- UART (GPIO 14 = TXD, GPIO 15 = RXD)

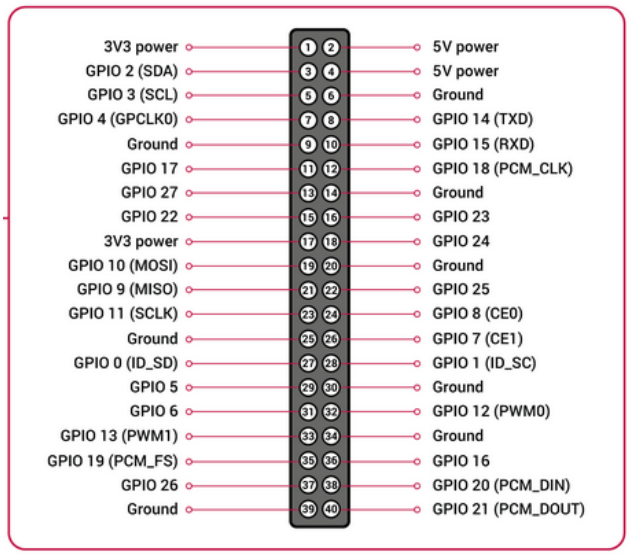


Figure 23 : Raspberry Pi pinout

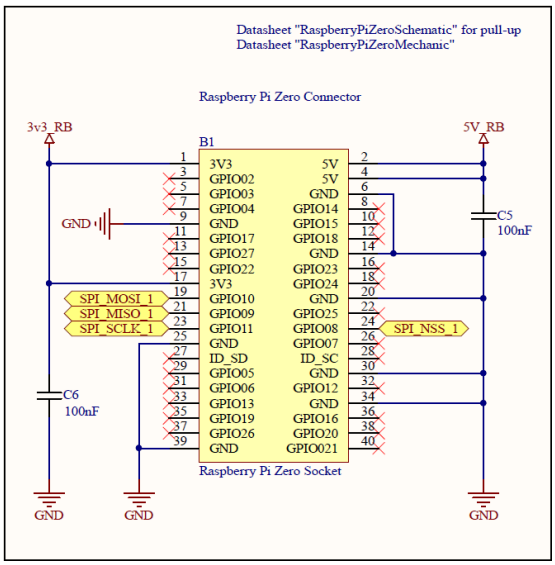


Figure 24 : Raspberry Pi Schematic

As said before, we'll be working with the SPI protocol for MASTER-GATEWAY communication.

2.7.4 *TM32F072V8T6*

We have integrated a reset button to restart the processor if needed.

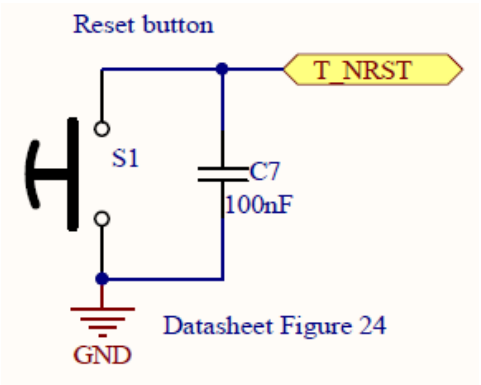


Figure 25 : Recommended external reset button

In case the accuracy of the internal oscillator is not sufficient, we have added an external oscillator to ensure sufficient accuracy.

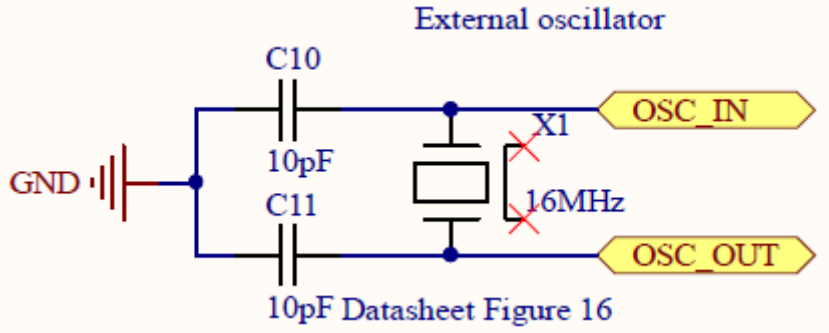


Figure 26 : Typical application external oscillator

We have to be careful about the current delivered by the GPIOs to control the motors. With the worst case, 8mA can be driven by the GPIOs.

The GPIOs (general purpose input/outputs) can sink or source up to +/-8 mA, and sink or source up to +/- 20 mA (with a relaxed V_{OL}/V_{OH}).

Figure 27 : Output driven current of processor

This is sufficient because engines consume in the worst case (peak consumption) 2.5mA/engine.

Motor average consumption (1 step)	I_{mot}	µAs	-	3.8	4.2
Motor peak consumption	I_{peak}	mA	-	2	2.5

Figure 28 : Current consumption of motors

In order to ensure the power supply of the processor, do not forget to add the decoupling capacitors. (See figure 13: "Power supply scheme" of the processor datasheet).

2.7.5 MCP2542

In order to be able to communicate correctly with the CAN protocol, we need to create an electrical signal adaptation. The MCP 2542 module does this automatically.

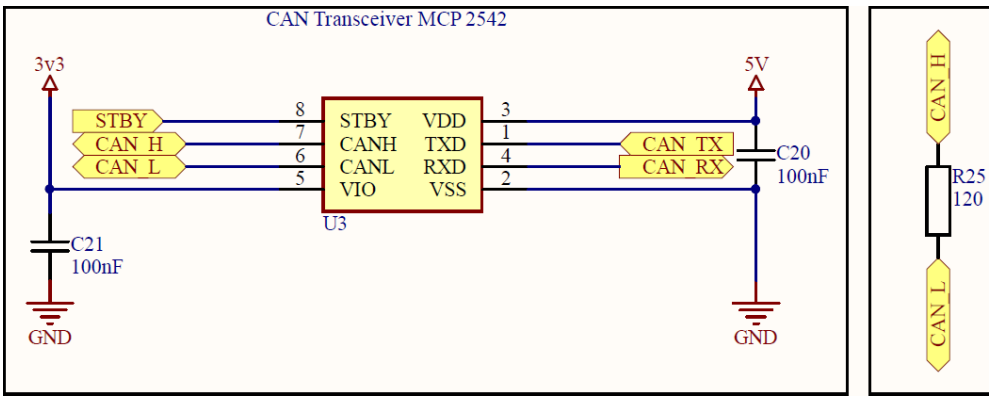


Figure 29 : Typical use of MCP2542

It is important not to forget the 120 Ohm resistor for the start and end node of the CAN bus to ensure electrical operation.

2.7.6 Requirements summary

Requirements	Fulfilled ?
Be able to debug with the STLINK V3SET	✓
Have the possibility to power the matrix either through the Raspberry Pi or through an external power supply	✓
Assign an address to each processor	✓
Be able to connect the raspberry pi with the appropriate connector	✓
Use the processor correctly, i.e. think about the decoupling capacitor, the reset, the external oscillator	✓
Use the electrical converter used for CAN communication	✓
Use connectors to connect PCBs to each other	✓

Table 11: Schematic requirements summary

All requirements are fulfilled.

3 SOFTWARE

3.1 Bluetooth between the two bachelor thesis

As said in the description of the HYPNOSIA project, this one is decomposed into two diploma works. The goal is to create a communication between these two in order to link the two projects.

To do this, we chose to communicate via a well-known protocol, Bluetooth. This protocol is present in many everyday objects, including our smartphones and the Raspberry Pi. Note that the Raspberry Pi will be in this case the slave of the Smartphone. Communication is initially unidirectional (Smartphone → Raspberry Pi).

We transmit the information using a serial socket over Bluetooth. This is the easiest way to communicate between two devices. To structure the information to be sent, we will work with the JSON format. This format is widespread and well known because we have worked with it for some laboratories.

This diagram shows the communication between the two projects.

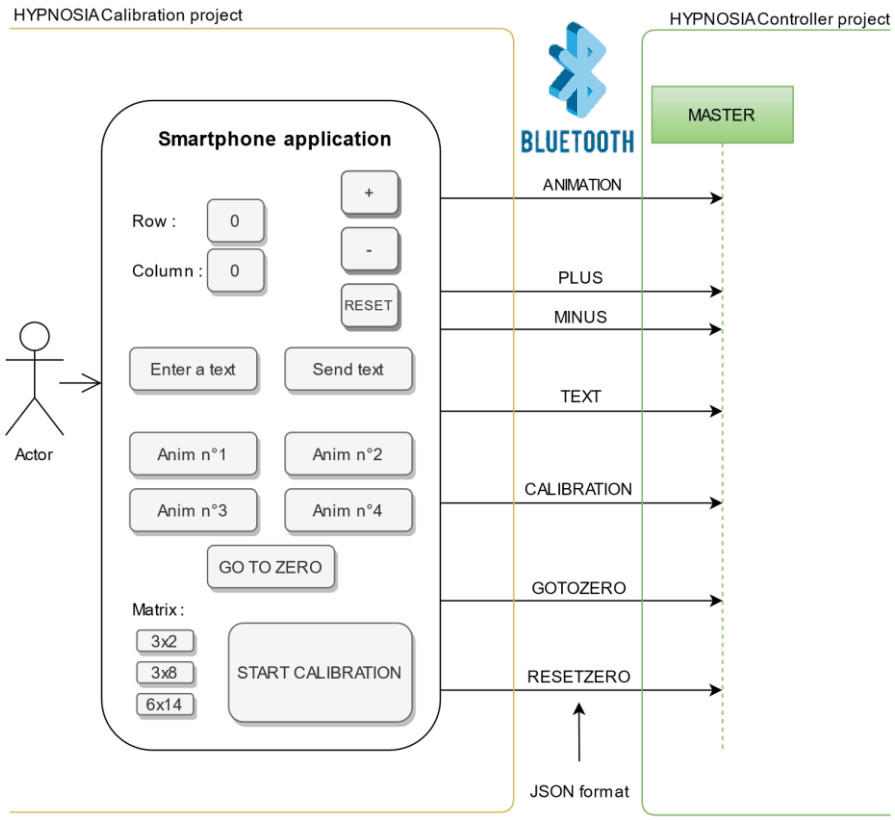


Figure 30 : Communication between "HYPNOSIA Calibration" & "HYPNOSIA Controller" project

3.1.1 JSON file description

In order to make the communication work between two devices, the structure of the JSON file must be precisely defined.

We have defined two key parts.

- "**header**": this will allow to know which action has been selected by the user.
- "**body**": this one will vary according to the "header". Its purpose is to give the information necessary to carry out a given action. It is important to note that with the JSON format, data arrays can be sent. This system makes it possible to efficiently structure the data to be sent. When reading the data, the operation becomes very easy.

Here is an example of a JSON file. All others are attached.

```
{
  "header": "CALIBRATION",
  "body":
  [
    clock_jsonObject n,
    clock_jsonObject n+1,
    ...
    clock_jsonObject n+x
  ]
}


//clock_jsonObject
{
  "processorID" : 0,     //0 to 13
  "clockID" : 0,         //0 to 5
  "watchpointerID" : 0, //0 or 1
  "moveWP1" : 180,       //0 to 359
  "moveWP2" : 90         //0 to 359
}
```

Figure 31 : JSON file example

3.2 Matrix structure

In the chapter "Research and Development", we defined the concept of MASTER-GATEWAY - SLAVE as well as the communication between each of them.

This is what the matrix looks like now:

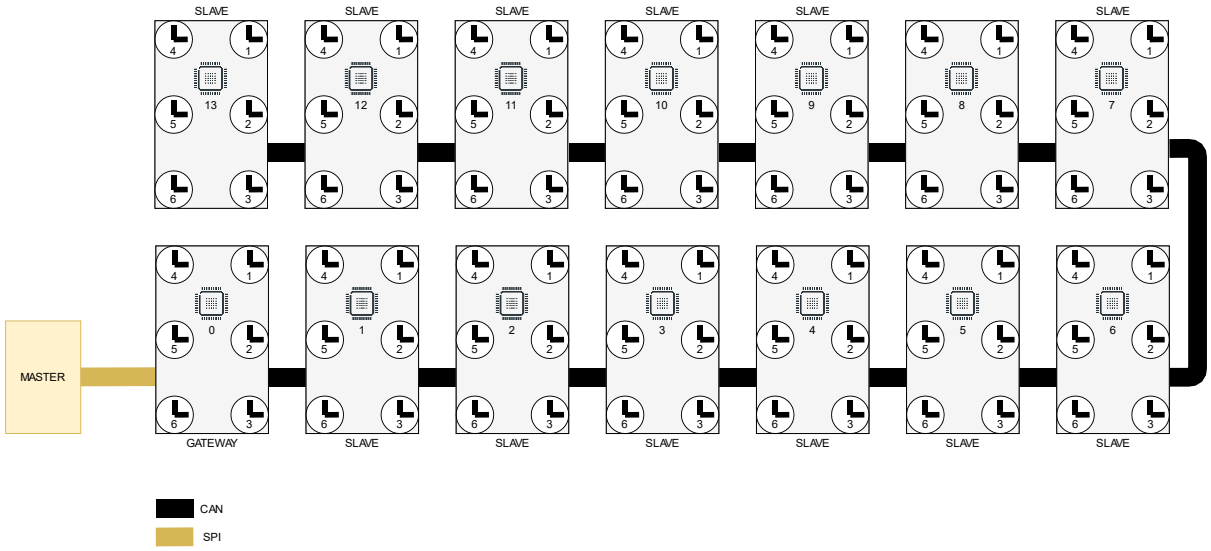


Figure 32 : MASTER – GATEWAY – SLAVE block diagram with used protocol

How it can be seen on the diagram, if we look at the matrix as a whole there is no logical sequence between the index of processors and clocks. So we need to simplify the communication to the watch pointers to make the animations easier to program afterwards. That's why I created a *motor* type array.

Motor is a structure that contains a *processorID* and a *clockID*. This way, we can communicate directly to a single watch pointer because we know which processor and clock it is linked to.

```
struct motor
{
    int processorID;
    int clockID;
};
motor matrix_6x14[6][14];
```

Figure 33 : "motor" structure & "motor" array

3.3 MASTER - Software description

3.3.1 Used programs

Here are the programs used to program the Raspberry Pi:

Tool	Comment
RaspberryPi Imager	Use to image the microSD card with Raspbian
QtCreator	Use to develop the program

Table 12: Used tools for MASTER

3.3.2 Schema bloc

MASTER requirements
Decode the JSON files received via Bluetooth
Send via SPI the data used to manage the watch pointers

Table 13: MASTER requirements

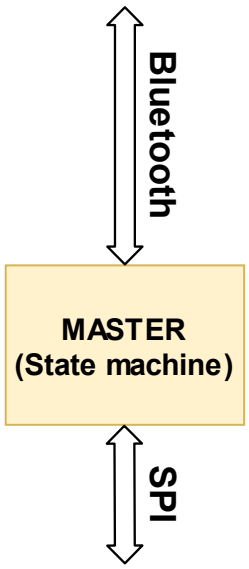


Figure 34 : MASTER block diagram

3.3.3 *Class diagram*

In order to properly structure my code and logic, I chose to use the MVC (Model-View-Controller) pattern because it is perfectly suited to our needs and because I've already used it in the lab.

I also use the Factory pattern which allows me to instantiate all my objects at code start-up.

It is very easy to use the MVC pattern. This pattern is composed of three parts:

- **Model** (here *Data*, *Processor*, *Clock* and *WatchPointer)* these classes contain all the data and the logic related to this data
- **View** (here *View* and *Bluetooth*) these classes contain all the graphical interface of the application
- **Controller** (here *Controller*) this class processes the user's actions, modifies the model and view data

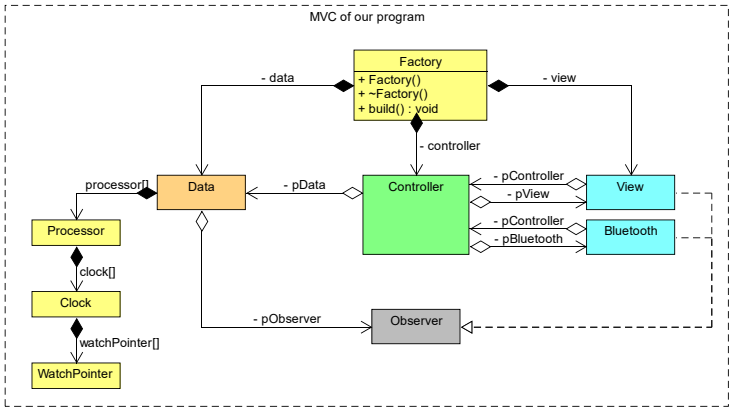


Figure 35 : MVC pattern, simplified class diagram

Here we have two classes that manage the View part. This is due to the fact that before creating the Bluetooth link between the smartphone application and the Raspberry Pi (V2.0), I created a GUI application that does exactly the same thing (V1.0). This allowed me not to depend on my colleague who developed the smartphone application.

The role of the View part is to generate events to run the main state machine present in the *Controller* class.

3.3.4 State machine

As said before, there is a main state machine that manages the data.

All the events that allow to change state are generated following an instruction given by the View part.

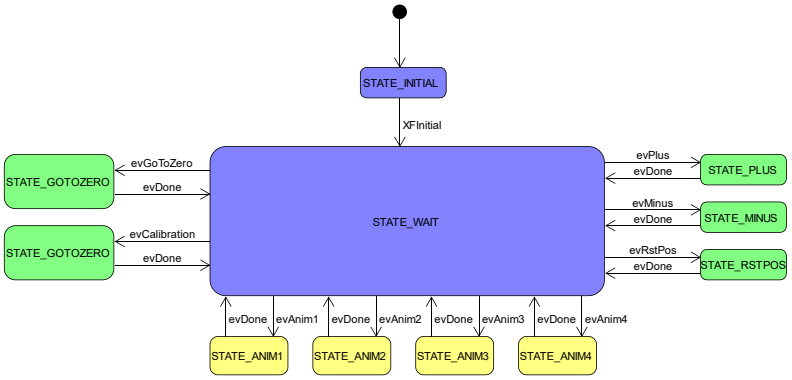


Figure 36 : MASTER main state machine

For each animation, there is actually a very simple state machine.

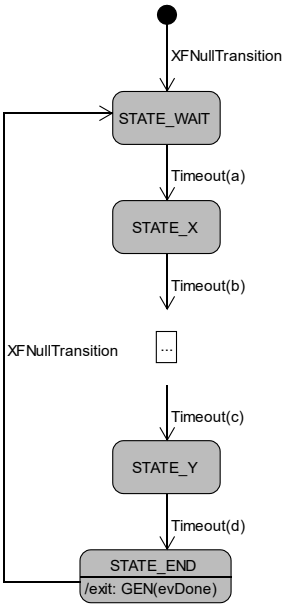


Figure 37 : MASTER animation state machine

All state machines dealing with animations follow the same pattern. Timeout allows to run the animation.

To manage all state machines, I use a nano OS (Operating System), an XF (eXecution Framework). The one I use is developed within the HEI. The description of this one is made below (see point 3.5).

3.3.5 *Flow chart*

Here is how the program behaves when the user clicks on a smartphone button.

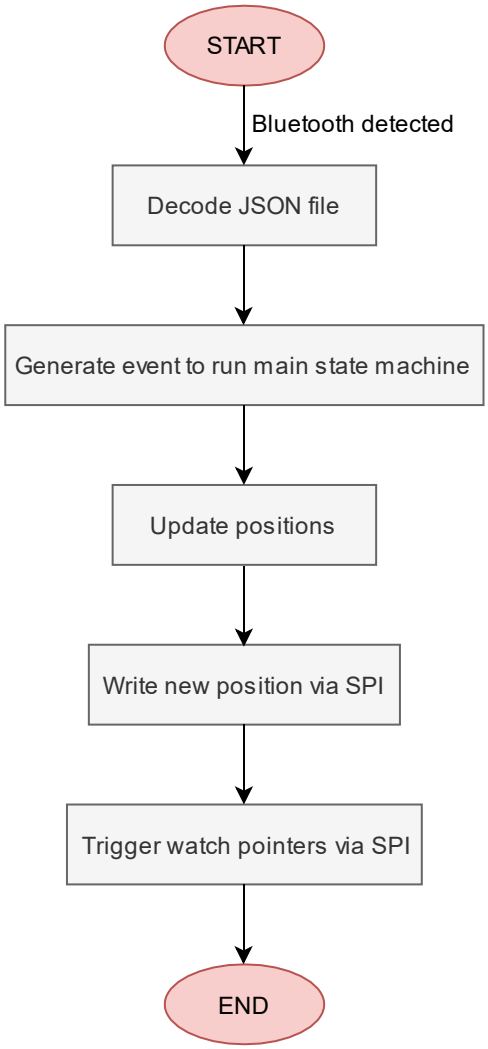


Figure 38 : Flow chart, program behaviour

3.3.6 *Sequence diagrams*

The sequence diagram below shows how the program reacts when the user selects an animation on the application.

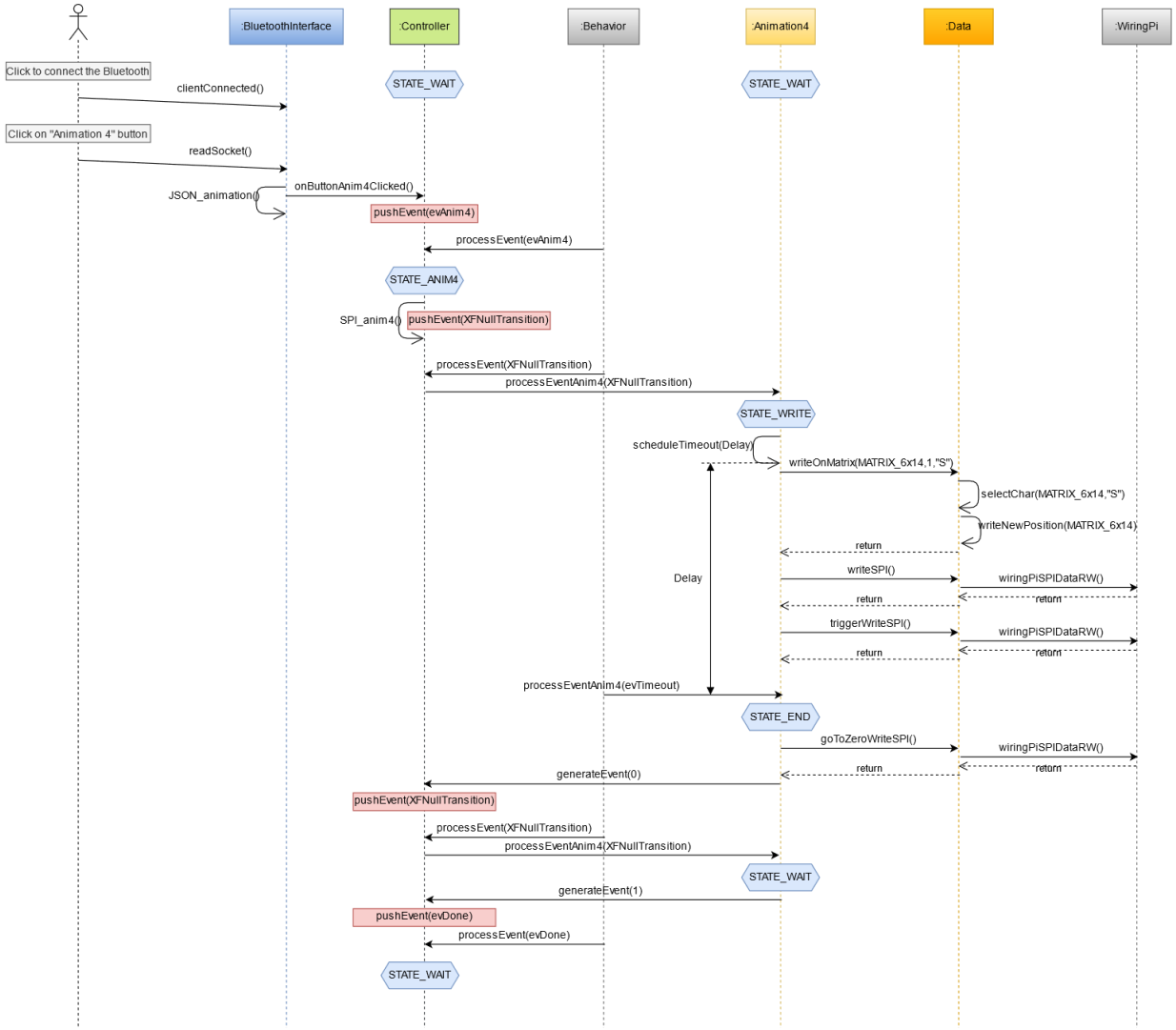


Figure 39 : Sequence diagram when choosing an animation on the smartphone

3.4 GATEWAY & SLAVE - Software description

3.4.1 Used programs

Here are the programs used to program the microprocessors:

Tool	Comment
STM32 CubeIDE	Use to program the STM32
STM32 CubeMX	Use to configure the processor pins
STM32 CubeProgrammer	Use to update the ST-Link

Table 14: Used tools for GATEWAY/SLAVE

3.4.2 Schema bloc

GATEWAY requirements
Detect SPI frames sent by the MASTER
Sent via CAN on the bus if SPI frames are not intended for it
Manage these own movements if SPI frames are intended for it

Table 15: GATEWAY requirements

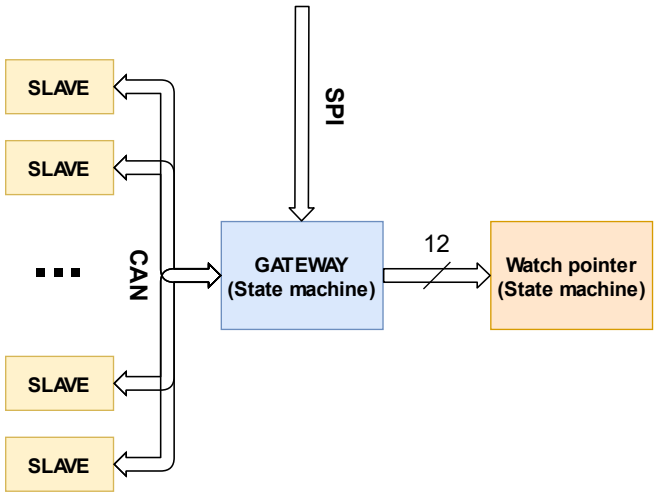


Figure 40 : GATEWAY block diagram

SLAVE requirements
Decode CAN frames intended for him
Manage these own movements according to the detected CAN frame

Table 16: SLAVE requirements

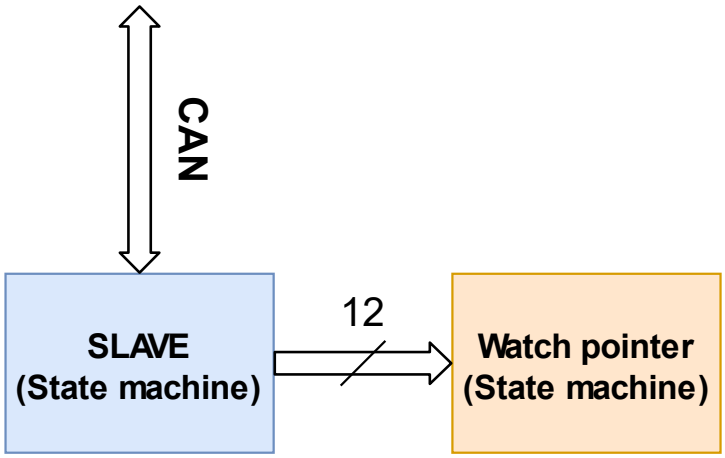


Figure 41 : SLAVE block diagram

3.4.3 State machines

In order to simplify the programming and programming process of the processors, it is essential to have exactly the same code for the GATEWAY and for all the SLAVES.

Two state machines are used to perform the work of each processor. The state machines below describe their operation.

1. The first is the main state machine. It manages the frames (SPI&CAN) and controls the movements by influencing the secondary state machine (see figure 42).
2. The secondary state machine is only used to drive a single watch pointer. Each processor controls 6 bi-axes movements, so 12 watch pointers in total (see figure 43).

In total, for each processor there are **13** state machines (1x global state machine + 12x secondary state machines) running to operate the system.

Below, state diagrams describe more precisely the function of each of the state machines.

To manage all state machines, I use a nano OS (Operating System), an XF (eXecution Framework). The one I use is developed within the HEI. The description of this one is made below (see point 3.5).

1. The main state machine consists of two parts:

   1.1. On the left, this is the frame processing part. GATEWAY will only enter the *STATE_SPI* state and SLAVES will only enter the *STATE_CAN* state.

   1.2. On the right, this is the motion processing part. This part will indicate to the secondary state machines the number of steps and in which direction it should be done.

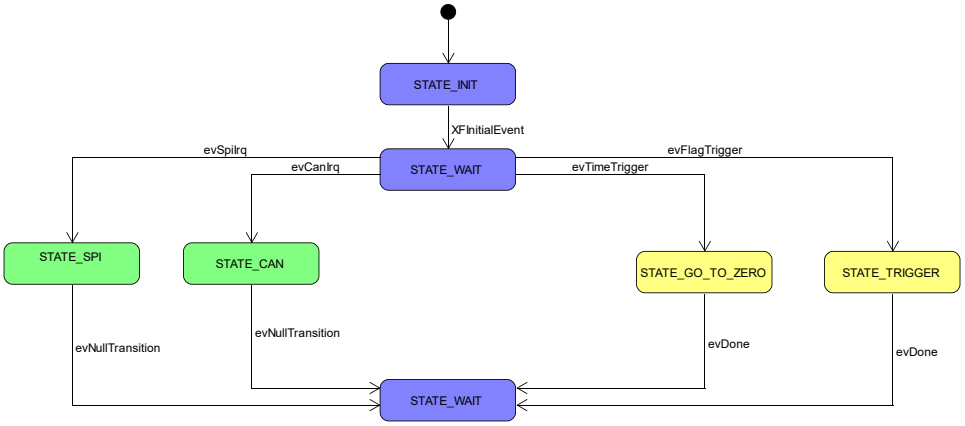


Figure 42 : GATEWAY/SLAVE main state machine

2. The secondary state machine allows to manage the timing to correctly control each watch pointer (description of the timings of the movements below, see point 4.1). This state machine receives events only from the main state machine.

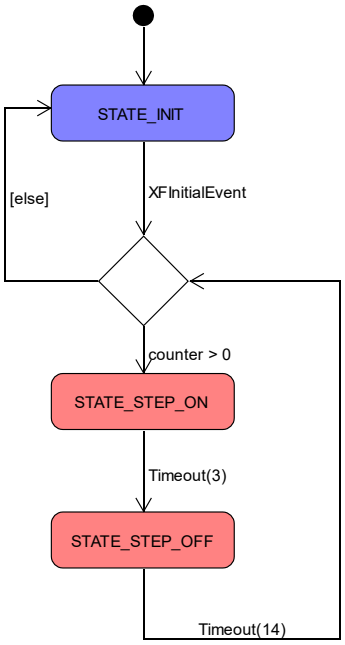


Figure 43 : GATEWAY/SLAVE secondary state machine

3.4.4 *Flow chart*

The flow charts below allow a better understanding of the main state machine. Since the second state machine is only used for timings, a flowchart is useless.

1. When an SPI frame is detected

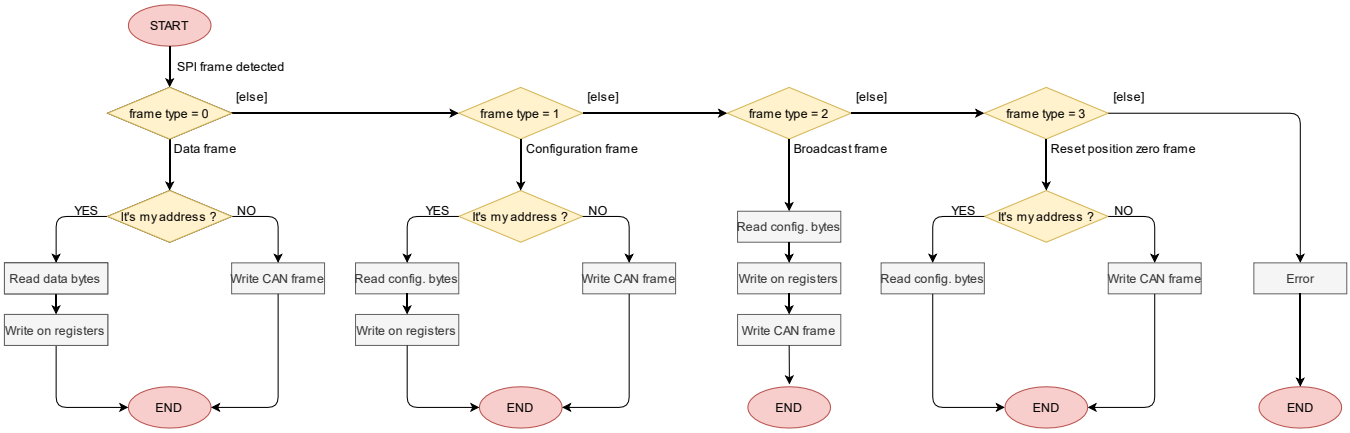


Figure 44 : Flow chart – SPI frame detected

2. When an CAN frame is detected

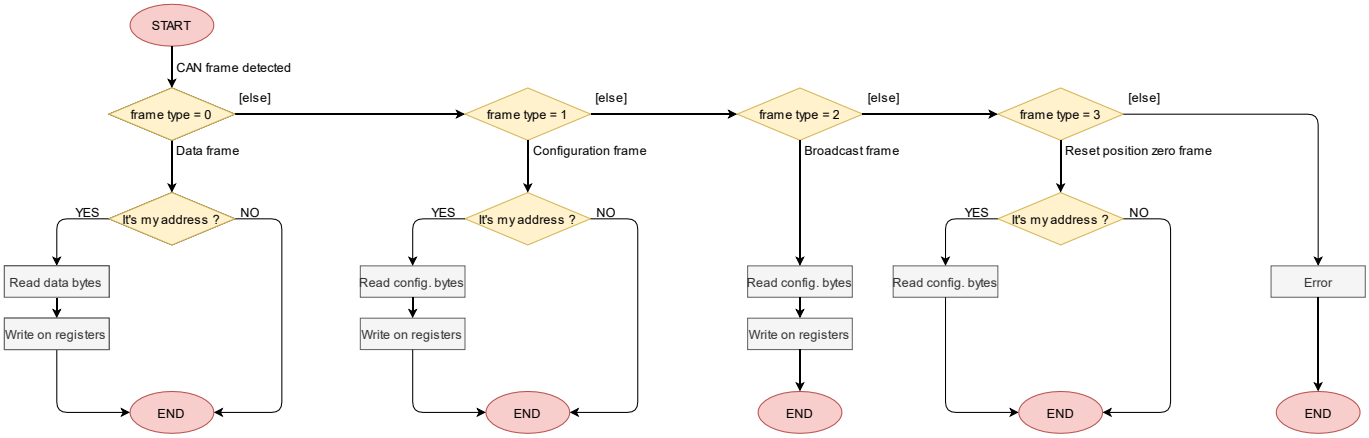


Figure 45 : Flow chart – CAN frame detected

3. When reading a configuration byte

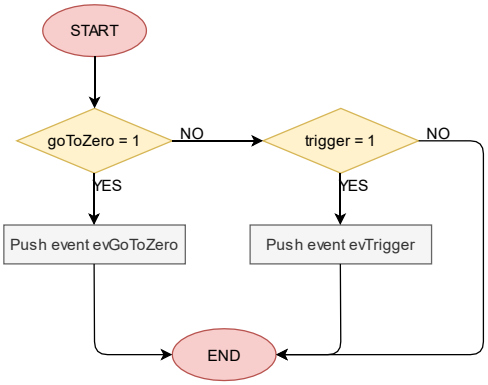


Figure 46 : Flow chart – read configuration byte

4. When there is an event to manage the movements

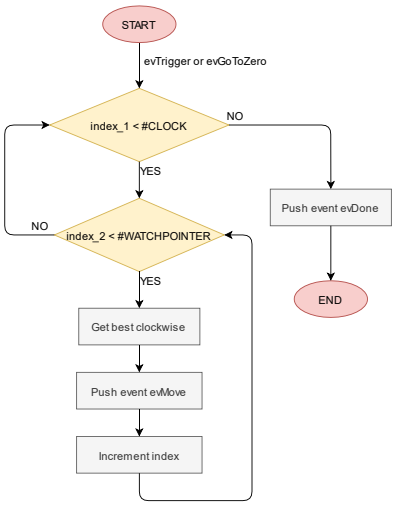


Figure 47 : Flow chart – manage movements

3.4.5 Class diagram

Here is the simplified class diagram describing the classes used. The complete class diagram is attached, see figure 90.

I use the *Factory* pattern which allows me to instantiate all my objects at code start-up. The *Controller* class has the main state machine (see figure 42). The *WatchPointer* class has the secondary state machine (see figure 43).

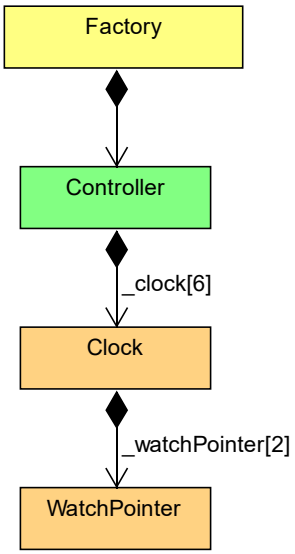


Figure 48 : GATEWAY/SLAVE simplified class diagram

3.4.6 *Sequence diagrams*

Sequence diagrams are very useful for understanding the methods used in a specific case.

Below are the most relevant sequence diagrams:

1. When an SPI data frame is detected. For CAN frame detection, the sequence diagram is similar. Only the names of the methods change

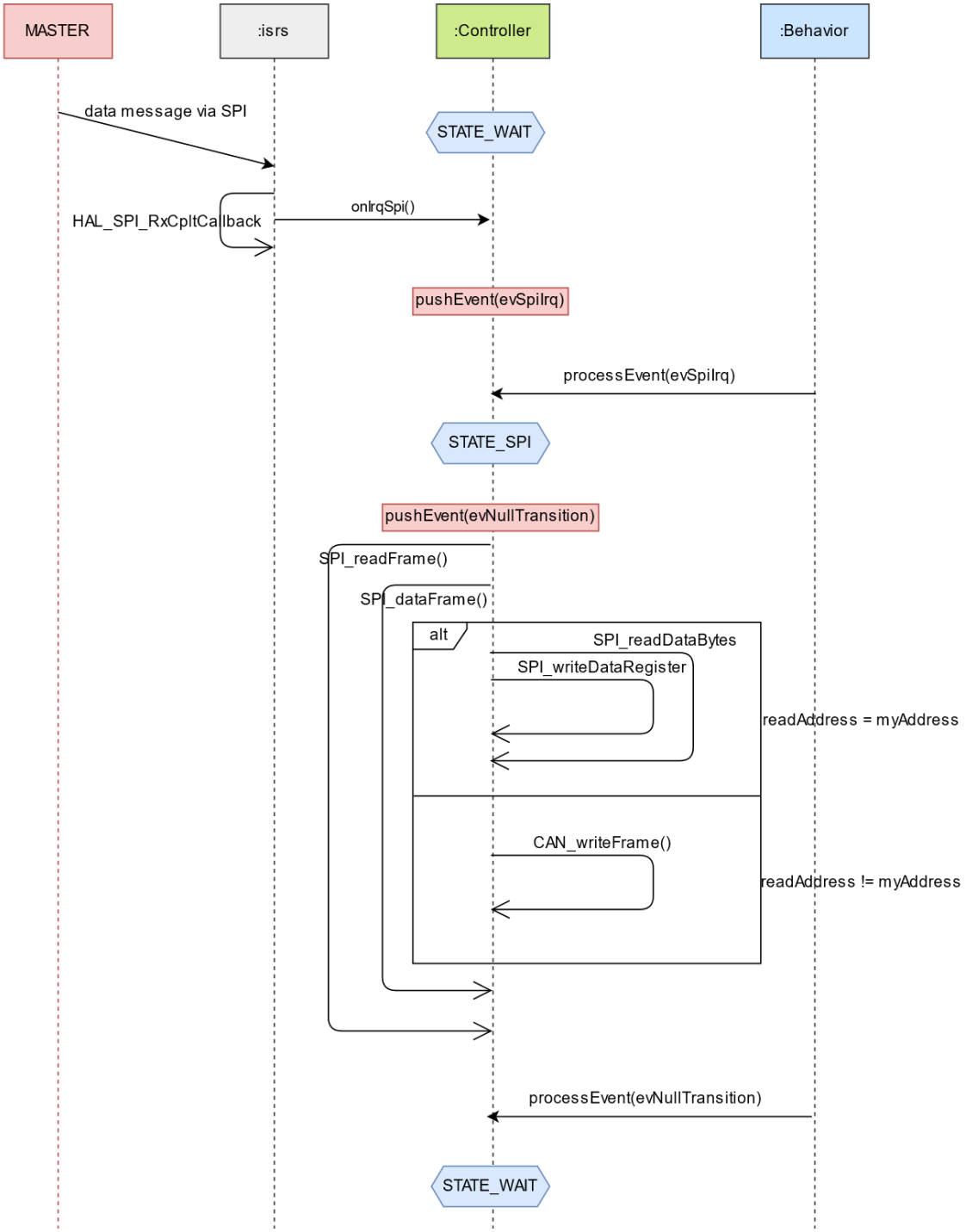


Figure 49 : Sequence diagram – SPI frame detected

2. When the event to trigger the movements is managed

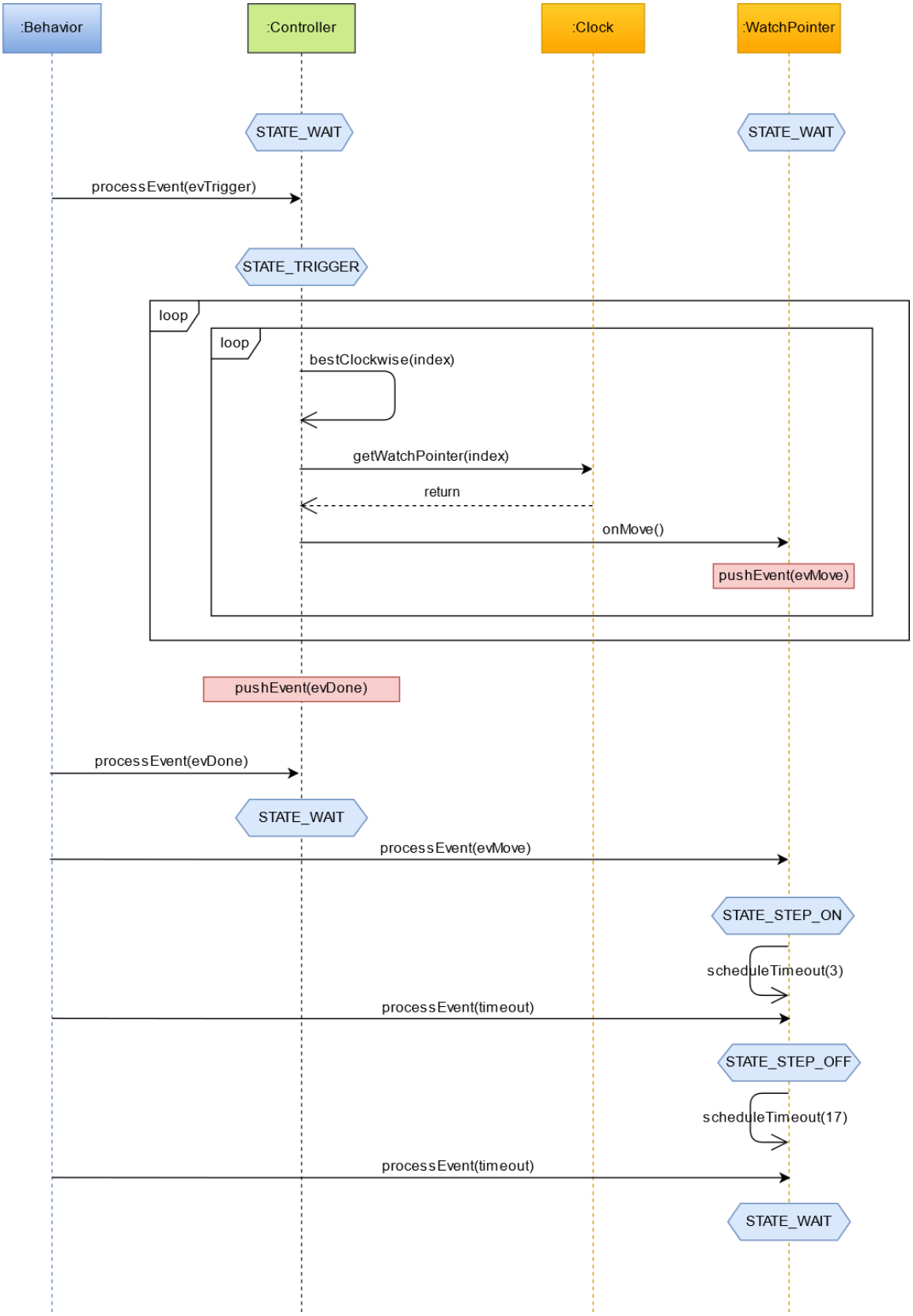


Figure 50 : Sequence diagram – trigger event detected

3.5 XF (eXecution Framework)

3.5.1 Introduction

As said before, I use an XF to run state machines for both MASTER and GATEWAY/SLAVE.

But finally, in a few words, what is an XF?

An XF is the tiniest form of an operating system. It has the following specifications:

- Event queue
- Timer manager
- Event dispatcher
- Protection mechanisms

XF can be used in two different ways:

- With an Operating System
- Without an OS Operating System

3.5.2 XF used with an Operating System

When an XF is used with an Operating System (e.g. FreeRTOS), then it is called an OXF (Operating eXection Framework).
It is still an XF controlled by timers and events but it provides several other features:

- Threads
- Protection
- Inter threads
- Better timer

It acts as an abstraction between the operating system and the user's applications. In this way, serialization and multi-tasking is possible.

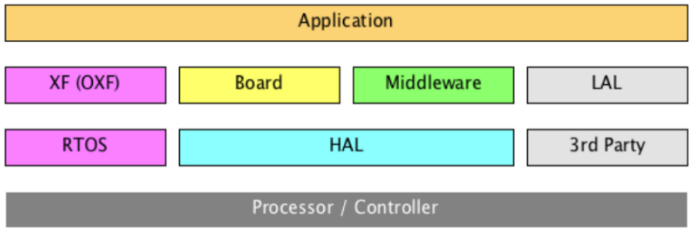


Figure 51 : XF as OXF architecture

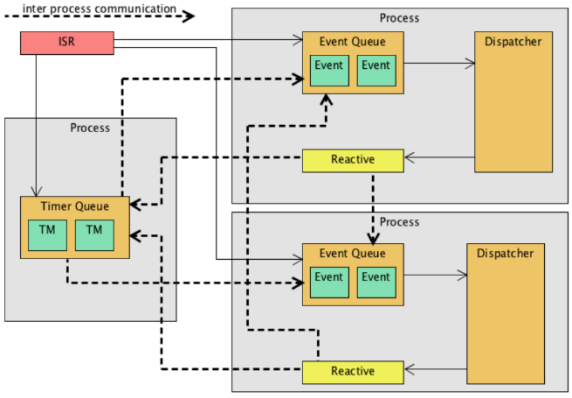


Figure 52 : XF as OXF behaviour

3.5.3 *XF used without an Operating System*

When used without an Operating System, then it is called an IDF (Interrupt Driver Framework). That's what I use for this project.
It is called IDF because two elements are important:

- Timers
- Events

The XF has the role of interface and allows to perform the most important functions of an Operating System, i.e. synchronization and pseudo-parallel execution, all with a high degree of abstraction.

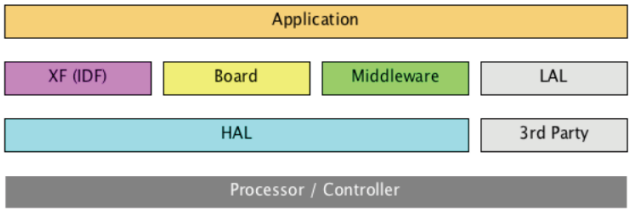


Figure 53 : XF as IDF architecture

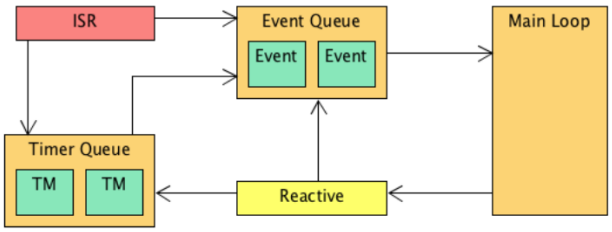


Figure 54 : XF as IDF behaviour

3.5.4 *Conclusion*

In conclusion, the XF offers a simple interface (independent of the use of an Operating System), very little memory space consuming and which allows the realization of state machine. These are practically indispensable when talking about embedded systems.

3.6 Animations

Now that the software part allows to control all the movements correctly, it is necessary to create animations in order to hypnotize the people who watch it.
Here are the sketches I made to get an idea of how the animations will look like.

All animations are represented in video. Here is the link:

- https://www.youtube.com/watch?v=ay2jXVQjWyM

3.6.1 City animation

It can be interesting to display the city in which the object is located. Here we are located in Sion (VS).
Here is the animation:

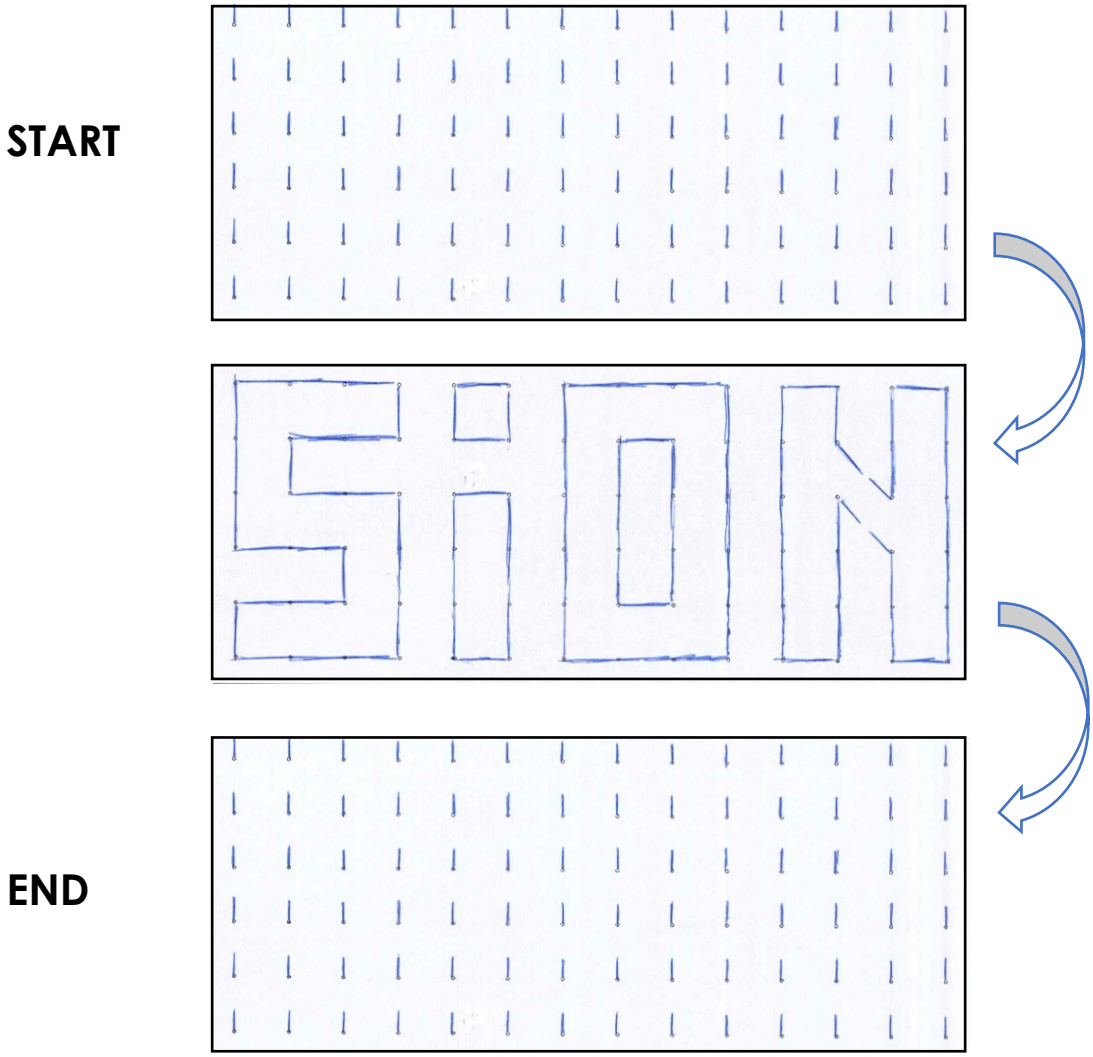


Figure 55 : City animation

3.6.2 *Time animation*

Although this is not the primary purpose, the matrix can display the time (Raspberry Pi).

Here is the animation:

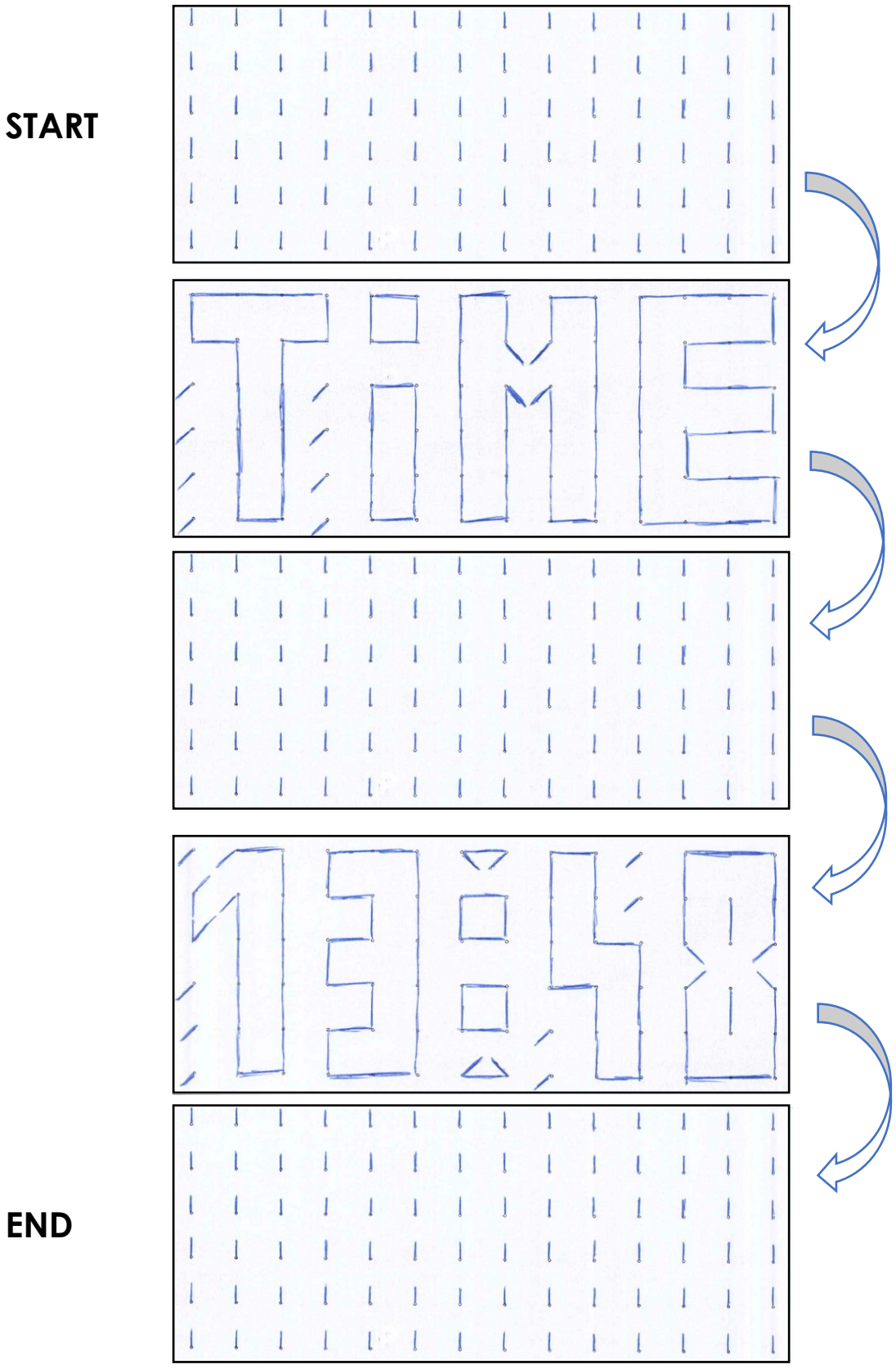


Figure 56 : Time animation

3.6.3 *School animation*

I have chosen to display also the school (and its different branches) which allowed the realization of this project.

Here is the animation:

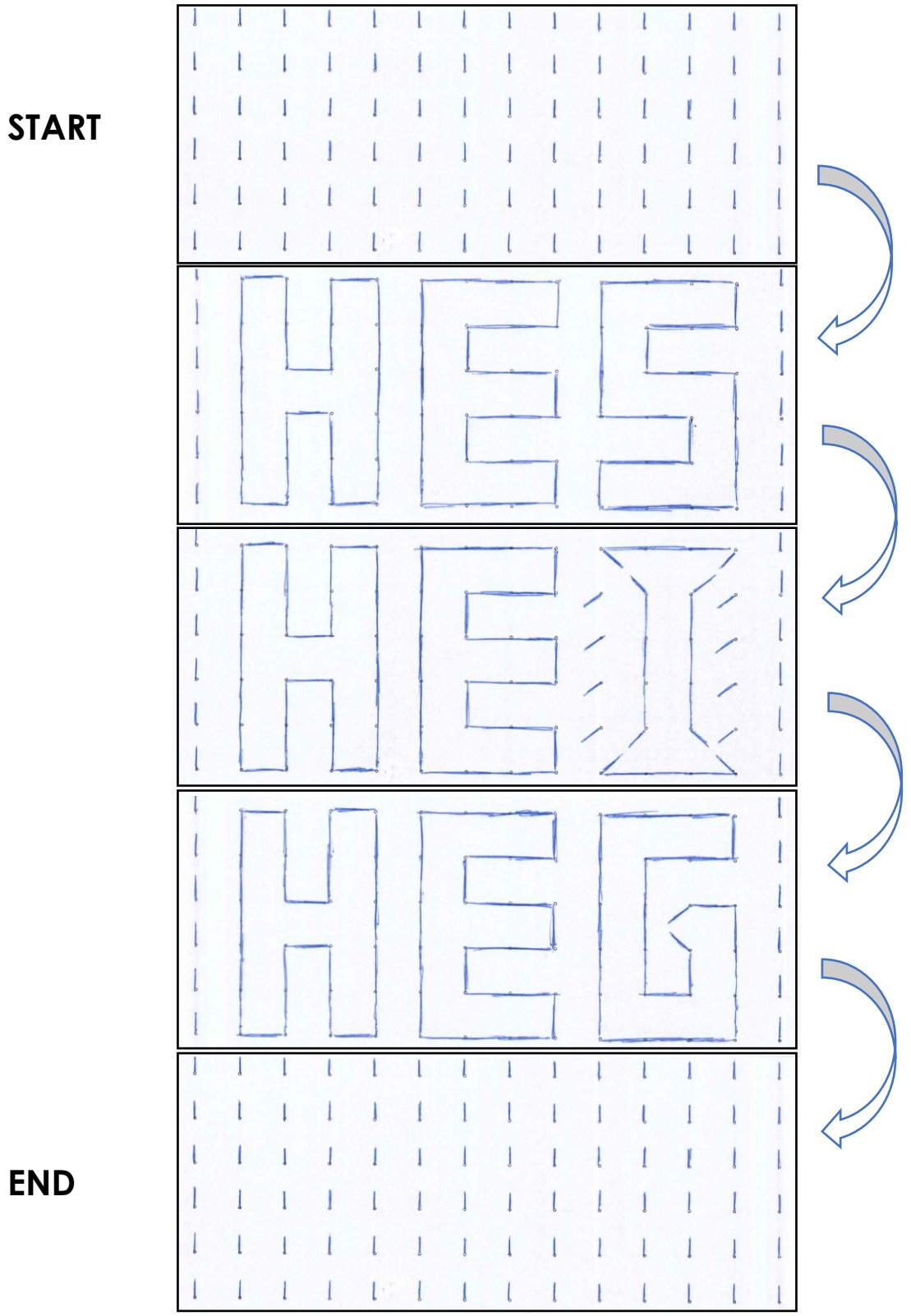


Figure 57 : School animation

4 MECHANICS

4.1 SOPROD movement

To drive all the watch pointers and to create the different animations, I use the movements produced by the company SOPROD SA.

Each axis is composed of two coils (see figure 61 & 63, coils are red). These are piloted with impulses with timing to be respected to ensure a good functioning (see figure 60).

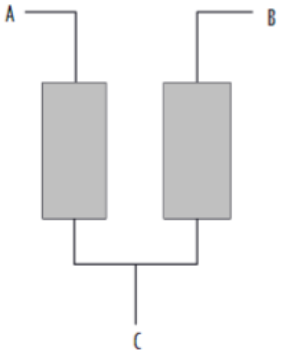


Figure 58 : Movement coils schematic

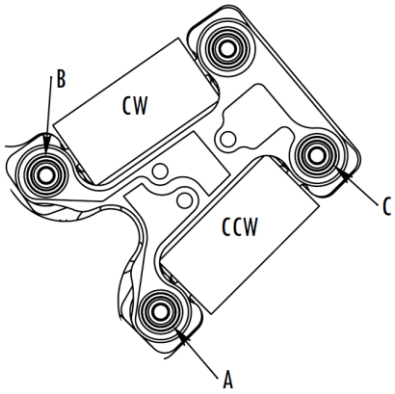


Figure 59 : Movement coils, bottom view

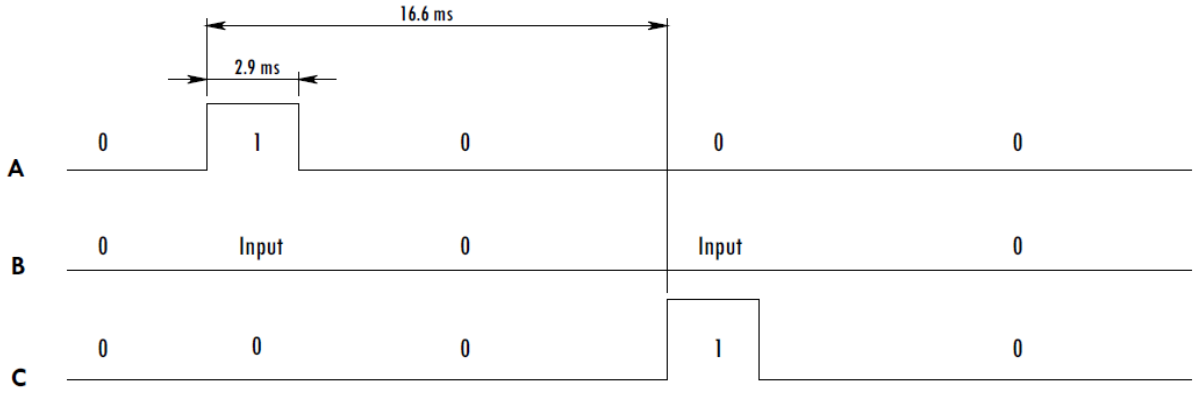


Figure 60 : Timing for step clockwise

Care must be taken to always alternate between A and C pine so as not to miss a step.

4.1.1 Bi-axes vs Tri-axes

- Bi-axes :

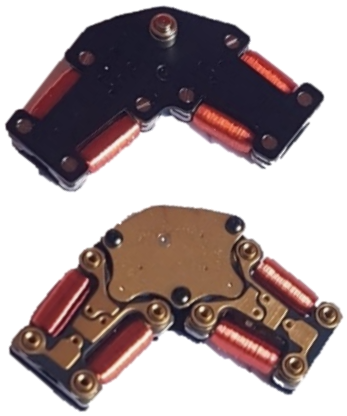


Figure 61 : Bi-axes movement

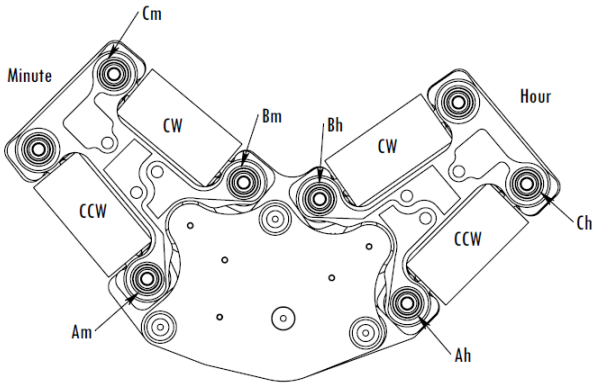


Figure 62 : Bi-axes movement – bottom view

- Tri-axes :

Figure 63 : Tri-axes movement

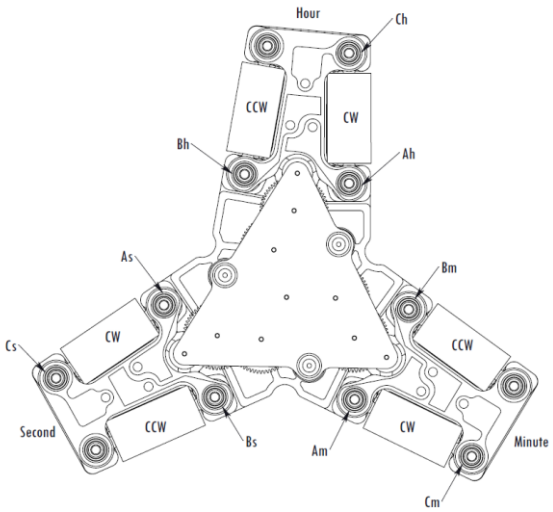


Figure 64 : Tri-axes movement – bottom view

Used movements	
V1.0	**V2.0**
▪ 3x bi-axes / processor ▪ 3x tri-axes / processor	▪ 6x bi-axes / processor

Table 17: Used movements for each version

The first idea was to create a PCB that was compatible with bi-axes and tri-axes movements. The problem is that technically it is impossible to make a pattern for this. Because the pins and holes of the movements cut each other.

For V1.0, we still wanted to test tri-axes motor drive.
For V2.0, we concentrated only on bi-axes movements.

4.1.2 Encountered problems

The precision of the direction of the watch pointers is not totally perfect. Here are some explanations:

1. Mechanical accuracy problem. The gears have a slight backlash and this is unavoidable. Although this backlash is small, it is perceptible to the human eye.
2. Mechanical accuracy problem. Not all movements are placed in the same direction on the PCB. This can create an offset between each gear and this only increases the inaccuracy.
3. Mechanical accuracy problem. The watch pointers, made at the HES with the tools available, do not allow a professional result to be obtained. This creates a problem between the axis and the watch pointer. In addition, the watch pointer position must be positioned very precisely to cancel the play of the gears. However, this is not possible because I do not have the tools and capabilities to do this.
4. Mechanical problem. The wire used to make the coils is very thin (um). An appropriate delicacy is required so as not to break the wire. This would imply that a bobbin would not work and therefore the movement could only turn in one direction.

4.2 Watch pointers production

The production of the watch pointers turned out to be much more complicated than expected. That is why we concentrated on producing the watch pointers only for the bi-axes movement.
Indeed, as the movements have very small axes (see figure 65), no inaccuracy is allowed with the watch pointers production.

Here is a very simplified view of the axis to allow an understanding of the problems encountered.

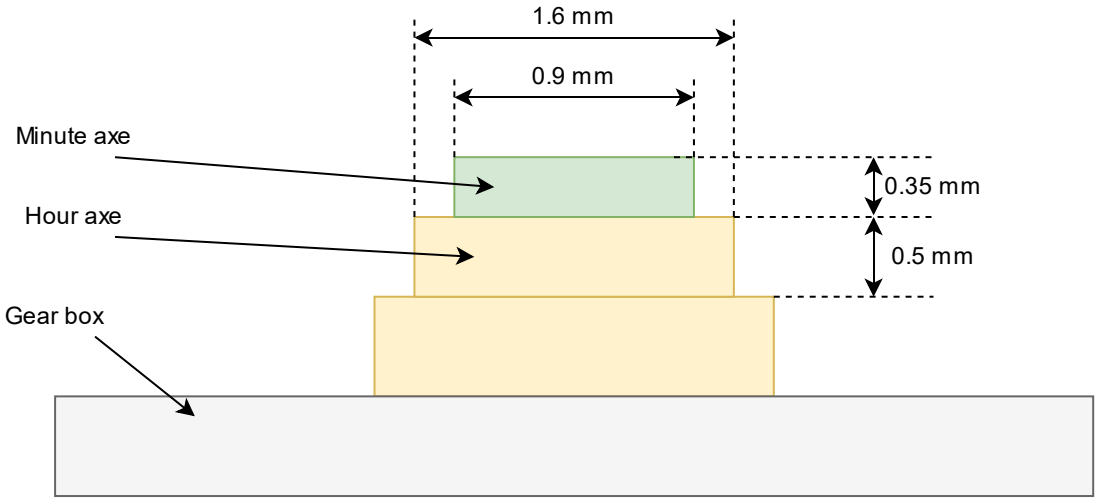


Figure 65 : Movement axes, simplified illustration

Several constraints complicated the task.

Watch pointers constraints
The watch pointers must not exceed a certain weight
The thickness of the watch pointers must be thin (< 0.35 mm) enough so that the watch pointers do not touch each other
The watch pointers must be rigid enough to stand upright and not collapse under its own weight
The material used must be flexible enough so that the watch pointer can be clipped onto the shaft without the need to add glue.

Table 18: Watch pointers constraints

For all these constraints mentioned below, we have chosen to work with plastic. It is the material that allows us to meet these constraints in the best possible way.

We have tried several manufacturing processes for watch pointers:

- 3D printing

  The precision of the printing, especially concerning the thickness of the watch pointer, is not sufficient in our case.

  When the object is printed in 3D, the plastic is brought to a certain temperature to make it modular. Because of this, managing the thickness of a 3D piece becomes very complicated.

- Laser cutting

  The accuracy of the hole is insufficient. Indeed, the laser cutting does not engrave in the material in a perpendicular way. This offset creates a hole that is not perfectly perpendicular to the surface but is conical in shape. This prevents the watch pointer from being held correctly on the axis.

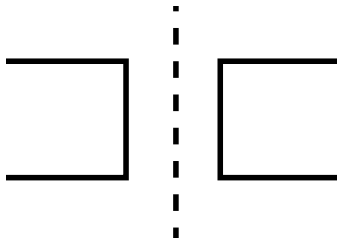


Figure 66 : Correct hole

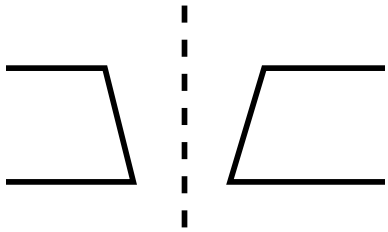


Figure 67 : Incorrect hole

- Millings

  We use a milling machine that allows us to drill the hole precisely and cut the contour of the watch pointer in a plastic part.

  It is the best option we could come up with. We used this option to make the watch pointers for the first prototype.

Despite this different manufacturing process, the result is not optimal but can do the job for a first prototype. Some watch pointers hook between it and some have difficulty to hold on to the axis. This is also due to the quality of the plastic used.

In the future, custom-made metal watch pointers will be indispensable.

4.3 Box design

It is always more interesting, even if it is a prototype, to see the product in a case.

Here is the case made. It is composed:

- **A back plate**
  Carry out with the milling machine. The PCBs are fixed there and a space is provided for the programming of the processor. A space is also provided to plug in the Raspberry Pi Zero. The front plate will also be fixed there.
- **A front plate**
  Carry out with the milling machine. The role is purely aesthetic. Space is provided so that the PCBs can come at the same level as the board.
- **Two supports**
  Realize with a 3D printer. Used to hold the whole on a horizontal surface on the ground.

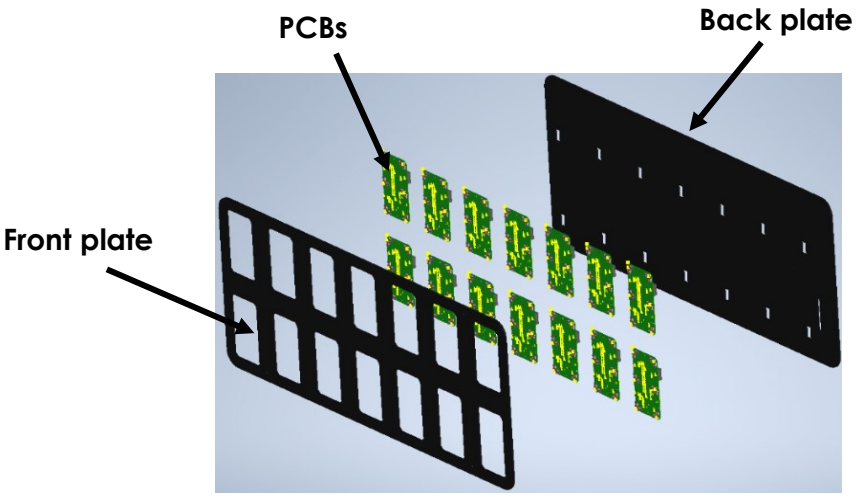


Figure 68 : Front and back plate for the box

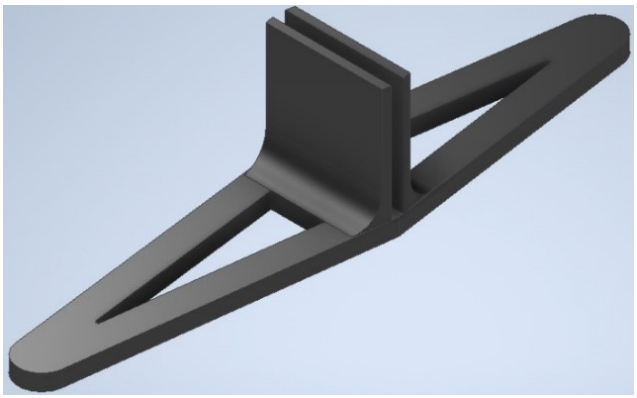


Figure 69 : Box support

This is what the first prototype looks like:

Figure 70 : Back view

Figure 71 : Lateral view

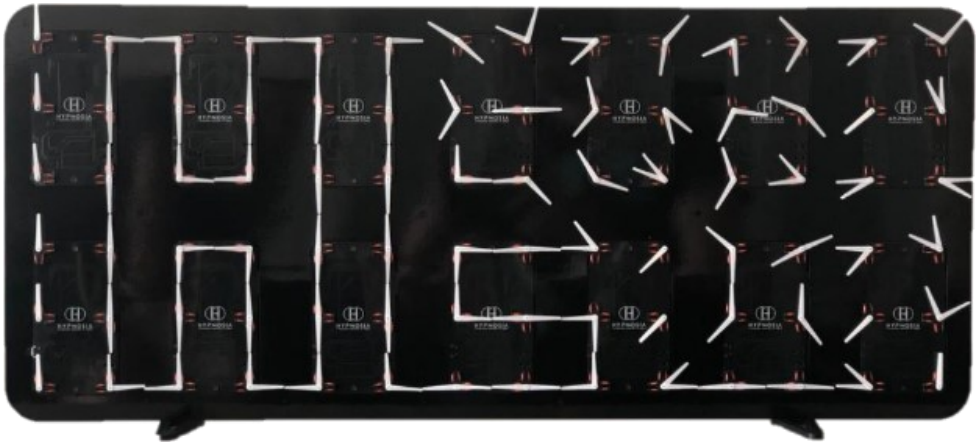


Figure 72 : Front view

5 TESTS

5.1 Used tools

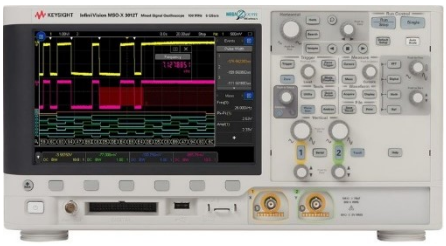
Tool	Comment
Logic analyser (Saleae Logic Pro 16) 	Used to measure digital signals. Very easy to use. Connects via USB to the PC.
Oscilloscope (MSOX3012T) 	Used to analyse the behaviour of an analog signal over time.
Multimeter (Agilent U1252B) 	Used to measure consumption.

Table 19: Used tools for tests

I would like to point out that the tests carried out below remain global and do not test the system in the slightest detail. I did not carry out further tests due to lack of time. But they remain essential to make a product reliable.

As it is often said in engineering, it takes 20% of the time to perform the 80% of the product and 80% of the remaining time to perform the remaining 20% of the product.

With the time made available in my case, the tests allowed me to determine whether or not the first prototype was working properly.

5.2 Tests performed

5.2.1 Bluetooth communication

To test the proper functioning of the Bluetooth communication between the smartphone and the MASTER, I am in debug mode and I analyse the information received.

Below is an example of information received when clicking on the "Animation 1" button on the smartphone. This corresponds exactly to the JSON format we defined beforehand (see point 3.1.1).

⊟ jsonObject	<2 items>	QJsonObject
"body"	1	QJsonValue (Number)
"header"	"ANIMATION"	QJsonValue (String)

Figure 73 : JSON information received by the MASTER

I then tested for all other JSON formats. All the information I received is correct.
So I validate the Bluetooth communication between the smartphone and the MASTER.

5.2.2 MASTER – GATEWAY communication

With this test, I make sure that the MASTER - GATEWAY communication via SPI is correct.
To perform the measurement, I use the logical analyser.

Test performed:

1. Send a well-defined frame

   For this test, the MASTER sends:

   - A data frame

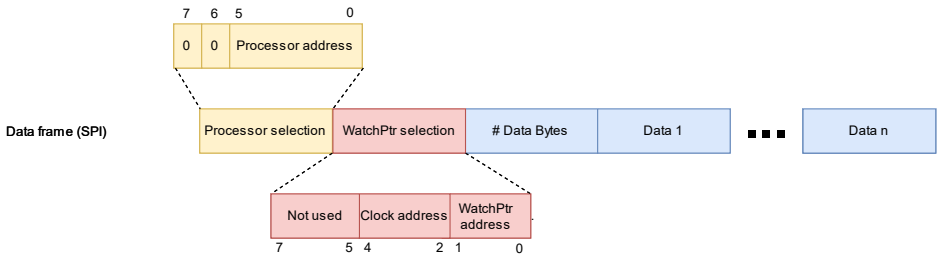


Figure 74 : SPI Data frame – sent frame

In this example, the processor address is 0. The clock address is 6. The pointer address is 0. The number of bytes to be sent is only 3 (Config.1&2, Position 1, Timing 1). See table 2. The initial position of the motor is 0, we want to run clockwise.

To sum up, the bytes sent are: **0x00, 0x0C, 0x03, 0x08, 0x01, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF**

- A configuration frame

Figure 75 : SPI Configuration frame – sent frame

In this example, the processor address is 0. The number of bytes to be sent is 2 (Index Register and Status) see table 2. Index Register is equal to 0 because we want to write to the first configuration byte. For the status byte, we just want to set to 1 the flag corresponding to the watch pointers trigger.

To summarize, the bytes sent are: **0x40, 0x02, 0x00, 0x10, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF**

2. <u>Measure the frame</u>

The GATEWAY is receiving:

- A data frame

Figure 76 : SPI Data frame – received frame

- A configuration frame

Figure 77 : SPI Configuration frame – received frame

3. <u>Compare the sent frame and the received frame</u>

We can now compare the frames sent and received. They are identical. I also ran the tests with the other types of frames as well. The results obtained are correct. So I validate the MASTER - GATEWAY communication.

5.2.3 GATEWAY – SLAVES communication

With this test, I make sure that the GATEWAY – SLAVES communication via CAN is correct. To perform the measurement, I use the logical analyser.

For this test, I relied on the above test validity. Indeed, I check that the GATEWAY transmits the information to the CAN bus when it receives a frame which is not addressed to it. To do this, the selected processor address must be different from 0 (because GATEWAY processor address = 0).

Below is the measurement of the CAN bus. We can see the two SPI frames sent by the MASTER to the GATEWAY which will be immediately transmitted to the CAN bus by the GATEWAY. Here the values of the frame are not important, it is the immediate retransmission system that is tested.

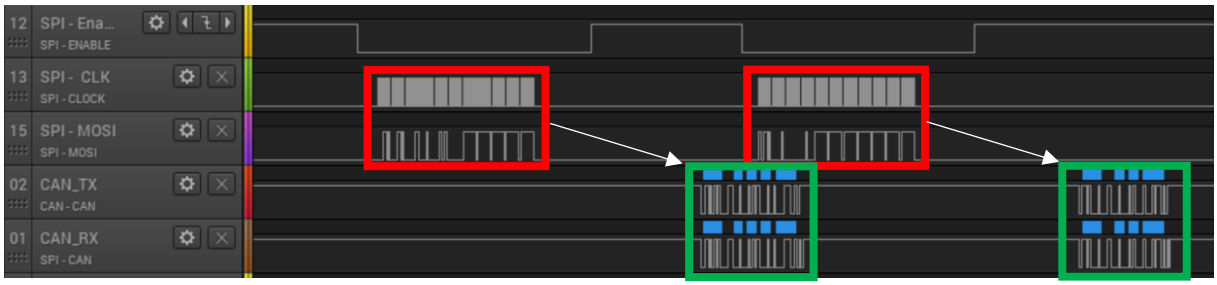


Figure 78 : SPI – CAN retransmission test

Sent by the MASTER

Sent by the GATEWAY

Now I also have to check that the sent values follow the predefined CAN protocol. Test performed:

1. <u>Send a well-defined frame</u>

   For this test, the MASTER sends:

   - A data frame

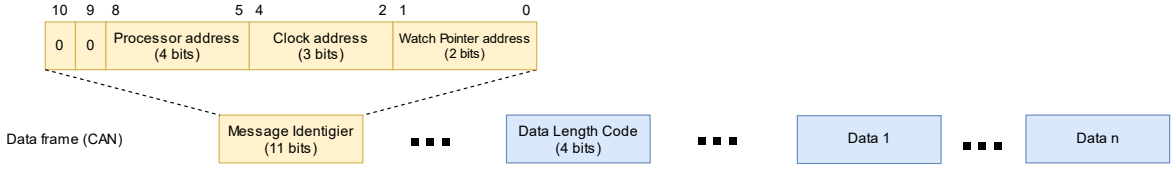


Figure 79 : CAN Data frame – sent frame

- A configuration frame

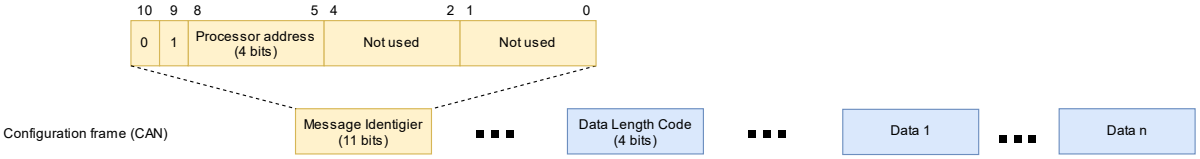


Figure 80 : CAN Configuration frame – sent frame

The role of the GATEWAY is to transform the SPI frame into a CAN frame while respecting the predefined fields of the CAN protocol.

2. Measure the frame

   All SLAVES are receiving:

   - A data frame

Figure 81 : CAN Data frame – received frame

   - A configuration frame

Figure 82 : CAN Configuration frame – received frame

The logic analyser knows the CAN protocol and allows direct display of the protocol fields for more efficient visibility.

The red crosses are bit stuffing. These bits occur when the previous 5 bits are identical. The goal is to increase the number of transitions is to ensure better synchronization of the nodes.

3. Compare the sent frame and the received frame

   The frames received correspond to the content received by the SPI frame. The GATEWAY performs its role as a frame transmitter well. The GATEWAY-SLAVE communication is therefore validated.

5.2.4 *Movements control*

With this test, I check that the timings used allow the steps of the movements to be controlled correctly.

Several tests have been carried out:

1. Take a single step.

   Here the MASTER sends the following information. First SPI frame indicates the new watch pointer position (+1). The second frame sends the information to the trigger GATEWAY and thus to go to the new position read.

   Below, we can see these two SPI frames sent and we can see that an pulse is given to the movement.

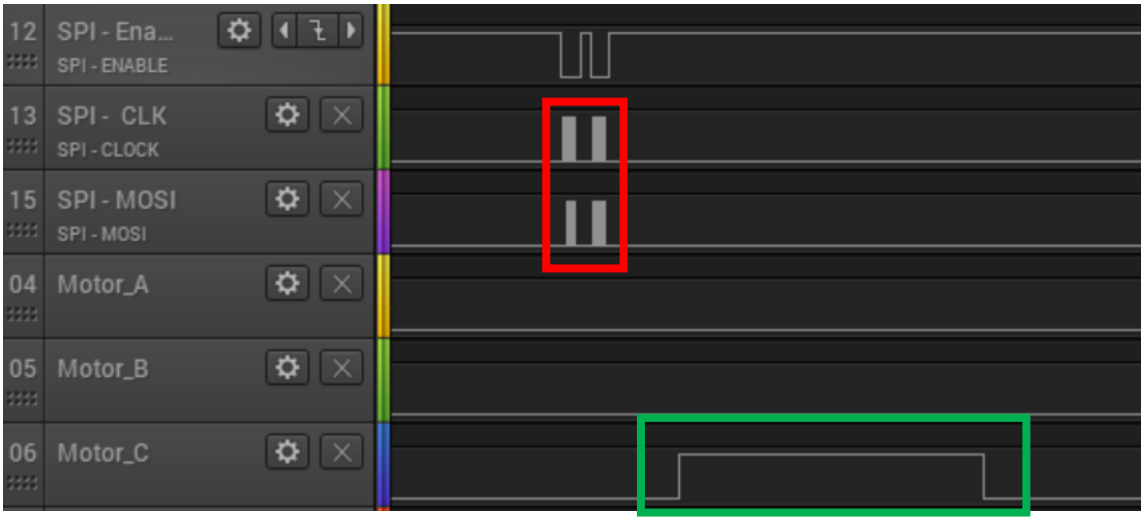


Figure 83 : Movement test – single step

SPI frames

Pulse given to the movement

2. Take several steps

   For the next test, I test that the pulses alternate each time as required by datasheet.

   Below are four pairs of SPI frames sent to the gateway to inform it to move the watch pointer by four step. Then we see that the pulses sent to the movement is well alternated.

Figure 84 : Movement test – four steps

SPI frames

Pulses given to the movement

3. Checking timings

   As mentioned in the datasheet, it is important to maintain the time of the impulses mentioned to ensure the proper functioning of the movement.

   Below are the measurements taken. On the right you can see that the time for the ON pulse is 3ms and the time for the next pulse is 14ms. This is in accordance with the times given in the datasheet (see figure 60).

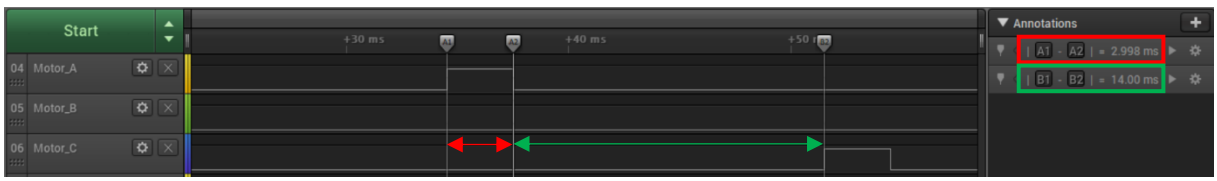


Figure 85 : Movement test - timing

4. Visual test

   I have also done different tests, this time visual. Like for example doing a quarter turn, a half turn, a full turn, two full turns, etc... All the tests were conclusive.

   So I can validate the movement control.

5.2.5 Power supply consumption

In many embedded systems, power is a very important factor. In our case, it is not necessary to control any nA consumption because the system is not powered by batteries but by the electrical grid.

If the consumption of the system is not supported by our Raspberry Pi, we always have the choice to power it with an external power supply, different from the power supply used for the Raspberry Pi (see point 2.7.2).
To perform the measurement, I use the multimeter.

I proceeded in three steps:

1. I first made a prediction based on the consumption values for each datasheet (datasheet capture are annexed)

Element	Theoretical consumption
1x Processor	19 [mA] (see figure 102)
12x watch pointers	1 [mA] (see figure 103)
1x CAN module	400 [uA] (see figure 104)
Total	20.4 [mA]
Total for the 6x14 matrix	14x 20.4 [mA] = **285.6 [mA]**

Table 20: Theoretical consumption

2. I then measured the actual consumption of the 6x14 matrix

Element	Real consumption
Total for the 6x14 matrix	**276 [mA]**

Table 21: Real consumption

3. Finally I compared the two values and determined if it's plausible.

Theoretical consumption	Real consumption	Plausible ?
285.6 [mA]	276 [mA]	✓

Table 22: Consumptions comparison

Since the theoretical measurements were made with a small margin each time, the measurement performed is plausible. The Raspberry Pi is therefore capable of powering the entire system.

5.2.6 Power supply behaviour

The power supplied by the Raspberry Pi is not a very stable power supply. However it can be sufficient for our needs.
So I measured the behaviour of the power supply with the oscilloscope to check that the power supply remains stable despite the movements of the watch pointers.

The result is good and the power supply remains stable.

5.2.7 Long-term testing

To ensure long-term operation, it must be ensured that the program does not crash after several minutes/hours.

So I let the program run and waited to see if a problem occurred. Indeed, after a few seconds/minutes, some processors start to crash…
When a problem of this type appears, you have to try to determine if it is always the same processors that crashes, if it is always after the same running time, if it is in the same order, etc...
The problem here is that the order of the processors and the runtime are totally independent and random. Since each processor has the same code and receives exactly the same information via CAN (and therefore should do exactly the same job), this made it difficult to detect the problem in practice.
So I made several hypotheses:

1. Power supply problem. As said before, I checked the power supply and it remains stable regardless of the use of the movements. So I discard this hypothesis.
2. Electrical problem related to the coils of the movements. As there are potentially 84 biaxial movements in motion (this represents 336 coils because each biaxial movement has 4 of them), I thought that this could have created an electrical effect that could influence the power supply of the processor. So I run the code but without using the coils. The problem persists. So I dismiss this hypothesis.
3. Memory allocation problem. Indeed when the processor crashes it enters the *HardFault_Handler()* method. When you enter this method, the problem is often related to a memory problem (often but not always). As the events generated

in the XF are dynamic, I modified the secondary state machine that manages the motion control (see figure 43) to optimize the number of events generated. This did not solve the problem. I then switched all the events used in the XF to static instead of dynamic. This improved the system but did not totally solve the problem. The problem happens much less frequently (usually after several minutes) but still happens.

4. Asynchronism problem. I hypothesized that we receive an interrupt indicating the detection of a frame (SPI or CAN) at exactly the same time as an interrupt of the timers used in the XF. This could explain the non-periodicity of the order of the crashing processors and of the running time. I have changed the interrupt priorities to give more importance to one type of interrupt rather than another. The problem persists.
5. Problem related to processor errata. In the processor errata, there is nothing to suggest that the processor used contains a problem related to the SPI/CAN protocol or even memory. So I dismiss this hypothesis.

When a totally random problem appears, it can be interesting to create a test code to allow to force the problem in order to better analyse it. However, with the time available, I have not been able to create an adapted test code. This is therefore an unsolved problem.

5.3 Tests summary

Test description	Validated ?
Bluetooth communication	✓
MASTER – GATEWAY communication	✓
GATEWAY – SLAVES communication	✓
Movement control	✓
Power supply consumption	✓
Power supply behaviour	✓
Long term testing	✗

Table 23: Tests summary

All the tests carried out were successful except for the long-term test. This is an essential problem to be solved for the rest of the project.

6 RESULTS

6.1 Project objectives summary

Here I take up again the objectives defined at the beginning of the diploma work.

Description	**Achieved ?**
Development of a prototype of 6 rows and 14 columns (84 movements in total).	✓
Development of a universal electronic board responsible for the operation of several movements (2x3), accounting with or without master and with bi-axes or tri-axes movements.	±
Development of a communication protocol between the master and all the slaves.	✓
Development of a master that can manage all the animations available.	✓
Development of at least two animations.	✓
Development of Bluetooth communication between the smartphone (Android) and the master.	✓
Packaging of the system.	✓
Minimize production strokes (< 400CHF).	✓

Table 24: Project objectives summary

We can therefore see that all the main objectives have been successfully achieved. However, the objective concerning the development of a universal electronic board has not been fully achieved, because, as explained in point 4.1.1, the pattern for biaxes and triaxes movement is technically impossible.

6.2 Future improvements

Concerning the continuation of the project, the main improvements to be achieved are:

1. Fixing the problem related to the long-term test
2. Improve the quality of the watch pointers so that they have a better hold on the axis.
3. Making a complete case
4. Make larger PCBs to limit the number of connectors used.
5. Limit material costs
6. Limit production costs
7. Make it totally reliable in the long term.

More improvements are essential to increase the precision of the animations to create an effect of satisfaction for the customer.

The most important thing is that we have fixed the important points to be improved, but the first prototype remains functional.

6.3 Products prices

When this product is presented to a potential client or investor, one of the questions that comes up every time is: "*What is its price ?*".

That's why I made a price list depending on the model chosen and the number of pieces produced.

When I made this price list, I noticed that a large percentage of the prices were for connectors. That's why I created versions with one large PCB rather than several small ones that require connectors.
Two curves are present on the two graphs below. One curve represents the material costs with the use of several 2x3 size PCBs. The other curve represents the material costs using a single large PCB. The production costs for small PCBs are lower, but this implies expenses for connectors. The production costs for large PCBs are higher but this does not imply an expense for connectors. So I tried to finish the cheapest solution.

Below are the prices for the 3x8 matrix model. You can see that the material costs decrease exponentially up to 100 pieces produced. Thereafter, the decrease in material costs remains linear.

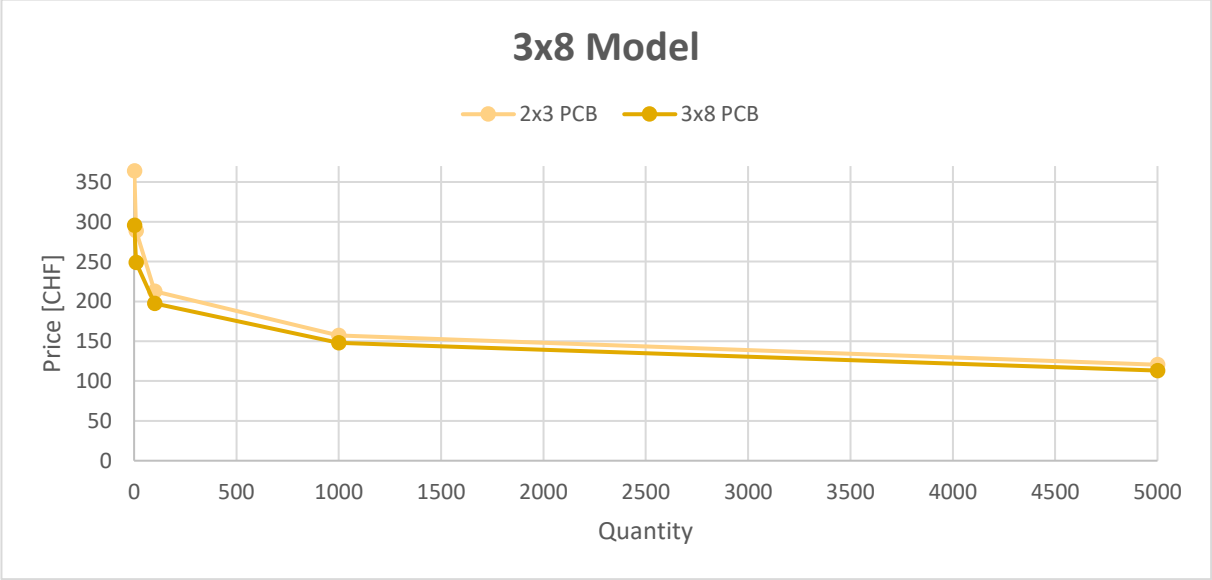


Figure 86 : Prices comparison for 3x8 model

Below are the prices for the 6x14 matrix model. It can be seen that the trend of the curve remains very similar to that of the 3x8 matrix model.

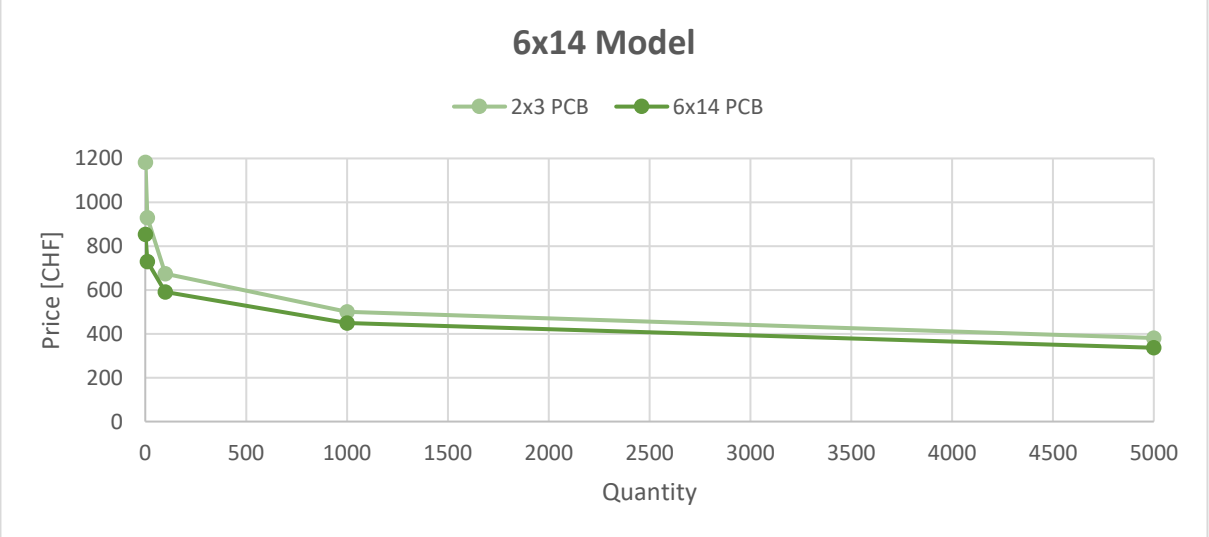


Figure 87 : Prices comparison for 6x14 model

Saving money by using several small PCBs or a single large one seems derisory... Yet this is absolutely not the case. To realize this big difference in price, we have to create a graph that shows the savings made according to the number of units produced.

The graph below shows the savings made with the use of large PCBs according to the number of units produced.

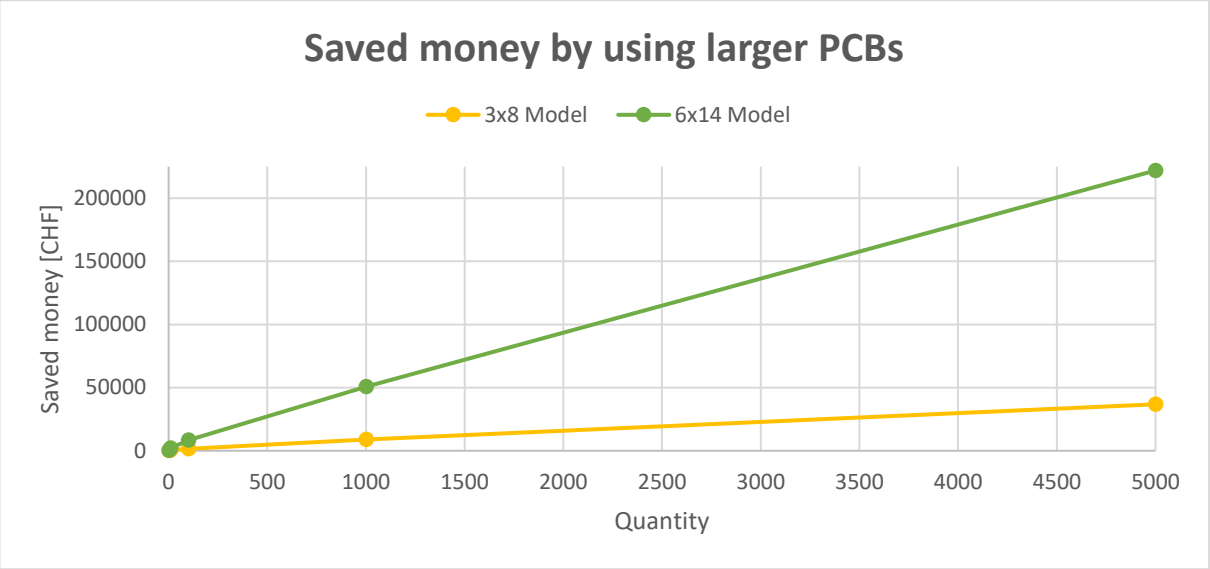


Figure 88 : Saved money by using larger PCBs

There is no doubt that the version with one large PCB rather than several small PCBs is much more cost effective, regardless of the model chosen.

All price details are attached.

6.4 The future of HYPNOSIA

HYPNOSIA, as mentioned in the introduction, is composed of two engineers and two economists. These two economists did not wish to continue the adventure because of personal projects. Personally, I will not be able to continue the development because I am continuing my studies with a full-time Master's degree, and the time available for external projects is very limited. Only Ruben De Campos will continue the development of the product. Nevertheless, I will be there to help with technical support or simply to make people talk about the product.

7 CONCLUSION

7.1 Project conclusion

This project has been very enriching for me and is very close to the values that I hold dear in terms of work, i.e. attention to detail and perfectionism.
Overall, I managed to achieve all the objectives set and I remain very satisfied with the final result.

This HYPNOSIA start-up project is going alive for a year now. Thanks to these two diploma theses, the project has greatly evolved over the last three months. There is still work to be done to make the product more reliable. But with this first prototype, approaching potential investors becomes less complicated. I think that the great adventure can still continue and I sincerely hope that we will end up with a reliable and robust product that can then be marketed.

7.2 Acknowledgment

I would like to sincerely thank all the people who believed in this project and helped me make it happen.
I congratulate the collaboration between the HEI (Haute Ecole d'Ingénierie) and the HEG (Haute Ecole de Gestion) for having set up the BeX (Business eXperience) program, which allowed the creation of the HYPNOSIA startup.

Concerning the realization of this project, I would like to thank Carmine Arcudi for all the routing part of the PCBs as well as for the realization of the plastic watch pointers. Thanks also to the company Soprod SA who believed in this project and provided us with all the motors necessary to realize our first prototype.
To conclude, a special thanks to Silvan Zahno, director of the diploma work, for his availability, reactivity and for all the good advices concerning the realization of this project.

Signature

Geraci Gregory

Sion, 21 august 2020

8 ANNEXES

A. References
B. Planning
C. Guide for Raspberry Pi
D. MASTER software
E. GATEWAY/SLAVE software
F. Altium Schematic V1.0
G. Altium Schematic V2.0
H. JSON files
I. Consumption
J. Price details

A. References

[1] 'Hypnosia | L'élégance dépasse le temps', *Hypnosia*. https://www.hypnosia.ch (accessed May 28, 2020).
[2] 'Comparatif des modèles de Raspberry PI | Tableaux comparatifs - SocialCompare'. https://socialcompare.com/fr/comparison/raspberrypi-models-comparison (accessed May 26, 2020).
[3] 'Teach, Learn, and Make with Raspberry Pi – Raspberry Pi'. https://www.raspberrypi.org/ (accessed May 25, 2020).
[4] 'UART vs I2C vs SPI – Communication Protocols and Uses', *Latest open tech from seeed studio*, Sep. 25, 2019. /blog/2019/09/25/uart-vs-i2c-vs-spi-communication-protocols-and-uses/ (accessed Jul. 29, 2020).
[5] 'UART Based Networking for Microcontrollers'. https://blog.thegaragelab.com/uart-based-networking-for-microcontrollers/ (accessed Jun. 04, 2020).
[6] 'STM32 32-bit Arm Cortex MCUs', *STMicroelectronics*. https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html (accessed May 28, 2020).
[7] 'STM32 Mainstream MCUs', *STMicroelectronics*. https://www.st.com/content/st_com/en/products/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus/stm32-mainstream-mcus.html (accessed May 28, 2020).
[8] 'STM32F0 Series', *STMicroelectronics*. https://www.st.com/en/microcontrollers-microprocessors/stm32f0-series.html (accessed May 28, 2020).
[9] 'STM32F0x2', *STMicroelectronics*. https://www.st.com/content/st_com/en/products/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus/stm32-mainstream-mcus/stm32f0-series/stm32f0x2.html (accessed Jun. 05, 2020).
[10] 'STM32F072V8T6 STMicroelectronics | Mouser', *Mouser Electronics*. https://www.mouser.ch/ProductDetail/511-STM32F072V8T6 (accessed Jun. 05, 2020).
[11] 'Microcontrollers | Microchip Technology'. https://www.microchip.com/design-centers/microcontrollers (accessed May 28, 2020).
[12] 'STM8 8-bit MCUs', *STMicroelectronics*. https://www.st.com/en/microcontrollers-microprocessors/stm8-8-bit-mcus.html (accessed May 28, 2020).
[13] 'MSP430 Ultra-Low-Power MCUs | Overview | Microcontrollers (MCU) | TI.com'. http://www.ti.com/microcontrollers/msp430-ultra-low-power-mcus/overview.html (accessed May 28, 2020).
[14] 'Nordic Semiconductor - Home - nordicsemi.com'. https://www.nordicsemi.com/ (accessed May 28, 2020).
[15] 'STM8S Series', *STMicroelectronics*. https://www.st.com/content/st_com/en/products/microcontrollers-microprocessors/stm8-8-bit-mcus/stm8s-series.html (accessed May 28, 2020).
[16] 'STM8S Value Line', *STMicroelectronics*. https://www.st.com/content/st_com/en/products/microcontrollers-microprocessors/stm8-8-bit-mcus/stm8s-series/stm8s-value-line.html (accessed May 28, 2020).
[17] 'STM8S003F3U6TR STMicroelectronics | Mouser', *Mouser Electronics*. https://www.mouser.ch/ProductDetail/511-STM8S003F3U6TR (accessed May 28, 2020).
[18] 'STM8S003F3P6TR STMicroelectronics | Mouser', *Mouser Electronics*. https://www.mouser.ch/ProductDetail/511-STM8S003F3P6TR (accessed May 28, 2020).

B. Planning

The Gantt Chart being relatively large, it is preferable to annex it via this link for a much better visibility.

- https://github.com/GeraciGregory/HypnosiaController/blob/master/Documentation/Planning/ganttChart.xlsx

However, here are a few comments about the Gantt Chart:

- I underestimated the delivery time for both components and PCBs. I was more than two weeks late for a single order from Mouser (usually 2-3 days).
- The manufacturing of the watch pointers turned out to be much more complicated than expected and therefore took much longer.
- I decided not to make a case for the V1.0, because in the end it was useless and it allowed me to invest more time on other more important tasks.
- I should have spent more time on testing and bug fixing.

When you are very close to the end date, it is interesting to make another small schedule with even more precise tasks.

- https://github.com/GeraciGregory/HypnosiaController/blob/master/Documentation/Planning/ganttChart_bis.xlsx

C. Guide for Raspberry Pi

This guide serves as a memory aid.

1. First boot

- https://www.raspberrypi.org/blog/raspbian-update-june-2018/

1) Use *RaspberryPi Imager* program to write *Raspbian* on SD card
2) Plug SD card to Raspberry Pi → configure option (language, wifi, ..)
3) Raspi-config (config keyboard, ssh enable, spi enable)
4) Install *wiringPi* library
5) Install *Qt5*
6) Install *GIT*
7) Limit access of used CPU cores if necessary
8) Git clone my project (https://github.com/GeraciGregory/HypnosiaController)

2. Qt Creator

- http://tvaira.free.fr/projets/activites/activite-qt5-rpi.html

1) ! Having the same version of Qt on your PC and on your Raspberry Pi !
2) ***qmake –version*** → know information about Qt
3) ! If SIGSTOP → UNCHECK the checkbox Project/Run Setting/Run in terminal !

3. Limit numbers of used CPU cores

- https://www.raspberrypi.org/forums/viewtopic.php?t=175745

Add ***maxcpus=2*** at the end of the line of the file ***/boot/cmdline.txt***

4. Edit a text with command line

1) Go to the right repository (use cd)
2) ***sudo nano <nameOfFile>***
3) Edit
4) Ctrl + x → yes → press enter

5. WiringPi libraries

- http://wiringpi.com/download-and-install/
- SPI: https://projects.drogon.net/raspberry-pi/wiringpi/spi-library/
- Code example: https://learn.sparkfun.com/tutorials/raspberry-pi-spi-and-i2c-tutorial/all

6. Connexion to RaspbberyPi with Putty

1) Enable SSH on Raspberry: ***sudo raspi-config*** → Interfacing option → SSH → YES
2) Find IP address of Raspberry → ***hostname -I*** (capital i)
3) Connect via SSH (*Putty*) to the Raspberry → ***putty -ssh 172.22.22.67 -l pi -pw hypnosia***

7. GitHub first boot

1) ***sudo apt update***
2) ***sudo apt install git***
3) Clone repository → ***git clone https://github.com/GeraciGregory/HypnosiaController***
4) Config → ***git config user.email gregory.geraci@students.hevs.ch***
5) Status → ***git status***
6) Add files to be committed → ***git add <file or repository>***
7) Commit last changes → ***git commit -m "…"***
8) Push on github → ***git push origin master***
9) View all commits → ***git log***

8. GitHub update repository

1) ***git status***
2) ***git add <file or repository>***
3) ***git commit -m "blabla"***
4) ***git push origin master***
5) ***git log***

9. GPIO info's

1) ***gpio -v***
2) ***gpio readall***

D. MASTER software

Here is the complete MASTER class diagram .Also present on GitHub:

- https://github.com/GeraciGregory/HypnosiaController/blob/master/Documentation/Diagrams/RPI_diagrams.pdf

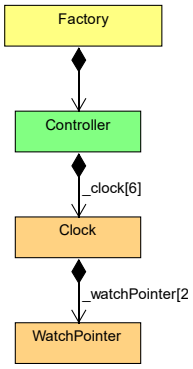


Figure 89 : MASTER simplified class diagram

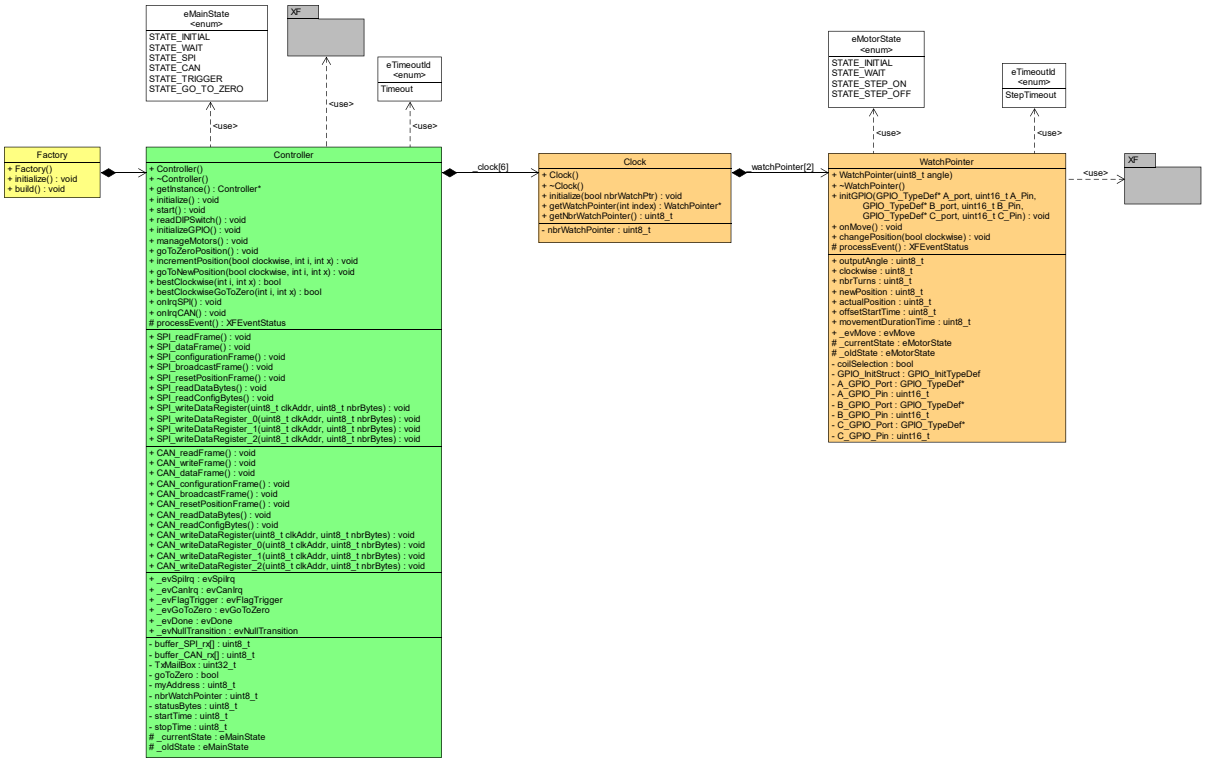


Figure 90 : MASTER detailed class diagram

All the code is present on GitHub:

- https://github.com/GeraciGregory/HypnosiaController/tree/master/RaspberryPiZero/HypnosiaController

E. GATEWAY/SLAVE software

Here is the complete MASTER class diagram .Also present on GitHub:

- https://github.com/GeraciGregory/HypnosiaController/blob/master/Documentation/Diagrams/STM32_diagrams.pdf

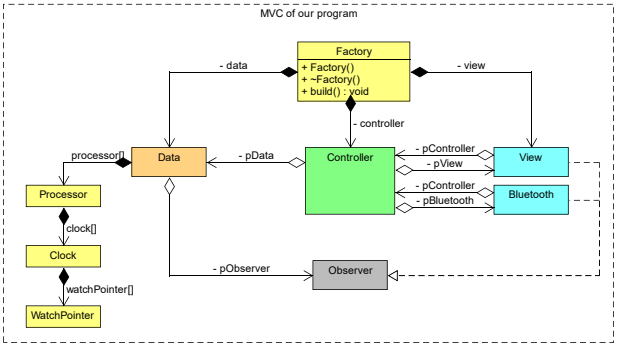


Figure 91 : GATEWAY/SLAVE simplified class diagram

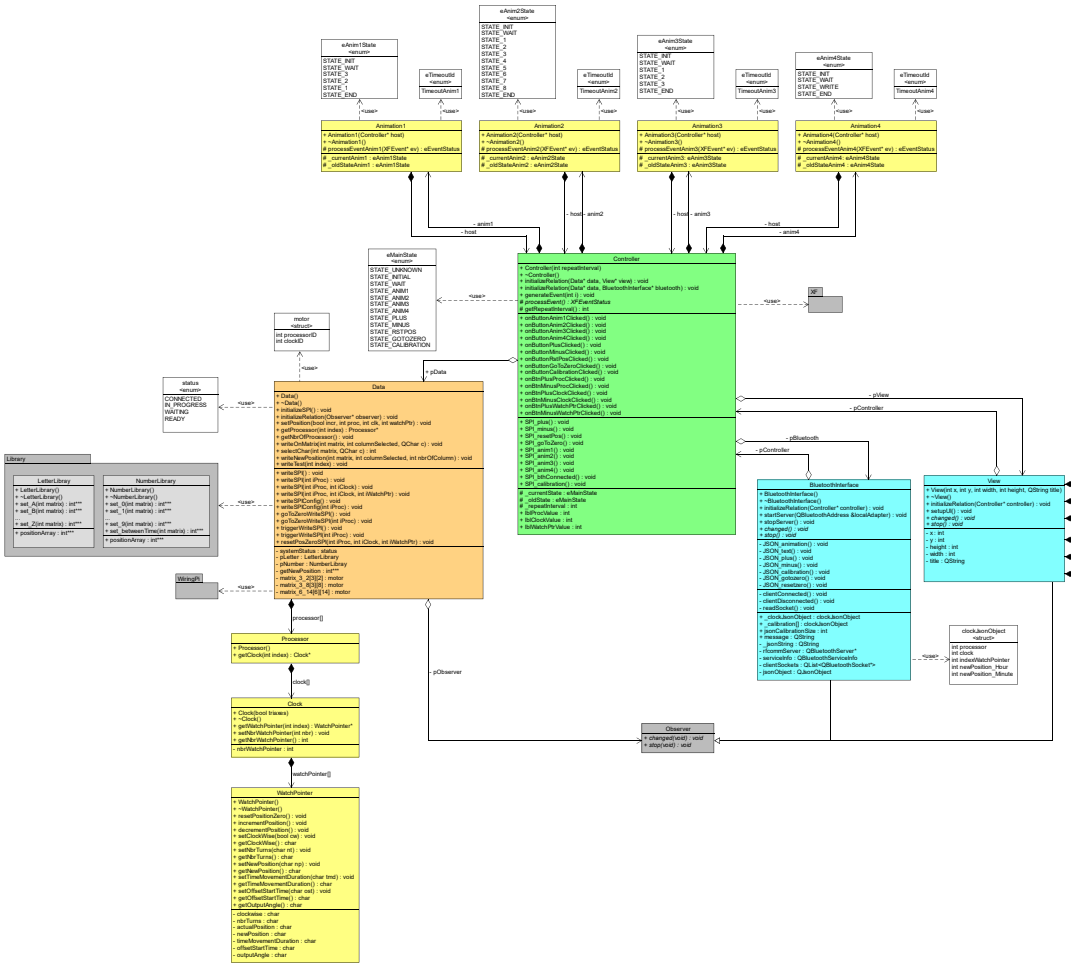


Figure 92 : GATEWAY/SLAVE detailed class diagram

All the code is present on GitHub:

- https://github.com/GeraciGregory/HypnosiaController/tree/master/STM32

F. *Altium schematic V1.0*

Schematic updated on GitHub.

- https://github.com/GeraciGregory/HypnosiaController/blob/master/Documentation/Altium/HYPNOSIA_controller_V1_CAN.pdf

Figure 93 : TOP view V1.0

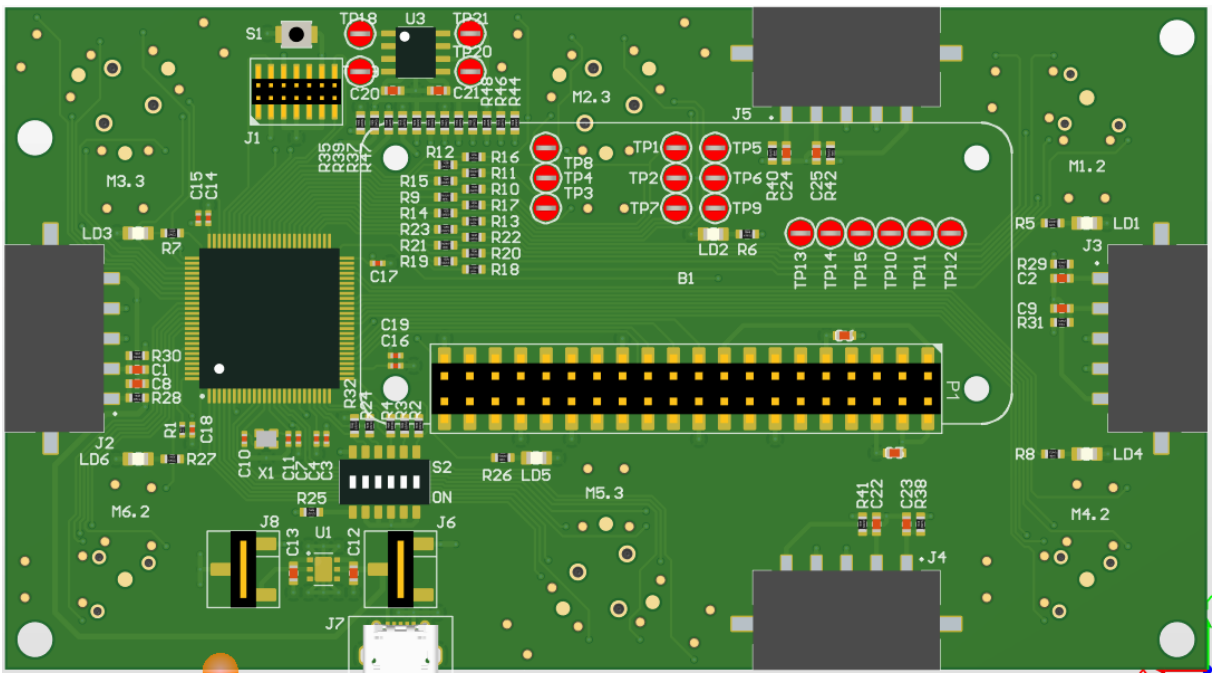


Figure 94 : BOTTOM view V1.0

G. Altium schematic V2.0

Schematic updated on GitHub.

- https://github.com/GeraciGregory/HypnosiaController/blob/master/Documentation/Altium/HYPNOSIA_controller_V2.pdf

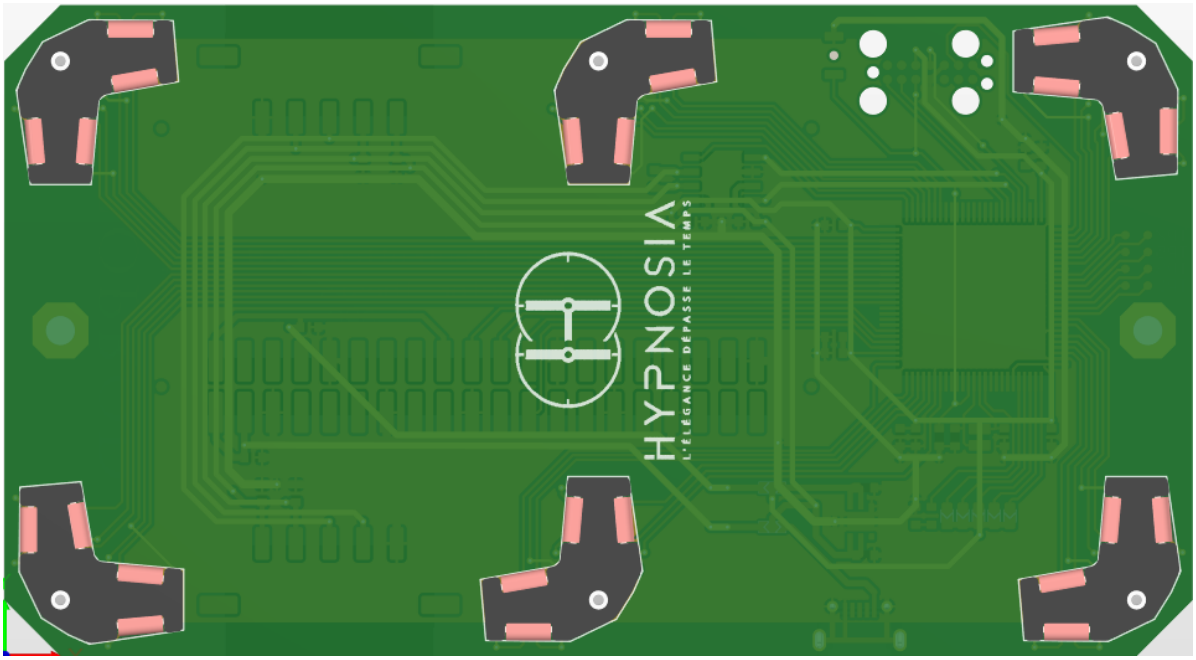


Figure 95 : TOP view V2.0

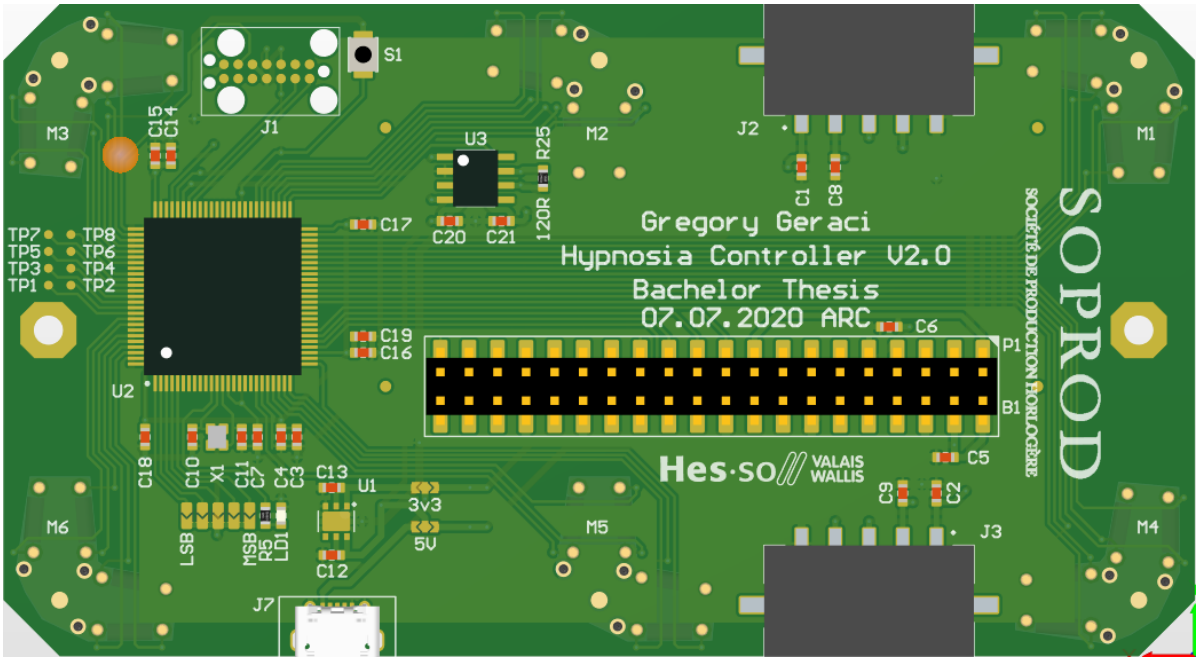


Figure 96 : BOTTOM view V2.0

H. JSON files

Here are all the JSON formats used for Bluetooth communication between the "HYPNOSIA Calibration" & "HYPNOSIA Controller" project".

```
{
  "header": "ANIMATION",
  "body": 1  //Select animation 1,2,3 or 4
}
```

Figure 97 : JSON file in order to select the animation

```
{
  "header": "TEXT",
  "body": "HYPNOSIA"  //Text to be written on the matrix
}
```

Figure 98 : JSON file in order to write a text

```
{
  "header": "GOTOZERO",
  "body": /  //Unused body
}
```

Figure 99 : JSON file in order to select the "GO TO ZERO" button

```
{
  "header": "PLUS", //Same for MINUS & RESETZERO header
  "body":
  {
    "processorID" : 0,     //0 to 13
    "clockID" : 0,         //0 to 5
    "watchpointerID" : 0, //0 or 1
    "moveWP1" : 180,       //0 to 359
    "moveWP2" : 90         //0 to 359
  }
}
```

Figure 100 : JSON file in order to select the "+", "_" or "RESET button"

```
{
  "header": "CALIBRATION",
  "body":
  [
    clock_jsonObject n,
    clock_jsonObject n+1,
    ...
    clock_jsonObject n+x
  ]
}


//clock_jsonObject
{
  "processorID" : 0,     //0 to 13
  "clockID" : 0,         //0 to 5
  "watchpointerID" : 0, //0 or 1
  "moveWP1" : 180,       //0 to 359
  "moveWP2" : 90         //0 to 359
}
```

Figure 101: JSON file in order to make the calibration of the matrix

I. Consumption

1. Processor consumption

Here is the information on the processor (STM32F072V8T6) datasheet:

Table 29. Typical and maximum current consumption from V_{DD} supply at V_{DD} = 3.6 V

Symbol	Parameter	Conditions	f_{HCLK}	All peripherals enabled				All peripherals disabled				Unit
				Typ	Max @ T_A(1)			Typ	Max @ T_A(1)			
					25 °C	85 °C	105 °C		25 °C	85 °C	105 °C	
I_{DD}	Supply current in Run mode, code executing from Flash memory	HSI48	48 MHz	24.3	26.9	27.2	27.9	13.1	14.8	14.9	15.5	mA
		HSE bypass, PLL on	48 MHz	24.1	26.8	27.0	27.7	13.0	14.6	14.8	15.4	
			32 MHz	16.0	18.3	18.6	19.2	8.76	9.56	9.73	10.6	
			24 MHz	12.3	13.7	14.3	14.7	7.36	7.94	8.37	8.81	
		HSE bypass, PLL off	8 MHz	4.52	5.25	5.28	5.61	2.89	3.17	3.26	3.34	
			1 MHz	1.25	1.39	1.58	1.87	0.93	1.06	1.15	1.34	
		HSI clock, PLL on	48 MHz	24.1	27.1	27.6	27.8	12.9	14.7	14.9	15.5	
			32 MHz	16.1	18.2	18.9	19.3	8.82	9.69	9.83	10.7	
			24 MHz	12.4	14.0	14.4	14.8	7.31	7.92	8.34	8.75	
		HSI clock, PLL off	8 MHz	4.52	5.25	5.35	5.61	2.87	3.16	3.25	3.33	

Figure 102: Processor consumption

We are in the "HSE bypass, PLL on" case at 48 [MHz].

There are still some values depending on the peripherals enable. They are either all on or all off. In our case we use only a small part of the peripherals. We are between the two values (in red). I arbitrarily set the consumption to **19 [mA].**

2. CAN converter (MCP2542) consumption

Here is the information on the MCP2542 datasheet:

VIO Pin						
Digital Supply Voltage Range	VIO	1.7	—	5.5	V	
Supply Current on VIO	IIO	—	7	20	µA	Recessive; VTXD = VIO
		—	200	400		Dominant; VTXD = 0V

Figure 103: MCP2542 consumption

In order to have margin, I take into account the worst-case scenario, i.e. **400uA**.

3. Motor consumption

Here is the information on the motor datasheet:

Motor consumption @ $T_0 = 2.9ms$	I_{mot}	µAs	-	3.8	4.2

Figure 104: Motor consumption

Here the values consumed by the motor are negligible [uAs] compared to the rest. I'm going to round up to **1mA** of consumption for the totality of the movements driven by a processor.

All the datasheets used in this project are present on GitHub:

- https://github.com/GeraciGregory/HypnosiaController/tree/master/Documentation/Datasheet

J. Price details

I would like to make it clear that some of the prices set may not be fully representative of reality. The purpose of these graphs is to get an idea of the price that such an object may be worth.

All numbers used to set costs are attached on GitHub for better visibility:

- https://github.com/GeraciGregory/HypnosiaController/blob/master/Documentation/HYPNOSIA_Prices_Chart.xlsx