TREBALL FI DE GRAU

Grau en Enginyeria Electrònica Industrial i Automàtica

PROGRAMACIÓ D'UN ALGORISME DE COMUNICACIÓ PER MODULACIÓ PPM PER A TRANSMISSORS ACÚSTICS PEL SEGUIMENT D'ESPÈCIES MARINES

Volum I

Memòria

Autor: Josep Mir Orfila

Director: Spartacus Gomariz Castro

Departament: EEL

Co-Director: David Sarrià Gandul

Convocatòria: Maig 2023

Programació de la comunicació per modulació PPM per a un TAG acústic per a la detecció d'espècies marines

Resum

En aquest treball es dissenya el programari necessari per a la comunicació entre dues etiquetes acústiques (TAGs) per a la localització d'objectes sota l'aigua, pretenent oferir una alternativa al dispositius comercials, que actualment no resulten adequats per al seguiment d'espècies marines . Es fan servir ones acústiques per a la comunicació, ja que es veuen menys atenuades que les ones electromagnètiques en un medi aquàtic.

S'han programat dos microcontroladors de la família STM32 de manera que un actuï de transmissor i l'altre de receptor, enviant-se missatges codificats fent servir un protocol basat en la PPM. També es detalla el maquinari necessari per a provar el funcionament d'aquests transmetent informació acústicament a través de l'aigua, fent servir transductors piezoelèctrics i els circuits adients per a comunicar els senyals entre els microcontroladors i aquests. S'ofereix també una explicació de les característiques dels transductors d'aquest tipus.

Per a la programació dels microcontroladors s'ha fet servir el llenguatge C, i a més s'ha creat una interfície programada en Python per al control del microcontrolador transmissor.

S'ha assajat exitosament el sistema amb dos muntatges diferents: en primer lloc, connectant directament ambdós microcontroladors amb un cable, per tal de comprovar el funcionament correcte del programari sense interferències externes. En segon lloc, realitzant un muntatge del maquinari per provar un exemple real de comunicació a través d'aigua. Els missatges assajats s'han transmès correctament i sense presentar interferències. De totes formes, al muntatge per cable s'han simulat interferències de manera artificial per provar la robustesa del receptor davant aquestes.

***Paraules clau**: transductor, piezoelèctric, modulació per posició de polsos, TAG, etiquetat acústic, microcontrolador, interfície, comunicació acústica subaquàtica*

Abstract

This paper features the development of the required software to allow communication between two acoustic tags for the localisation of underwater objects. It aims to offer an alternative to commercial devices, which are currently not sufficient for the tracking of

marine species. Acoustic waves are used for communication, as they are less attenuated than electromagnetic waves in an aquatic environment.

Two STM32 microcontrollers have been programmed so that one acts as a transmitter and the other one as a receiver. The communication between them uses a pulse-position modulation (PPM) based protocol. Also detailed is the required hardware to test the functionality of these devices via the acoustic transmission of information through water.

The microcontrollers have been programmed using C language. In addition, an interface programmed in Python has been created to control the transmitter.

The system has been successfully tested in two ways: first, by connecting both microcontrollers with a cable to verify the correct functioning of the program without external interference; secondly, by using hardware to mimic a verisimilar instance of underwater acoustic communication. The messages received were transmitted correctly and without interference. However, during the wired connection, interference was artificially simulated to test the robustness of the receiver against it.

***Keywords**: transducer, piezoelèctric, pulse position modulation, acoustic tags, microcontroller, interface, underwater acoustic communication*

Sumari

Sumari de figures

Sumari de taules

Sumari d'extractes de codi

1. Introducció

L'estudi dels mars i oceans i la seva fauna està en avançament constant, i té una gran importància en alguns àmbits industrials. Al sector pesquer, és especialment rellevant el control de poblacions animals per a conèixer l'impacte de l'explotació sobre l'ecosistema marí i conseqüentment millorar les pràctiques d'explotació per tal de preservar els seus recursos.

D'aquesta necessitat d'estudi del mar surten varies aplicacions tecnològiques. Una d'elles són els TAGs: dispositius d'etiquetat electrònic que permeten la transmissió a distància d'informació sobre els exemplars observats, normalment consistint d'un identificador del dispositiu i alguna dada complementària, mesurada per algun sensor integrat [1]. La informació es transmet codificant-la dins ones sonores seguint algun protocol de modulació, degut a la impossibilitat de fer servir ones electromagnètiques en el medi aquàtic. A dia d'avui, aquests TAGs són distribuïts comercialment per empreses com VEMCO [1], Lotek [2] o HTI [3]. De totes maneres, els models existents no satisfan els requeriments de les investigacions dutes a terme pels científics actualment. Si, per exemple, es vol detectar la posició d'un exemplar animal que ha estat etiquetat, només és possible fer-ho amb els models de més altes prestacions, utilitzant tècniques de triangulació; així i tot, segueix sent necessari un mínim de tres receptors [4, 5], la resolució dels quals deixa molt a desitjar: arriba a un màxim de cinc metres en condicions òptimes, difícils d'aconseguir en una situació real (a grans profunditats) arran la dificultat de posicionar els receptors correctament [6].

El projecte SASES (Sistemes acústics submarins per a la monitorització del comportament espacial d'espècies), desenvolupat pel SARTI-UPC, té com a objectiu crear dispositius que millorin les prestacions i funcionalitats ofertes actualment per les solucions comercials. Actualment s'està treballant en el desenvolupament d'un TAG que permeti la comunicació bidireccional sense afectar la resolució. La bidireccionalitat permetria reduir els costos de desplegament d'infraestructura, ja que, per tal de localitzar un animal etiquetat, no caldria posicionar diversos receptors a diversos punts de la mar com es fa actualment, sinó que es podria fer servir un sol receptor instal·lat en un vehicle autònom que es mogui per l'àrea d'interès, realitzant així l'adquisició des de diferents punts en l'espai i amb un únic dispositiu.

2. Objectius

En les etapes inicials del projecte SASES, s'ha desenvolupat el hardware necessari per a un TAG bidireccional [7]. Per tal de provar el funcionament d'aquest, és necessari un algorisme que dugui a terme la comunicació seguint algun tipus de protocol. En aquest treball es proposa dissenyar un programa que imiti el procediment d'una etiqueta de VEMCO, aconseguint així la compatibilitat amb aquests.

Els TAGs tradicionals de VEMCO, segons han observat membres del SARTI, fan servir una codificació mitjançant la Modulació per Posició de Polsos (d'ara endavant PPM, de l'anglès *Pulse Position Modulation*). Amb aquesta modulació es crea un protocol de comunicació que consisteix en l'enviament d'una sèrie de trens de polsos, on es codifica una cadena de bits, en la qual el valor de cada dos dígits ve donat pel temps passat entre dos d'aquests trens.

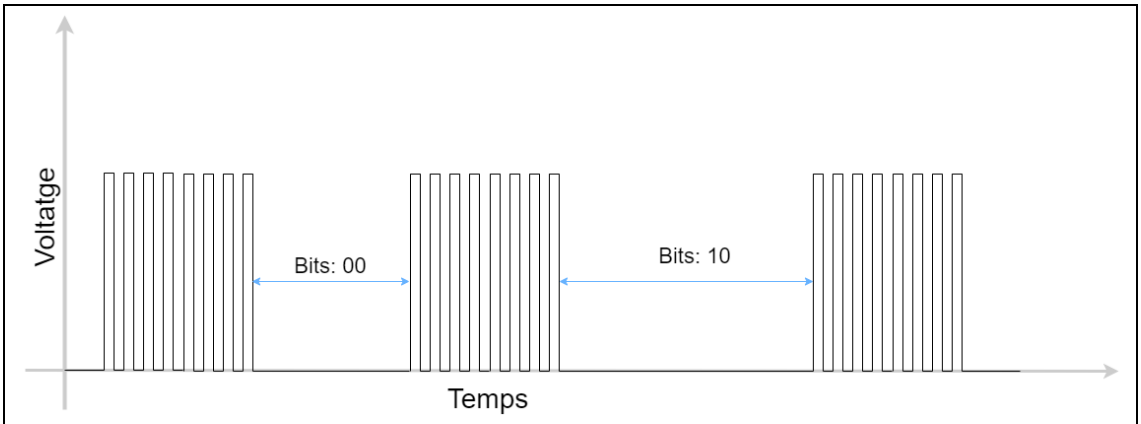

Figura 1. *Exemple d'una cadena de 4 bits codificada en PPM*

L'objectiu del present treball és dur a terme la comunicació exitosa entre dos dispositius a través del medi acuàtic, enviant un missatge codificat dins un senyal acústic que serà emès per un transductor piezoelèctric i rebut per un altre. Es faran servir dos microcontroladors: un ha de generar un senyal, on estarà codificat un missatge (s'anomenarà d'ara endavant transmissor); l'altre l'ha de rebre i descodificar correctament (s'anomenarà d'ara endavant receptor). Per a fer arribar la informació d'un microcontrolador a l'altre, es compta amb uns circuits condicionadors que adapten el senyal de la sortida del transmissor per a ser reproduït per un altaveu; i, el senyal de la sortida d'un micròfon a l'entrada del receptor. Aquest treball es centra en el programari

d'aquests dos microcontroladors, encara que s'hi explicarà breument el maquinari utilitzat.

Es programaràn dos microcontroladors de la família STM32, que actuaran respectivament com a emissor i receptor. Entre ells, es transmetran missatges fent servir un protocol establert amb PPM. L'efectivitat de la comunicació es comprovarà enviant un missatge de dues maneres: mitjançant polsos per cable i mitjançant ones sonores, transmeses per aire o per aigua.

A l'inici de la tasca es disposava ja d'un microcontrolador amb un software ja instal·lat, proporcionat per un membre del SARTI, que du a terme part de la transmissió. Concretament, permet l'enviament de cadenes de trens de polsos, controlat per comandes comunicades mitjançant USB, podent programar per avançat el temps passat entre els trens per a poder enviar un missatge. En un principi, pot ser operat manualment enviant les comandes necessàries amb un software de comunicació serial (com pot ser RealTerm), però el procés requereix enviar un nombre extens de comandes i la realització de certs càlculs per part de l'operador. Per tant, es desenvoluparà un programa en llenguatge Python que faci l'operació del transmissor molt més senzilla, on l'usuari simplement introdueixi els bits que es volen enviar i el software s'encarregui d'enviar totes les comandes corresponents.

A més, es programarà des de zero un algorisme capaç de rebre i descodificar els senyals transmesos en el protocol que s'ha fixat, comunicant per USB els polsos rebuts, el temps des del darrer pols i, si el missatge és correcte, la informació rebuda descodificada. També ha de ser capaç de detectar si els senyals que rep no són consistents amb el protocol i classificar-los com a erronis. Aquest algorisme serà un programari en llenguatge C i serà carregat a un altre microcontrolador STM32.

Com ja s'ha mencionat prèviament, els sistemes es provaran de dues maneres. Primer, es realitzarà l'enviament de polsos per cable, connectant el pin d'entrada del receptor al pin de sortida del transmissor. Una vegada s'hagi comprovat el funcionament d'aquesta manera, es realitzaran proves condicionant el senyal per poder ser transmès auditivament a través de transductors piezoelèctrics, i es provarà la comunicació enviant aquest senyal a través de l'aigua.

En resum, els objectius són:

1. Implementar un programa en llenguatge C per a un microcontrolador que sigui capaç de rebre i descodificar correctament missatges codificats amb PPM seguint un protocol establert.
2. Aconseguir que el programa sigui capaç d'identificar senyals inconsistents amb el protocol com erronis.
3. Implementar una interfície en llenguatge Python que controli, mitjançant l'enviament de comandes, el microcontrolador programat per a la transmissió del que ja es disposa.
4. Comprovar la comunicació correcta entre els dispositius transmissor i receptor transmetent missatges per cable.
5. Comprovar la correcta detecció per part del receptor de senyals erronis.
6. Comprovar la comunicació correcta entre els dispositius transmissor i receptor transmetent missatges acústicament per aigua fent ús de transductors piezoelèctrics.

3. Eines, protocols, maquinari i programari utilitzats

3.1. La PPM i el protocol de comunicació

Un TAG que fa servir la PPM transmet una sèrie de polsos, i cada sèrie defineix un valor numèric en format binari. Aquest valor queda codificat en els intervals de temps entre els polsos successius [8].

Per a assegurar la compatibilitat amb els TAGs de VEMCO, el protocol que es farà tindrà les següents característiques:

- Els missatges que s'enviaran seran de 14 bits: 6 bits consistiran en un identificador (ID) del TAG transmissor, i 8 que contindran una dada (D):

ID5	ID4	ID3	ID2	ID1	ID0	D7	D6	D5	D4	D3	D2	D1	D0

Figura 2. *Disposició dels bits d'un missatge*

- L'ona transmesa consistirà en una sèrie de trens de polsos. Aquests polsos tindran una duració de 5ms i es produiran a una freqüència de 69 kHz, la qual s'ha escollit prenent com a referència els TAGs comercials [7] i el protocol obert de l'European Tracking Newtork [8].
- L'interval de temps entre el final d'un tren i el principi d'un altre indicarà el valor d'un parell de bits del missatge. Llavors, un missatge es compondrà de set trens de polsos més un darrer pols que indicarà el final del missatge. Els bits es transmetran de més significant a menys significant, començant per l'ID i acabant per la Dada. A la taula 1 s'indiquen els valors de temps que equival a cada valor binari.

Temps entre tren de polsos	Equivalència en bits
50 ms	00
100 ms	01
150 ms	10
200 ms	11

Taula 1. *Equivalència entre els intervals entre polsos i els bits enviats*

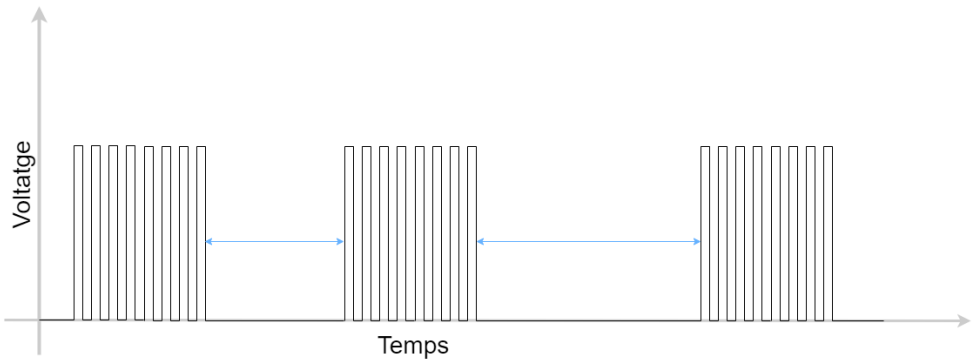

Figura 3. *Representació del temps entre polsos*

3.2. La família de microcontroladors STM32

Els microcontroladors utilitzats per al desenvolupament del projecte han estat tots de la família de STM32. Aquests controladors disposen d'un controlador de 32 bits amb un processador Arm Cortex®-M [9]. Encara que a l'inici del desenvolupament es van fer servir plaques amb microcontroladors STM32G031K8TX i STM32F767ZITX, finalment s'ha optat pels microcontroladors de la branca STM32L4, concretament el STM32L432KCUX. "[The STM32 family of 32-bit microcontrollers] offers products combining very high performance, real-time capabilities, digital signal processing, low-power / low-voltage operation, and connectivity, while maintaining full integration and ease of development" [9]. La migració d'un codi a un altre microcontrolador no presenta obstacles, ja que el programa fa ús de pocs perifèrics i la tasca d'adaptar-los entre microcontroladors és senzilla: només consisteix en trobar els perifèrics del mateix tipus al nou controlador.

S'ha triat un microcontrolador d'aquesta família perquè estan dissenyats per a tenir un rendiment òptim i poques fuites de potència [10]. D'aquesta manera, s'assegura la durabilitat de la bateria que s'assigni al microcontrolador en un futur dispositiu.

3.3. La STM32CubeIDE

Per a facilitar-ne la programació, els propis fabricants dels microcontroladors STM32 proporcionen un IDE (entorn de desenvolupament integrat, o *integrated development environament*, en anglès) pensada per a aquests, anomenada STM32CubeIDE. Disposa d'una interfície gràfica que permet ajustar la configuració dels perifèrics del microcontrolador d'una manera senzilla, escrivint el codi necessari per a inicialitzar els seus paràmetres de forma automàtica. Té l'opció de baixar la informació de cada microcontrolador o placa de desenvolupament que ofereix l'empresa, que permetrà a l'IDE mostrar una representació gràfica del controlador corresponent, fent possible per a l'usuari triar quin perifèric s'associa a cada pin, és a dir, si vol que actuï d'input, output, de línia per a interrupció externa, port de comunicació UART, entrada de l'ADC i moltes opcions més. També es poden configurar els paràmetres de tots els perifèrics, vagin associats a un port com els que s'han mencionat, o siguin interns al funcionament, com els temporitzadors TIM.

En crear un projecte, es pot accedir a "Device configuration tool", que mostra una representació gràfica del microcontrolador que s'està fent servir. Allà s'hi troba una llista de perifèrics que es poden configurar. També es pot clicar damunt els pins per configurar-los. Accedint a la pantalla "clock configuration" es pot observar una representació dels temporitzadors del controlador, que s'haurà de tenir en compte al programar els temporitzadors.

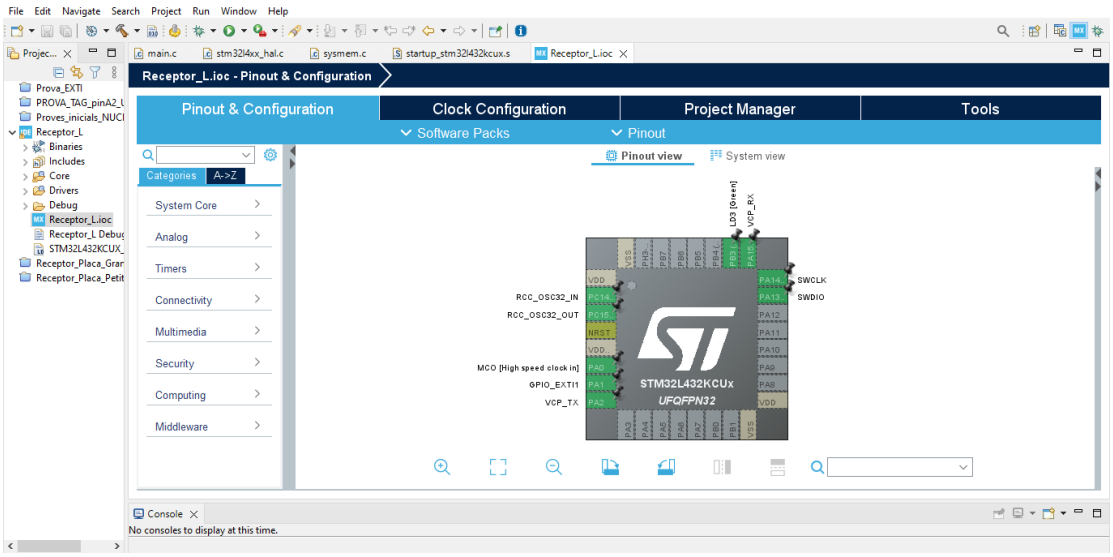

Figura 4. *Vista de la "Device configuration tool" a la STM32CubeIDE*

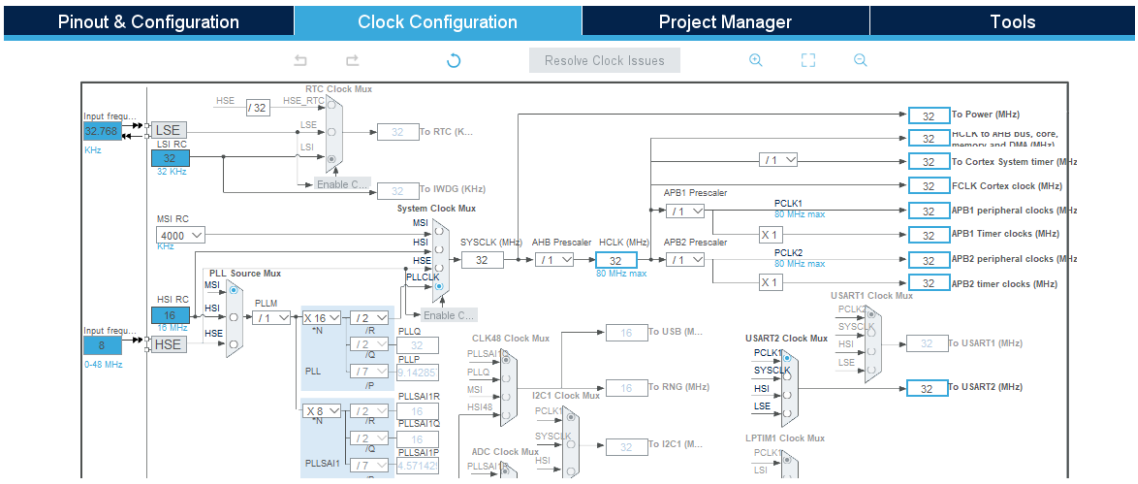

Figura 5. *Vista de la "clock configuration" a la STM32CubeIDE*

Una vegada programats els perifèrics, si es guarda el projecte l'IDE generarà automàticament alguns fitxers de codi. Navegant fins a aquests fitxers es poden editar per a programar els algorismes necessaris. Encara que altres també es poden, es treballa principalment a core/source/main.c. Per tant, el llenguatge C és el principal de la part del projecte corresponent al receptor.

3.4. Perifèrics de l'STM32 utilitzats per al programa

A continuació s'explicarà el funcionament dels perifèrics que s'han fet servir per al desenvolupament del programa receptor. Aquest funcionament és aplicable als STM32L4 i pot ser-ho també a altres microcontroladors de la família, encara que pot presentar-hi variacions.

La manipulació dels perifèrics es pot dur a terme a través de la capa d'abstracció de hardware (*Hardware Abstraction Layer* o *HAL*). "The HAL offers high-level and feature-oriented APIs, with a high-portability level. They hide the MCU and peripheral complexity to end-user" [11].

3.4.1. EXTI

"The EXTI is not considered as a standalone peripheral but rather as a service used by other peripheral, that are handled through EXTI HAL APIs" [12 p. 28]. El servei EXTI permet realitzar una operació quan es detecta un flanc en algun dels pins. Per a configurar-lo, cal seleccionar algun dels pins i triar l'opció GPIO_EXTIx ("x" generalment serà el nombre del pin). Això farà que els flancs detectats a aquell pin creïn una interrupció i consegüentment s'executi una rutina associada.

A la categoria GPIO, si es tria el pin d'interrupció es pot configurar com es crea la interrupció (en flanc de pujada o baixada) i si el pin ha de fer servir la resistència interna de *pull-up* o *pull-down*. A l'apartat NVIC s'ha de seleccionar la casella "enable" per permetre a la interrupció funcionar [13].

Per a programar l'acció realitzada quan es crida la interrupció, al fitxer *main.c* s'ha d'afegir una funció a l'apartat /*USER CODE BEGIN 4*/ amb la declaració void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin). Aquesta funció s'executarà cada vegada que EXTI detecti una interrupció a qualsevol pin, pel que cal comprovar que aquesta hagi estat creada pel pin desitjat. La millor manera de fer això és afegir un *if* que comprovi GPIO_Pin==GPIO_PIN_x (on x és el nombre de pin) i dins aquest *if* programar les accions corresponents.

3.4.2. TIM

"The TIM peripheral is a multi-channel timer unit, available in various configurations, depending on the instance used. There are basically following categories: advanced-

control timers, general-purpose timers and basic timers" [14]. Aquest perifèric permet al microcontrolador mesurar temps en segon pla sense ser interromput per altres funcions ni bloquejar l'execució d'altres instruccions. A la taula 2 es poden trobar els diferents temporitzadors dels quals disposa aquest perifèric a un dispositiu STM32L4x i les seves característiques.

Timer type	Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/ compare channels	Complementary outputs
Advanced control	TIM1, TIM8	16-bit	Up, down, Up/down	Any integer between 1 and 65536	Yes	4	3
General-purpose	TIM2, TIM5	32-bit	Up, down, Up/down	Any integer between 1 and 65536	Yes	4	No
General-purpose	TIM3, TIM4	16-bit	Up, down, Up/down	Any integer between 1 and 65536	Yes	4	No
General-purpose	TIM15	16-bit	Up	Any integer between 1 and 65536	Yes	2	1
General-purpose	TIM16, TIM17	16-bit	Up	Any integer between 1 and 65536	Yes	1	1
Basic	TIM6, TIM7	16-bit	Up	Any integer between 1 and 65536	Yes	0	No

Taula 2. *Timers disponibles a un STM32L4X [15]*

Per a configurar un temporitzador, a *Device configuration tool* se selecciona la categoria *Timers* i se'n tria un. Allà es poden configurar els paràmetres que pertoquin. Generalment, s'ha d'activar la interrupció al NVIC i a continuació triar un valor per al *prescaler* i el període, que marcaran cada quant temps es produeix una interrupció. El *prescaler* és un nombre enter pel que es dividirà el rellotge del sistema HCLK, alterant la freqüència a la que es produeix un pols del temporitzador, mentre que el període indica cada quants polsos es produeix una interrupció. A la pestanya *clock configuration* es pot consultar i fins i tot modificar la freqüència del HCLK. S'ha de tenir en compte que hi ha que restar 1 al valor que es vulgui tenir tant de *prescaler* com de període, ja que el microcontrolador no contempla un valor de 0.

Cada vegada que es dispari la interrupció per un *overflow* del temporitzador, s'executarà la funció void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) , que cal

declarar a l'apartat *User code 4* al fitxer *main.c*. Igual que amb l'EXTI, cal comprovar amb un *if* que la interrupció hagi estat generada pel temporitzador que correspongui.

3.4.3. NVIC

El *Nested Vectored Interrupt Controller* decideix quines interrupcions estan habilitades i quin nivell de prioritat tenen, podent controlar fins a 16 nivells de prioritat [15]. Per tant, sempre que s'utilitza un perifèric que fa servir una interrupció se n'ha de configurar la prioritat al NVIC.

3.4.4. UART

Aquest mòdul s'encarrega de la comunicació serial. En aquest projecte es fa servir per a enviar i rebre caràcters al PC [16]. A l'eina de configuració es pot habilitar el dispositiu i modificar-ne els paràmetres.

Per a enviar un *string* per USB, es fa servir la funció del HAL *HAL_UART_Transmit.* Aquesta funció demana el perifèric d'UART que es vol emprar per tractar la comunicació, un punter a una variable que faci de memòria intermèdia on està guardat el missatge (el punter ha de tenir un format *uint8_t*, la llargària del missatge que es vol enviar, i un temps límit, el *timeout*, per a la transmissió).

Com a memòria intermèdia es fa servir una variable anomenada *msg* (de l'anglès *message*). Es declara com un vector de *chars*, en el cas d'aquest projecte, de 50 caràcters. Amb un punter a *msg*, la funció del HAL és capaç de llegir els caràcters que contengui el vector.

La manera en què s'utilitza aquesta funció en aquest projecte és fent ús d'una variable tipus *uint8_t* anomenada *msglen*, on es guarda la llargària del *string* que s'enviï. Amb la funció *sprintf* del llenguatge C es pot introduir el *string* formatat (tal i com es formataria amb *printf*) dins la memòria intermèdia. A més, la funció retorna la llargària d'aquest, així que es pot guardar dins *msglen* per a usar-se després. D'aquesta manera, en cridar HAL_UART_Transmit es pot fer servir de paràmetre. Això es veu exemplificat a l'extracte de codi 1:

```
376
377                 msglen=sprintf(msg, "ERROR, s'ha rebut un temps invalid\r\n");
378                 HAL_UART_Transmit(&huart2, (uint8_t*)msg, msglen, 10);
379
```

***Extracte de codi 1**. Enviament d'un missatge via UART*

3.5. RealTerm

RealTerm ha estat una eina útil per a la comprovació del funcionament del codi, encara que és substituïble per qualsevol altre eina que permeti la comunicació serial amb un dispositiu per USB. "RealTerm is an engineers terminal program specially designed for capturing, controlling and debugging binary and other difficult data streams. It is the best tool for debugging comms"[17].

3.6. La llibreria Pyserial

Pyserial és una llibreria de Python que permet la comunicació serial a partir d'un programa en aquest llenguatge. És útil si s'han d'enviar moltes comandes o automatitzar processos que impliquen la comunicació serial amb un altre dispositiu.

3.7. Els transductors piezoelèctrics

Per a la comunicació acústica, s'han fet servir dos transductors idèntics: un funciona com altaveu i l'altre com a micròfon. Ambdós han d'estar composts d'un material piezoelèctric que pugui generar una freqüència sonora en resposta a un senyal elèctric i també fer el procés a la inversa (de senyal elèctric a acústic). El material i les dimensions del transductor han d'estar dissenyades de manera que la freqüència de ressonància del dispositiu resultant sigui propera als 69 kHz. També s'ha de tenir en compte que les seves dimensions han de ser prou petites per anar muntades en un espècimen animal, que és la funció final que tindrà, i a més ha d'anar muntat dins una càpsula que l'aïlli de l'aigua.

El material més utilitzat en la fabricació de transductors piezoelèctrics ceràmics és el titanat de zirconat de plom (abreujat PZT), ja que entre els materials capaços de generar una càrrega elèctrica en ser deformats (i viceversa), presenta una alta sensitivitat i una temperatura d'operació alta, en comparació amb altres materials com el titanat de bari ($BaTiO_3$). La fórmula bàsica del PZT és $Pb[Zr_{(x)}Ti_{(1-x)}]O_3$. La proporció de zirconi i titani s'ajusta per a obtenir unes propietats idònies per a cada propòsit i, a més, existeix la possibilitat de dopar el material per a aconseguir altres efectes [18]. Generalment, es

classifiquen en ceràmiques blanes i dures. Les primeres solen ser utilitzades en sensors per les seves amplades de banda més àmplies i majors desplaçaments en resposta a la tensió, mentre que les segones tenen un factor d'acoblament més baix i són resistents a temperatures més altes, pel que són útils per a aplicacions de potència [19].

Algunes de les formes més comuns per als transductors ceràmics d'aquest tipus són els discs, els anells, les plaques i els cilindres [20]. A partir de la forma, les dimensions i la constant de freqüència del material de cada piezoelèctric, es pot calcular la freqüència de ressonància d'aquest seguint les fórmules indicades a la figura 7. S'ha de tenir en compte que la freqüència varia segons l'eix del piezoelèctric en què vibri.

De totes maneres, aquesta també es pot trobar empíricament mitjançant un escombratge de freqüències. La freqüència a la qual el transductor presenti una impedància mínima serà la de ressonància; aquella on sigui màxima, serà la freqüència de ressonància paral·lela, que serà sempre més alta que l'altra [21].

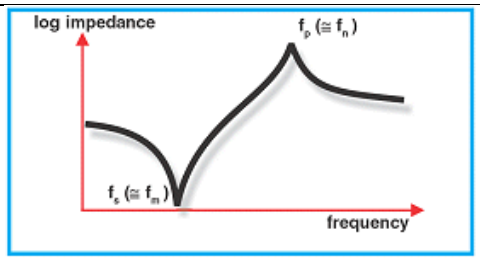

Figura 6. *Relació entre la freqüència i impedància d'un piezoelèctric, on fs és la freqüència de ressonància i fp la de ressonància paral·lela [21]*

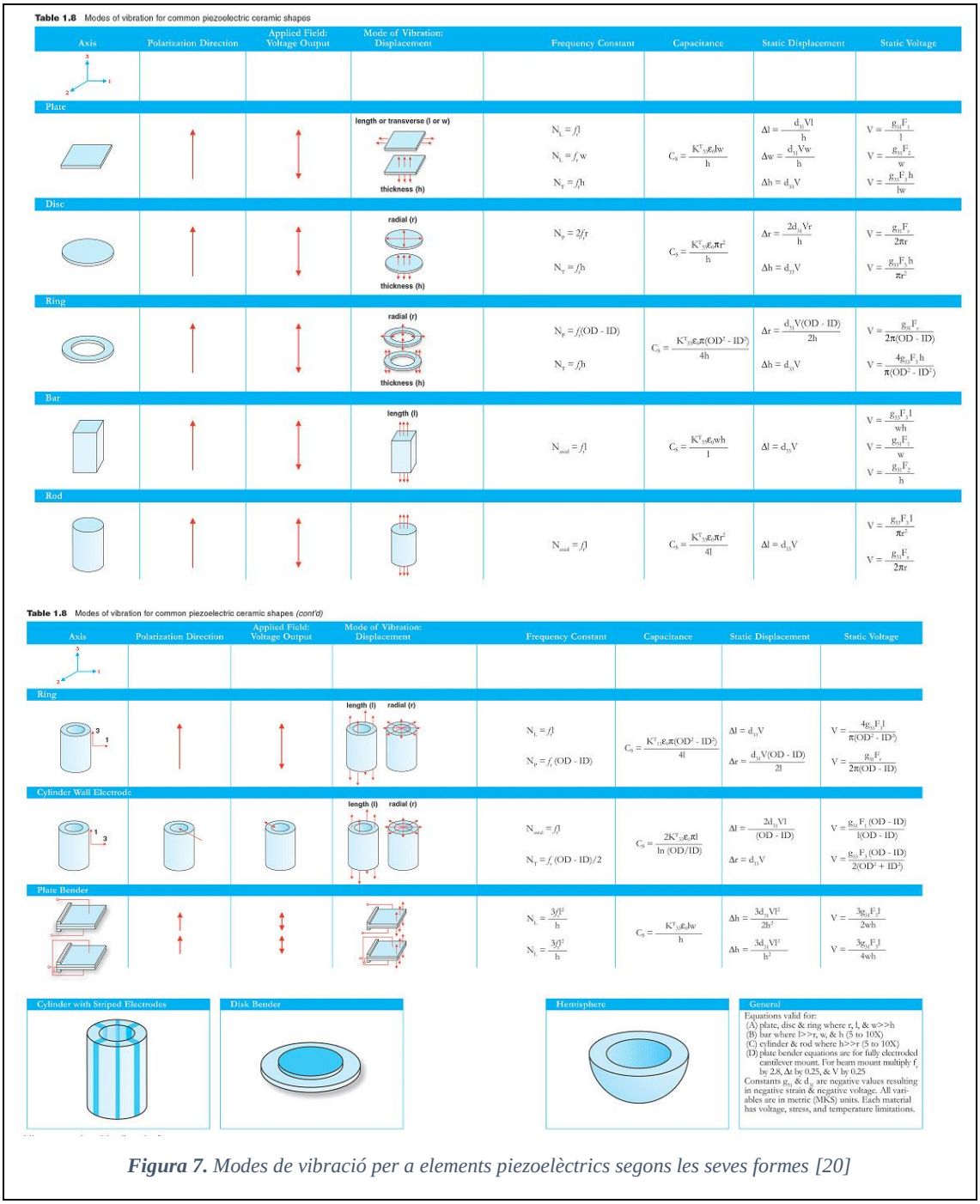

Figura 7. *Modes de vibració per a elements piezoelèctrics segons les seves formes [20]*

El piezoelèctric que s'ha triat per a la present aplicació és de forma cilíndrica, amb diàmetres extern i intern de 14 mm i 12 mm respectivament, i 10 mm de longitud, dimensions amb les quals presenta una freqüència de ressonància de 78 kHz radial, 1823 kHz a la paret, i 150 kHz axial, de les quals la primera és la més propera a la desitjada de 69 kHz. La ceràmica utilitzada és del tipus PZ26, que presenta el menor nombre de variacions en la fabricació de la indústria i, per tant, és un material estable amb un

comportament consistent i dissenys estandarditzats. També presenta poc desgast en el temps, baixes pèrdues dielèctriques i una temperatura d'ús alta. És àmpliament utilitzat en transductors subaquàtics i hidròfons. El piezoelèctric s'ha encapsulat amb una resina epoxy de duresa D85 per a garantir la resistència del transductor a l'aigua.

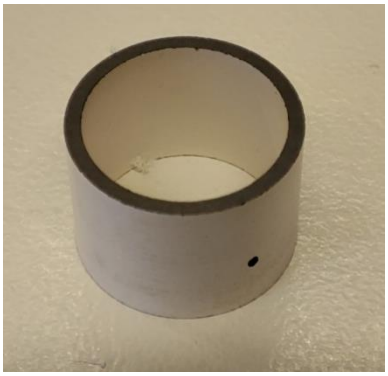

***Figura 8**. Piezoelèctric utilitzat com a transductor acústic en aquest treball*

***Figura 9.** Encapsulat i cablejat del transductor piezoelèctric*

4. Programació del programa receptor

La clau del programa receptor rau en tres aspectes principals: detectar els polsos, mesurar el temps entre ells i descodificar aquest temps en bits.

4.1. Planificació de l'algorisme

En el seu estat inicial, el programa no realitza cap funció i el temporitzador està aturat fins que el servei EXTI detecta un flanc al pin d'entrada. Quan passa això, el temporitzador s'inicialitza, o bé, es reinicia si ja estava funcionant. Aquest temporitzador està programat perquè a una variable *uint8_t* anomenada *tempsActual* s'hi pugui consultar el temps des del darrer flanc amb una precisió d'1ms. Si ja havia començat a comptar, abans de reiniciar el temporitzador es guarda el temps actual en una altra dada *uint8_t* anomenada *tempsPassat*, que guarda el temps des del darrer flanc en el moment que es detecta un nou pols, per tal de poder-lo processar sense ser interferit per valors canviants. Si no s'havia començat a comptar, el valor que s'obtindrà és 0, que és el valor d'inicialització de *tempsActual*.

Una vegada s'ha obtingut el temps entre dos polsos, s'ha de descodificar la informació que aquest indica. A la taula 1 es pot consultar el protocol establert. Si el temps obtingut es troba dins els marges acceptables (és a dir, entre 50 ms i 200 ms, amb un marge d'error de 10 ms), es passarà a una funció anomenada *descodificador* que interpretarà el seu significat en bits i el guardarà dins la variable que correspongui. En canvi, si el temps detectat entre polsos és menor que 40 ms, això s'interpretarà com que el controlador ha rebut dos polsos consecutius d'un tren de polsos, i serà ignorat pel descodificador. Per altra banda, si en qualsevol moment el temporitzador indica un temps passat major que 210 ms, s'interpretarà com que s'ha perdut informació i s'esborraran les dades rebudes fins al moment que pertanyin a la cadena de 14 bits que s'estava completant, i a continuació s'aturarà el timer i es reiniciarà *tempsActual*.

Les dades que siguin considerades vàlides a la rutina d'interrupció i siguin conseqüentment interpretades a la funció descodificadora, s'aniran guardant per parells de bits. Els 6 primers bits s'estojaran a una variable corresponent a l'identificador, anomenada *IDrebut*, i els altres 8 a una variable que indicarà una dada, anomenada *dadaRebuda*. Ambdues variables són de tipus *int*.

Finalment, si no hi ha hagut cap interrupció de la recepció o cap temps entre polsos invàlid, el microcontrolador comunicarà per USB l'identificador i la dada rebuda ja descodificats. De tota manera, per a facilitar la detecció d'errors, cada vegada que es rebi un tren de polsos nou es comunicarà per USB el temps passat des del darrer pols vàlid, així com quants polsos considerats part de la mateixa cadena s'han rebut fins al moment.

El format de missatge que comunica la recepció d'un tren de polsos és [$PPM,NNN,TTT], on NNN indica el nombre de trens part de la mateixa cadena rebuts i TTT el temps des del darrer pols, en format decimal. Per a la comunicació d'un missatge complert rebut, el format és [$TAG,III,DDD], on III indica l'indicador i DDD la dada rebuts, en format hexadecimal.

4.2. Recepció de polsos: les interrupcions EXTI

Per a detectar l'arribada de polsos, es fa el servei EXTI, que és capaç de detectar flancs en un pin determinat. El pin escollit és el PA1 (marcat com pin A1 a la placa). A l'eina de configuració del dispositiu, es selecciona sobre la representació gràfica del microcontrolador aquest pin i es tria l'opció GPIO_EXTI1. A continuació, a la categoria *System Core>GPIO* es selecciona el PA1. Es desplegarà una sèrie d'opcions amb les que es pot configurar el funcionament de la interrupció.

Per defecte el senyal del transmissor està en nivell baix, i els polsos s'envien en nivell alt. Al pin s'haurà de detectar, per tant, un dels dos flancs possibles (ascendent o descendent). En aquest cas s'ha triat l'ascendent. Per tant es selecciona "*External Interrupt Mode with Rising edge trigger detection"* al *GPIO Mode.* Es selecciona també *GPIO pull-down,* al ser el senyal per defecte quan no es rep un pols de nivell baix.

A la pestanya *NVIC* s'activa la interrupció marcant la casella que li correspon. D'aquesta manera el funcionament del servei EXTI ja queda configurat.

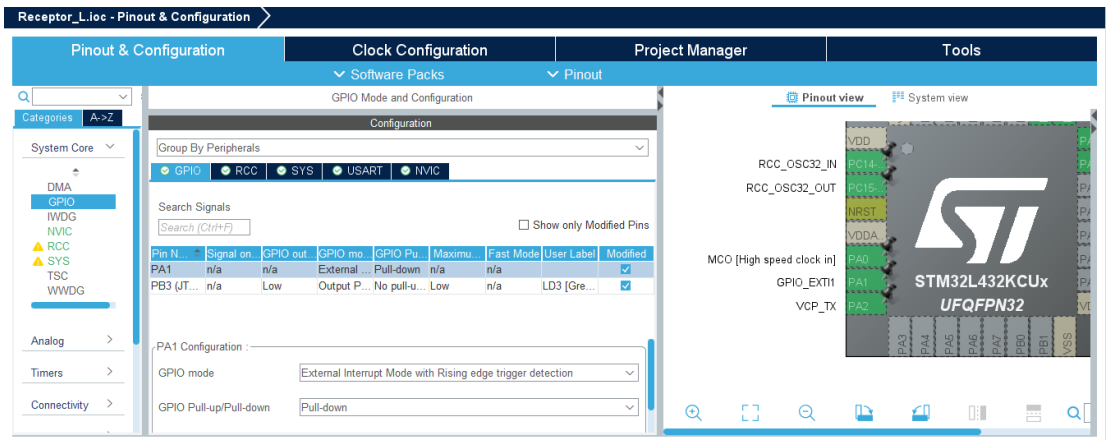

Figura 10. *Configuració del servei EXTI*

Una vegada s'ha fet això, s'ha de configurar la rutina que realitzarà el microcontrolador quan rebi una d'aquestes interrupcions. Per això, s'ha d'afegir una funció anomenada *HAL_GPIO_EXTI_Callback* al fitxer *main.c*, dins la qual es programarà tot el que es vulgui executar quan es detecti un flanc.

```
308 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
309
310     if(GPIO_Pin == GPIO_PIN_1) {
311         //Aquí es guarda el temps passat des del darrer pols
312         HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_3);
313         tempsPassat=tempsActual;
314         //Comunica temps per uart
315         comptapolsos++;
316         tempsActual=0;
317
```

Extracte de codi 2. *Accions que realitza el codi al activar-se una interrupció externa (primera part)*

El primer que es fa dins la funció de *callback* és comprovar amb un *if* que la interrupció ve del pin 1 i no d'un altre GPIO. Si es compleix aquesta condició, es guarda el valor de *tempsActual* a *tempsPassat.* A continuació, es reinicialitza el valor de *tempsActual* a 0. Després, es comprova si el temporitzador funciona. Això ho indica una variable anomenada *timercorre*, que durant tot el programa es tracta de manera que el seu valor sigui 0 si aquest està aturat i 1 si no ho està. S'utilitza un *if* de manera que si *timercorre* és 0, s'executi la instrucció *HAL_TIM_Base_Start_IT(&htim7)* que inicialitza el temporitzador, is canviï el valor de *timercorre* a 1.

```
319         if (!timercorre) {
320             HAL_TIM_Base_Start_IT(&htim7);
321             timercorre=1;
322         }
```

Extracte de codi 3. *Accions que realitza el codi al activar-se una interrupció externa (segona part)*

Un cop fet això, s'haurà obtingut el temps des del darrer pols. El primer que es fa és comprovar que el temps sigui major al marge d'error. Si ho és, l'UART comunicarà la recepció d'un nou pols.

```
324         if (tempsPassat>10){
325             if (DEBUG) msglen=sprintf(msg, "temps %u\r\n", tempsPassat);
326             if (DEBUG) HAL_UART_Transmit(&huart2, (uint8_t*)msg, msglen, 10);
327             msglen=sprintf(msg, "$PPM,%d,%u\r\n",(int)((12-shiftbits)/2+1),tempsPassat);
328             HAL_UART_Transmit(&huart2, (uint8_t*)msg, msglen, 1);
329         }
```

Extracte de codi 4. *Accions que realitza el codi al activar-se una interrupció externa (tercera part)*

Seguidament, es comprova que el temps des del darrer pols sigui vàlid per a passar al descodificador. Això vol dir que es trobarà dins un marge de 10 ms de 50, 100, 150 o 200 mil·lisegons. Si el requisit es compleix, el valor es passa a la funció descodificadora, el funcionament de la qual s'explica a l'apartat 5.4.

```
331         if (tempsPassat>=40 && tempsPassat<=210 && shiftbits>=0) {
332             descodificador(tempsPassat);
333         }
334         else if(tempsPassat>=40 && (tempsPassat>=210 || shiftbits<0)){
335
336
337             //Serà una nova dada
338             shiftbits=12;
339             dadaRebuda=0;
340             IDrebut=0;
341
342             msglen=sprintf(msg, "ERROR, no s'ha reinicialitzat bé la dada\r\n");
343             HAL_UART_Transmit(&huart2, (uint8_t*)msg, msglen, 10);
344         }
345     }
```

Extracte de codi 5. *Accions que realitza el codi al activar-se una interrupció externa (quarta part)*

En cas contrari, s'interpretarà com un error. Els possibles errors que pot detectar el microcontrolador en aquest punt del programa són:

- **Ha passat massa temps des del darrer pols, i encara no s'ha transmès del tot la dada actual:** apareix si el comptador de bits (*shiftbits*, veure apartat 4.4) encara no ha arribat al seu límit i el temps entre polsos és major a 210 ms. Aquest error és redundant, ja que el comptador, pel seu disseny, mai hauria d'arribar a aquest valor. De totes formes, per fer el codi més robust, es comprova que no passi això. Si passés, es pararia el temporitzador i es reiniciarien totes les variables relacionades amb el procés de recepció.
- **El comptador de bits passa el valor mínim:** això indicaria que el receptor ha rebut tots els bits necessaris per a completar un missatge, però per alguna raó el reinici de variables no s'ha fet correctament. Per l'estructura del codi això tampoc hauria de passar mai, però afegeix robustesa i ajuda a detectar possibles errors.
- **L'interval entre trens rebuts és invàlid:** indica que l'interval rebut no entra dins els marges de cap dels valors que poden correspondre a un valor de bits. Aquest error pot donar-se per factors externs al receptor, pel que és important detectar-lo. Quan es detecta, a part de comunicar-se l'incident per UART, es canvia el valor d'un *flag* anomenat *dadaFallada* a 1 (el seu valor inicial és 0). En arribar al final de la recepció d'un missatge, es comprova l'estat d'aquest *flag*. Si no té valor nul, es considerarà invàlid i per tant no es comunicarà la informació corresponent. S'ha decidit fer això d'aquesta manera, en lloc d'interrompre i reiniciar la recepció immediatament al detectar l'error (com es fa amb els altres errors), perquè, d'aquesta manera, si es troba un error per causes externes al receptor es poden seguir rebent la resta de polsos de la transmissió, que no haurien de venir errats.

4.3. Temporitzadors

La millor solució per a mesurar el temps entre polsos és l'ús dels perifèrics TIM, que fan la funció de temporitzadors. A la taula 3 es poden apreciar tots els temporitzadors de què disposa i les seves especificacions. S'ha triat el TIM14, un *general purpose timer* amb una resolució de 16 bits, suficient per fer la funció que es necessita (el marge d'error que s'ha fixat és de 10ms). A la vista de configuració de dispositiu, se selecciona el temporitzador per a tocar-ne els paràmetres. Es tria un *prescaler* que divideixi la

freqüència del *clock* intern HCLK, que es pot trobar i modificar a la pestanya "clock configuration". En aquest cas, la freqüència està ajustada a 96 MHz. També es tria el període del comptador, que indicarà cada quants ticks es provoca una interrupció.

Amb un *prescaler* de 9600 s'aconsegueix una freqüència de 96 MHz/9600=10 kHz, posant el període a 10 ticks, s'obtindrà una interrupció cada 1 ms. Fent que aquesta interrupció incrementi una variable (int tempsActual) en valor de 1 cada vegada que es cridi, tindrem una variable que compta mil·lisegons. Això farà prou senzilla la tasca de mesurar el temps entre polsos.

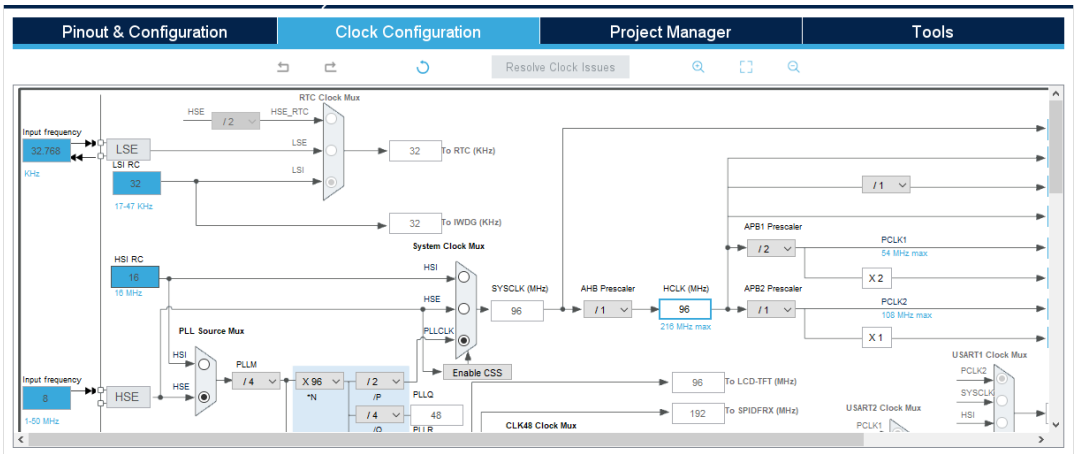

Figura 11. *Vista de l'esquema de configuració dels clocks*

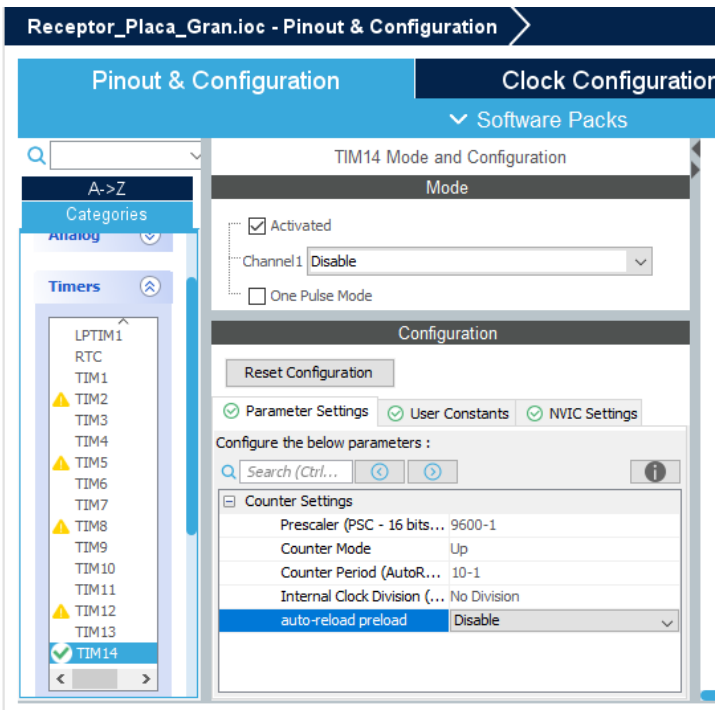

Figura 12. *Paràmetres del temporitzador*

Timer type	Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/ compare channels	Complementary output	Max interface clock (MHz)	Max timer clock (MHz)(1)
Advanced -control	TIM1, TIM8	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	Yes	108	216
General purpose	TIM2, TIM5	32-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	54	108/216
	TIM3, TIM4	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	54	108/216
	TIM9	16-bit	Up	Any integer between 1 and 65536	No	2	No	108	216
	TIM10, TIM11	16-bit	Up	Any integer between 1 and 65536	No	1	No	108	216
	TIM12	16-bit	Up	Any integer between 1 and 65536	No	2	No	54	108/216
	TIM13, TIM14	16-bit	Up	Any integer between 1 and 65536	No	1	No	54	108/216
Basic	TIM6, TIM7	16-bit	Up	Any integer between 1 and 65536	Yes	0	No	54	108/216

1. The maximum timer clock is either 108 or 216 MHz depending on TIMPRE bit configuration in the RCC_DCKCFGR

Taula 3. *Temporitzadors disponibles a un microcontrolador STM32F7xxxx*

```
506 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
507
508     if (htim == &htim14 ) {
509         //Incrementa 1 el temporitzador
510         if (tempsActual<210){
511                 tempsActual++;
512         }
513         else if (tempsActual>=210){
514             tempsActual=0;
515             HAL_TIM_Base_Stop_IT(&htim14);
516             timercorre=0;       //Flag que indica si funciona el timer
517         }
518
519     }
520 }
521
```

Extracte de codi 6*. Accions que realitza el codi quan el temporitzador genera una interrupció*

Cada vegada que s'activa la interrupció, es crida la rutina *HAL_TIM_PeriodElapsedCallback*, que incrementa en valor de 1 una variable tipus *int* anomenada *tempsActual* sempre que el seu valor es trobi per davall de 210, que és el

temps màxim entre polsos que pot contenir un missatge en el protocol utilitzat. Si el valor és igual o major a 210, es tornarà *tempsActual* al valor inicial de 0. També s'aturarà el temporitzador, ja que no interessa mesurar valors majors a 210. Així, s'evita l'error que succeiria si un nou pols fos rebut quan el temporitzador s'acabés de refrescar.

4.4. Descodificació

Els temps entre polsos vàlids que es rebin seran passats a la funció *descodificador* per a ser interpretats. La seva tasca és traduir aquests temps al seu significat en parell de bits, i col·locar-los al seu lloc corresponent a la variable que pertoqui, sigui *IDrebut* o *dadaRebuda*.

El primer que es fa és guardar el valor corresponent al temps rebut en una variable anomenada *bits*. Això es fa amb una sèrie d'*ifs* que comproven si el temps està dins els marges de cada valor:

Temps entre trens de polsos	**Marges**	**Valor en bits**	**Valor decimal**
50 ms	40 ms $\leq t_p \geq$ 60 ms	00	0
100 ms	90 ms $\leq t_p \geq$ 110 ms	01	1
150 ms	140 ms $\leq t_p \geq$ 160 ms	10	2
200 ms	190 ms $\leq t_p \geq$ 210 ms	11	3

Taula 4. *Equivalències entres temps entre polsos, marges, valors binaris i valors decimals*

```
372 void descodificador(uint8_t tempsPassat){
373
374     short int bits;
375     //if (DEBUG) msglen=sprintf(msg, "temps %u\r\n", tempsPassat);
376     //if (DEBUG) HAL_UART_Transmit(&huart2, (uint8_t*)msg, msglen, 10);
377     if (tempsPassat<=60 && tempsPassat>=40){
378         bits=0;
379     }
380     else if(tempsPassat<=110 && tempsPassat>=90){
381         bits=1;
382     }
383     else if(tempsPassat<=160 && tempsPassat>=140){
384         bits=2;
385     }
386     else if(tempsPassat<=210 && tempsPassat>=190){
387         bits=3;
388     }
```

Extracte de codi 7. *Funció descodificadora (primera part)*

Una vegada s'ha obtingut el valor del parell de bits, s'ha de col·locar a la posició de la variable que li pertoqui. Com a comptador dels parells que es reben dins una transmissió,

es fa servir una variable *int* anomenada *shiftbits*. Aquesta té un valor inicial de 12 bits i es fa servir per a calcular quantes posicions s'han de desplaçar cap a l'esquerra els parells de bits rebuts abans de ser afegits a la variable corresponent.

Pel que fa a les variables *IDrebut* i *dadaRebuda*, ambdues tenen un valor inicial de 0. Això fa que serveixin com espais en blanc on es pot afegir el valor dels bits rebuts i desplaçats mitjançant un *or* bit a bit.

Per a saber si un parell de bits d'una transmissió correspon a l'identificador o a la dada, es fa servir la mateixa variable *shiftbits.* Com que per protocol s'ha establert que el transmissor primer ha d'enviar l'ID i després la dada, ambdós en ordre de bit més significant a menys, es comprova que els 3 primers parells seran part del primer i els 4 restants seran part del segon. *Shiftbits* es decrementa en un valor de 2 cada vegada que la funció descodificadora acaba d'executar-se i es torna a 12 cada vegada que es reinicia la transmissió d'un missatge, pel que es pot fer servir per saber quants de bits s'han rebut fins al moment. Per tant, si és major o igual que 8, els bits tractats es consideraran part de l'ID, i si és menor que 8 i major que 0, part de la dada.

En haver-se triat en quina variable aniran estojats, els parells de bits s'han de desplaçar a l'esquerra el nombre de posicions que li siguin necessàries. Per a l'identificador, s'haurà de desplaçar el parell per la resta del valor de *shiftbits* menys 8, mentre que per a la dada bastarà desplaçar-lo pel valor de dita variable sense fer-li cap operació.

Valor de *shiftbits*	12	10	8	6	4	2	0
Variable que pertoca	IDrebut	IDrebut	IDrebut	dadaRebuda	dadaRebuda	dadaRebuda	dadaRebuda
Posició a la variable	5, 4	3, 2	1, 0	7, 6	5, 4	3, 2	1, 0
Posicions a desplaçar (La posició original és 1, 0)	4	2	0	6	4	3	0

Taula 5. *Equivalència entre el valor de la variable* shiftbits*, la variable on s'estojarà el parell de bits, la posició que li correspon a la variable, i el nombre de posicions a desplaçar*

```
435     if (shiftbits>=8){
436         //Bits 13 a 8 van a l'ID
437         IDrebut= IDrebut | (bits << (shiftbits-8));
438         if (DEBUG) msglen=sprintf(msg, "shifted %d\r\n", shiftbits);
439         if (DEBUG) HAL_UART_Transmit(&huart2, (uint8_t*)msg, msglen, 10);
440     }
441     else if (shiftbits<8){
442         //Bits 7 a 0 van a dada
443         dadaRebuda= dadaRebuda | (bits << shiftbits);
444         if (DEBUG) msglen=sprintf(msg, "shifted %d\r\n", shiftbits);
445         if (DEBUG) HAL_UART_Transmit(&huart2, (uint8_t*)msg, msglen, 10);
446     }
447
448     shiftbits-=2;
```

Extracte de codi 8. *Funció descodificadora (segona part)*

Amb els bits ja estojats dins la variable corresponent, hi ha un decrement de dos en el valor del comptador de bits. A continuació, es comprova el seu valor: si és menor que 0, voldrà dir que el missatge complert ja s'ha rebut i per tant el microcontrolador comunicarà la informació obtinguda per UART. Abans de transmetre això, es comprova que el flag *dadaFallada* tingui un valor de 0, ja que això voldria dir que en algun moment la rutina d'EXTI ha detectat un interval entre trens erroni, i la dada no seria correcta. Finalment, es reiniciaran les dades relatives a tot el procés als seus valors originals (*tempsPassat=0, tempsActual=0, shiftbits=12, dadaRebuda=0, IDrebut=0, timercorre=0, dadaFallada=0).*

5. Programació de la interfície d'usuari per al microcontrolador transmissor

Com s'ha mencionat prèviament, a l'inici d'aquest treball es disposava d'un microcontrolador ja programat per un altre participant del projecte SASES. El microcontrolador és capaç d'enviar trens de polsos seguint unes comandes que se li poden enviar per USB i que rep mitjançant UART. Per aconseguir, però, que enviés missatges seguint el protocol de PPM (capaç d'entendre el microcontrolador receptor), se li ha d'enviar una quantitat extensa de comandes. Per facilitar aquesta tasca, s'ha programat una interfície fent servir el llenguatge Python, que demana a l'usuari l'identificador i la dada que vol enviar i, quan se li dona l'ordre, envia per USB les comandes necessàries al transmissor.

5.1. El programa transmissor

El programa del que disposa el microcontrolador compta amb diversos modes, tots relacionats amb l'enviament de trens de polsos. Està pensat per a ser versàtil i servir per a altres protocols a més del que s'està desenvolupant en el present treball.

Els modes de funcionament es configuren mitjançant l'enviament de comandes. Aquestes es fan mitjançant el protocol serial UART, configurat amb els següents paràmetres:

Baudrate: 115200 bps.

Data Bits: 8 bits.

Stop bits: 1 bit.

Paritat: cap.

Flow control: cap

Totes les comandes disponibles comencen pel caràcter "$" i han de seguir-se d'un caràcter de retorn (CR, \r) per a ser vàlides. No s'ha d'enviar un caràcter de nova línia (LF, \n) al final d'una comanda. Si es fa, aquesta es considera invàlida. Quan el microcontrolador rep una comanda vàlida, la respon repetint-la, però seguida d'un caràcter de retorn i un de canvi de línia.

5.1.1. Comprovació de la connexió

Aquest mode rep la instrucció "$AT" i en retorna els mateixos caràcters. És útil per a comprovar si la connexió entre l'ordinador i el microcontrolador funciona correctament.

5.1.2. Mode Continu

Aquest mode genera una ona de polsos quadrats quan se li dona l'ordre "$SC", que s'atura quan rep la instrucció d'aturada general "$ST".

La freqüència i potència de l'ona que es genera és configurable mitjançant la instrucció "$CC,FFF,000,P". FFF són tres dígits amb els quals s'indica la freqüència en kHz que es desitja (han de ser necessàriament tres dígits, afegint zeros a l'esquerra si és necessari); i P és la potència, amb un valor de 0 a 3 on 0 indicaria una ona inexistent i 3 la màxima potència.

5.1.3. Mode "Single shot"

Aquest mode envia un sol tren de polsos en el moment just en què se li dona l'ordre "$SS". Es pot triar la freqüència dels polsos i quants de polsos conté un tren amb la instrucció "$CS,FFF,OOO,P", on FFF indica la freqüència en kHz, OOO el nombre de polsos i P la potència. Abans d'enviar un nou pols, s'ha de parar l'acció amb la instrucció "$ST".

5.1.4. Mode Tren de polsos

El mode tren de polsos envia una cadena de trens de polsos quan se li dona l'ordre amb la comanda "$SP". Aquest és el mode que es fa servir per a enviar missatges seguint el protocol establert en aquest treball. Els paràmetres configurables són: freqüència, nombre de trens, potència, nombre de polsos a cada un dels trens i temps d'espera fins al següent pols.

Per a enviar un missatge, primer s'han d'ajustar els paràmetres generals d'aquest. Això es fa amb la comanda "$CP,FFF,OOO,P", on FFF indica la freqüència en kHz i OOO el nombre de trens de polsos que contindrà el missatge.

A continuació, es configuren les característiques de cada tren de polsos. Això es fa tren per tren, podent crear-ne de diferents durades amb diferents espais entre ells, on l'únic que és constant és la freqüència dels polsos que contenen. S'utilitza la comanda

"CT,NNN,PPP,SSSS", on NNN és el nombre del pols que es configura, començant per 1; PPP és la quantitat de polsos que conté aquest tren; i SSSS és el temps d'espera després d'enviar el tren que s'està configurant fins que s'envia el següent, en dècimes de mil·lisegon. S'ha d'enviar per tant, una comanda per cada tren de polsos que utilitzi el missatge.

Per exemple, si es volen enviar 3 trens a 50 kHz, potència de 2, on el primer conté 200 polsos, el segon 250, i el tercer 300, i després de cada un d'ells hi ha un retard de 40, 50 i 60 ms respectivament, s'envien les següents quatre comandes:

$CP,050,003,2\r

$CT,001,200,0400\r

$CT,002,250,0500\r

$CT,003,300,0600\r

Finalment, es llança el missatge fent servir "$SP" i, en acabar (o si es volgués interrompre abans de temps l'enviament), s'envia la comanda "$ST". Fins que aquesta darrera no s'enviï, el microcontrolador no admetrà una nova comanda.

Així doncs, un missatge en el protocol emprat en aquest treball ha de constar de 8 trens: 7 primers trens, en el retard dels quals està codificada la informació; i un darrer tren, el temps després del qual és irrellevant, per tancar el retard després del tren nombre 7. La freqüència ha de ser de 69 kHz i a cada tren se li han d'assignar 345 polsos, de manera que la durada total d'aquest sigui de 5 ms. El temps després de cada pols ha de ser de 50, 100, 150 o 200 mil·lisegons, depenent de quins siguin els bits que es volen transmetre. El temps després del vuitè pols pot ser qualsevol, però és convenient ajustar-lo als 211 ms, de manera que superi el temps màxim d'espera del receptor.

5.2. Característiques de la interfície

La interfície disposa d'un menú on s'ofereixen varies opcions a l'usuari: connectar-se al microcontrolador, modificar l'ID, modificar la dada, transmetre i sortir del programa. L'usuari selecciona quina opció vol fer servir en introduir un caràcter. A més, al menú inicial es mostren, en valor hexadecimal, l'identificador i la dada que hi ha carregats al programa en aquell moment. Ambdós es poden modificar en qualsevol moment fent

servir l'opció pertinent. Una vegada es té carregat el missatge que es vol, l'usuari pot connectar-se al microcontrolador i donar l'ordre per a que enviï els trens de polsos que pertoquin.

5.3. Programació de la interfície

L'estructura del programa consta d'un bucle principal i una sèrie de funcions que s'executen com a subrutines des d'aquest bucle.

A l'inici del codi, s'importa la llibreria *pyserial*, que efectuarà les comunicacions entre el PC i el microcontrolador transmissor. També es declaren algunes variables amb un valor inicial, que seran utilitzades per més d'una funció, i per tant és més convenient tractar-les com a globals. Aquestes són, en primer lloc, la variable *ser*, que més endavant contindrà la instància de la llibreria serial amb la que es gestionaran les comunicacions. També hi tenim definits un identificador i una dada per defecte a ser enviats, per tal d'agilitzar la comprovació del sistema, i un nombre de port de comunicacions predefinit.

```
1  #Llibreries
2  import serial
3
4  #VARIABLES GLOBALS
5  ser=None
6  ID=0b111100
7  Dada=0b11001010
8  port = '7'
```

Extracte de codi 9. *Importació de llibreries i de variables globals*

5.3.1. Menú principal

En iniciar el programa, a la consola s'imprimeix el menú principal, on s'explica a l'usuari quines opcions té disponibles per a executar. Això s'assoleix amb una sèrie de funcions *print*. A continuació, es demana un caràcter amb una funció *input*, que es guarda en una variable anomenada *mode*. Tot el codi relatiu al menú principal està col·locat dins un bucle *while*, que només es talla si la variable *surt* canvia el seu valor a *true* (per defecte, en inicialitzar-se el seu valor és fals), que passa quan l'usuari selecciona el caràcter "S" (executant així l'opció "Sortir del programa"). D'aquesta forma, qualsevol opció que

realitzi el programa anirà seguida d'un retorn al menú principal, des del qual es podrà seguir amb una altra instrucció.

```
165  #BUCLE MENÚ PRINCIPAL
166  surt=False
167  while(not(surt)):
168      #Selecció opció
169      print("ID: ", hex(ID), "Dada:", hex(Dada))
170      print("Port: ", 'COM'+port)
171      print("Introdueixi un caràcter per a seleccionar el mode:")
172      print("C: Connectar amb el microcontrolador")
173      print("I: Modificar l'ID")
174      print("D: Modificar la dada")
175      print("T: Transmetre")
176      print("P: Canvi de port")
177      print("S: Sortir del programa")
178      mode=input("Esculli una opció: ")
```

***Extracte de codi 10**. Bucle que genera el menú principal (primera part)*

A continuació, amb una sèrie d'*ifs* i *elifs* es selecciona una funció a executar segons el valor de *mode*. Aquests *ifs* segueixen tots la mateixa estructura, que és en primer lloc confirmar l'opció que han triat a l'usuari amb un *print*, cridar la funció obtenint-ne o no paràmetres i finalment demanar un input a l'usuari per a continuar amb el programa una vegada finalitzada l'execució de la funció.

```
167      if mode=='C':
168          print("Ha seleccionat: Connectar amb el microcontrolador")
169          connexio=comprovaConnexio()
170          obert=connexio[0]
171          ser=connexio[1]
172          input("Premi 'Enter' per a continuar")
173
174      elif mode=='I':
175          print("Ha seleccionat: Modifica l'ID")
176
177          ID=triaID()
178
179          input("Premi 'Enter' per a continuar")
180
181      elif mode=='D':
182          print("Ha seleccionat: Modifica la dada")
183
184          Dada=triaDada()
185
186          input("Premi 'Enter' per a continuar")
187
188      elif mode=='T':
189          print("Ha seleccionat: Transmetre")
```

***Extracte de codi 11.** Bucle que genera el menú principal (segona part)*

5.3.2. Connectar amb el microcontrolador

Si l'usuari selecciona el caràcter "C", el programa cridarà la funció *obreConnexio()*. Aquesta funció pren la variable global *ser* i hi obre una instància *Serial* per a poder comunicar-se amb un port. A continuació, es trien les característiques d'aquesta instància. El port serà un *string* amb els caràcters "COM" més el nombre de port corresponent, que el programa guarda a la variable *port*, també en forma de *string*. El *baudrate* és de 115200, tal i com s'ha explicat a l'apartat 5.1, i el temps d'espera màxim (*timeout*) que s'ha triat és de 0,1 segons.

Una vegada definits els paràmetres, es fa servir una clàusula *try-except* per intentar obrir la connexió amb la funció *ser.open()*. Si s'obre correctament, es segueix endavant amb el codi. En cas contrari, es comunica amb un *print* que hi ha hagut un error i també s'imprimeix el missatge d'error generat per la llibreria.

Seguidament, es comprova si s'ha obert el port revisant el valor de *ser.is_open*. Si el resultat és negatiu, s'imprimeix un missatge que indica que no s'ha trobat el dispositiu, es tanca preventivament el port amb la instrucció *ser.close()* (encara que aquesta acció és redundant, aporta robustesa al codi) i es retorna *False* com a resultat de la funció. En canvi, si el port sí és obert, es crida la funció *comprovaConnexio()*, que comprova si el dispositiu al que estem connectats és un microcontrolador amb el programari de transmissió carregat. Finalment es retorna el resulta de la funció de comprovació, el funcionament de la qual s'explica en el proper apartat.

```
11  def obreConnexio():
12      global ser
13      #Obre serial
14      ser = serial.Serial()
15      #Paràmetres Port (FALTA: Fer editables)
16      ser.port = 'COM'+port
17      ser.baudrate = 115200
18      ser.timeout=0.100
19      #Obrir port
20      try:
21          ser.open()
22      except serial.SerialException as errorMsg:
23          print("S'ha trobat una excepció:")
24          print("->", errorMsg)
25
26      if ser.is_open:
27          comunica=comprovaConnexio()
28          return comunica
29      else:
30          print("No s'ha trobat un dispositiu")
31          ser.close()
32          print("S'ha tancat el port serial. Si us plau, torni's a connectar")
33          return False
```

Extracte de codi 12. *Funció obreConnexio*

5.3.3. Comprovació de la connexió

La funció *comprovaConnexió* s'ha implementat per a poder ser cridada abans d'executar qualsevol acció que impliqui la comunicació amb el microcontrolador, per tal de comprovar de que aquest està connectat i respon correctament. Això es fa enviant la comanda "$AT" i revisant que s'obtingui la resposta corresponent (veure apartat 5.1.1).

Per a enviar una comanda, es fa servir la funció *ser.write*, que fa servir com a paràmetre la versió codificada d'un *string*, En aquest cas, es fa servir la codificació *ascii*.

Una vegada enviada la comanda, es dona una ordre *.read_until()* que llegeix el que respon el microcontrolador fins a trobar un caràcter de nova línia. El missatge que es rebi serà comparat amb la comanda que s'ha enviat mitjançant un *if*.

Cada vegada que el transmissor rep una comanda, la contesta amb una d'idèntica, amb un caràcter de nova línia afegit al principi d'aquesta. Com a excepció, la primera comanda que rebi al obrir-se una nova connexió amb un dispositiu serà repetida idènticament. Per tant, es compara el missatge rebut (guardat a la variable *msg*) amb les dues variants de resposta possibles. Si coincideixen, es comunicarà l'èxit de la connexió i la funció de comprovació retornarà un booleà vertader. En cas contrari, tornarà un valor fals i comunicarà l'error.

```python
def comprovaConnexio():
    global ser
    comanda='$AT\r'
    ser.write(comanda.encode('ascii'))
    print(comanda)
    msg=ser.read_until('\n'.encode('ascii'))
    print(msg)
    if msg == (comanda+'\n').encode('ascii') or msg == ('\n'+comanda+'\n').encode('ascii'):
        print("S'ha connectat correctament")
        return True
    else:
        print(msg)
        print("El microcontrolador no respon correctament")
        ser.close()
        print("S'ha tancat el port serial. Si us plau, torni's a connectar")
        return False
```

Extracte de codi 13. *Funció comprovaConnexio*

5.3.4. Modificació de l'identificador i/o la dada a enviar

Si l'usuari introdueix el caràcter 'I' o 'D', es cridaran respectivament les funcions *triaID* o *triaDada*. Ambdues funcions són prou senzilles: consisteixen en un input amb el qual s'introdueix un nombre enter, que després és comunicat a l'usuari en la seva forma binària i guardat a la variable que pertoqui.

```python
def triaID():

    ID=int(input("Introdueixi un Identificador: "),2)
    print(bin(ID))
    return ID

def triaDada():

    Dada=int(input("Introdueixi una Dada: "),2)
    print(bin(Dada))
    return Dada
```

Extracte de codi 14*. Funcions triaID i triaDada*

5.3.5. Selecció del port

Si l'usuari introdueix el caràcter 'P', es realitzen una sèrie d'operacions per a canviar el port al que es connectarà el programa en cas de seleccionar l'opció "Connectar amb el microcontrolador".

En primer lloc, es comprova que no hi hagi cap connexió serial oberta i, si n'hi ha, s'ha de tancar, ja que pot estar associada a un port no desitjat. A continuació, es demana un *input* a l'usuari per a que insereixi el nombre de port que vol utilitzar. Aquest nombre ha de ser numèric; per comprovar-ho, s'hi aplica la funció de Python *isnumeric*. Si ho és, es

guarda a la variable *port* en forma de *string*. En cas contrari, es torna a demanar una entrada a l'usuari fins que sigui correcta.

```
210     elif mode=='P' or mode=='p':
211         print("Ha seleccionat: Canvi de port")
212         if ser!=None:
213             if ser.is_open:
214                 ser.close()
215         port=input("Insereixi un nombre de port (nombre enter): ")
216         if port.isnumeric():
217             port=str(port)
218         else:
219             print("Entrada invàlida")
220             port=input("Si us plau, insereixi un nombre de port (nombre enter): ")
```

Extracte de codi 15. *Bucle que genera el menú principal (tercera part)*

5.3.6. Transmissió del missatge

Una vegada estiguin configurats el port i el missatge (identificador i dada) que es vol enviar, l'usuari pot seleccionar el caràcter 'T' per a donar l'ordre d'enviament al transmissor.

```
199     elif mode=='T' or mode=='t':
200         print("Ha seleccionat: Transmetre")
201         if ser.is_open:
202             comprovaConnexio()
203             configurat=configuraTren()
204             if configurat:
205                 enviaTren()
206         else:
207             print("Connecti's al microcontrolador")
```

Extracte de codi 16. *Bucle que genera el menú principal (quarta part)*

Primerament, es comprova que la connexió serial estigui oberta i es crida la funció *comprovaConnexio* (veure apartat 5.2.3) per a assegurar que s'estigui comunicant amb un dispositiu transmissor.

A continuació, es crida la funció *configuraTren()*, la qual envia les comandes necessàries per a configurar la cadena de polsos que enviarà el microcontrolador. En executar-la, es comprova que hi hagi una connexió serial oberta amb *ser.is_open*. En cas afirmatiu, s'envia una comanda de configuració dels paràmetres de la cadena. Aquesta ordre és "$CP,069,008,3" i, seguint les convencions explicades a l'apartat 5.1.4, indica que la

freqüència dels polsos serà 69 kHz; que s'enviaran 8 trens de polsos; i que el nivell de potència utilitzat serà el més alt. Seguidament, s'envia el missatge codificada en ASCII i es comprova que el microcontrolador respongui amb un *string* idèntic. Si alguna cosa falla, es comunica l'error i es tanca la connexió serial. En canvi, si no falla res, procedeix amb el següent pas, que consisteix en la configuració de cadascun dels 8 trens de polsos que ha de contenir la cadena.

```
64  def configuraTren():
65      global ser
66      #Es comprova que estigui obert
67      if ser.is_open:
68          comanda='$CP,069,008,3\r'
69          print(comanda)
70          ser.write(comanda.encode('ascii'))
71
72          msg=ser.read_until('\n'.encode('ascii'))
73          if msg != (comanda+'\n').encode('ascii') and msg != ('\n'+comanda+'\n').encode('ascii'):
74              print(msg)
75              print("El microcontrolador no respon correctament")
76              ser.close()
77              print("S'ha tancat el port serial. Si us plau, torni's a connectar")
78              return False
79
80          parelles=[]
81          timestring=[]
82          strID=(str(bin(ID)).split('b')[1]).zfill(6)
83          strDada=(str(bin(Dada)).split('b')[1]).zfill(8)
84          for i in range(0, 6, 2):
85              parelles.append(strID[i]+strID[i+1])
86
87          for i in range(0, 8, 2):
88              parelles.append(strDada[i]+strDada[i+1])
```

Extracte de codi 17. *Funció configuraTren (primera part)*

Per a realitzar la configuració, es necessita enviar 8 comandes del tipus "$CT" al microcontrolador que contenguin la informació sobre la quantitat de polsos i el temps d'espera posterior de cadascú dels trens. Per tant, cada una presenta uns paràmetres diferents que s'han de codificar segons els valors de les variables *ID* i *Dada*.

La creació d'aquestes comandes comença amb la inicialització de dos vectors de dades anomenats *parelles* i *timestring*. Seguidament, es converteixen els valors de les variables *ID* i *Dada* en dues *strings* i queden guardats a les variables *strID* i *strDada* en format binari. Per fer això, es converteix la forma binària amb la funció *str()*, s'extreu amb la funció *bin()* i, consegüentment, s'obté una cadena amb el format "0bXXXXXX". El "0b" inicial s'elimina amb la funció *.split('b')*, que divideix el *string* en dues cadenes al lloc on trobi el caràcter 'b' i les guarda en un vector. D'aquest vector, es fa servir només el segon element resultant. Tot seguit, es fixa que *strID* i *strDada* tinguin una llargària de 6 i 8 caràcters respectivament amb la funció *zfill()*, la qual posa zeros al principi d'un *string* per assegurar que tingui una llargària mínima, que se li dona com a paràmetre.

Una vegada es tenen els formats correctes de *strID* i *strDada*, els dígits d'aquests s'han de separar per parelles de bits, ja que cada una es veurà codificada en un tren de polsos. Per a fer això, es fan servir dos bucles *for*, un per a cadascuna d'aquestes variables, que fan aquesta separació i estogen les parelles resultants en format *string* dins el vector *parelles*.

```
90          for bits in parelles:
91              #Podria ser un switch case (match case a python) però a versions més antigues que python 3.1
92              if bits=='00':
93                  timestring.append('0500')
94              elif bits=='01':
95                  timestring.append('1000')
96              elif bits=='10':
97                  timestring.append('1500')
98              elif bits=='11':
99                  timestring.append('2000')
100         print(timestring)
101         for i in range(0,7):
102             print(i)
103             comanda='$CT,00'+str(i+1)+',345,'+timestring[i]+'\r'
104             print(comanda)
105             ser.write(comanda.encode('ascii'))
106 
107             msg=ser.read_until('\n'.encode('ascii'))
108             print(msg)
109             if msg != (comanda+'\n').encode('ascii') and msg != ('\n'+comanda+'\n').encode('ascii'):
110                 print(msg)
111                 print("El microcontrolador no respon correctament")
112                 ser.close()
113                 print("S'ha tancat el port serial. Si us plau, torni's a connectar")
114                 return False
115         #Tren de tancament
```

Extracte de codi 18. *Funció configuraTren (segona part)*

A continuació, fent un bucle amb els elements del vector de parelles, es descodifiquen aquestes en valors de temps seguint el codi descrit a l'apartat 4.4 i es guarden en format *string* al vector *timestring*. Això es fa amb una sèrie d'*ifs* i *elifs* que comparen el valor de cada parella de bits amb els possibles valors binaris de dos dígits. Encara que seria més convenient implementar aquest pas amb una clàusula *match-case* (semblant a l'estructura *switch-case* d'altres llenguatges com el C), aquesta s'ha implementat recentment a les darreres versions de Python. Per tant, s'ha decidit no fer-ne servir per evitar problemes de compatibilitat amb dispositius que tinguin versions més antigues.

Fet això, s'ha creat un bucle de 8 iteracions, que prepara i envia les comandes tipus "$CT" necessàries per a marcar les característiques de cada un dels trens de polsos. Com s'ha explicat a l'apartat 5.1.4, la comanda ha de tenir la forma "$CT,NNN,PPP,SSSS". PPP ha d'indicar el nombre de polsos fixe de 345, ja que així tots els trens de polsos tindran la mateixa durada, que a la freqüència de 69 kHz a la que s'està treballant, és de 5 ms. Els altres paràmetres depenen de la iteració del bucle, ja que son diferents per cada pols.

El nombre de pols NNN equival a l'índex de la iteració del bucle més 1 (i+1), ja que els polsos es comencen a contar a partir del número u. Com que només hi ha 8 polsos, aquest nombre sempre serà d'una xifra, pel que és suficient afegir dos zeros en forma de *string* al davant. Pel que fa al temps SSSS, aquest valor ja està guardat al vector *timestring* amb el format i codificació correctes. Per tant, s'afegeix a la comanda sense fer-hi canvis.

Quan ja es té la comanda muntada, s'escriu al serial per enviar-la al microcontrolador. Llavors es comprova que el microcontrolador torni la resposta correcta. Si hi ha algun error, la funció s'interromprà i tornarà *False*. Si no n'hi ha, seguirà enviant les comandes, que en total han de ser 8. Les set primeres codifiquen els valors a enviar en el seu temps de retràs, mentre que la darrera tindrà sempre un temps de 210 mil·lisegons, que és el temps màxim que pot registrar el receptor abans de donar per tancada la transmissió d'un missatge. En haver enviat la darrera comanda es torna a comprovar la resposta del microcontrolador, i si funciona, la funció tornarà *True*.

```
116         #Tren de tancament
117         comanda='$CT,008,345,2100\r'
118         print(comanda)
119         ser.write(comanda.encode('ascii'))
120 
121         msg=ser.read_until('\n'.encode('ascii'))
122         print(msg)
123         if msg != (comanda+'\n').encode('ascii') and msg != ('\n'+comanda+'\n').encode('ascii'):
124             print(msg)
125             print("El microcontrolador no respon correctament")
126             ser.close()
127             print("S'ha tancat el port serial. Si us plau, torni's a connectar")
128             return False
129     else:
130         print("Connecti's al microcontrolador")
131         return False
132     return True
```

Extracte de codi 19. *Funció configuraTren (tercera part)*

Tornant al codi fora de la funció, es comprova el resultat de la funció *configuraTren*, i si ha estat *True*, es crida la funció *enviaTren*. Aquesta funció, havent comprovat que la connexió serial segueix oberta, envia la comanda "$SP" i comprova que el transmissor respongui correctament. Fet això, espera un *input* de l'usuari per tal de parar l'enviament, que es fa enviant la comanda "$ST" i comprovant que el microcontrolador respongui de manera correcta.

```
def enviaTren():
    global ser

    if (not ser.is_open):
        print("Connecti's al microcontrolador")
        return False
    comanda="$SP\r"
    print(comanda)
    ser.write(comanda.encode('ascii'))

    msg=ser.read_until('\n'.encode('ascii'))
    print(msg)
    if msg != (comanda+'\n').encode('ascii') and msg != ('\n'+comanda+'\n').encode('ascii'):
        print(msg)
        print("El microcontrolador no respon correctament")
        return False
    input("Premi 'Enter per parar l'enviament")

    comanda="$ST\r"
    print(comanda)
    ser.write(comanda.encode('ascii'))

    msg=ser.read_until('\n'.encode('ascii'))
    print(msg)
    if msg != (comanda+'\n').encode('ascii') and msg != ('\n'+comanda+'\n').encode('ascii'):
        print(msg)
        print("El microcontrolador no respon correctament")
        return False
```

Extracte de codi 20. *Funció enviaTren*

6. Proves

6.1. Proves per cable

Per a provar el correcte funcionament del sistema transmissor i el programa receptor, s'han muntat els dos microcontroladors en una *breadboard*, i amb un cable s'ha connectat la sortida del transmissor amb l'entrada del receptor. Ambdós dispositius s'han connectat per USB a un PC per poder realitzar les comunicacions per UART.

En el moment de fer les proves, la disponibilitat de plaques NUCLEO-L432KC eren limitades. Per això, per al transmissor s'ha fet servir una NUCLEO-G031K8. Tanmateix, aquesta substitució no afecta al funcionament del sistema que es vol demostrar.

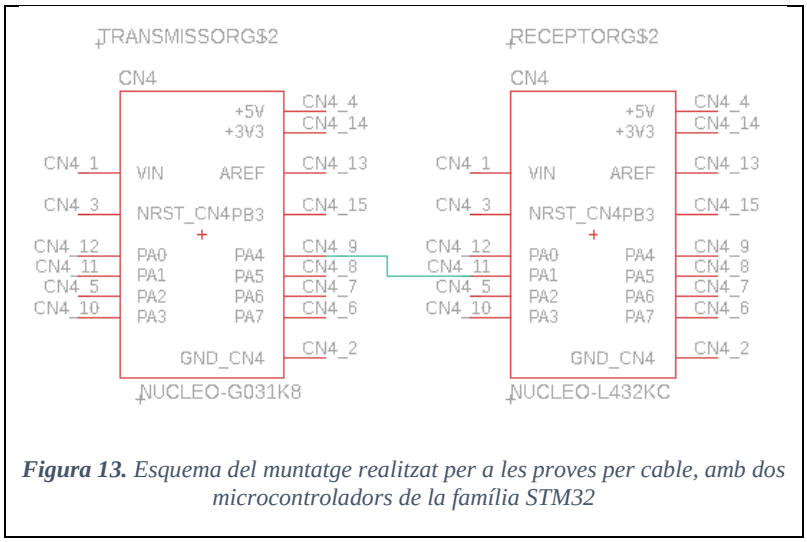

Figura 13. *Esquema del muntatge realitzat per a les proves per cable, amb dos microcontroladors de la família STM32*

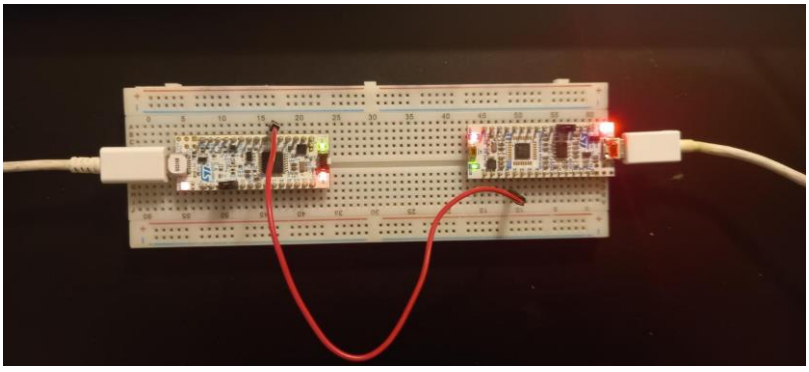

Figura 14. *Fotografia del muntatge realitzat per a les proves per cable, amb dos microcontroladors de la família STM32*

A la interfície, es selecciona "C" per a connectar-se amb el controlador transmissor. Observant que la comunicació correcta, es selecciona el mode "Modificar l'ID" i després "Modificar la dada" per triar els paràmetres a enviar.

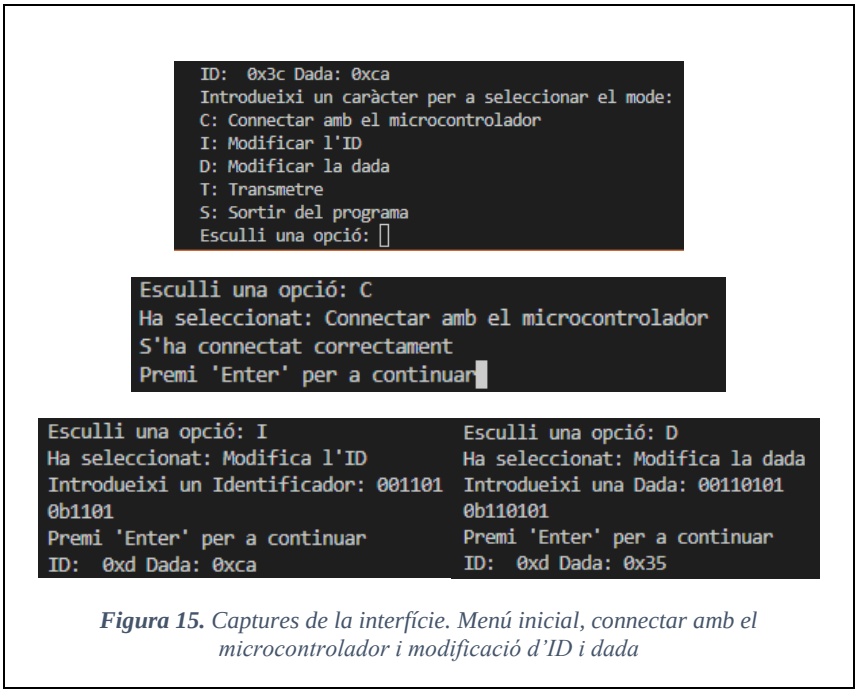

Figura 15. *Captures de la interfície. Menú inicial, connectar amb el microcontrolador i modificació d'ID i dada*

A continuació, es selecciona "Transmetre" per a enviar la dada. A la consola de Python es poden veure les comandes enviades al microcontrolador durant el procés de transmissió:

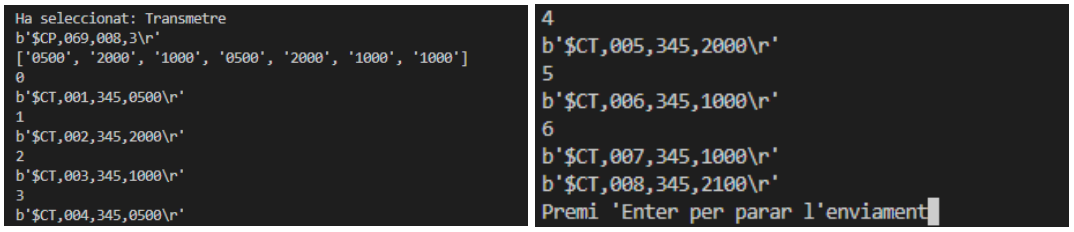

Figura 16*. Captures de la interfície al transmetre un missatge*

A una finestra de *RealTerm* connectada al port corresponent al microcontrolador receptor, es comprova que efectivament la mateixa informació que s'ha seleccionat per enviar és rebuda i descodificada per aquest dispositiu.

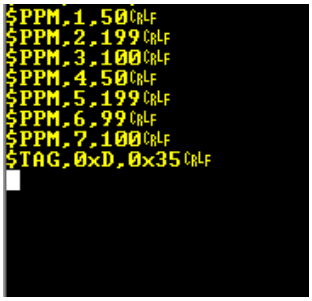

Figura 17. *Captura de la sortida de RealTerm durant l'enviament*

Per a comprovar el comportament del receptor quan aquest rep cadenes de polsos errònies, s'ha modificat el codi de Python de manera que aquest enviï comandes amb temps equivocats. A la funció *configuraTren*, s'han modificat els temps que el programa codifica per a cada valor de bits.

S'han provat dos possibles casos d'error. En primer lloc, s'han modificat dos dels temps possibles, de manera que generin un error quan s'intentin enviar els parells 00 i 10.

```
87          for bits in parelles:
88              #Podria ser un switch case (match case a python) però a versions més antigues que python 3.
89              if bits=='00':
90                  timestring.append('0610')
91              elif bits=='01':
92                  timestring.append('1000')
93              elif bits=='10':
94                  timestring.append('1390')
95              elif bits=='11':
96                  timestring.append('2000')
97          print(timestring)
```

Extracte de codi 21. *Tros de la funció configuraTren amb dos dels possibles temps modificats per a què generin un error.*

Com es pot comprovar a la figura 18, el receptor ha sabut detectar aquests errors i comunicar-los.

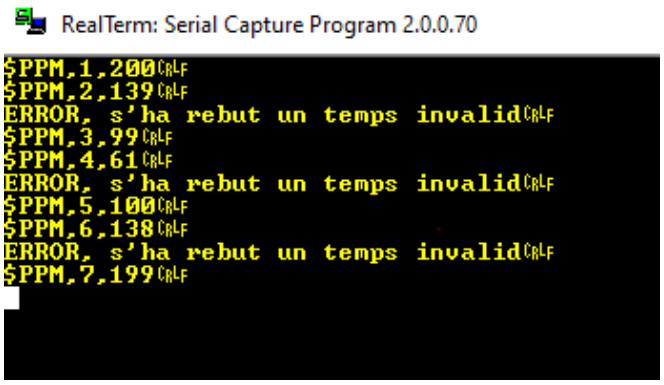

Figura 18. *Captura de la sortida de RealTerm durant l'enviament d'un missatge amb dos temps erronis*

El segon cas d'error que s'ha provat consisteix en introduir un temps major al límit de recepció de 2100 mil·lisegons. Això s'ha fet canviant la conversió del parell 11 a 2200 ms enlloc de 2000.

```
87          for bits in parelles:
88              #Podria ser un switch case (match case a python) però a versions més antigues que python 3.10
89              if bits=='00':
90                  timestring.append('0500')
91              elif bits=='01':
92                  timestring.append('1000')
93              elif bits=='10':
94                  timestring.append('1500')
95              elif bits=='11':
96                  timestring.append('2200')
97          print(timestring)
```

Extracte de codi 22. *Tros de la funció configuraTren amb un valor modificat per a què superi el límit de temps*

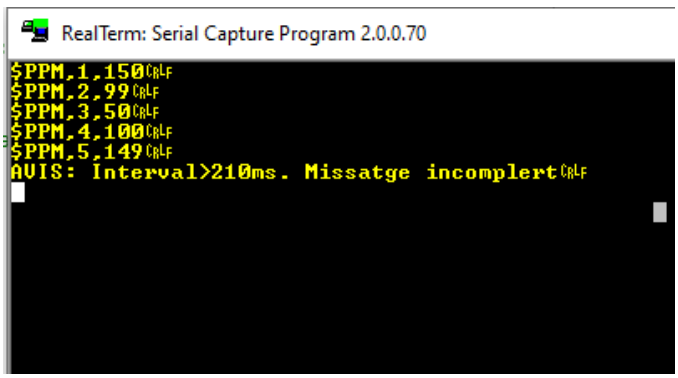

Figura 19. *Captura de la sortida de Realterm durant l'enviament d'un missatge amb un temps erroni major al límit d'espera*

Mirant la figura 19, es veu com salta l'error *AVIS. Interval>210ms. Missatge incomplert.* Per tant, l'error és detectat correctament, però es té l'inconvenient de que en aquest cas el microcontrolador és incapaç de seguir rebent la resta de trens del missatge. Això dificultaria la tasca de trobar la causa d'error manualment.

Malgrat això, s'ha comprovat el correcte funcionament de la detecció d'intervals no vàlids i la reinicialització del procés de recepció d'un missatge en ser detectar-se aquest.

6.2. Proves en aigua

Una vegada realitzades les proves per cable, s'ha volgut comprovar el funcionament del programari en una situació més propera a l'aplicació real del sistema. Per això s'ha provat a transmetre el missatge a través de dos piezoelèctrics submergits dins un recipient d'aigua, medi en el qual es poden donar més interferències.

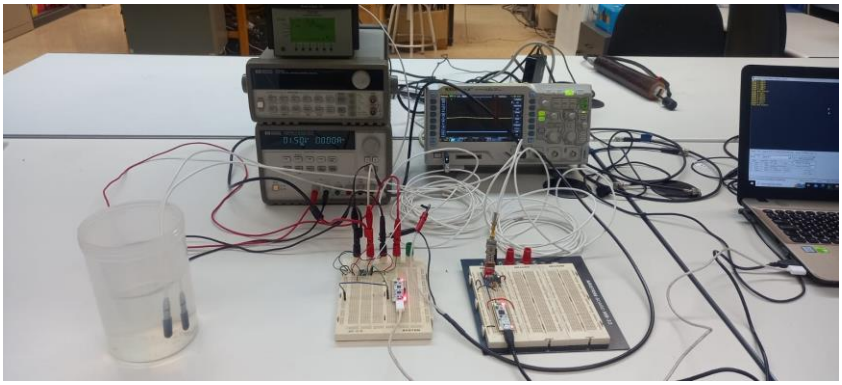

Figura 20. *Muntatge de les proves de transmissió per ones sonores. A l'esquerra, els piezoelèctrics submergits en aigua. Per al detall de les plaques, veure figura 26*

El recorregut general del senyal a enviar és el següent: en primer lloc, és generat pel microcontrolador transmissor. Tot seguit ha de passar per un circuit de condicionament, que adapta el senyal per a que a continuació pugui ser reproduït pel piezoelèctric que fa d'altaveu. En sortir de l'altaveu, el senyal acústic viatja per l'aigua fins a ser rebut pel transductor que fa de micròfon. Seguidament passa per un segon circuit de condicionament, per a ser rebut pel receptor, on finalment serà descodificat.

Per a condicionar el senyal de la sortida del transmissor al piezoelèctric, s'ha fet servir el circuit integrat PAM8904, que és un *driver* per a transductors d'aquest tipus, que amplifica el senyal que surt del microcontrolador, que té un rang de 0 a 3.3 V i el converteix en un senyal altern amb una amplitud de fins a 24 V_{pp}. [22]. D'aquesta manera podrà ser reproduïda pel transductor. La figura 21 mostra un esquema del muntatge genèric del CI. Per a la present aplicació, la tensió d'alimentació (Vin) ha de ser de 3.3 V, igual que els pins EN1 i EN2 que, en estar ambdós connectats a nivell lògic alt, configuren la màxima potència de sortida del circuit. El senyal d'entrada DIN és el senyal que es vol amplificar, que en aquest cas és la sortida del microcontrolador. Com es pot observar, les sortides Vo_1 i Vo_2 són el senyal condicionat que ha de connectar-se al piezoelèctric. Aquest senyal es transmetrà de l'altaveu a través de l'aigua i arribarà al micròfon, que el convertirà novament en un senyal elèctric

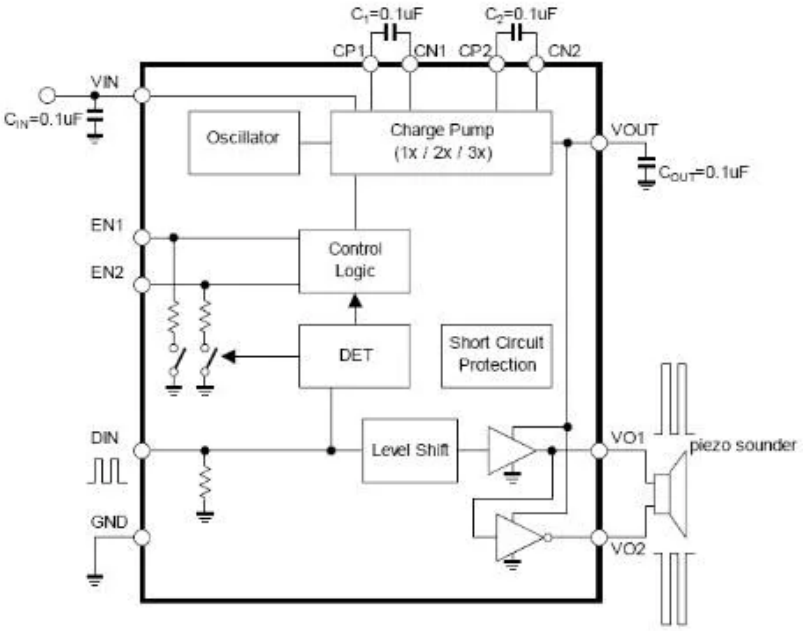

***Figura 21**. Muntatge genèric del driver PAM8904*

Com que es preveu que en el medi aquàtic s'atenuï l'ona i també es generin algunes interferències, es fa servir un segon circuit de condicionament abans d'entrar el senyal al microcontrolador receptor. Aquest circuit consisteix d'una etapa amplificadora, per a compensar l'atenuació, un filtre passa-banda amb freqüència de pas propera als 69 kHz (la freqüència del senyal que es vol transmetre), i un comparador que tornarà la forma digital a l'ona, que durant la transmissió pot haver adquirit algun harmònic no desitjat. Amb aquest condicionament, el senyal podrà ser llegit correctament pel microcontrolador.

A les figures a continuació hi figuren, en ordre: l'esquema del circuit (fig. 22), la simulació de la resposta en freqüència (fig. 23) i de la resposta en el temps (fig. 24), realitzades amb el programari *MultiSim*. Per a simular el senyal d'entrada, s'ha fet servir una font de tensió alterna de 20mV i 69kHz, i un condensador de 4nF (a la fig. 22, V_PZT i C_PZT).

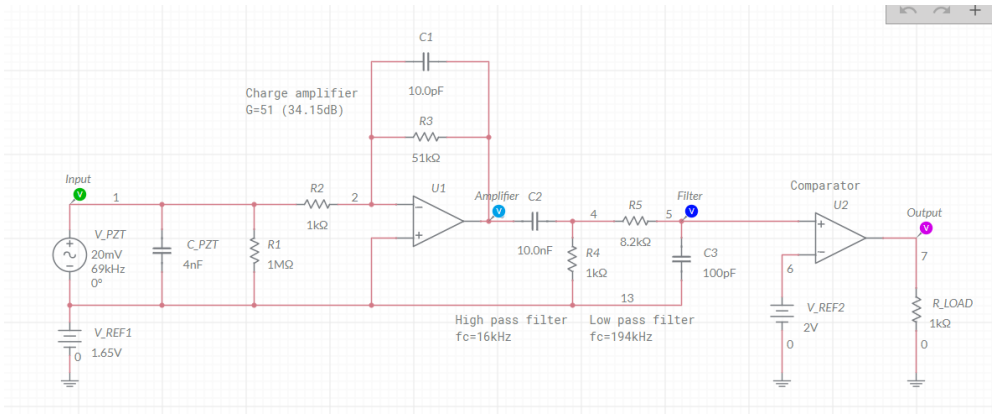

***Figura 22**. Esquema del circuit de condicionament utilitzat*

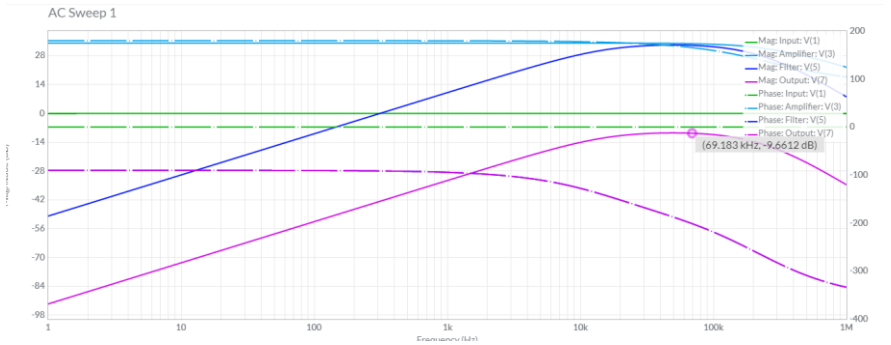

***Figura 23**. Resposta en freqüència del circuit de condicionament. En línia contínua, les magnituds, i en discontínua, les fases. En verd, l'entrada del circuit;, en blau fosc, la sortida del filtre; en blau clar, la sortida de l'amplificador; i en rosa, la sortida*

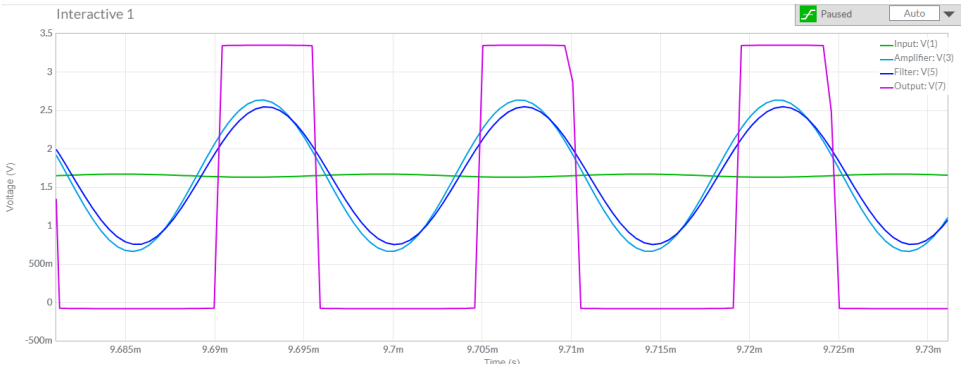

***Figura 24.** Simulació en el temps del circuit de condicionament. Segueix el mateix codi de colors que la figura 23*

Com es pot veure a les figures 23 i 24, en entrar un senyal altern al circuit de condicionament del receptor, s'amplificarà i es convertirà en un senyal de polsos. El circuit amplificarà les freqüències més properes als 69 kHz i filtrarà les altres.

A la figura 25, es pot apreciar un diagrama de blocs general del sistema que s'ha muntat per a fer les proves, que mostra el procés pel que passa el senyal des que és generat pel transmissor fins que és rebut pel receptor.

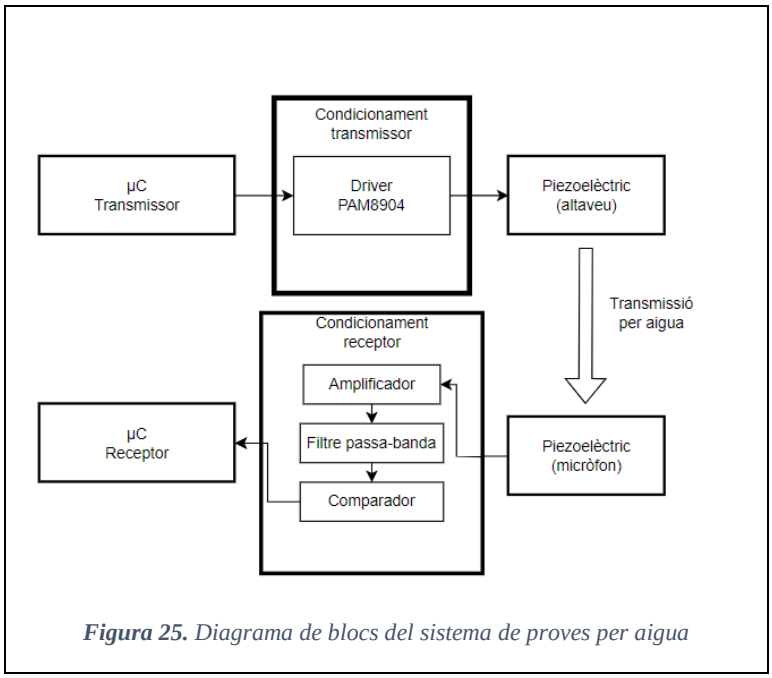

Figura 25. *Diagrama de blocs del sistema de proves per aigua*

Una vegada muntats els circuits, s'ha provat a enviar una sèrie de missatges. El circuit de condicionament ha estat capaç de filtrar les interferències de manera força efectiva, i per això els resultats obtinguts han estat els mateixos que en les proves per cable: els missatges s'han rebut i descodificat sense problemes.

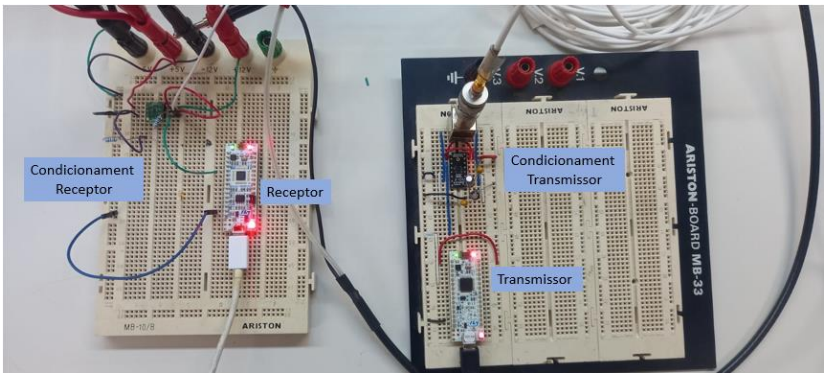

Figura 26. *Muntatge dels circuits sobre plaques de proves*

També s'ha observat amb un oscil·loscopi el senyal a la sortida del circuit de condicionament del transmissor durant l'enviament de varis missatges, per tal de comprovar el seu correcte funcionament. Primer de tot, s'ha comprovat que els trens de polsos tinguin la durada adequada. A la figura 27 es pot observar com aquesta és de 5 ms, que és la durada prevista quan s'emeten 345 polsos a 69 kHz. També s'hi observa el comportament del pin, que passa de tenir un *output* en estat flotant (que pel circuit de condicionament es mostra com neutre) a una successió d'estats positius i negatius (que correspondrien a una alternació de sortides positives i neutres del microcontrolador transmissor).

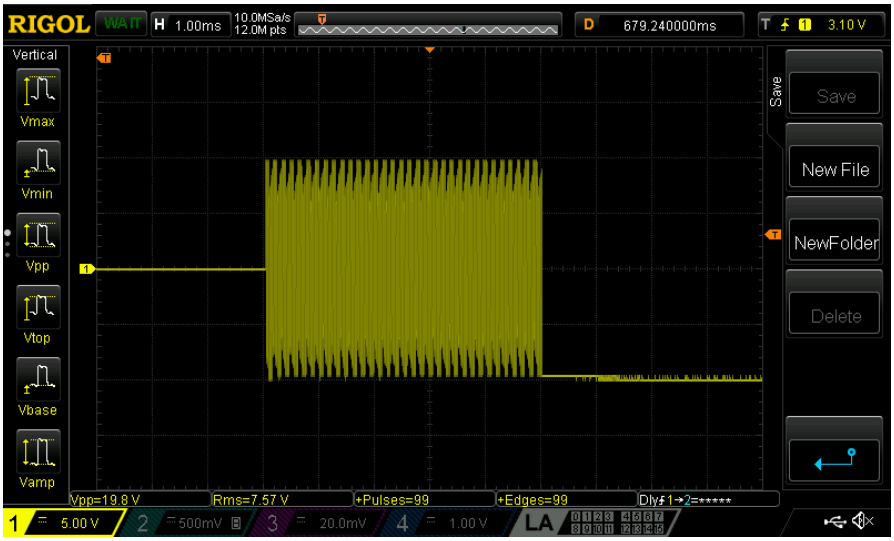

Figura 27. *Detall d'un tren de polsos a la sortida del transmissor*

A continuació, s'ha observat el correcte funcionament de la codificació de bits a temps entre polsos. A les figures 28, 29, 30 i 31 es pot veure com, per a cada comanda $PPM enviada pel receptor, hi ha un pols amb el temps d'espera corresponent a la captura d'oscil·loscopi.

En primer lloc, s'ha provat a enviar un missatge amb l'identificador 0x03 i la dada 0x39. En format binari, aquests valors són 000011 i 00111001 respectivament. Això correspon a una sèrie de trens de polsos amb els següents temps entre ells: 50 ms, 50 ms, 200 ms, 50 ms, 200 ms, 150 ms, 100 ms. Es pot observar com s'envien i es descodifiquen correctament a les figures 28 i 29.

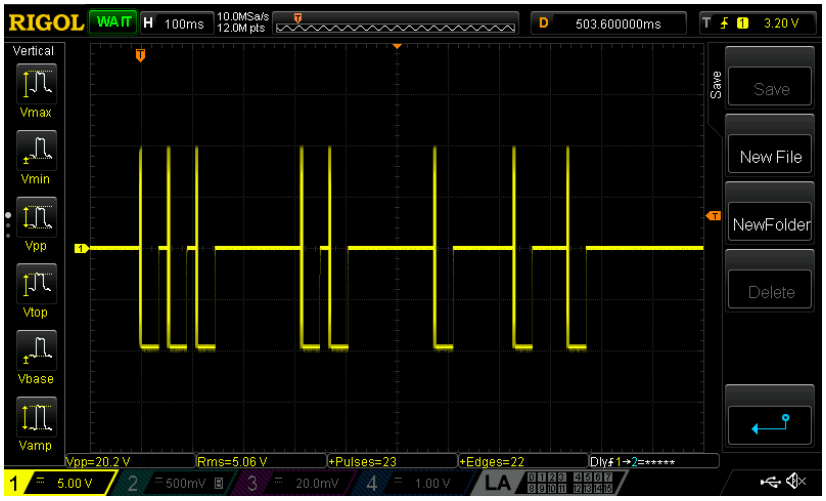

Figura 28. *Captura d'oscil·loscopi de la sortida del transmissor durant l'enviament del missatge ID:0x03, Dada: 0x39*

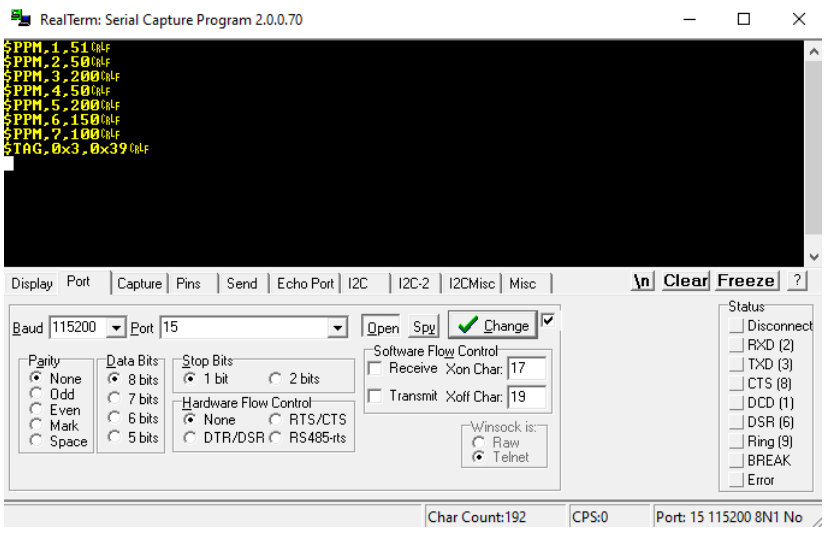

Figura 29. *Captura d'oscil·loscopi de la comunicació serial del receptor durant la recepció del missatge ID: 0x03, Dada: 0x39*

En segon lloc, s'ha enviat un missatge amb l'identificador 0x06 i la dada 0xE4. En format binari, aquests valors són 000110 i 1110 0100 respectivament, que correspon a una sèrie de trens de polsos amb temps: 50 ms, 100 ms, 150 ms, 200 ms, 150 ms, 100 ms, 50 ms. Es pot observar com s'envien i es descodifiquen correctament a les figures 30 i 31.

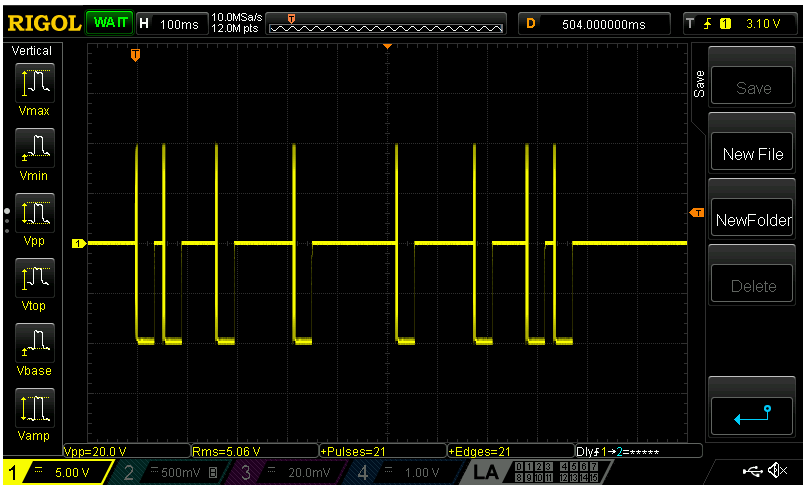

Figura 30. *Captura d'oscil·loscopi de la sortida del transmissor durant l'enviament del missatge ID:0x06, Dada:0xE4*

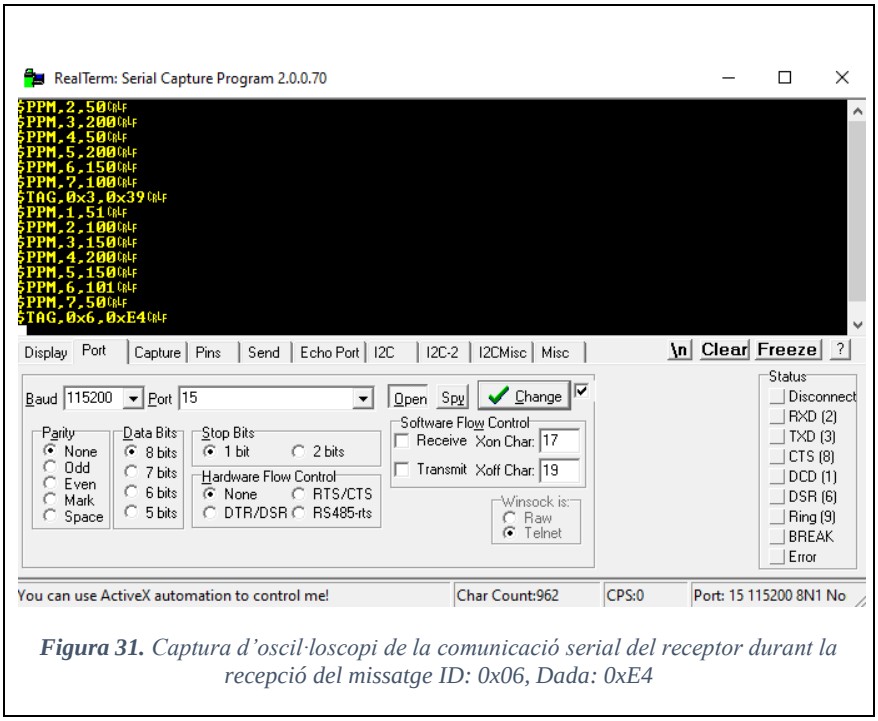

Figura 31. *Captura d'oscil·loscopi de la comunicació serial del receptor durant la recepció del missatge ID: 0x06, Dada: 0xE4*

Així doncs, s'ha provat amb varis missatges la correcta comunicació, fent servir valors que contenguin els 4 tipus de parelles de bits possibles (00, 01, 10, 11), i per tant, els 4 temps de retard que es poden donar en el protocol de comunicació emprat (50 ms, 100 ms, 150 ms, 200 ms). Queda així demostrat el correcte funcionament del sistema.

7. Conclusions

L'algorisme que s'ha programat per al receptor és capaç de rebre i descodificar correctament els missatges enviats en el protocol basat en modulació per posició de polsos que s'ha establert. També s'ha aconseguit una interfície funcional per a dur a terme el control del transmissor. Els dos dispositius són capaços de comunicar-se entre ells i de detectar possibles incoherències causades per interferències en el medi aquàtic de propagació.

Així i tot, aquest és només un punt de partida si es consideren totes les potencials innovacions que es poden afegir a un sistema d'etiquetatge de desenvolupament propi. Com s'ha descrit anteriorment, la implementació d'un dispositiu bidireccional que integri les funcions de transmissor i receptor programades en aquest treball, permetria una reducció de costos de desplegament d'infraestructura en el monitoratge d'espècies marines. També es podrien implementar nous protocols de modulació més com la OFDM (*Ortogonal Frequency Division Multiplexing*), que és més robusta que la PPM davant la propagació multicamí i té majors tasses de transmissió [23].

El programari de recepció de missatges es podria millorar implementant alguna mena d'algorisme de reconeixement de freqüències, per assegurar que no confongui canals de comunicació. En el sistema que s'ha presentat en aquest treball, els senyals rebuts es filtren analògicament amb el circuit de condicionament que s'ha implementat, però un filtre digital afegiria robustesa i podria obrir la possibilitat de rebre missatges codificats en més d'una freqüència.

Pel que fa a la interfície de control del transmissor, és totalment funcional però simple; es tracta d'una sèrie d'instruccions i comandes que l'usuari rep i envia a través de la consola de Python. Aquest programa es podria millorar fent més visual la interfície, encara que això no afegiria funcionalitats addicionals sinó que simplement augmentaria l'accessibilitat.

Encara que les possibilitats són moltes, en aquest treball s'han complert amb èxit els objectius que s'havien proposat. S'ha desenvolupat un programari funcional per a l'aplicació descrita amb la que es poden dur a terme comunicacions subaquàtiques amb èxit. Amb això es té un software des del qual provar els circuits que formaran part dels

TAGs de producció pròpia del projecte SASES i que servirà de base de totes les innovacions que es desenvolupin sobre ells.

Revisant els objectius assolits, es té que:

1. S'ha implementat un programa que dona al microcontrolador receptor la capacitat de rebre i descodificar missatges codificats amb el protocol de PPM establert.
2. S'ha implementat dins aquets programa la identificació de senyals erronis.
3. S'ha programat una interfície en llenguatge Python capaç de controlar el dispositiu transmissor per tal d'enviar missatges en el protocol establert.
4. S'ha comprovat la comunicació correcta entre els dos dispositius transmetent els senyals per cable.
5. S'ha comprovat que, davant la recepció de senyals que contenen errors, el receptor respon correctament identificant-los com a tals.
6. S'ha comprovat que la comunicació entre els dispositius funciona si el missatge es transmet a través de senyals acústics en un medi aquàtic.

8. Bibliografia

[1] InnovaSea Systems, Inc., 2021. *How do Innovasea coded tags work?*. Disponible a: https://support.vemco.com/s/article/How-do-VEMCO-coded-tags-work. [Consulta: 4 de desembre del 2022].

[2] LOTEK Wireless INC. (sense data). *Products | Lotek*. Disponible a: https://www.lotek.com/products/. [Consulta: 23 de febrer de 2023].

[3] HT Italia S.r.l.(sense data). *HTI - 290 - Acoustic Tags - Acoustic Tag Receiver*. Disponible a: https://www.environmental-expert.com/products/hti-model-290-acoustic-tag-receiver-630873. [Consulta: 23 de febrer de 2023].

[4] TAN, Hwee Pink et al., 2011. A survey of techniques and challenges in underwater localization. *Ocean Engineering*. Vol 38, no. 14-15, pp. 1663-1676. ISSN 0029-8018.

[5] CHAN, Yiu Tong and Ho, Kenneth C., 1994. A simple and efficient estimator for hyperbolic location. *IEEE Transactions on Signal Processing*, vol. 42, no. 8, pp. 1905-1915. DOI: 10.1109/78.301830

[6] ANDREWS, Kelly S. et al., 2011. Comparison of fine-scale acoustic monitoring systems using home range size of ademersal fish. *Marine Biology*. Vol. **158**, pp. 2377–2387. DOI: 10.1007/s00227-011-1724-5

[7] CORREGIDOR, Daniel et al., 2021. Analysis and initial design of bidirectional acoustic tag modulation schemes and communication protocol. *IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*. DOI: 10.1109/I2MTC50364.2021.9460108.

[8] European Tracking Network (sense data). *Important information for acoustic telemetry users*. Disponible a: https://www.europeantrackingnetwork.org/en/important-information-acoustic-telemetry-users. [Consulta: 30 de desembre del 2022].

[9] STMicroelectronics N.V (sense data). *STM32 Arm Cortex MCUs - 32-bit Microcontrollers - STMicroelectronics*. Disponible a: https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html [Consulta: 1 de gener del 2023].

[10] STMicroelectronics N.V (sense data). *STM32 Ultra Low Power Microcontrollers (MCUs)*. Disponible a:

https://www.st.com/content/st_com/en/products/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus/stm32-ultra-low-power-mcus.html. [Consulta: 1 de gener del 2023]

[11] MAGDY, Khaled. (sense data). STM32 HAL Library Tutorial – HAL Library Examples. A: *DeepBlue* [en línia]. Disponible a: https://deepbluembedded.com/stm32-hal-library-tutorial-examples/ [Consulta: 1 de gener de 2023].

[12] STMicroelectronics N.V., 2021. *Introduction Description of STM32L4/L4+ HAL and low-layer drivers UM1884 User manual*. Disponible a: https://www.st.com/resource/en/user_manual/um1884-description-of-stm32l4l4-hal-and-lowlayer-drivers-stmicroelectronics.pdf. [Consulta: 1 de gener de 2023].

[13] STMicroelectronics N.V., 2022. *Getting started with EXTI*. Disponible a: https://wiki.st.com/stm32mcu/wiki/Getting_started_with_EXTI. [Consulta: 1 de gener de 2023].

[14] STMicroelectronics N.V., 2022. *TIM internal peripheral - stm32mpu*. Disponible a: https://wiki.st.com/stm32mpu/wiki/TIM_internal_peripheral. [Consulta: 2 de gener de 2023].

[15] ST Microelectronics N. V., 2019. *STM32L476xx Datasheet*. Disponible a: https://www.st.com/resource/en/datasheet/stm32l476je.pdf. [Consulta: 1 de gener de 2023].

[16] MAGDY, Khaled. (sense data). STM32 USART / UART Tutorial - Example Interrupt DMA. A: *DeepBlue* [en línia]. Disponible a: https://deepbluembedded.com/stm32-usart-uart-tutorial/ [Consulta: 2 de gener de 2023].

[17] RealTerm (sense data). *RealTerm: Serial Terminal*. Disponible a: https://realterm.sourceforge.io/ [Consulta: 1 de gener de 2023].

[18] APC International, Ltd. (sense data). PZT Properties & PZT Manufacturing. A: *American Piezo* [en línia]. [Consulta: 23 d'abril de 2023]. Disponible a: https://www.americanpiezo.com/piezo-theory/pzt.html

[19] APC International, Ltd. (sense data). Soft Piezo & Hard Piezo Materials. A: *American Piezo* [en línia]. Disponible a:

https://www.americanpiezo.com/piezo-theory/ceramics.html. [Consulta: 23 d'abril de 2023].

[20] APC International, Ltd. (sense data). Piezo Vibration | Piezoelectric Elements Vibrator. A: *American Piezo* [en línia]. Disponible a: https://www.americanpiezo.com/knowledge-center/piezo-theory/vibration.html. [Consulta: 23 d'abril de 2023].

[21] APC International, Ltd. (sense data). Minimum & Maximum Impedance. A: *American Piezo* [en línia]. Disponible a: https://www.americanpiezo.com/knowledge-center/piezo-theory/determining-resonance-frequency.html. [Consulta: 23 d'abril de 2023].

[22] Diodes Incorporated, 2017. *18VPP output piezo sounder driver. Datasheet del PAM8904*. Disponible a: https://docs.rs-online.com/acd6/0900766b814b921a.pdf

[23] NAKASHIMA, Yusuke, MATSUOKA, Hosei Matsuoka and YOSHIMURA, Takeshi, 2006. Evaluation and Demonstration of Acoustic OFDM. *Fortieth Asilomar Conference on Signals, Systems and Computers*. California: IEEE. pp. 1747-1751, DOI: 10.1109/ACSSC.2006.355061.

Annex 1: Codi C del programa receptor. (Fitxer main.c)

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2022 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "string.h"
#include "stdio.h"
/* USER CODE END Includes */
```

```
/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
#define DEBUG 0         //Si es defineix com 1 es generen una sèrie de missatges via UART prescindibles però útils per a trobar errors de programació.
/* USER CODE END PTD */


/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */


/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */


/* USER CODE END PM */


/* Private variables ---------------------------------------------------------*/
TIM_HandleTypeDef htim7;


UART_HandleTypeDef huart2;


/* USER CODE BEGIN PV */


/* USER CODE END PV */


/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM7_Init(void);
/* USER CODE BEGIN PFP */
void descodificador(uint8_t tempsPassat);
```

```
//proves (provisional)
void trenpolsos();            //Funció



/* USER CODE END PFP */


/* Private user code ---------------------------------------------------
--------*/
/* USER CODE BEGIN 0 */


//Comunicació UART
char msg[50];
uint8_t msglen;
//Variables relatives a la recepció de
dades.....................................................


      //Comunicació UART
      short int dadaFallada=0;            //Flag que s'empra en cas de
què arribi un temps incorrecte


      //Temps
      uint8_t tempsActual=0;              //en milisegons, valor max
255
      uint8_t tempsPassat=0;
      short int timercorre=0;             //Flag per a comprovar si
corre el temporitzador


      // tractament dada
      int shiftbits=12;                         //Comptador per fer la
decodificació
      int dadaRebuda=0b0000000000000000;   //Els dos primers bits són
indiferents 0bXX00000000000000
      int IDrebut=0;


//.....................................................................
.............................
```

```
/* USER CODE END 0 */


/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */







  /* USER CODE END 1 */


  /* MCU Configuration--------------------------------------------------------*/


  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();


  /* USER CODE BEGIN Init */


  /* USER CODE END Init */


  /* Configure the system clock */
  SystemClock_Config();


  /* USER CODE BEGIN SysInit */


  /* USER CODE END SysInit */


  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
```

```
  MX_TIM7_Init();
  /* USER CODE BEGIN 2 */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_SET);
  //Es comunica inici del programa per uart
  msglen=sprintf(msg, "Iniciant codi\r\n");
  HAL_UART_Transmit(&huart2, (uint8_t*)msg, msglen, 100);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) !=
HAL_OK)
  {
    Error_Handler();
  }
```

```
  /** Configure LSE Drive Capability
  */
  HAL_PWR_EnableBkUpAccess();
  __HAL_RCC_LSEDRIVE_CONFIG(RCC_LSEDRIVE_LOW);


  /** Initializes the RCC Oscillators according to the specified
parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType =
RCC_OSCILLATORTYPE_LSE|RCC_OSCILLATORTYPE_MSI;
  RCC_OscInitStruct.LSEState = RCC_LSE_ON;
  RCC_OscInitStruct.MSIState = RCC_MSI_ON;
  RCC_OscInitStruct.MSICalibrationValue = 0;
  RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_6;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_MSI;
  RCC_OscInitStruct.PLL.PLLM = 1;
  RCC_OscInitStruct.PLL.PLLN = 16;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV7;
  RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
  RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }


  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK

|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
```

```
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) !=
HAL_OK)
  {
    Error_Handler();
  }

  /** Enable MSI Auto calibration
  */
  HAL_RCCEx_EnableMSIPLLMode();
}

/**
  * @brief TIM7 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM7_Init(void)
{

  /* USER CODE BEGIN TIM7_Init 0 */

  /* USER CODE END TIM7_Init 0 */

  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM7_Init 1 */

  /* USER CODE END TIM7_Init 1 */
  htim7.Instance = TIM7;
  htim7.Init.Prescaler = 3200-1;
  htim7.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim7.Init.Period = 10-1;
  htim7.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim7) != HAL_OK)
  {
```

```
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim7, &sMasterConfig) !=
HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM7_Init 2 */

  /* USER CODE END TIM7_Init 2 */

}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
```

```
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD3_GPIO_Port, LD3_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : PA1 */
  GPIO_InitStruct.Pin = GPIO_PIN_1;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
```

```
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : LD3_Pin */
  GPIO_InitStruct.Pin = LD3_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD3_GPIO_Port, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI1_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI1_IRQn);

}

/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
      /*
       * Aquesta funció s'activa cada vegada que es rep un pols a
l'entrada del microcontrolador
       * */
      if(GPIO_Pin == GPIO_PIN_1) {        //Comprovació PIN 1

            HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_3);          //Canvi
d'estat LED intern del microcontrolador
            tempsPassat=tempsActual;
            tempsActual=0;

            if (!timercorre) {
                  HAL_TIM_Base_Start_IT(&htim7);
                  shiftbits=12;
                  timercorre=1;
            }

            if (tempsPassat>10){                //Així es filtren els
polsos individuals d'un tren
```

```
                    if (DEBUG) msglen=sprintf(msg, "temps %u\r\n",
tempsPassat);

                    if (DEBUG) HAL_UART_Transmit(&huart2, (uint8_t*)msg,
msglen, 10);

                    msglen=sprintf(msg, "$PPM,%d,%u\r\n",(int)((12-
shiftbits)/2+1),tempsPassat);

                    HAL_UART_Transmit(&huart2, (uint8_t*)msg, msglen, 1);



                    if (((tempsPassat>=40 && tempsPassat<=60) ||
(tempsPassat>=90 && tempsPassat<=110) || (tempsPassat>=140 &&
tempsPassat<=160) || (tempsPassat>=190 && tempsPassat<=210)) &&
shiftbits>=0) {

                          //Es crida el descodificador si els temps entre
polsos es troba dins els marges correctes

                          descodificador(tempsPassat);

                    }

                    //Errors possibles:

                    else if(tempsPassat>=210 && shiftbits>=0){

                          //Ha passat molt de temps des del darrer pols i
encara no s'ha acabat de transmetre la dada

                          //Es reinicia el procés per obtenir una nova
dada

                          HAL_TIM_Base_Stop_IT(&htim7);

                          timercorre=0;


                          shiftbits=12;

                          dadaRebuda=0;

                          IDrebut=0;

                          tempsActual=0;

                          tempsPassat=0;

                          dadaFallada=0;


                          msglen=sprintf(msg, "ERROR, massa temps des del
darrer pols\r\n");

                          HAL_UART_Transmit(&huart2, (uint8_t*)msg,
msglen, 10);

                    }

                    else if( (shiftbits<0)){
```

```
                        //Nova dada
                        HAL_TIM_Base_Stop_IT(&htim7);
                        timercorre=0;

                        shiftbits=12;
                        dadaRebuda=0;
                        IDrebut=0;
                        tempsActual=0;
                        tempsPassat=0;
                        dadaFallada=0;

                        msglen=sprintf(msg, "ERROR, no s'ha
reinicialitzat bé la dada\r\n");
                        HAL_UART_Transmit(&huart2, (uint8_t*)msg,
msglen, 10);
                }
                else{
                        //Es vol que el codi segueixi rebent la resta
de la transmissió pero que no ho interpreti com una dada
                        dadaRebuda=0;
                        IDrebut=0;
                        dadaFallada=1;

                        msglen=sprintf(msg, "ERROR, s'ha rebut un temps
invalid\r\n");
                        HAL_UART_Transmit(&huart2, (uint8_t*)msg,
msglen, 10);

                        shiftbits-=2;

                }
            }
        }
        else {
            __NOP();
        }
}
```

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
	//Funcó per al control del temporitzador
	if (htim == &htim7 ) {
		HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_3);
		//Incrementa en 1 el temporitzador (Passa cada 1 ms)
		if (tempsActual<210){		//Temps màxim del temporitzador
			tempsActual++;
		}
		else if (tempsActual>=210){
			HAL_TIM_Base_Stop_IT(&htim7);

			if (shiftbits>0 && shiftbits!=12){
				//Si abans d'acabar de rebre un missatge hi ha un interval massa llarg
				msglen=sprintf(msg, "AVIS: Interval>210ms. Missatge incomplert\r\n");
				HAL_UART_Transmit(&huart2, (uint8_t*)msg, msglen, 10);
			}

			tempsActual=0;
			tempsPassat=0;
			shiftbits=12;
			dadaRebuda=0;
			IDrebut=0;
			timercorre=0;
			dadaFallada=0;



		}

	}
}
```

```
void descodificador(uint8_t tempsPassat){

	short int bits;
	//if (DEBUG) msglen=sprintf(msg, "temps %u\r\n", tempsPassat);
	//if (DEBUG) HAL_UART_Transmit(&huart2, (uint8_t*)msg, msglen,
10);
	if (tempsPassat<=60 && tempsPassat>=40){
		bits=0;
	}
	else if(tempsPassat<=110 && tempsPassat>=90){
		bits=1;
	}
	else if(tempsPassat<=160 && tempsPassat>=140){
		bits=2;
	}
	else if(tempsPassat<=210 && tempsPassat>=190){
		bits=3;
	}

	if (shiftbits>=8){
		//Bits 13 a 8 van a l'ID
		IDrebut= IDrebut | (bits << (shiftbits-8));
		if (DEBUG) msglen=sprintf(msg, "shifted %d\r\n",
shiftbits);
		if (DEBUG) HAL_UART_Transmit(&huart2, (uint8_t*)msg,
msglen, 10);
	}
	else if (shiftbits<8){
		//Bits 7 a 0 van a dada
		dadaRebuda= dadaRebuda | (bits << shiftbits);
		if (DEBUG) msglen=sprintf(msg, "shifted %d\r\n",
shiftbits);
		if (DEBUG) HAL_UART_Transmit(&huart2, (uint8_t*)msg,
msglen, 10);
	}
```

```
	shiftbits-=2;


	if (shiftbits<0){
		//Final de la transmissió d'un missatge
		HAL_TIM_Base_Stop_IT(&htim7);
		tempsPassat=0;
		tempsActual=0;
		shiftbits=12;
		if (!dadaFallada){
			//Comunicació de la informació rebuda
			msglen=sprintf(msg, "$TAG,0x%X,0x%X\r\n", IDrebut,
dadaRebuda);
			HAL_UART_Transmit(&huart2, (uint8_t*)msg, msglen,
10);
			shiftbits=12;
		}


		dadaRebuda=0;
		IDrebut=0;
		timercorre=0;
		dadaFallada=0;
	}
}



void trenpolsos(){


	//Funció provisional que simula un tren de 8 polsos
	//Aquesta funció va ser implementada per a provar la funció
"descodificador" aïllant-la de la funció de recepció.
	/*Cada crida a HAL_GPIO_EXTI_Callback(GPIO_PIN_10); Simula una
interrupció causada per la rebuda d'un pols.
	 * Per a simular un missatge, al bucle infinit de "main" es
cridava la present funció separada per uns HAL_Delay(), els valors
	 * de temps dels quals corresponien a la codificació d'una
parella de bits
	 * */
	HAL_GPIO_EXTI_Callback(GPIO_PIN_10);
```

```
        HAL_Delay(1);
        HAL_GPIO_EXTI_Callback(GPIO_PIN_10);
        HAL_Delay(1);
        HAL_GPIO_EXTI_Callback(GPIO_PIN_10);
        HAL_Delay(1);
        HAL_GPIO_EXTI_Callback(GPIO_PIN_10);
        HAL_Delay(1);
        HAL_GPIO_EXTI_Callback(GPIO_PIN_10);
        HAL_Delay(1);
        HAL_GPIO_EXTI_Callback(GPIO_PIN_10);
        HAL_Delay(1);
        HAL_GPIO_EXTI_Callback(GPIO_PIN_10);
        HAL_Delay(1);
        HAL_GPIO_EXTI_Callback(GPIO_PIN_10);
        HAL_Delay(1);

}

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error
return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}
```

```
#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

Annex 2: Codi Python de la interfície de control del transmissor

```
#Llibreries
import serial

#VARIABLES GLOBALS
ser=None
ID=0b111100
Dada=0b11001010
port = '7'

#FUNCIONS
def obreConnexio():
    global ser
    #Obre serial
    ser = serial.Serial()
    #Paràmetres Port
    ser.port = 'COM'+port
    ser.baudrate = 115200
    ser.timeout=0.100
    #Obrir port
    try:
        ser.open()
    except serial.SerialException as errorMsg:  #Trobar error
        print("S'ha trobat una excepció:")
        print("->", errorMsg)


    if ser.is_open:
        comunica=comprovaConnexio()
        return comunica
    else:
        print("No s'ha trobat un dispositiu")
        ser.close()
        print("S'ha tancat el port serial. Si us plau, torni's a
connectar")
        return False
```

```
def comprovaConnexio():
    global ser
    comanda='$AT\r'
    ser.write(comanda.encode('ascii'))
    print(comanda)

    #Comprovació resposta
    msg=ser.read_until('\n'.encode('ascii'))
    print(msg)
    if msg == (comanda+'\n').encode('ascii') or msg ==
('\n'+comanda+'\n').encode('ascii'):
        print("S'ha connectat correctament")
        return True
    else:
        print(msg)
        print("El microcontrolador no respon correctament")
        ser.close()
        print("S'ha tancat el port serial. Si us plau, torni's a
connectar")
        return False

def triaID():

    ID=int(input("Introdueixi un Identificador: "),2)
    print(bin(ID))
    return ID

def triaDada():

    Dada=int(input("Introdueixi una Dada: "),2)
    print(bin(Dada))
    return Dada

def configuraTren():
    global ser
```

```
    #Es comprova que estigui obert
    if ser.is_open:
        #Configuració paràmtres de Cadena
        comanda='$CP,069,008,3\r'
        print(comanda)
        ser.write(comanda.encode('ascii'))

        #Comprovació resposta
        msg=ser.read_until('\n'.encode('ascii'))
        if msg != (comanda+'\n').encode('ascii') and msg !=
('\n'+comanda+'\n').encode('ascii'):
            print(msg)
            print("El microcontrolador no respon correctament")
            ser.close()
            print("S'ha tancat el port serial. Si us plau, torni's a
connectar")
            return False

        parelles=[]
        timestring=[]
        #Conversió ID i Dada a string, separant l'inici "0b"
        strID=(str(bin(ID)).split('b')[1]).zfill(6)
        strDada=(str(bin(Dada)).split('b')[1]).zfill(8)
        for i in range(0, 6, 2):
            parelles.append(strID[i]+strID[i+1])

        for i in range(0, 8, 2):
            parelles.append(strDada[i]+strDada[i+1])

        for bits in parelles:
            #Podria ser un match-case però no seria compatible amb
versions més antigues que Python 3.10
            if bits=='00':
                timestring.append('0500')
            elif bits=='01':
                timestring.append('1000')
```

```
            elif bits=='10':
                timestring.append('1500')
            elif bits=='11':
                timestring.append('2000')
        print(timestring)
        for i in range(0,7):
            #Configuració de cada tren de polsos
            print(i)
            comanda='$CT,00'+str(i+1)+',345,'+timestring[i]+'\r'
            print(comanda)
            ser.write(comanda.encode('ascii'))

            #Comprovació resposta
            msg=ser.read_until('\n'.encode('ascii'))
            print(msg)
            if msg != (comanda+'\n').encode('ascii') and msg !=
('\n'+comanda+'\n').encode('ascii'):
                print(msg)
                print("El microcontrolador no respon correctament")
                ser.close()
                print("S'ha tancat el port serial. Si us plau, torni's
a connectar")
                return False
        #Tren de tancament
        comanda='$CT,008,345,2100\r'
        print(comanda)
        ser.write(comanda.encode('ascii'))

        #Comprovació resposta
        msg=ser.read_until('\n'.encode('ascii'))
        print(msg)
        if msg != (comanda+'\n').encode('ascii') and msg !=
('\n'+comanda+'\n').encode('ascii'):
            print(msg)
            print("El microcontrolador no respon correctament")
            ser.close()
```

```
            print("S'ha tancat el port serial. Si us plau, torni's a
connectar")
            return False
    else:
        print("Connecti's al microcontrolador")
        return False
    return True



def enviaTren():
    global ser

    if (not ser.is_open):
        print("Connecti's al microcontrolador")
        return False
    #Comanda d'enviament
    comanda="$SP\r"
    print(comanda)
    ser.write(comanda.encode('ascii'))

    #Comprovació resposta
    msg=ser.read_until('\n'.encode('ascii'))
    print(msg)
    if msg != (comanda+'\n').encode('ascii') and msg !=
('\n'+comanda+'\n').encode('ascii'):
        print(msg)
        print("El microcontrolador no respon correctament")
        return False
    input("Premi 'Enter per parar l'enviament")

    comanda="$ST\r"
    print(comanda)
    ser.write(comanda.encode('ascii'))

    #Comprovació resposta
    msg=ser.read_until('\n'.encode('ascii'))
```

```
    print(msg)
    if msg != (comanda+'\n').encode('ascii') and msg !=
('\n'+comanda+'\n').encode('ascii'):
        print(msg)
        print("El microcontrolador no respon correctament")
        return False


#BUCLE MENÚ PRINCIPAL
surt=False
while(not(surt)):
    #Selecció opció
    print("ID: ", hex(ID), "Dada:", hex(Dada))
    print("Port: ", 'COM'+port)
    print("Introdueixi un caràcter per a seleccionar el mode:")
    print("C: Connectar amb el microcontrolador")
    print("I: Modificar l'ID")
    print("D: Modificar la dada")
    print("T: Transmetre")
    print("P: Canvi de port")
    print("S: Sortir del programa")
    mode=input("Esculli una opció: ")

    if mode=='C' or mode=='c':
        print("Ha seleccionat: Connectar amb el microcontrolador")
        obreConnexio()
        input("Premi 'Enter' per a continuar")

    elif mode=='I' or mode=='i':
        print("Ha seleccionat: Modifica l'ID")

        ID=triaID()

        input("Premi 'Enter' per a continuar")

    elif mode=='D' or mode=='d':
```

```
        print("Ha seleccionat: Modifica la dada")

        Dada=triaDada()

        input("Premi 'Enter' per a continuar")

    elif mode=='T' or mode=='t':
        print("Ha seleccionat: Transmetre")
        #Comprovació connexió
        if ser.is_open:
            comprovaConnexio()
            configurat=configuraTren()
            if configurat:
                enviaTren()
        else:
            print("Connecti's al microcontrolador")

        input("Premi 'Enter' per a continuar")
    elif mode=='P' or mode=='p':
        print("Ha seleccionat: Canvi de port")

        #Comprovar que hi hagi una connexió oberta
        if ser!=None:
            if ser.is_open:
                ser.close()
        port=input("Insereixi un nombre de port (nombre enter): ")
        if port.isnumeric():
            port=str(port)
        else:
            print("Entrada invàlida")
            port=input("Si us plau, insereixi un nombre de port
(nombre enter): ")

    elif mode=='S' or mode=='s':
        if ser.is_open:
            ser.close()
```

```
        print("Sortint")
        surt=True

    else:
        print("Caràcter invàlid")
```