Universitat de Girona
Escola Politècnica Superior

Treball final de grau

Estudi: Grau en Enginyeria Electrònica Industrial i Automàtica

Títol: Control del pitch i monitorització de l'estructura mecànica i del vent d'un aerogenerador de 2,2 kW

Document: 1. Memòria

Alumne: Pol Carreras i Viñas

Tutor: Lluís Pacheco Valls

Departament: ATC

Àrea: ATC

Convocatòria (mes/any): juny/2023

ÍNDEX

1. INTRODUCCIÓ

El present projecte s'emmarca dins el projecte "Llavor" concedit per la Generalitat de Catalunya a la Universitat de Girona. Concretament concedit als grups de recerca GREFEMA, MICELAB i VICOROB dels departaments d'Enginyeria Mecànica i de la Construcció Industrial, Enginyeria Elèctrica, Electrònica i Automàtica, i Arquitectura i Tecnologia de Computadors, que investiguen sobre la millora de sistemes de producció d'energia renovable.

Aquest projecte consisteix en dissenyar un aerogenerador de petita escala ubicat a l'ECOgranja Vilà del veïnat de Pols, situat al municipi alt-empordanès d'Ordis. Es basa en desenvolupar una turbina eòlica de dos kilowatts adaptada a diferents règims de vent. Inicialment, no existia cap element electrònic en l'aerogenerador, és a dir, s'ha hagut de desenvolupar tot el sistema. El projecte Llavor consta de diferents parts: l'estructura mecànica, el mecanisme de pitch, la part elèctrica i electrònica.

1.1. Antecedents

En el conjunt del projecte Llavor, aquest treball final de grau es centra en el control del pitch i la monitorització, que es duen a terme mitjançant dos microcontroladors, un per cada tasca. Aquests es comuniquen amb un altre microcontrolador, que té a càrrec el control del consum de potència del molí, mitjançant una xarxa multi-mestre d'I2C autònoma sense necessitat de control extern. A més, el sistema queda interconnectat, actuant com a esclau, en una xarxa Modbus amb un PLC mestre que permet adquirir i modificar paràmetres per tenir un control total dels processos de l'aerogenerador.

1.2. Objecte

L'objecte d'aquest treball consisteix a desenvolupar un control actiu de pitch fiable, així com una monitorització que permeti comprovar l'estat de l'estructura mecànica i del vent en tot moment. El conjunt haurà de treballar de manera coordinada amb els altres dispositius del molí perquè l'aerogenerador sigui el més eficient possible, robust a pertorbacions i segur per l'entorn i les persones.

1.3. Especificacions i abast

L'abast del treball final de grau es basa en controlar la velocitat de rotació del rotor. Per portar-ho a terme es dissenyarà un sistema en el qual el microcontrolador pugui

modificar l'angle d'inclinació de les pales. Es recolliran dades d'un sensor de posició, un encoder i un sensor de temperatura que determinaran el moviment a realitzar per mantenir la velocitat de rotació desitjada, s'aplicarà a la senyal resultant un control per PI des del software de l'entorn de treball del microcontrolador.

També s'obtindran i es tractaran dades de diversos sensors amb l'altre microcontrolador per tal de monitoritzar l'estructura i el vent. La velocitat i direcció del vent es llegiran d'un anemòmetre, les vibracions de l'estructura es mesuraran amb un acceleròmetre i per determinar els esforços dels cables tensors s'utilitzaran unes galgues extensiomètriques. El microcontrolador podrà enviar un senyal d'alerta a la resta del sistema quan les variables mesurades superin certs límits prèviament establerts.

Per tal d'aconseguir els objectius d'aquest sistema es farà el disseny de les PCBs per poder interconnectar els microcontroladors amb els busos de comunicació i els circuits de potència pertinents. A l'hora d'escriure el codi s'ha de vetllar que el dispositiu tingui el menor consum possible, és a dir, en situacions on no intervingui, aquest s'ha de posar en mode baix consum.

2. SISTEMA DE L'AEROGENERADOR

L'energia eòlica es basa en aprofitar la força del vent per moure unes aspes que transformen l'energia cinètica del vent en energia elèctrica. Aquesta energia constitueix un dels sectors energètics que més ha crescut en els últims anys, degut a la necessitat de fonts d'energia renovables, i en alguns països ja cobreix una fracció notable del subministrament. No obstant això, el seu principal inconvenient recau en la inestabilitat de la producció, ja que depèn directament de fenòmens meteorològics.

La màquina que permet aprofitar l'energia eòlica per convertir-la en energia elèctrica s'anomena aerogenerador. A nivell domèstic no es solen utilitzar degut a l'inconvenient abans mencionat. Per tal d'extreure'n el màxim rendiment es requereixen bateries que emmagatzemin l'energia, provocant que la inversió inicial sigui difícilment recuperable fins un període de temps significatiu.

Actualment, hi ha dos tipus principals d'aerogeneradors, els d'eix vertical i els d'eix horitzontal. Aquests últims, els més usats, permeten cobrir un ampli rang d'aplicacions, des d'instal·lacions aïllades de petita potència fins a instal·lacions en grans parcs eòlics.

L'aerogenerador del projecte, veure figura 1, no es troba connectat a la xarxa elèctrica, forma part d'una instal·lació aïllada d'una granja situada al poble d'Ordis, Alt Empordà, és d'eix horitzontal, de 15 metres i té una potència màxima de 2,2 kW. L'aerogenerador es troba a 1 km de distància de la granja.

Figura 1: Aerogenerador d'Ordis

El projecte en qüestió es centra en dos parts concretes del molí, amb tasques clarament diferenciades.

Es farà un control del "pitch" de l'aerogenerador, és a dir de l'angle d'inclinació de les pales, per tal d'obtenir el màxim rendiment de l'energia del vent. Aquest control es realitzarà mitjançant una senyal PWM connectada a un relé MOSFET que activarà un motor de desplaçament lineal connectat a les pales. Es recolliran dades d'un sensor lineal per tal de tancar el llaç de control i s'utilitzarà la velocitat de rotació de les pales per determinar la consigna de pitch.

Per altra banda es recolliran les dades dels sensors de velocitat i direcció del vent, esforços de l'estructura, esforços dels cables tensors i vibracions. Aquests sensors serveixen de suport per altres tasques, aporten seguretat i informació de l'estat del sistema de l'aerogenerador. Si les dades obtingudes es troben fora dels rangs de funcionament normal i es preveu que es poden causar danys en l'estructura s'ha d'activar l'estat d'emergència per tal de frenar el molí.

Cadascuna d'aquestes tasques serà realitzada per un microcontrolador, i aquests es trobaran connectats a PCBs instal·lades en la pròpia estructura de l'aerogenerador, en la base d'aquest. En elles es realitzen les connexions entre el microcontrolador i la resta de dispositius necessaris per realitzar les funcions.

El sistema complet que es vol instal·lar en l'aerogenerador inclou un tercer microcontrolador que no és objecte d'aquest projecte. Aquest s'encarrega d'administrar l'energia obtinguda, alimentant la bateria, emplenant dipòsits mitjançant bombes hidràuliques o activant focus per alliberar energia sobrant, segons correspongui.

El funcionament de l'aerogenerador és autònom utilitzant el bus I2C entre els microcontroladors. Es configuren una sèrie de missatges predeterminats per tal de transmetre informació entre els dispositius, aquests es poden veure en el subapartat 5.1.1. Per altra banda s'habilita també una comunicació externa per tal de permetre la monitorització i modificar la configuració del sistema mitjançant un PLC situat a la granja.

En la figura 2 es pot veure un esquema del sistema electrònic al complet del molí. Dins del sistema de l'aerogenerador s'anomena als microcontroladors esclaus, en referència a la comunicació RS-485 entre elles i el PLC, que actua com a mestre.

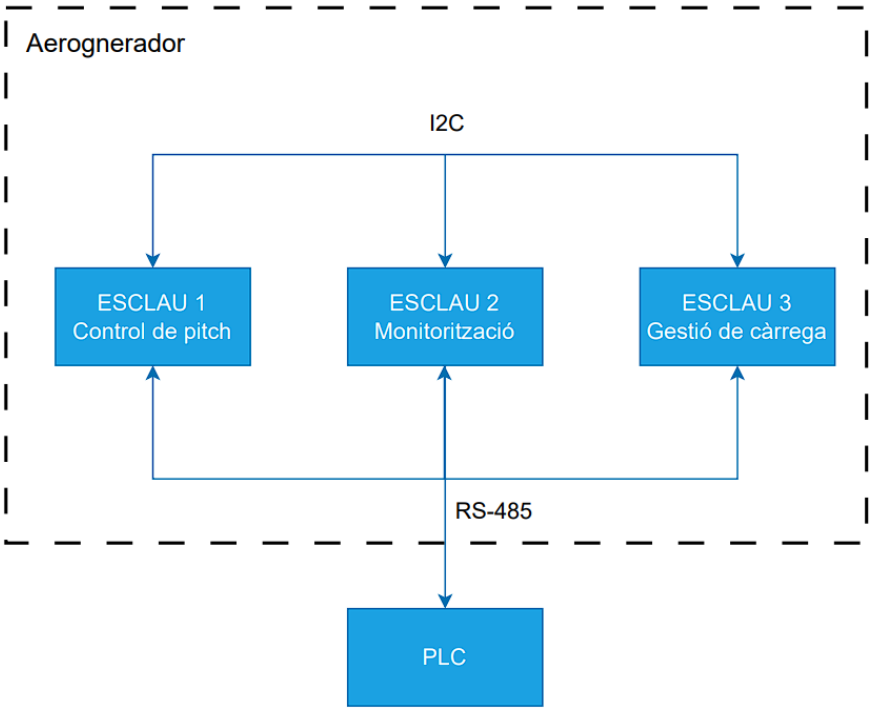

Figura 2: Sistema electrònic de l'aerogenerador

Aquest primer model inicial de petit aerogenerador controlat servirà com a prototip per tal de valorar la viabilitat de la seva producció.

2.1. Sensors, actuadors i connexions

Per tal de realitzar les accions del control de pitch s'utilitza:

Un actuador lineal de 24V connectat mecànicament a les pales a través de l'estrella. El seu moviment lineal es converteix per mitjà d'un sistema biela-manovella en moviment rotatiu dels eixos de les pales. El motor, té un parell màxim de 2500N i una velocitat màxima de moviment de 47,2mm/s a plena càrrega. Es troba ubicat a la punta del molí juntament amb dos fi de cursa per bloquejar el moviment.

En la figura 3 es pot veure el mecanisme connectat als tres eixos de les pales. Aquesta estructura correspon a un prototip imprès en 3D i no forma part del mecanisme de pitch de l'aerogenerador real. Quan s'aplica una força en la peça triangular del centre del mecanisme (estrella), aquesta es desplaça endavant o endarrere provocant la rotació dels eixos, que es troben connectats a unes ranures en les arestes del triangle. L'estructura resultant s'assimila a un jou escocès on el moviment rectilini d'una guia es transmet a moviment rotatori de la manovella i el seu arbre.

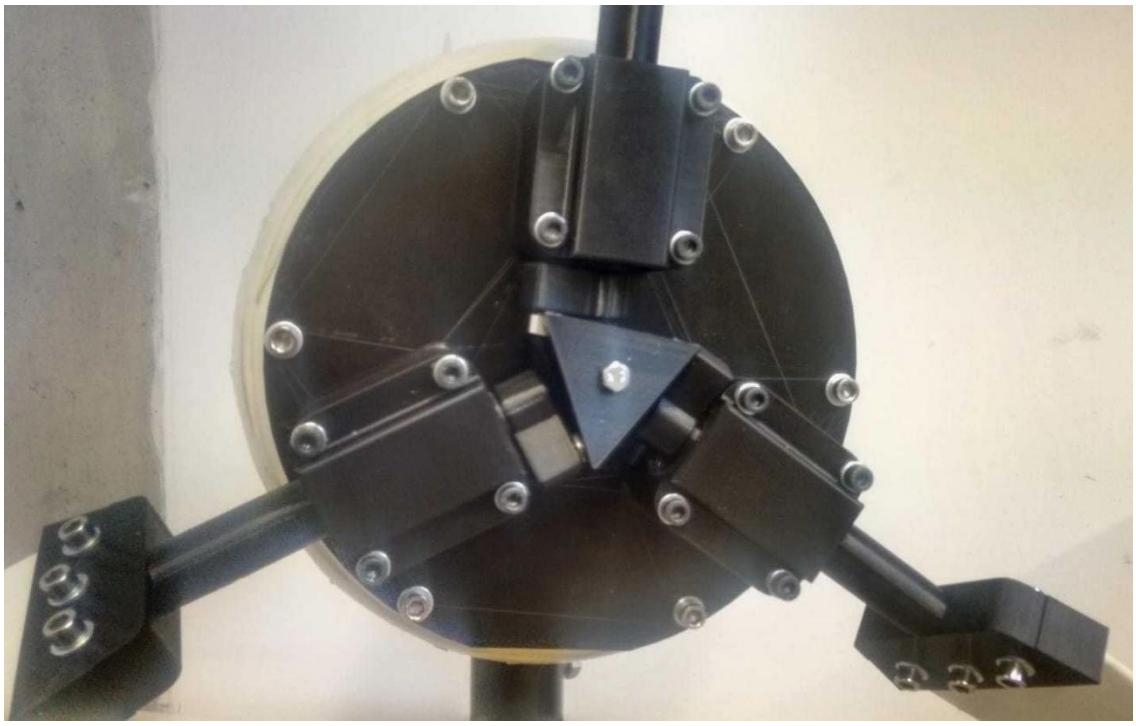

Figura 3: Prototip del mecanisme de pitch

En el dimensionament dels cables d'alimentació del motor pel que fa al criteri d'intensitat màxima admissible del conductor, s'ha seguit la ITC-BT-19. Els aïllants tenen un nivell de 450/750 V, i estan fabricats de XLPE, que és un material lliure d'halògens. Els cables unipolars usats son del tipus H07Z1-K. A plena càrrega, el motor té un consum màxim de 316,8W, per tant s'escull una secció de 25mm^2 per tal de reduir, en la mesura del possible, la caiguda de tensió en els cables.

El sistema d'alimentació del motor es troba en la base del molí i consta d'un fusible de 20A per protegir tot el circuit de potència de la PCB, un mòdul de relé de 2 canals per fer la inversió de gir, un MOSFET per alimentar el disparador del relé de 2 canals a 24V i un altre MOSFET per augmentar el senyal PWM provinent del microcontrolador de 5V a 24V. S'explica més extensament el seu funcionament en l'apartat 3.1. S'alimenta el sistema a la tensió de 24V que proporciona el controlador de càrrega. Aquest controlador de càrrega forma part del sistema de control de potència i s'encarrega d'alimentar també l'inversor i el mecanisme de frenat elèctric.

Per adquirir les dades de control s'utilitza un sensor lineal per adquirir els valors de posició i un encoder per llegir la velocitat de rotació de les pales de l'aerogenerador, ambdós col·locats a la punta de l'aerogenerador. S'utilitza també un sensor de temperatura, col·locat a la base, per compensar la temperatura del sensor lineal.

Per últim es disposa dues plaques extensores I2C i un mòdul UART/RS-485 per les comunicacions; un convertidor voltatge/intensitat, un convertidor intensitat/voltatge i dos altres mòduls UART/RS-485, per fer arribar les senyals del sensor de posició i de l'encoder, respectivament.

La PCB amb el microcontrolador i les connexions de tots els components anteriors es trobarà situada a la part inferior de l'aerogenerador, en el quadre elèctric.

Per altra banda, per les accions de monitorització de l'estructura mecànica i del vent es fa servir:

Un anemòmetre per llegir la velocitat i direcció del vent, permet detectar quan la velocitat del vent supera els límits de l'aerogenerador. 16 galgues extensiomètriques en estructura de pont connectades a 4 mòduls amplificadors de voltatge, un d'aquests ponts es trobarà situat en el màstil, els altres 3 es trobaran en els cables tensors que el suporten, a partir de la senyal de sortida es determinen els esforços als quals està subjecte l'estructura. Per últim una placa d'acceleròmetre per llegir les vibracions, un excés de vibracions pot indicar que s'ha malmès una pala. Aquests sensors, juntament amb el sensor de posició i l'encoder del sistema de control de pitch, es trobaran col·locats a una PCB pròpia en la gòndola de l'aerogenerador. En aquesta PCB es realitzaran les connexions entre els sensors i els mòduls per tal de transportar els senyals fins al microcontrolador.

Els mòduls utilitzats en aquest cas inclouen una placa convertidora de nivell i una placa extensora pel bus I2C de l'acceleròmetre, dos mòduls UART/RS-485 per la senyal de velocitat de l'anemòmetre, 5 convertidors voltatge/intensitat i 5 convertidors intensitat/voltatge per la senyal de direcció de l'anemòmetre i les senyals de sortida dels 4 amplificadors.

Per les comunicacions es fan servir dues plaques extensores I2C i un mòdul UART/RS-485.

La PCB del microcontrolador en aquest cas també estarà situada en la base de l'aerogenerador, dins el quadre elèctric. En ella es col·loquen els mòduls amb totes les connexions provinents dels sensors.

Per les alimentacions de 24V i 5V de tots els components i PCBs s'utilitzen cables unipolars de tipus H07Z1-K, d'1,5mm^2 o 2,5mm^2 segons correspongui per longitud del cable. El color dels cables s'ha escollit segons els criteris marcats per la norma UNE-EN 60204-1. Segons aquesta mateixa norma els cables de control provinents dels

convertidors voltatge/intensitat tindran aïllament de PVC i una secció de 1,5mm^2. Pel bus I2C s'utilitzen cables de parell trenat RJ45 i cables de parell trenat apantallat per l'RS-485. S'aprofita la pròpia estructura del molí per passar els cables de la base a la gòndola, on hi ha la PCB dels sensors i el motor.

2.2. Microcontrolador

Els microcontroladors són petits ordinadors formats per un sol circuit integrat. Contenen una o més CPUs juntament amb memòria i perifèrics d'entrada/sortida programables. Tenen una mida i un preu més reduïts comparats amb un disseny que utilitzi un microprocessador, memòria i dispositius d'entrada/sortida per separat.

Els dispositius escollits per processar les accions del sistema són microcontroladors PIC18F47Q10 de la marca Microchip. Els PIC són microcontroladors senzills i econòmics. A diferència d'altres microcontroladors, com ESP32 o Arduino, són més configurables a baix nivell, i compten amb una amplia gamma de models segons els mòduls, nombre d'entrades/sortides i potència de processament desitjada.

Aquests PIC18 es troben incorporats en una placa de desenvolupament Curiosity HPC, que es pot observar en la figura 4. S'ha escollit aquesta placa ja que inclou un programador/"debugger" integrat amb interfície USB, varies opcions a nivell d'usuari (potenciòmetre i polsadors), molt útils a l'hora de realitzar proves, i suport Mikrobus™ per connectar plaques complementàries, estalviant espai de la PCB.

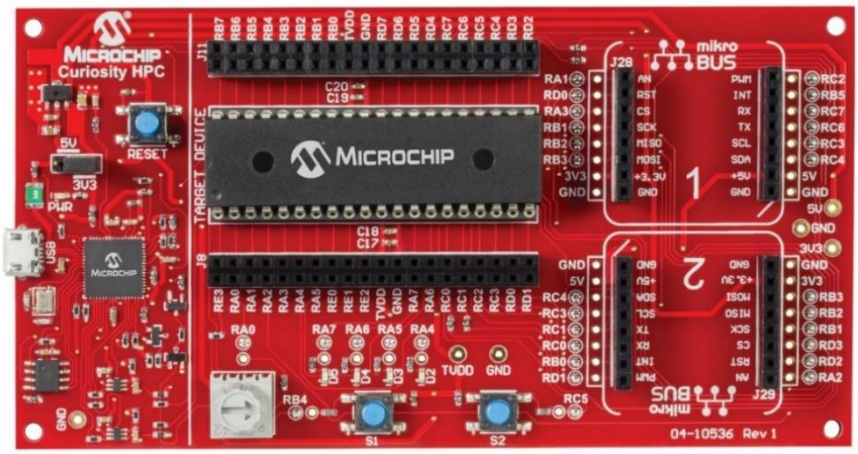

Figura 4: Placa Curiosity HPC

Aquestes plaques són els nuclis de les tres tasques que es volen portar a terme en el sistema de l'aerogenerador explicades en l'apartat anterior. Aquestes plaques es

mencionaran a partir d'ara com a Esclau 1 (control de pitch), Esclau 2 (tractament de dades) i Esclau 3 (gestió de càrrega).

2.2.1. Característiques i mòduls principals

Els PIC18F47Q10 són microcontroladors configurables amb perifèrics analògics i de comunicació independents, per a una àmplia gamma de propòsits generals i aplicacions de baixa potència.

Consten d'una arquitectura RISC optimitzada per compilació en llenguatge C, amb una velocitat de cicle d'instrucció de 62,5 ns i prioritats d'interrupcions programables.

Tenen 3 temporitzadors de 8 bits i 4 temporitzadors de 16 bits amb freqüència configurable, a més d'un temporitzador extra que realitza la funció de "watchdog".

Els microcontroladors disposen de mòduls/perifèrics específics per realitzar funcions concretes, aquests poden ser activats o desactivats des de l'entorn de treball des d'on es configuren i s'associen a pins.

Cada microcontrolador disposa de vuit mòduls CLC per tal de realitzar lògica seqüencial i combinacional integrada, un mòdul CWG per generar formes d'ona associades a ponts de MOSFETs i de senyal PWM, dos mòduls CCP que permeten activar accions durant un cert temps concret o passat un cert temps concret, així com activar senyals PWM de freqüència i "duty cycle" variable, dos mòduls PWM addicionals, permetent crear 4 senyals PWM en total, un mòdul de CRC programable per tal de verificar la integritat dels missatges dels busos de comunicacions, un mòdul ADC de 10 bits per convertir senyals analògiques a digitals i un DAC per convertir-ne de digitals a analògiques, un mòdul ZCD per controlar el pas per zero de senyals alternes, dos mòduls CMP que permeten comparar senyals, un mòdul HLT que monitoritza el hardware per tal de detectar fallades, un DSM que habilita la modulació de senyals internes o externes, un mòdul CVD que permet adquirir senyals utilitzant una matriu de condensadors "Sample-and-Hold" ajustable i un mòdul FVR per aplicar nivells de voltatge precisos als pins.

També inclou quatre mòduls de comunicacions, dos MSSP, per comunicació SPI i I2C, i dos EUSART amb suport per LIN.

A part dels perifèrics mencionats, els PIC18 disposen de 35 pins d'entrades i sortides separats en 5 ports. Cada pin disposa de resistències de "pull-up" individuals i programables. Cada mòdul/perifèric està associat a un o varis ports de pins específics.

Per últim es disposa de un ICSP per tal de programar el microcontrolador i un ICD amb tres "breakpoints" per poder fer "debug".

2.2.2. Entorn de treball

Per tal de programar i configurar el PIC i les seves connexions s'utilitza l'IDE de Microchip, MPLAB X. El programa permet crear un projecte amb el microcontrolador a utilitzar per començar a programar directament. Tot i així per tal de simplificar la programació s'utilitzarà el "plugin" MCC.

L'MCC és un entorn de programació gràfic que genera codi en llenguatge C de fàcil comprensió que pot ser introduït en un projecte. Utilitza una interfície intuïtiva que permet activar i configurar els perifèrics i pins necessaris per l'aplicació desitjada. El "plugin" MCC permet crear la base sobre la qual construir el nostre programa. A partir de les opcions escollides en l'entorn de treball s'encarrega d'escriure els registres necessaris en el codi C, d'aquesta manera s'eviten els coneixements de programació a baix nivell requerits per configurar el microcontrolador des de zero. Tota la configuració de connexions feta en l'entorn de treball ha de coincidir amb les connexions reals físiques de la PCB.

2.2.3. Programació

A partir del codi generat per l'MCC s'escriuen les funcions i accions necessàries en l'arxiu "main.c". Es poden incloure també llibreries o altres arxius amb funcions complementaries en el directori principal del projecte.

A l'hora d'escriure el codi, cal tenir en compte que per tal d'estalviar recursos i portar un seguiment molt més precís de les accions que es realitzen en el microcontrolador, el flux del programa es basarà en interrupcions, eliminant en la mesura que sigui possible tots els processos que es realitzin de forma continuada sense un període definit. D'aquesta manera, mitjançant un vector d'interrupcions, es poden ordenar les accions del programa. Totes les interrupcions han de ser de curta durada per evitar que el programa quedi saturat realitzant una tasca concreta. Seguint aquest procediment, en les interrupcions s'habiliten "flags" que activen les accions corresponents lligades a la interrupció, en el bucle principal del programa. Aquestes interrupcions poden generar-se des dels temporitzadors, els mòduls de comunicacions o pins externs.

Aquesta estructura de programa acaba resultant amb un sistema multitasca que pot realitzar varies accions alhora sempre seguint l'ordre marcat de prioritat. Si salta una interrupció durant l'execució d'una acció, primer es gestiona ràpidament la interrupció i posteriorment es retorna al mateix punt on s'havia deixat l'acció pertinent.

Totes les interrupcions presenten el mateix ordre de prioritat, això implica que es gestionen una per una i no es generen casos d'interrupcions dins d'interrupcions. El rellotge del microcontrolador es configura a 8MHz que correspon a un temps per instrucció de 125ns, per tant totes les interrupcions es gestionen en menys de 1µs.

També s'habilitarà el "Watchdog" en tots els microcontroladors per tal d'evitar que el programa es quedi penjat degut a algun error de hardware i s'utilitzaran les funcions de "sleep" per estalviar energia quan no hi hagi vent o quan el molí es trobi a nivells crítics de bateria.

Finalment els arxius es compilaran i es bolcaran en el microcontrolador, a partir d'aquí el programa es pot executar amb normalitat.

3. CONTROL DEL PITCH

El moviment del pitch de l'aerogenerador es portarà a terme amb un actuador lineal alimentat a 24V, mitjançant les dues bateries de 12V del molí. Aquest motor canviarà el sentit de gir mitjançant un mòdul de relé i també disposarà de dos fi de cursa per no sobrepassar els límits de moviment marcats. La velocitat de rotació la marca un MOSFET controlat per PWM. Cambiant el "duty cycle" de la senyal que arriba al MOSFET es retalla l'alimentació del motor modificant la seva velocitat. En la figura 5 es pot veure una representació del sistema electrònic de control de pitch.

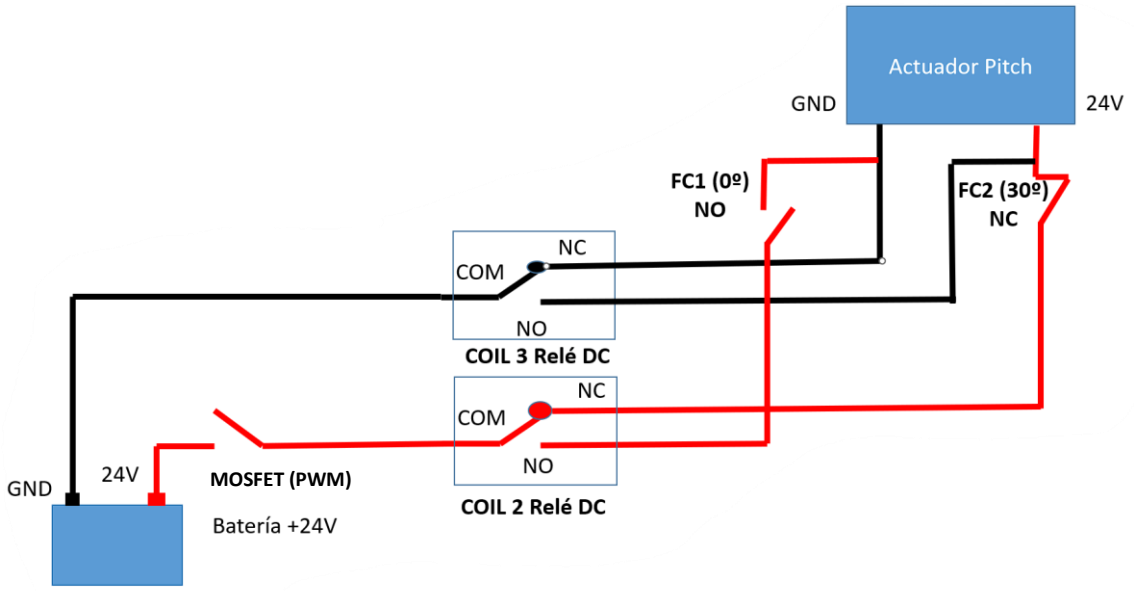

Figura 5: Esquema connexió actuador pitch

El relé de canvi de sentit es troba alimentat per un altre MOSFET, quan aquest rep un senyal des del microcontrolador actua sobre el disparador del mòdul de relé obrint els dos canals i invertint la polaritat de l'actuador, tal i com es pot veure en la figura 5.

El control d'aquest motor es realitzarà mitjançant l'esclau 1, PIC18F47Q10. Aquest recollirà dades d'un sensor lineal i amb elles calcularà la posició del pitch. Internament aquest sensor actua com un potenciòmetre de 10 kΩ, alimentat a 5V proporciona una resposta lineal que és llegida per l'ADC del microcontrolador.

Es disposa d'un sensor de temperatura per tal de negligir la possible desviació per temperatura del sensor lineal i obtenir una lectura més precisa. S'utilitza un LM35 alimentat també a 5V i connectat a una altra entrada ADC del PIC18. La compensació de temperatura es durà a terme durant l'etapa de posada en marxa de l'aerogenerador i podrà ser modificada posteriorment.

Per tal de determinar la consigna de posició es mesurarà la velocitat de rotació del rotor mitjançant un encoder incremental. Aquest compta amb tres senyals de sortida, A, B i Z. Els senyals A i B generen una ona quadrada cada 360º elèctrics de rotació i es troben desfasades 90º entre elles. El senyal Z en canvi genera un pols cada 360º mecànics de rotació i per tant permet determinar el nombre de voltes que ha donat l'aerogenerador. L'encoder utilitzat té una resolució de 600 impulsos/rotació, que correspon al nombre de flancs de pujada que s'observaran tant en el senyal A com en el B per cada gir complet de les pales. En la figura 6 es pot observar millor l'activació i desactivació dels polsos en funció dels graus elèctrics.

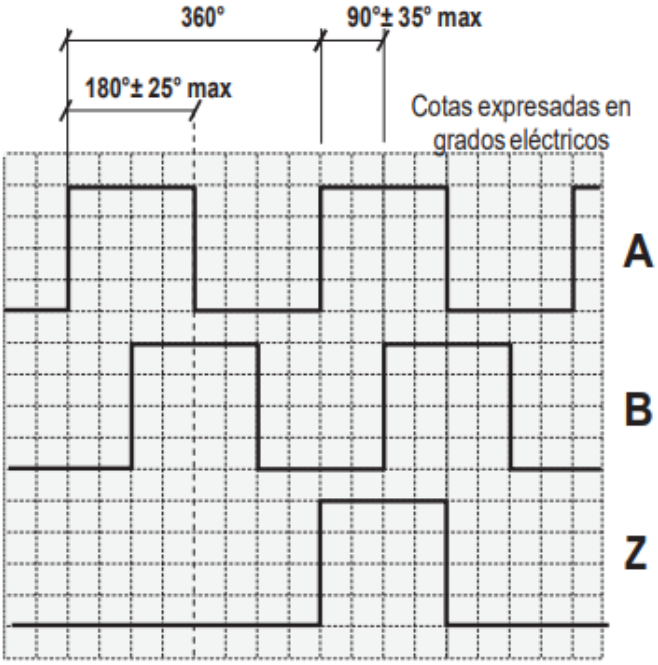

Figura 6: Representació gràfica de les senyals incrementals A, B i Z

El senyal de velocitat de rotació A es fa arribar des de la PCB situada a la punta de l'aerogenerador fins la base del molí on hi ha el microcontrolador utilitzant un mòdul de comunicacions UART/RS-485. Finalment es connecta el senyal a través d'una entrada de temporitzador. Aquest senyal és vist pel PIC com una font de rellotge i el propi hardware porta a terme un comptatge dels polsos. Finalment s'adquireix el valor a intervals constants per determinar la velocitat.

En la figura 7 es pot veure un esquema del diagrama de planta del sistema amb totes les connexions entre els dispositius. La línia discontinua "PC (Software)" engloba tot el que fa referència al programari del microcontrolador.

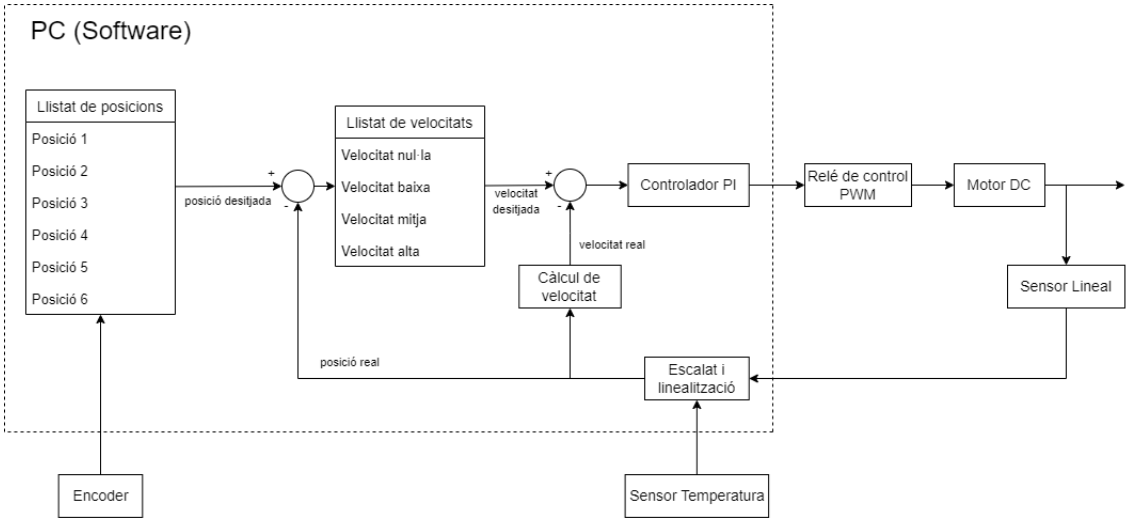

Figura 7: Diagrama de planta del control de pitch

Segons la velocitat obtinguda per l'encoder s'aplica un valor de posició del pitch des d'un llistat de posicions predeterminades. Aquest llistat funciona com una màquina d'estats, on es busca que la velocitat de rotació de l'aerogenerador es trobi dins d'un rang concret. Per exemple, quan la velocitat sigui massa elevada, la consigna de posició passa a la següent posició de la llista, inclinant les pales paral·lelament al vent i conseqüentment frenant l'aerogenerador i quan la velocitat sigui massa baixa es realitza l'acció contraria, la consigna passa a la posició anterior de la llista, inclinant les pales perpendicularment al vent i accelerant la seva rotació. Els valors de referència del rang de velocitats així com els del llistat de posicions s'ajustaran a les necessitats de l'aerogenerador un cop estigui muntat, durant l'etapa de posada en funcionament.

A partir del valor seleccionat de la llista i el valor de posició real, obtingut pel sensor lineal, es calcula l'error de posició, en funció d'aquest es seleccionen els paràmetres de control del PI discret i la velocitat de moviment del pitch. Es disposa de 3 consignes de velocitats de moviment, cadascuna ajustada per un controlador específicament sintonitzat per aquella velocitat concreta. Es contempla també un petit marge de zona morta al voltant de la consigna per tal de que el motor pugui aturar-se completament.

A partir dels valors de posició obtinguts a un període de temps específic es calcula la velocitat real de moviment del pitch i juntament amb la velocitat de consigna es calcula l'error de velocitat. Finalment aquest error s'envia a l'entrada del controlador.

La variable de sortida del controlador correspon al valor de "duty cycle" del senyal PWM que activarà i desactivarà el MOSFET. Aquest es troba alimentat a 24V i la seva sortida connecta amb el motor.

3.1. Control PI

En el sistema de l'aerogenerador s'opta per un controlador PI. La resposta d'un motor de corrent continu sol ser de primer o segon ordre, per tant no es requereix d'un sistema de control complex, tampoc es necessita un temps de resposta ràpid ja que el motor no treballa a velocitats tant altes. Per últim un sistema de control més simple provoca menys càrrega de dades i processament al microcontrolador, ja que aquest les requereix per altres serveis, com podrien ser les comunicacions.

El controlador PI (Proporcional i Integrador) és un controlador realimentat el propòsit del qual és fer que l'error en estat estacionari, entre el senyal de referència i el senyal de sortida de la planta, sigui el més proper a zero possible, la qual cosa s'aconsegueix mitjançant l'ús de l'acció integral.

Els controladors PI són suficients per a resoldre els problemes de control de moltes aplicacions en la indústria, particularment quan la dinàmica del procés ho permet (en general processos que poden ser descrits per dinàmiques de primer i segon ordre).

És un recurs molt utilitzat en la indústria, la gran majoria dels llaços de control que existeixen en les aplicacions industrials són del tipus PI, això demostra la preferència de l'usuari en l'ús de lleis de control més simples.

Per tal de portar a terme un control amb PI s'utilitza el principi de realimentació, aquest ha tingut èxit en els camps del control, comunicacions i instrumentació. El principi de realimentació es basa en incrementar la variable manipulada quan la variable del procés sigui més petita que la referència i disminuir-la quan aquesta sigui més gran.

Aquest tipus de realimentació es diu realimentació negativa, pel fet que la variable manipulada es mou en la direcció oposada a la variable del procés. El principi pot ser observat en el diagrama de planta vist a la figura 7.

3.1.1. Proporcional

El terme proporcional és el més simple dels tres i és el més comú en les tècniques de control amb sistemes de realimentació. El guany proporcional (Kp) és multiplicat per l'error. La correcció aplicada al sistema és directament proporcional a l'error, a mesura que el guany augmenta la correcció aplicada a la planta es torna més agressiva. Aquest tipus de control s'utilitza per disminuir l'error fins a un valor petit, però mai zero, deixant la resposta amb un error permanent.

Aquesta és la raó per la qual el control proporcional no és suficient en alguns sistemes, aquests requeriran termes integrals i/o derivatius per obtenir una bona resposta.

3.1.2. Integral

Contràriament al control proporcional que actua segons l'error actual, el control integral es fixa en els errors anteriors. L'error acumulat (suma dels errors anteriors) s'utilitza per calcular el terme integral a intervals fixes de temps. El període d'adquisició té un pes clau en la correcta estabilització del sistema, si el període és massa curt l'error acumulat creixeria massa ràpid com per deixar que el sistema respongui correctament, provocant una resposta inestable.

Un altre element del control integral que cal considerar és el "wind-up". Apareix quan la sortida de planta es troba saturada i l'error acumulat augmenta constantment. Per tal d'evitar aquesta situació cal establir límits a l'error acumulat o desactivar el terme integral quan la sortida es troba saturada.

Una altra característica és el guany excessiu, aquest pot comportar una sortida inestable, fent que el sistema oscil·li. El guany integral s'ha de provar per totes les possibles situacions del sistema per trobar la millor resposta global.

En resum, a mesura que l'error acumulat augmenta el terme integral té un efecte més significatiu en la resposta de la planta. En un sistema lent, el terme integral podria dominar el valor que s'envia a la planta, provocant inestabilitat.

3.2. Sintonització del controlador

Per tal de portar a terme un control precís, es realitza una monitorització de la resposta del sistema per tal d'identificar els seus paràmetres utilitzant programari especialitzat (MATLAB). Finalment, amb el sistema identificat es realitza una sintonització del controlador per tal d'obtenir els valors dels termes proporcional i integral.

Es realitzen també diferents simulacions en l'entorn Simulink per veure el comportament del conjunt motor-controlador.

3.2.1. Monitorització del sistema

Per tal de realitzar la sintonització dels controladors es porten a terme unes proves amb el motor a diferents consignes de velocitat.

Primer de tot es desactiva el sistema de control i s'apliquen valors de PWM directament a l'entrada del motor a llaç obert, en forma de graó unitari. Els valors introduïts corresponen a les tres velocitats (alta, mitja i baixa) de consigna del pitch,

aquestes consignes es seleccionen a partir de les respostes observades en aquesta prova.

Una interrupció de temporitzador de 100ms habilita l'adquisició de valors del sensor lineal, aquesta posició s'escala per tal d'obtenir la posició real en cm. Amb aquesta posició, la posició obtinguda en la interrupció anterior i coneixent el període de mostreig es calcula la velocitat real.

Aquest valor de velocitat s'envia a un període constant de 100ms a un mòdul de comunicacions UART/USB. Aquest mòdul es connecta directament sobre la placa Curiosity del microcontrolador mitjançant el suport Mikrobus i s'alimenta a través d'ell. En la figura 8 es mostra com és físicament el mòdul de comunicacions.

Per últim les dades es llegeixen a través de l'ordinador mitjançant el programari PuTTY.

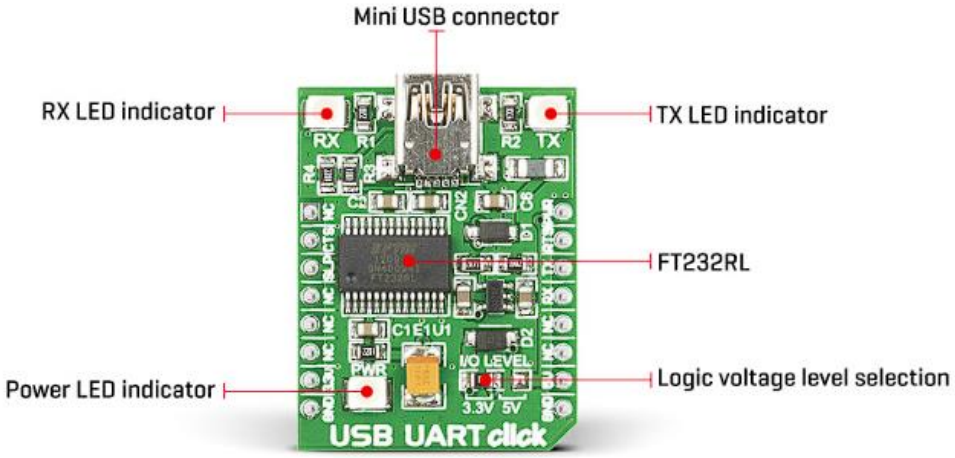

Figura 8: USB/UART Click board

Un cop s'han obtingut les dades de les tres velocitats es passen els arxius generats pel PuTTY a un Excel. Això permet ordenar les dades i fer un primer anàlisi abans d'introduir-les en un programari més avançat com podria ser el MATLAB. Si la resposta observada no és acceptable es repeteix l'adquisició de dades modificant el "duty cycle" de l'entrada del motor. Els paràmetres que determinen la validesa de l'assaig son la velocitat màxima assolida pel motor i la forma de la corba. Es busquen tres consignes que proporcionin velocitats clarament diferenciades i una acceleració ràpida i precisa.

Finalment quan les dades es donen per vàlides es processen a través de MATLAB, com es pot apreciar en la figura 9.

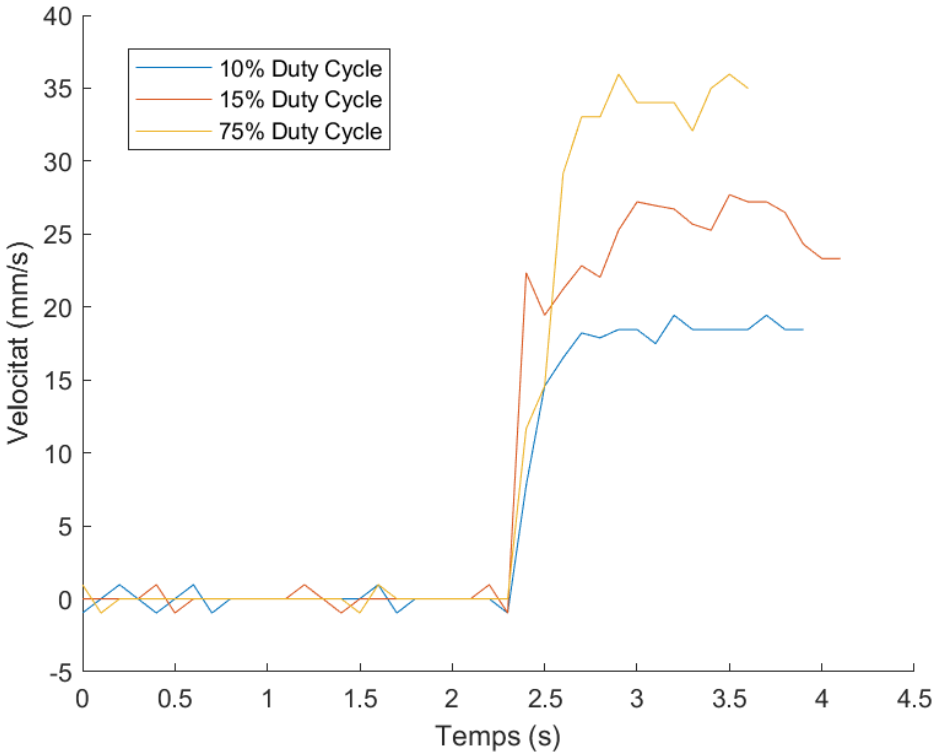

Figura 9: Resposta en graó unitari de les tres velocitats de consigna

En aquesta gràfica es poden observar les tres velocitats seleccionades i la resposta del motor aplicant un graó unitari per cadascuna. Les consignes son: 18,5mm/s (velocitat baixa), 25mm/s (velocitat mitja) i 35mm/s (velocitat alta).

Cal tenir en compte que la prova es realitza sense càrrega i per tant les velocitats poden estar associades a valors de PWM diferents en el model definitiu, però això no afectarà al moviment del motor.

L'objectiu de les tres velocitats és parametritzar un controlador específic per cada consigna.

3.2.2. Identificació del sistema

Partint de les dades obtingudes s'utilitza l'eina "System Identification Toolbox" per identificar el sistema i trobar una funció de transferència correlativa, com es pot veure en la figura 10. Cadascuna de les velocitat es tractarà com un sistema independent.

Aquesta extensió permet importar les dades i fer una estimació del model. S'han escollit funcions de transferència de primer ordre ja que son els sistemes que s'ajusten millor a les dades d'entrada.

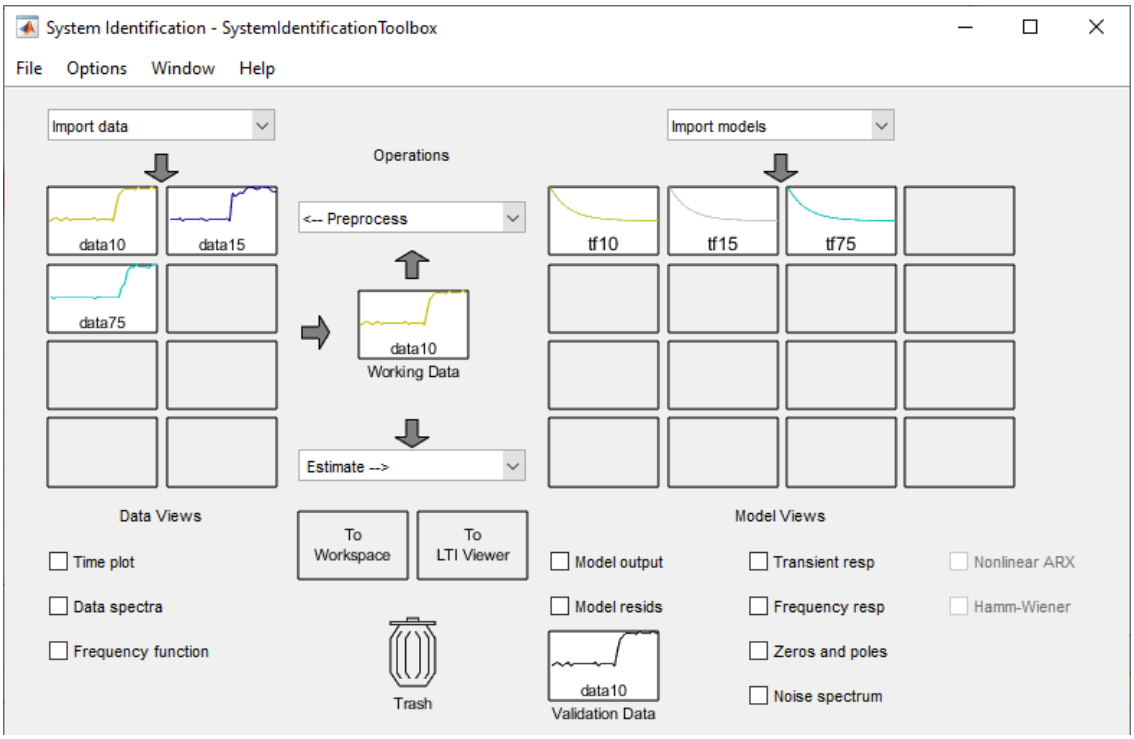

Figura 10: Eina System Identification Toolbox

En les figures 11, 12 i 13 es poden veure els models estimats resultants de cadascun dels 3 sistemes importats.

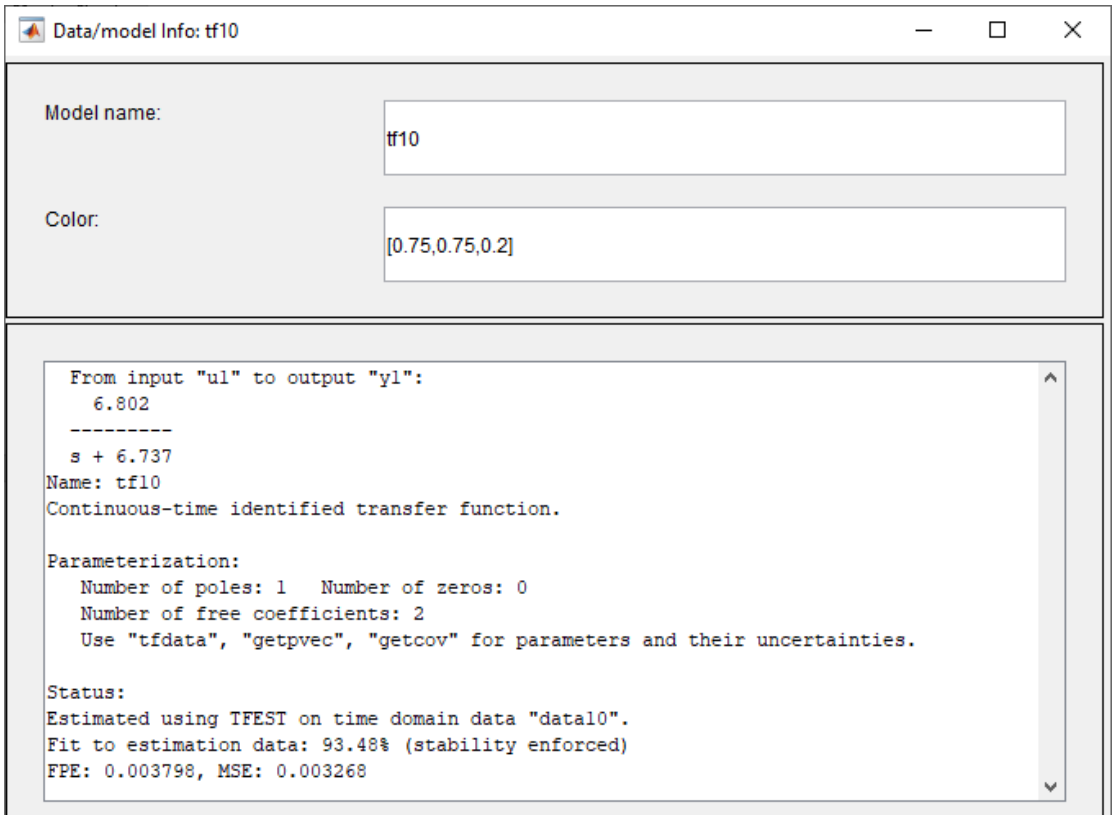

Figura 11: Funció de transferència del sistema a velocitat baixa

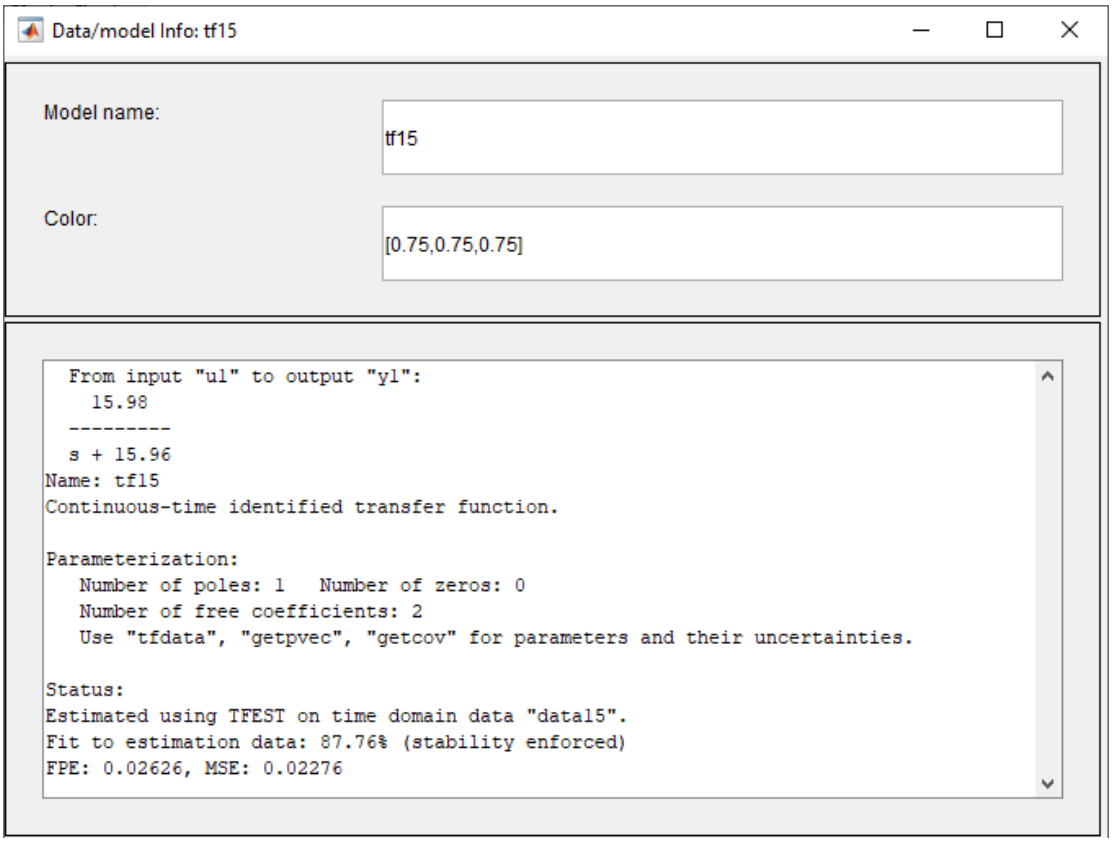

Figura 12: Funció de transferència del sistema a velocitat mitja

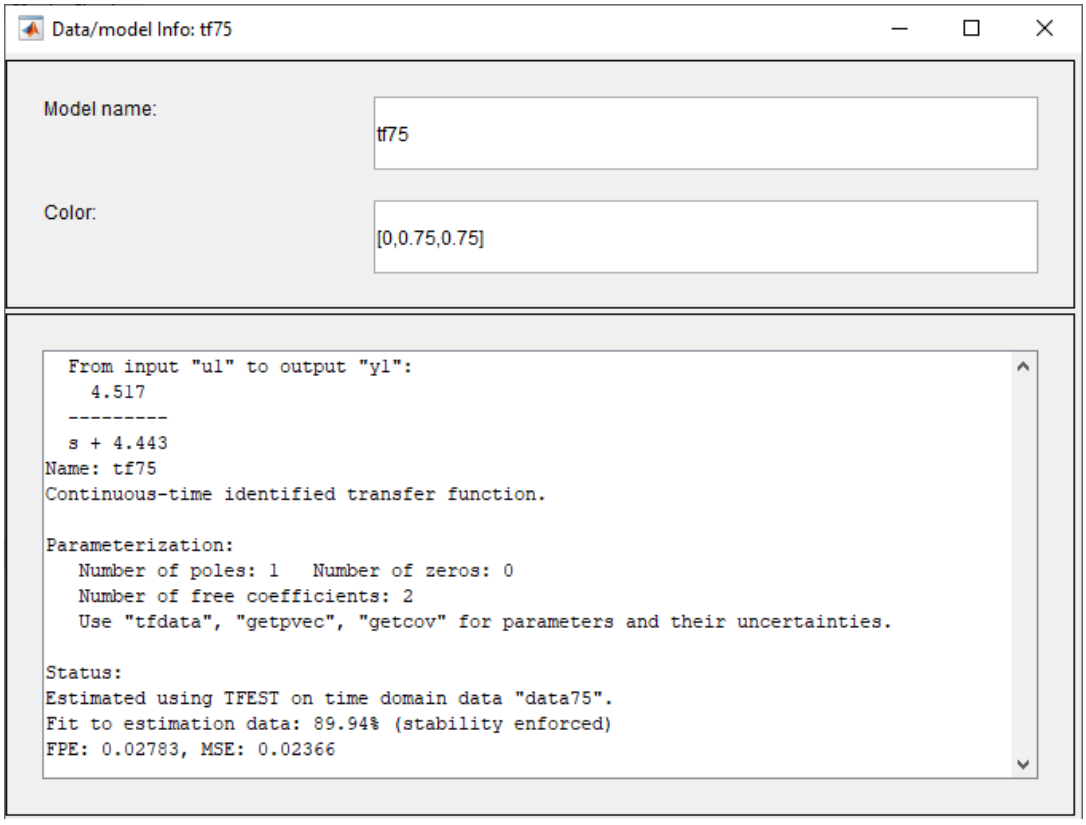

Figura 13: Funció de transferència del sistema a velocitat alta

Amb les funcions de transferència identificades es pot utilitzar l'eina "PID Tuner" per tal de trobar un controlador específic que s'ajusti a les nostres necessitats. La figura 14 mostra com s'importen els models i es selecciona un control per PI.

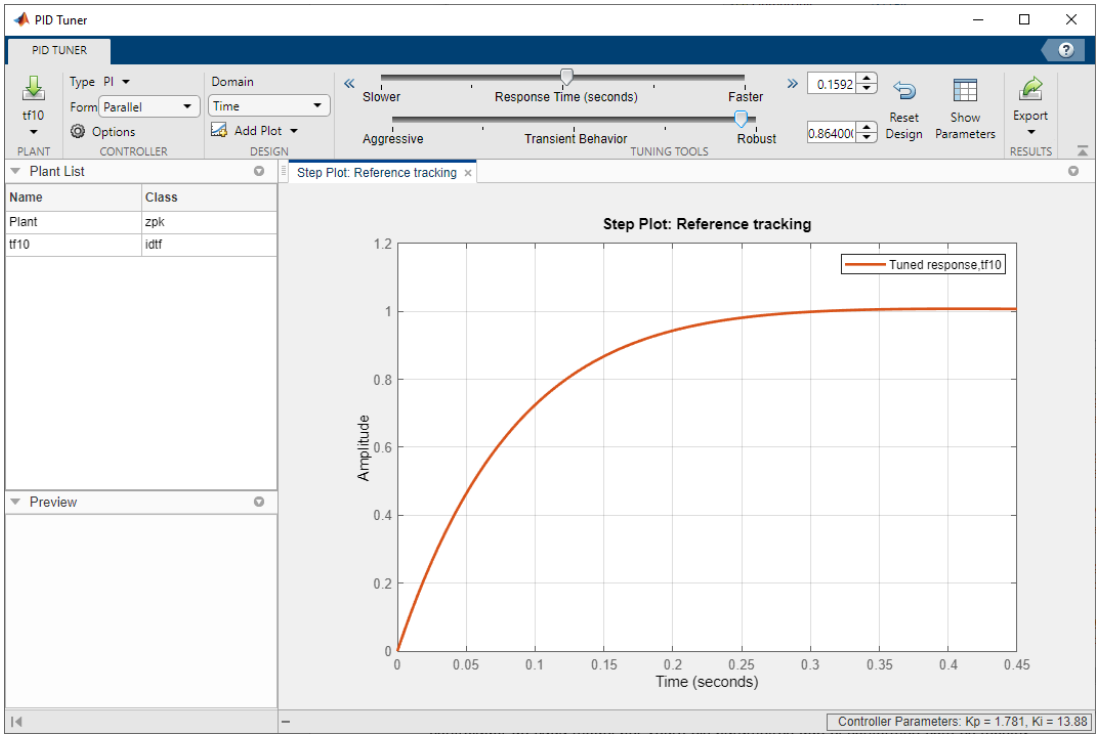

Figura 14: Eina PID Tuner

Es vol que la resposta sigui ràpida i robusta però no massa agressiva per tal de que es mantingui estable quan es discretitzin els controladors a l'hora d'introduir-los en el programa del PIC.

Un cop la resposta compleix amb les condicions especificades es pot exportar el controlador de cada model per veure els paràmetres que el conformen com es mostra en les figures 15, 16 i 17.

```
C10 =

             1
  Kp + Ki * ---
             s

  with Kp = 1.92, Ki = 11.6

Continuous-time PI controller in parallel form.
```

Figura 15: Paràmetres del controlador PI de velocitat baixa

```
C15 =

             1
  Kp + Ki * ---
             s

  with Kp = 0.999, Ki = 9.81

Continuous-time PI controller in parallel form.
```

Figura 16: Paràmetres del controlador PI de velocitat mitja

```
C75 =

             1
  Kp + Ki * ---
             s

  with Kp = 1.69, Ki = 11.5

Continuous-time PI controller in parallel form.
```

Figura 17: Paràmetres del controlador PI de velocitat alta

3.2.3. Simulació del sistema

Ara que disposem dels models de les tres velocitats del sistema i dels seus controladors podem crear una simulació utilitzant l'eina "Simulink".

El model de la figura 18 permet observar les velocitats de sortida del motor segons la velocitat de consigna aplicada, s'han tingut en compte les funcions de transferència de cadascuna de les respostes del motor i s'hi ha afegit el seu controlador corresponent.

Els controladors s'han discretitzat en introduir-los a la simulació per tal de que es corresponguin amb el model real. Per portar-ho a terme s'ha escollit un període de mostreig de 100ms, els termes proporcionals i integrals mantenen els valors de les figures 15, 16 i 17.

Les velocitats de consigna es troben representades en mm/s així com les velocitats de sortida de les funcions de transferència.

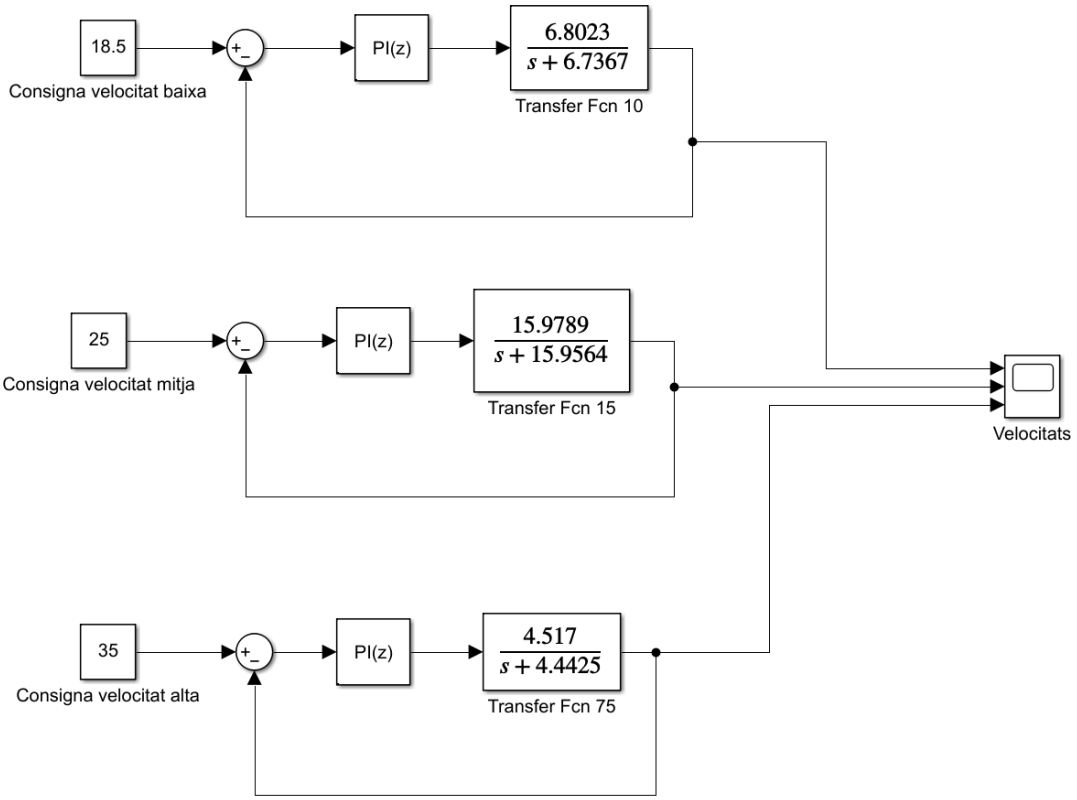

Figura 18: Model tres velocitats - simulink

Les gràfiques de velocitat obtingudes confirmen que els controladors calculats permeten assolir les velocitats desitjades tal i com es pot veure en la figura 19.

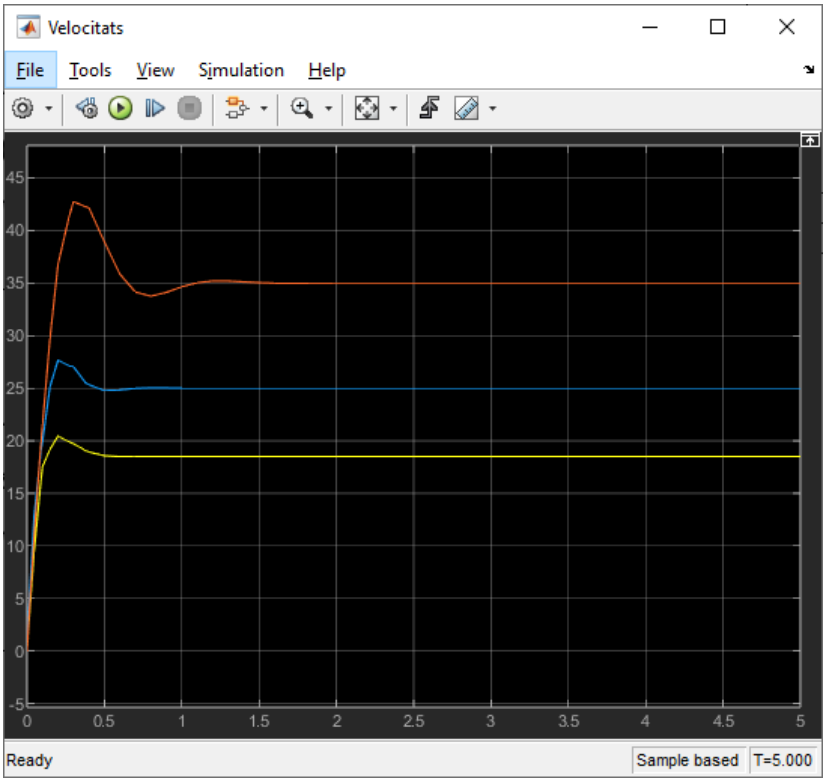

Figura 19: Gràfica tres velocitats - simulink

El color taronja correspon a la velocitat alta de 35mm/s, el color blau és la velocitat mitja de 25mm/s i el color groc és la velocitat baixa de 18,5mm/s.

A partir d'aquesta simulació es poden unir els tres models i afegir la consigna de posició per tal d'obtenir el sistema de moviment de pitch complet de la figura 20.

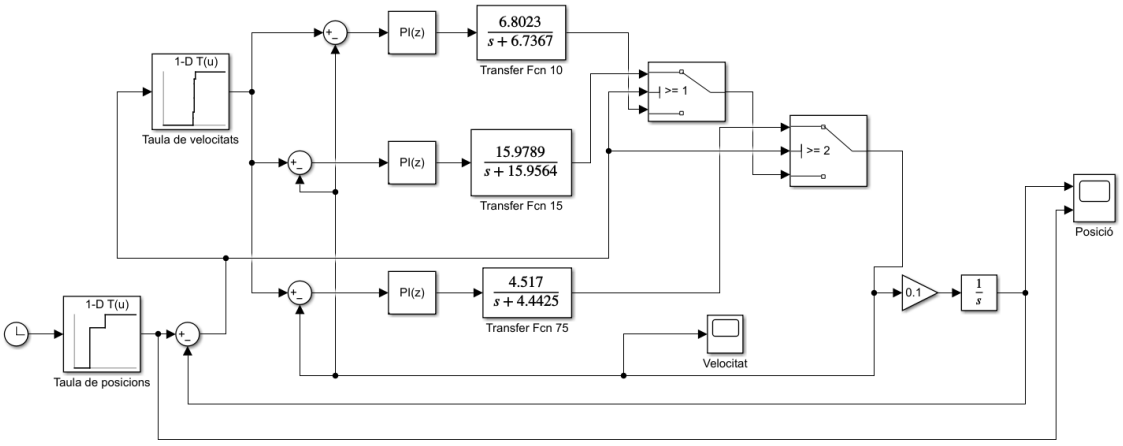

Figura 20: Model complet - simulink

En aquest model la velocitat a aplicar es determina a partir de l'error de posició representat en centímetres. La posició real del sistema s'obté a base d'integrar la resposta de velocitat del motor i la posició de consigna és introduïda per l'usuari a partir d'una taula de posicions que recorrerà el motor segons el temps de simulació.

L'error de posició determina la velocitat de consigna i també habilita la funció de transferència corresponent. Quan l'error de posició sigui més gran de 2cm s'escull la velocitat elevada de 35mm/s, quan l'error es troba entre 1cm i 2cm s'escull la velocitat mitja de 25mm/s i finalment quan l'error és menor de 1cm es selecciona la velocitat baixa de 18,5mm/s.

De la mateixa manera que el sistema real, en el model de simulink s'inclou una zona morta de 0,5cm. Aquesta zona morta s'implementarà únicament per evitar que els relés s'activin i desactivin constantment degut a les oscil·lacions de la resposta i la inèrcia del motor.

Per aquesta primera simulació, a la taula de posicions inicial s'introdueix que el motor vagi a la posició 6cm al cap de 5 segons i a la posició 8cm als 10 segons.

En la gràfica de la figura 21 es pot veure la posició del motor a cada instant de temps, es pot apreciar el canvi de velocitat quan s'assoleixen els 4cm i els 5cm ja que es modifica el pendent de la recta. En blau es mostra la consigna de posició i en groc la

posició simulada del motor, es recorda que no s'arriba exactament a la consigna ja que el sistema de control presenta una zona morta de 0,5cm.

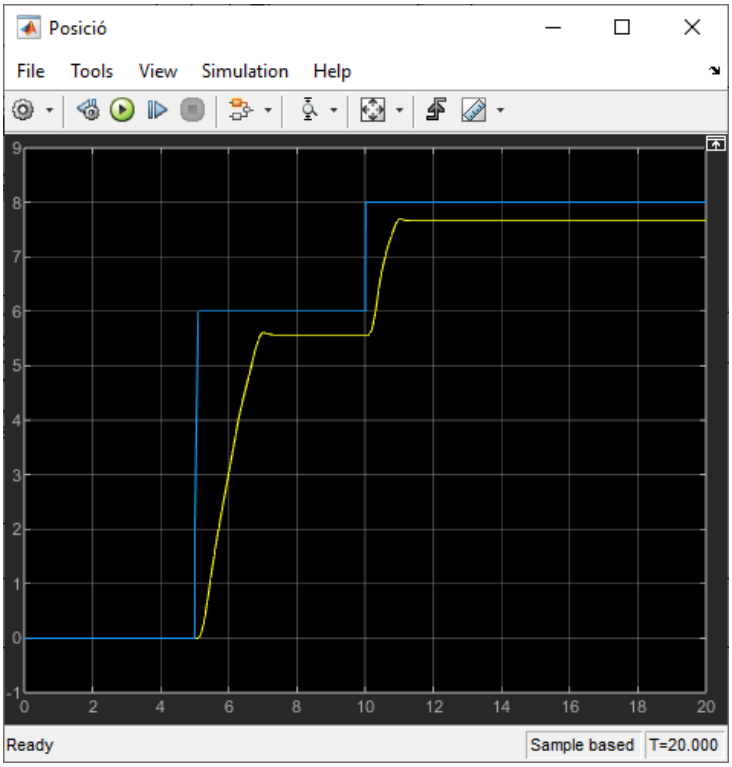

Figura 21: Gràfica posició – simulink

En la gràfica de velocitat de la figura 22 es pot observar com el motor assoleix les velocitats marcades sense cap problema.

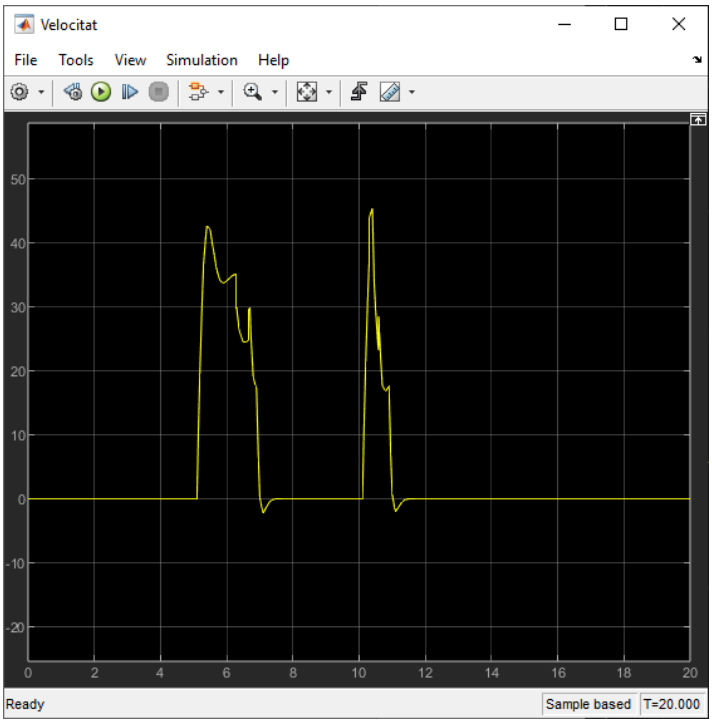

Figura 22: Gràfica velocitat - simulink

Per tal de comprovar la robustesa s'introdueixen pertorbacions externes en el sistema i s'observa la resposta. Es modifica el model per incloure una pertorbació negativa de 80mm/s en l'instant 6s entre el controlador i les funcions de transferència i una pertorbació positiva de 150mm/s en l'instant 15s a la sortida del sistema. En la figura 23 es poden apreciar els canvis realitzats en el model.

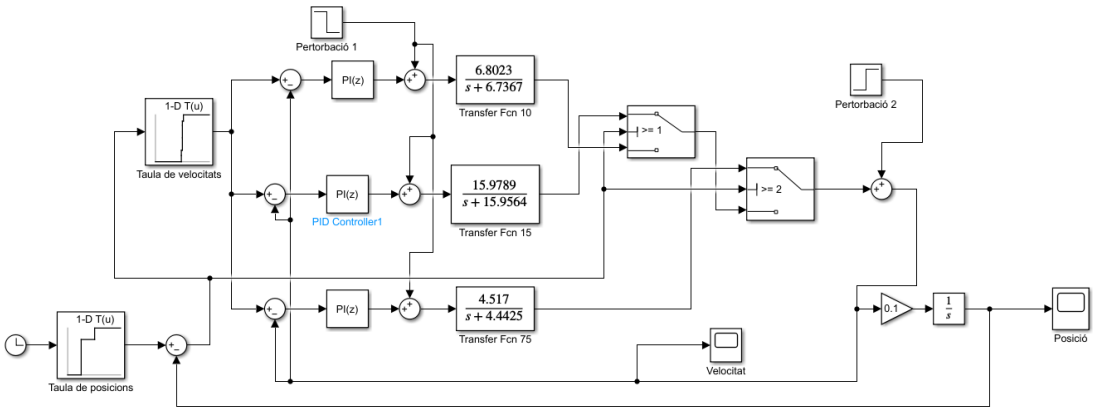

Figura 23: Model complet amb pertorbacions – simulink

La gràfica de la figura 24 mostra la posició del motor durant tota la seqüència en color groc i la consigna d'entrada en color blau.

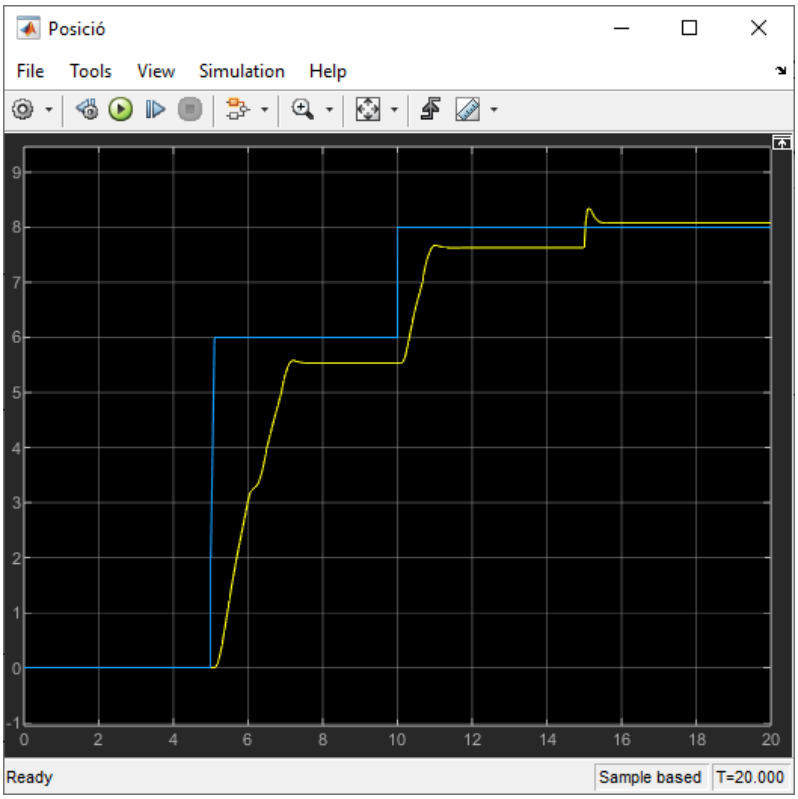

Figura 24: Gràfica posició amb pertorbacions 1 - simulink

Es pot apreciar com els controladors mantenen el sistema estable en tot moment i dins dels marges especificats (0,5cm de zona morta).

Ara s'inverteixen els signes de les pertorbacions, és a dir s'aplicarà una pertorbació positiva de 80mm/s en l'instant 6s i una pertorbació negativa de 150mm/s en l'instant 15s. La figura 25 mostra els resultats obtinguts, en blau es mostra la consigna i en groc la posició del motor.

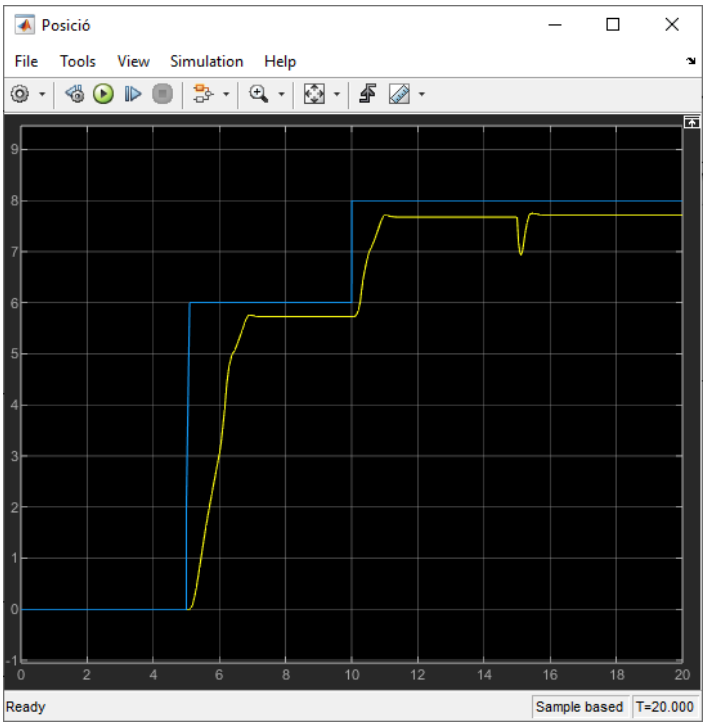

Figura 25: Gràfica posició amb pertorbacions 2 – simulink

Tot i que les pertorbacions, al ser de magnitud elevada, afecten al sistema, aquest aconsegueix recuperar-se ràpidament i retorna la posició dins dels marges especificats.

Es realitza una altra simulació amb el model sense pertorbacions, vist a la figura 20, desplaçant el motor fins a la posició 5cm en l'instant 5s i passats 5 segons més retornant fins la posició 4cm. D'aquesta manera es verifica que el canvi de sentit no afecta a la resposta del motor i que funciona tant a velocitats altres com velocitats baixes, els resultats es poden veure en la figura 26.

El motor reacciona com s'espera utilitzant la velocitat alta quan la consigna de posició es troba lluny i la velocitat baixa quan es troba a prop.

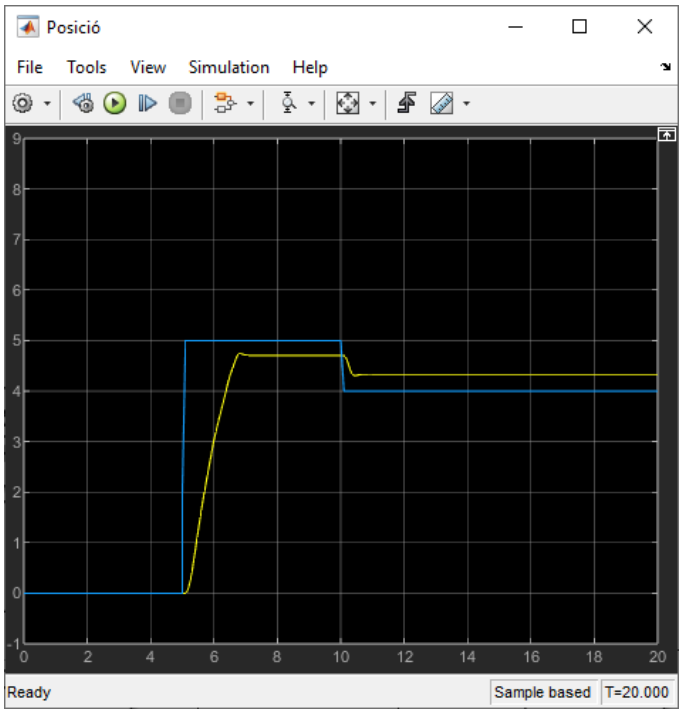

Figura 26: Gràfica posició amb retorn – simulink

Per últim es realitza una última simulació per comprovar el recorregut total del motor i simular el temps màxim de frenada en cas d'emergència. Per aquesta simulació el motor començarà a l'estat inicial de 0cm i al cap de 5 segons es desplaçarà fins els 9cm, que correspon al rang complet de moviment.

La gràfica de posició resultant d'aquesta simulació es pot veure en la figura 27.

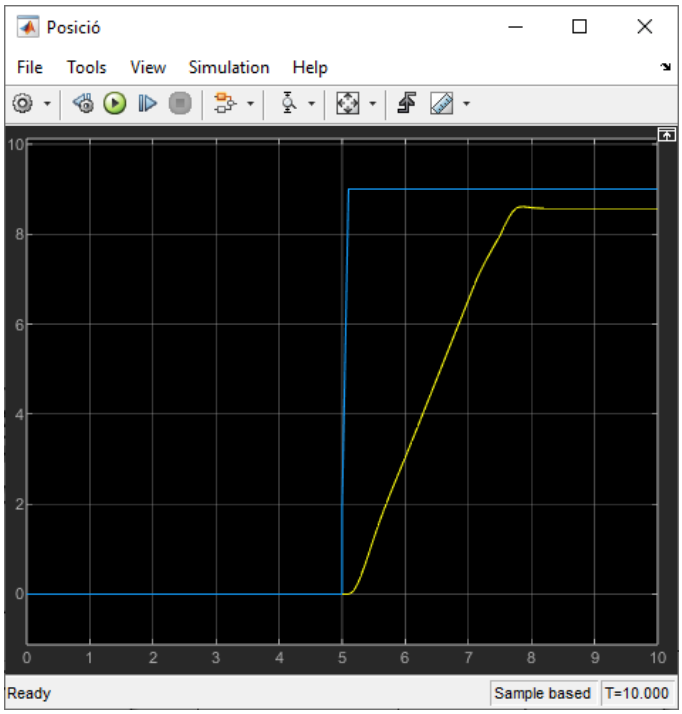

Figura 27: Gràfica posició rang complet – simulink

Amb aquesta simulació es pot determinar que el motor tardaria aproximadament 3 segons en realitzar tot el moviment complet per frenar les pales mitjançant el pitch. La posició 0cm representaria les pales en orientades perpendicularment al vent i la posició 9cm correspondria a les pales paral·leles al vent. Per tant en cas d'emergència es tardaria 3 segons en aplicar el fre màxim a les pales.

Finalment, amb els paràmetres trobats i comprovats en la simulació es poden integrar els tres controladors dins el programa del microcontrolador per cadascuna de les velocitats.

3.3. Implementació del controlador

El programa de l'esclau 1 treballa sobre dues rutines d'interrupció generades per temporitzador.

La primera rutina d'interrupció és d'1 segon i està generada per el temporitzador TMR0, aquesta llegeix la velocitat de rotació de les pales. La senyal de l'encoder està connectada a l'entrada del TMR1 de manera que el propi hardware del temporitzador realitza un comptatge dels polsos. A partir del nombre de polsos i del període d'adquisició d'un segon es determina la velocitat de gir. Amb aquesta velocitat s'activa la funció que canvia la consigna de posició de pitch, tal com s'ha vist en el diagrama de la planta de la figura 7. Es parteix d'un llistat de posicions, quan es detecta una velocitat de rotació massa elevada es passa a la següent posició del llistat i quan es detecta una velocitat massa reduïda es passa a la posició anterior. Si s'assoleixen els extrems de la taula s'activen les alertes corresponents, les accions que porten a terme s'expliquen més endavant, en l'apartat 5.1.

La segona rutina és de 100 mil·lisegons i està lligada al TMR3, és l'encarregada d'activar la funció d'adquisició de dades. En ella es llegeixen les dades del sensor lineal i del sensor de temperatura. Un cop processades les dades es calcula l'error de posició i la velocitat actual de moviment del pitch, finalment s'habilita una "flag" que conseqüentment inicia les accions del PI. Aquest temps de 100ms correspon al període de mostreig del controlador.

En les figures 28 i 29 es mostren els diagrames de flux de les dues interrupcions, el codi en C implementat es pot veure en l'Annex A.

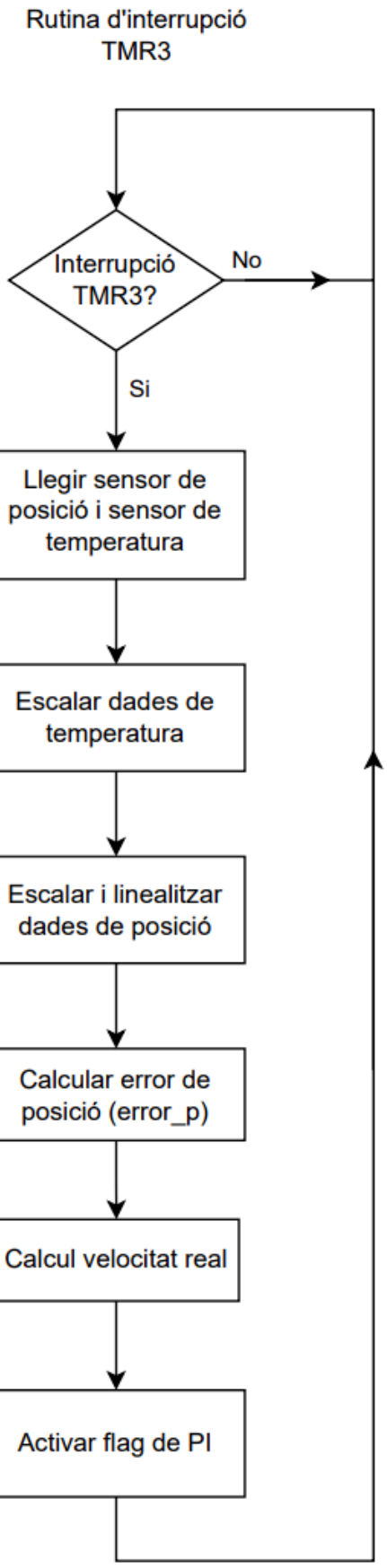

Figura 28: Diagrama de flux rutina d'interrupció TMR3 del control de pitch

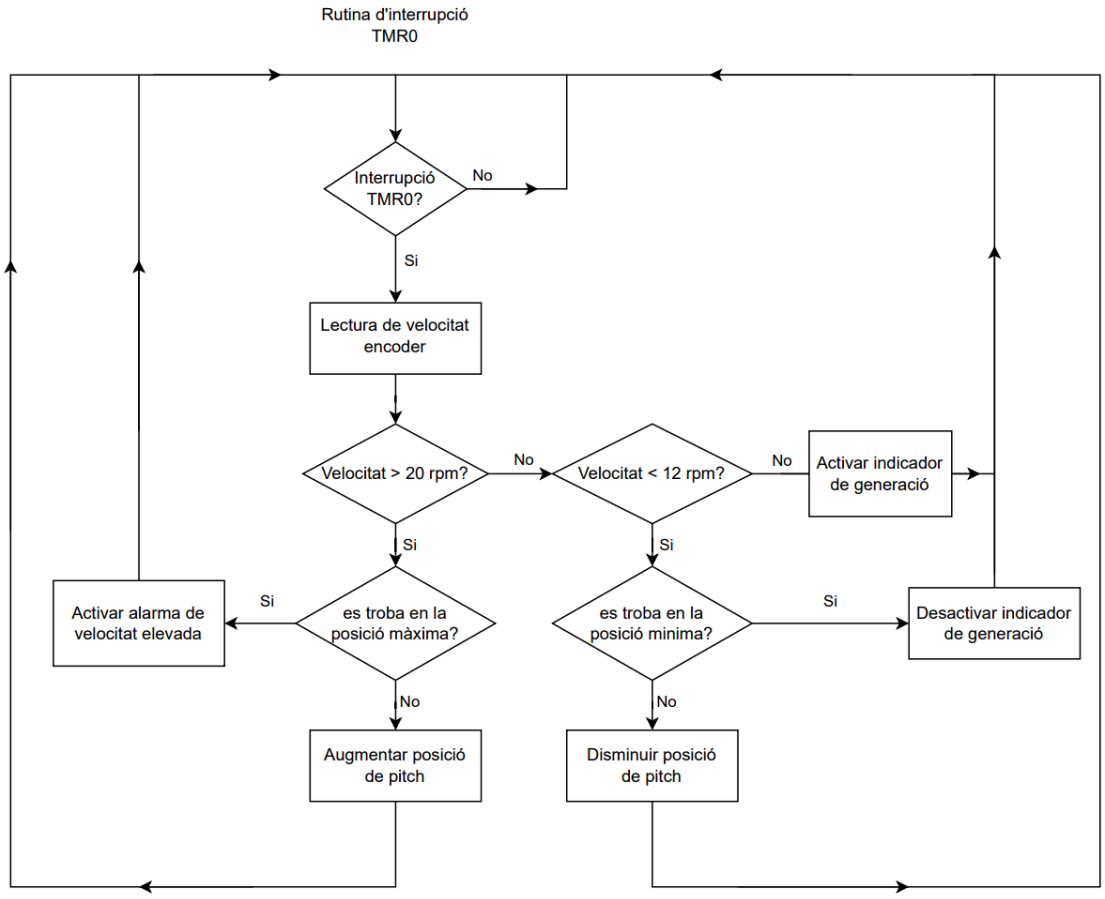

Figura 29: Diagrama de flux rutina d'interrupció TMR0 del control de pitch

La llista de posicions de pitch es mostra en la taula 1.

Posició	**Valor (cm)**
1	0
2	1,8
3	3,6
4	5,4
5	7,2
6	9

Taula 1: Llistat de consignes de posicions del control de pitch

El rang de moviment del motor i del sensor lineal és de 0cm a 9cm. S'ha determinat que amb 6 posicions de pitch ja es disposa de suficient control sobre la velocitat de rotació de les pales. Partint dels dos extrems s'han distanciat la resta de posicions equitativament.

Com es pot veure en la figura 29 el sistema anirà saltant entre les posicions segons la velocitat detectada per l'encoder. Quan es detecti una velocitat de rotació inferior a 12rpm es passarà a la posició inferior i quan es detecti una velocitat superior a 20rpm es passarà a la posició superior. Aquestes velocitats son provisionals i son suficients tant com per produir electricitat, com per no posar en perill l'aerogenerador. Tot i així, podran ser ajustades durant la posada en marxa de l'aerogenerador.

3.3.1. Algoritmes de control

Quan s'activa la rutina de PI s'inicialitzen les variables, un cop activada la "flag" i a partir de l'error de posició trobat anteriorment es determina la velocitat de consigna del pitch (alta, mitja, baixa o nul·la) així com els paràmetres del PI específics de la velocitat escollida com es pot veure en el codi següent.

```
if (error_p < 0) error_p = -error_p;
    if (error_p <= zona_morta){
        vel_consigna = 0;
        kp = 0;
        ti = 1000000;
        Cn_1 = 0;
    }else if (error_p > zona_morta && error_p <= limit_baix){
        vel_consigna = vel_baixa;
        kp = kp_vel_baixa;
        ti = ti_vel_baixa;
    } else if (error_p > limit_baix && error_p <= limit_alt){
        vel_consigna = vel_mitja;
        kp = kp_vel_mitja;
        ti = ti_vel_mitja;
    }else{
        vel_consigna = vel_alta;
        kp = kp_vel_alta;
        ti = ti_vel_alta;
    }
}
```

Amb la consigna i el valor actual de velocitat es calcula l'error i s'introdueix dins de l'equació del PI discret.

```
error_v = vel_consigna - vel_real;
q0=kp*(1+T/(2*ti));
q1=-kp*(1-T/(2*ti));
Cn = Cn_1 + q0*error_v + q1*error_v1; //llei del controlador PI discret
```

```
if(Cn >= 100) Cn = 100;  // Limitador de duty cycle per evitar warnings

if(Cn <= 5)   Cn = 0;                    // Zona morta de duty cycle

error_v1 = error_v;
Cn_1 = Cn;
valor_PWM = Cn;
```

La variable T correspon al període del controlador (100ms), error_v1 correspon a l'error de velocitat anterior i Cn_1 correspon a l'anterior resultat del PI. La llei de control s'obté de la discretització de l'equació continua Eq.1.

$$u(t) = k_c \left[e(t) + \frac{1}{\tau_i} \int_0^t e(t)dt \right] \quad \text{(Eq. 1)}$$

On k_c correspon al guany proporcional, T_i és el paràmetre integral, e(t) equival a l'error de la variable d'entrada i u(t) és la sortida del controlador.

Utilitzant el mètode de la integració trapezoidal i discretitzant l'equació s'acaba obtenint l'equació discretitzada Eq.2 amb els paràmetres de les equacions 3 i 4. Aquestes equacions son les que s'integren en el programa tal i com s'ha vist en el fragment de codi anterior.

$$u(k) = u(k-1) + q_0 e(k) + q_1 e(k-1) \quad \text{(Eq. 2)}$$

On:

$$q_0 = k_c \left[1 + \frac{T_s}{2\tau_i} \right] \quad \text{(Eq. 3)}$$

$$q_1 = -k_c \left[1 - \frac{T_s}{2\tau_i} \right] \quad \text{(Eq. 4)}$$

En les equacions anteriors, T_s correspon al període de mostreig (en segons), e(k) correspon a l'error actual, e(k-1) correspon a l'error anterior, u(k) és la sortida del controlador i u(k-1) és la sortida del controlador anterior. En el codi e(k) correspon a la variable error_v, e(k-1) seria error_v1, u(k) seria Cn, u(k-1) correspondria a Cn_1, T_s es representa com a T i k_c apareix com a kp.

Els paràmetres k_c i T_i de les equacions 3 i 4 mantenen el seu valor respecte l'equació 1, per això, com s'ha vist en l'apartat 3.2., un cop identificat el sistema i sintonitzat el controlador, aquest pot ser utilitzat en qualsevol rutina de control tant continua com discreta. Sense oblidar que amb períodes de mostreig elevats es requereix que el controlador presenti una resposta més suau, ja que una resposta agressiva podria desestabilitzar el sistema.

Per últim s'estableix el "duty cycle" del PWM segons el valor de velocitat obtingut i es limita dins del seu rang (0% a 100%) enviant el valor al MOSFET i aplicant el sentit de moviment al relé de dos canals.

Si el motor es troba en la seva posició màxima o mínima els fi de cursa s'activaran i s'anul·larà el moviment per hardware. Tanmateix cada vegada que es realitzi un canvi de sentit també s'anul·larà el moviment durant un sol cicle d'acció del PI (100ms) per evitar que el relé curtcircuiti la font d'alimentació.

```
if(valor_PWM == 0) PWM3_LoadDutyValue(valor_PWM);

else if(pos_consigna > posicio_cm){     //Si cal anar endavant
    if (IO_RA4_PORT == 1){
        valor_PWM = 0;      //anul·lar moviment
        PWM3_LoadDutyValue(valor_PWM);
        IO_RA4_PORT = 0;    //Canvi de sentit
    }else{
        PWM3_LoadDutyValue(valor_PWM);//Carregar sortida del controlador
    }
}
else if (pos_consigna < posicio_cm){       //Si cal anar endarrere
    if(IO_RA4_PORT == 0){
        valor_PWM = 0;          //anul·lar moviment
        PWM3_LoadDutyValue(valor_PWM);
        IO_RA4_PORT = 1;    //Canvi de sentit
    } else{
        PWM3_LoadDutyValue(valor_PWM); //Carregar sortida del controlador
    }
}
do_PI = false;
```

En la figura 30 es pot veure el flux del procés sencer del controlador PI, aquest inclou les accions dels fragments de codi que s'acaben de veure.

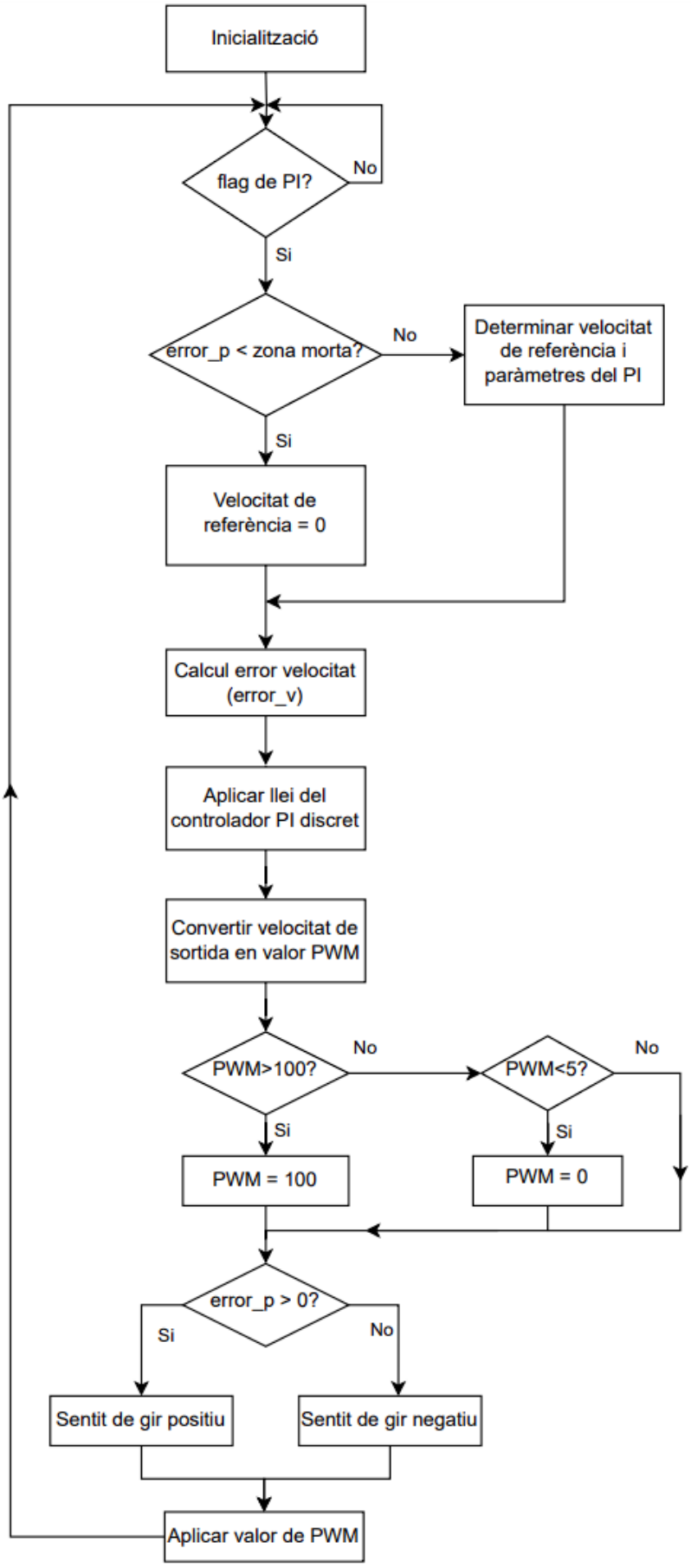

Figura 30: Diagrama de flux rutina PI del control de pitch

En l'annex C.1. es poden veure els resultats experimentals del sistema de control de pitch, utilitzant el microcontrolador per moure a voluntat el motor que s'instal·larà en l'aerogenerador. Gràcies a aquesta prova experimental es pot determinar el temps d'execució de l'algoritme de PI, és a dir de tots els fragments de codi presents d'aquest subapartat 3.3.1. Els resultats obtinguts indiquen un temps d'execució aproximat de 7ms, per tant dels 100ms del període de control, el PIC18 estarà ocupat amb la rutina PI un 7% del temps i disposant de la resta per realitzar altres rutines del programa.

Alternativament al sistema implementat, el PLC, mitjançant la comunicació Modbus, pot actuar directament sobre les variables de control del pitch per anular el moviment o modificar els paràmetres de Kp i Ki per optimitzar el control en l'etapa de posada en marxa.

Per altra banda l'esclau 2, encarregat de la monitorització de l'estructura i del vent, o l'esclau 3, encarregat del control de potència, poden enviar, en cas d'emergència, una ordre per situar el pitch en la seva posició màxima per tal de frenar l'aerogenerador. Quan el frenat de pitch es trobi en aquesta posició s'enviaria l'ordre a l'esclau 3 per tal d'activar el frenat elèctric i acabar d'aturar el generador.

Aquests dos processos de comunicació, juntament amb les maniobres d'execució de cada estat, s'expliquen més extensament en el capítol 5.

4. MONITORITZACIÓ DE L'ESTRUCTURA MECÀNICA I DEL VENT

El sistema de monitorització de l'aerogenerador consta d'un acceleròmetre per tal de mesurar les vibracions a la punta del molí, quatre ponts de Wheatstone formats per quatre galgues extensiomètriques units a amplificadors de voltatge per mesurar els esforços de flexió i tracció de l'estructura i dels tres cables tensors que la suporten, respectivament, i un anemòmetre per tal d'obtenir la velocitat i direcció del vent.

Aquests sensors es troben situats en una PCB independent a la punta de l'aerogenerador. Es realitzaran les connexions corresponents, utilitzant diversos mòduls, per tal de fer arribar les senyals a la base de l'aerogenerador, on hi haurà la PCB del microcontrolador PIC18, veure apartat 2.2., i els mòduls de comunicacions.

De la mateixa manera que en el sistema de control de pitch cada lectura de dades estarà lligada a una interrupció de temporitzador. Aquesta interrupció habilitarà les "flags" corresponents que activaran les accions de lectura des del bucle infinit del programa.

Les dades de vibracions s'obtenen cada 2,5ms, es requereix d'una freqüència d'adquisició elevada per tal de detectar les oscil·lacions de l'estructura, a part es fa una mitjana de les dades obtingudes cada 4 cicles per descartar les fluctuacions aleatòries presents en les lectures individuals i obtenir una mesura més precisa i fiable. També s'habilita un pin d'interrupció que detecta quan les vibracions son massa elevades, d'aquesta forma es pot detectar si el molí presenta desperfectes en l'estructura, com podria ser una pala trencada.

Les dades dels esforços tensors dels cables i de l'estructura es processen cada 500ms, és un període suficientment reduït com per aturar l'aerogenerador durant una ratxa de vent abans de que es provoquin desperfectes en l'estructura.

Les dades de velocitat i direcció del vent s'obtenen cada segon. La direcció s'obté únicament com a element informatiu ja que no aporta cap acció complementaria, els valors obtinguts podran utilitzar-se en un futur projecte per comprovar la viabilitat d'un mecanisme de rotació a la base de l'aerogenerador per tal d'encarar-lo al vent. La velocitat es processa utilitzant el "hardware" del microcontrolador, resultant amb un valor que representa la mitjana de velocitat del vent durant el segon d'adquisició, això permet detectar ràfegues de vent de curta durada i actuar en conseqüència.

4.1. Monitorització de les vibracions

Les vibracions es mesuraran a través d'una placa d'acceleròmetre MMA8451 i aquest es comunicarà amb el microcontrolador mitjançant bus I2C utilitzant l'adreça 0x1C. L'acceleròmetre es trobarà connectat en el mateix bus I2C multi-mestre que la resta de dispositius, s'escull una velocitat de transmissió de dades alta (400kbits/s) de manera que la lectura periòdica de les vibracions no obstaculitzarà la resta de comunicacions entre microcontroladors. A diferència de la resta de dispositius del bus, l'acceleròmetre funciona a 3,3V i per tant requereix d'una placa "level shifter" que permeti reduir la tensió del bus per tal de permetre la comunicació. L'adreça de dispositiu a utilitzar per l'acceleròmetre pot ser escollida d'entre dos valors seqüencials: 0x1C o 0x1D. La selecció d'aquesta adreça es realitza posant l'entrada SA0 del chip a nivell baix o nivell alt respectivament. En aquest projecte s'utilitza l'adreça 0x1C connectant el pin SA0 a terra (0V).

L'acceleròmetre té un ODR màxim de 800 Hz i disposa d'un buffer intern tipus FIFO, en la figura 31 es pot veure la placa amb les seves connexions.

Figura 31: Placa d'acceleròmetre MMA8451

El format de comunicació per la lectura i escriptura de l'acceleròmetre es pot veure a l'apartat 5.1. I2C.

Primer de tot es fa la inicialització de l'acceleròmetre, es modifiquen els registres de configuració segons les especificacions desitjades. Tota la configuració es realitza mitjançant escriptura de registres a través del bus I2C.

Es programa l'acceleròmetre a un rang màxim de ± 2g, l'ODR a 800Hz, el buffer intern es configurarà en mode cíclic i s'habilita el pin d'interrupció INT1 per tal d'avisar quan el valor d'acceleració superi el límit establert de 2g.

En els següents paràgrafs es pot veure la funció d'inicialització de l'acceleròmetre amb totes les escriptures als registres necessaris per implementar les configuracions que s'han mencionat. Alguns registres només poden ser escrits si el dispositiu es troba

apagat, per tant primer de tot es desactiva l'acceleròmetre per escriure en aquests registres concrets i després es torna a activar a la freqüència desitjada.

A continuació es pot veure la funció d'inicialització de l'acceleròmetre on es configuren tots els registres necessaris.

Quan es finalitza la configuració es llegeix el registre "WHO AM I" (0x0D) que conté l'identificador de dispositiu, 0x1A. Si el valor llegit difereix de l'esperat s'activa la "flag" d'error d'acceleròmetre activant la maniobra d'error, descrita en el subapartat 5.1.7.

```
_Bool Accelerometre_Init(void){  // Inicialització acceleròmetre
    //Apaguem accelerometre
    I2C_Master_Start(); //Start condition
    I2C_Master_Write(adreca_acc_escriptura);
    I2C_Master_Write(0x2A); //Adreça del registre CTRL_REG1
    I2C_Master_Write(0x00); //ASLP_RATE[1:0]--DR[2:0]--LNOISE--F_READ--ACTIVE
    I2C_Master_Stop(); //Stop condition

    //Desactivar o activar high-pass filter i configurar a 2g el max scale
    uint8_t XYZ_DATA_CFG = 0;
    if (filtre_pas_alt) XYZ_DATA_CFG|=0x10;
    if (escala_max < 3) XYZ_DATA_CFG|=escala_max;
    I2C_Master_Start(); //Start condition
    I2C_Master_Write(adreca_acc_escriptura);
    I2C_Master_Write(0x0E); //Adreça del registre XYZ_DATA_CFG Register
    I2C_Master_Write(XYZ_DATA_CFG); //000--HPF_OUT--00--FS[1:0]
    I2C_Master_Stop(); //Stop condition

    /*Permet activar una flag quan es supera cert valor d'acceleració marcat
     * pel  registre 0x17 (FF_MT_THS)*/
    I2C_Master_Start (); //Start condition
    I2C_Master_Write(adreca_acc_escriptura);
    I2C_Master_Write(0x15); //FF_MT_CFG Freefall/Motion Configuration Register
    I2C_Master_Write(0b11111000); //ELE--OAE--ZEFE--YEFE--XEFE--000
    I2C_Master_Stop(); //Stop condition

    //Activar interrupció motion CTRL_REG4
    I2C_Master_Start(); //Start condition
    I2C_Master_Write(adreca_acc_escriptura);
    I2C_Master_Write(0x2D); //Adreça del registre CTRL_REG4
    I2C_Master_Write(0b00000100); //INT_EN_FF_MT
    I2C_Master_Stop(); //Stop condition

    //Associar interrupció a pin INT1 CTRL_REG5
    I2C_Master_Start(); //Start condition
    I2C_Master_Write(adreca_acc_escriptura);
    I2C_Master_Write(0x2E); //Adreça del registre CTRL_REG5
    I2C_Master_Write(0b00000100); //INT_CFG_FF_MT
```

```
I2C_Master_Stop(); //Stop condition


//Engegar acceleròmetre, dades d'un byte i freq de 800hz CTRL_REG1
I2C_Master_Start(); //Start condition
I2C_Master_Write(adreca_acc_escriptura);
I2C_Master_Write(0x2A); //Adreça del registre CTRL_REG1
I2C_Master_Write(0x03); //ASLP_RATE[1:0]--DR[2:0]--LNOISE--F_READ--ACTIVE
I2C_Master_Stop(); //Stop condition


//Activar FIFO circular F_SETUP FIFO Setup Register
I2C_Master_Start(); //Start condition
I2C_Master_Write(adreca_acc_escriptura);
I2C_Master_Write(0x09); //Adreça del registre F_SETUP FIFO Setup Register
I2C_Master_Write(0b01000000); //F_MODE[1:0]--F_WMRK[5:0]
I2C_Master_Stop(); //Stop condition

//Configurar a 2g el nivell d'acceleració que activa el flag
uint8_t motion_threshold;
if(acceleracio_max<8){
    motion_threshold = acceleracio_max * 15.873; //resolucio=0.063g
}
I2C_Master_Start(); //Start condition
I2C_Master_Write(adreca_acc_escriptura);
I2C_Master_Write(0x17); //FF_MT_THS Freefall and Motion Threshold Register
I2C_Master_Write(motion_threshold);//DBCNTM--THS[6:0]  default: 0b00100000
I2C_Master_Stop(); //Stop condition

/*Configurar a 10ms el temps per el qual es supera el valor abans
 * d'activar la flag*/
I2C_Master_Start(); //Start condition
I2C_Master_Write(adreca_acc_escriptura);
I2C_Master_Write(0x18);//Adreça del registre FF_MT_COUNT Debounce Register
I2C_Master_Write(0b00000001); //D[7:0] (resolucio=10ms)
I2C_Master_Stop(); //Stop condition

//AJUSTAMENT D'OFFSET
//Offset X
int8_t correccio_offset_X;
correccio_offset_X = Offset_X>>1; //resolucio 2mg
I2C_Master_Start();  //Start condition
I2C_Master_Write(adreca_acc_escriptura);
I2C_Master_Write(0x2F); //OFF_X Offset Correction X Register
I2C_Master_Write(correccio_offset_X); //D[7:0]  default: 0b00000000
I2C_Master_Stop();   //Stop condition

//Offset Y
int8_t correccio_offset_Y;
```

```
    correccio_offset_Y = Offset_Y>>1; //resolucio 2mg
    I2C_Master_Start();         //Start condition
    I2C_Master_Write(adreca_acc_escriptura);
    I2C_Master_Write(0x30);     //OFF_Y Offset Correction Y Register
    I2C_Master_Write(correccio_offset_Y);     //D[7:0]  default: 0b00000000
    I2C_Master_Stop();          //Stop condition
    //Offset Z
    int8_t correccio_offset_Z;
    correccio_offset_Z = Offset_Z>>1; //resolucio 2mg
    I2C_Master_Start();         //Start condition
    I2C_Master_Write(adreca_acc_escriptura);
    I2C_Master_Write(0x31);     //OFF_Z Offset Correction Z Register
    I2C_Master_Write(correccio_offset_Z);     //D[7:0]  default: 0b00000000
    I2C_Master_Stop();          //Stop condition

    //Comprovació de comunicació
    I2C_Master_Start();
    I2C_Master_Write(adreca_acc_escriptura); //7 bit address + Write
    I2C_Master_Write(0x0D); //Registre WHO_AM_I
    I2C_Master_RepeatedStart();
    I2C_Master_Write(adreca_acc_lectura); //7 bit address + Read
    uint8_t ID = I2C_Master_Read(0); //Read + NotAcknowledge ID=0x1A?
    I2C_Master_Stop(); //Stop condition

    if (ID != 0x1A) return false;
    else return true;
}
```

Alguns valors de la configuració poden ser modificats de cara a la posada en marxa de l'aerogenerador, mitjançant la comunicació RS-485 amb el PLC, per tal d'adaptar-se a les necessitats del sistema. S'explica més en detall quins valors i quins rangs tenen en el subapartat 5.2.2.

Com ja s'ha explicat en l'inici del capítol es farà una lectura constant de dades mitjançant una interrupció del temporitzador TMR4 cada 2,5ms. S'aplicarà un filtratge senzill a les dades obtingudes, els valors d'acceleració es sumaran entre ells i cada 10ms es farà una mitjana dels 4 valors obtinguts fins al moment. En la figura 32 es pot veure un esquema de la maniobra descrita. Aquesta mitjana es guarda en la memòria del microcontrolador fins que és sobreescrita pel següent valor. El valor actual d'acceleració pot ser llegit en tot moment pel PLC mitjançant el bus RS-485 per tal de monitoritzar el sistema.

Mitjançant els valors obtinguts en els tres eixos, X, Y i Z es poden obtenir gràfiques de les vibracions i analitzant les gràfiques, calcular la freqüència de ressonància

fonamental de l'aerogenerador, molt útil en un anàlisis mecànic. En l'annex C.2. es mostren els resultats d'aquesta prova experimental, realitzada en un prototip d'aerogenerador més petit.

El pin d'interrupció pot determinar si hi ha desperfectes en l'estructura de l'aerogenerador (per exemple, si es trenca una pala), en el cas de que es detectés un nivell d'acceleració superior a 2g durant almenys 10ms, s'activaria el pin INT1. Aquest està connectat al microcontrolador i generaria una interrupció per GPIO en el programa, activant l'estat d'emergència de l'aerogenerador.

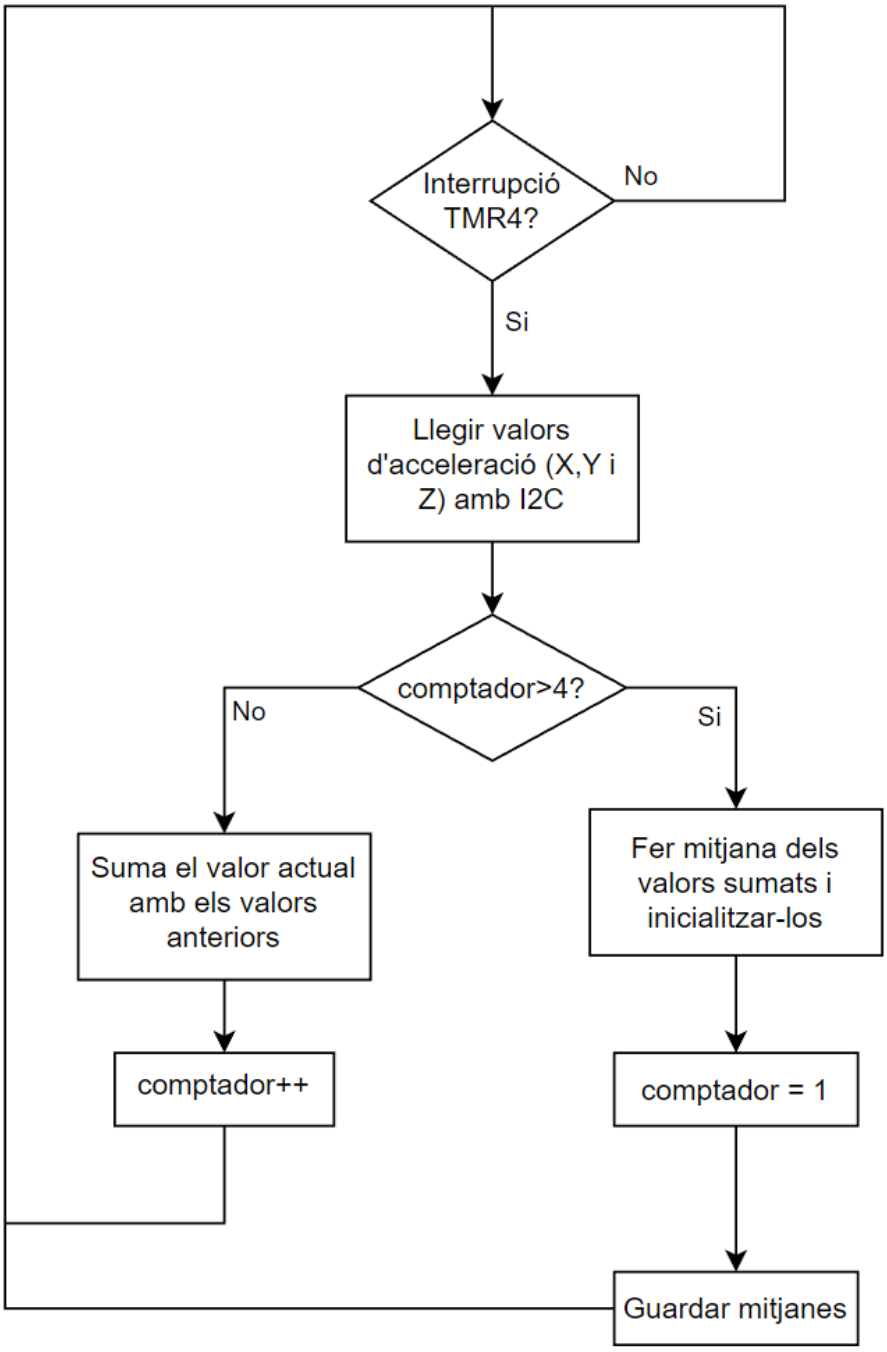

Figura 32: Diagrama de flux rutina d'interrupció TMR4 del tractament de dades

4.2. Monitorització dels esforços

Els esforços es mesuraran a partir de 16 galgues, 4 col·locades en la pròpia estructura de l'aerogenerador i 12 en els cables tensors que la suporten, 4 galgues en cada cable. Aquestes estaran connectades en un pont de Wheatstone. Les connexions d'entrada es realitzaran entre 5V i GND.

La configuració de pont de Wheatstone escollida pels cables es pot visualitzar en la figura 33 on s'indica la posició física i les connexions elèctriques de cada galga.

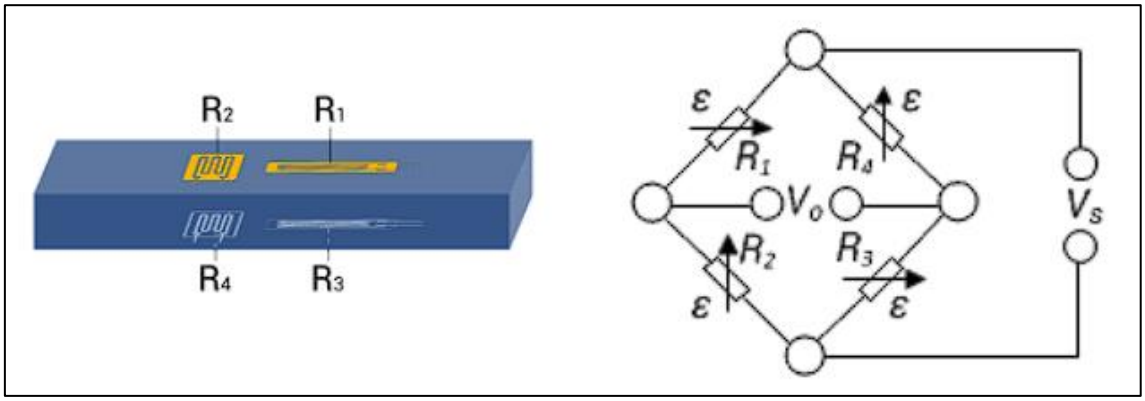

Figura 33: Posició de les galgues en els cables tensors

Aquesta configuració de pont complet permet mesurar les deformacions de tracció independentment de la flexió aplicada, quan els cables no es trobin generant esforços poden flexionar-se sols pel seu propi pes i ens interessa descartar aquest efecte.

Al ser un pont complet els efectes de la temperatura es compensen entre les galgues ja que la seva resistivitat variarà uniformement.

L'equació 5 descriu el comportament elèctric d'aquest pont.

$$\varepsilon = \frac{1}{2\cdot(1+\nu)} \cdot \frac{4}{k} \cdot \frac{V_0}{V_S} \qquad \text{(Eq. 5)}$$

On "Ɛ" correspon a la deformació aplicada (Ɛ = $\Delta L/L_0$), "ν" fa referència al coeficient de Poisson del material, "k" correspon al factor de galga o factor de calibre (en les galgues utilitzades en el projecte, k=2), V_S és la tensió d'alimentació del pont (5V) i V_0 la tensió de sortida.

Per altra banda el pont de Wheatstone de l'estructura de l'aerogenerador requereix d'una disposició de galgues diferent per tal de mesurar els esforços de flexió propis del màstil. La configuració escollida es pot veure en la figura 34.

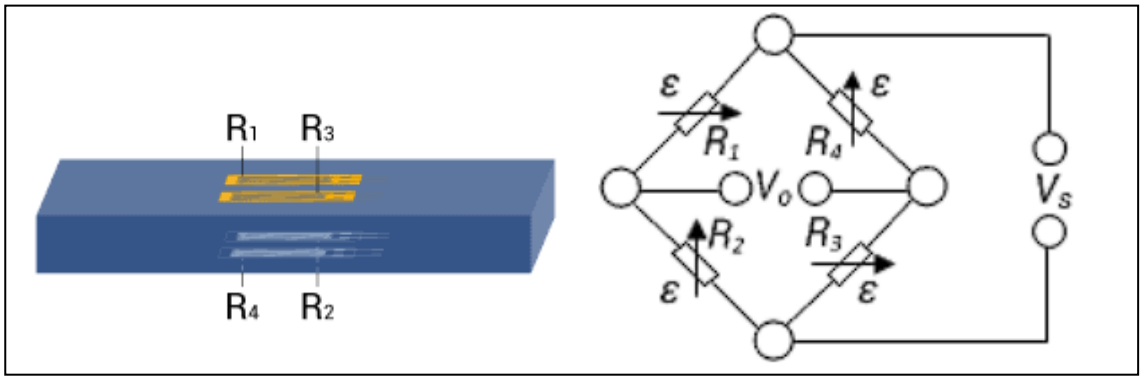

Figura 34: Posició de les galgues en l'estructura de l'aerogenerador

En aquesta disposició, al contrari que amb la de la figura 33, s'ignoren els esforços de tracció i compressió i únicament s'obtenen els valors de la deformació per flexió.

L'equació 6 ens descriu el seu comportament.

$$\varepsilon = \frac{1}{4} \cdot \frac{4}{k} \cdot \frac{V_0}{V_S} \qquad \text{(Eq. 6)}$$

Els termes utilitzats corresponen als mateixos vistos en l'equació 5, on "Ɛ" correspon a la deformació aplicada ($\varepsilon = \Delta L/L_0$), "k" correspon al factor de galga o factor de calibre (en les galgues utilitzades en el projecte, k=2), V_S és la tensió d'alimentació del pont (5V) i V_0 la tensió de sortida.

S'han escollit aquestes configuracions ja que presenten un senyal de sortida potent i un bon CMR (rebuig del mode comú).

Cada senyal de sortida dels 4 ponts es connecta a una placa amplificadora de voltatge AD620, situades en la PCB de la punta del molí. Aquesta placa té un guany regulable de 1,5 a 1000 i proporcionarà un voltatge de sortida de 0 a 3V. Amb les configuracions de pont vistes anteriorment, es regularan els amplificadors fins a obtenir un voltatge de sortida acceptable que pugui ser llegit pel microcontrolador. Cal tenir present que els tres amplificadors dels ponts dels cables tensors han d'estar configurats amb el mateix guany, per tal de que l'estat d'emergència salti correctament.

Es faran arribar les quatre senyals analògiques dels amplificadors al microcontrolador a través de plaques de conversió voltatge-intensitat. La intensitat d'un circuit en sèrie és la mateixa en qualsevol punt sense importar la resistència del cablejat, d'aquesta manera al transmetre senyals de la part superior a la part inferior de l'aerogenerador ens assegurem que es manté la precisió de la resposta evitant les interferències.

Finalment caldrà programar l'ADC del PIC18 per tal d'adquirir els valors de tensió dels quatre amplificadors i introduir-los en el codi dins d'una interrupció per temporitzador de 500ms. En cada període de la interrupció es llegiran els quatre valors dels amplificadors i si superen els límits marcats d'algun d'ells s'activarà l'estat d'emergència enviant el corresponent missatge d'avís a través de l'I2C. Al subapartat 5.1.6. es parla d'aquesta maniobra, de les seves condicions d'activació i de les seves accions conseqüents.

Tanmateix, els valors llegits poden ser demanats en qualsevol moment pel PLC i transmesos mitjançant el bus RS-485, com s'explica en el subapartat 5.2.2.

A continuació es pot veure la funció de lectura dels valors de les galgues.

```
void lectura_galgues(){ //TMR 2 - 500ms

    //0-1024 bits --> X * 5000/1024 mV/bits
    ADCC_DischargeSampleCapacitor();
    amp_galgues_estructura = ADCC_GetSingleConversion(channel_ANA1)*4.8828;

    //0-1024 bits --> X * 5000/1024 mV/bits
    ADCC_DischargeSampleCapacitor();
    amp_galgues_cable1 = ADCC_GetSingleConversion(channel_ANA2)*4.8828;

    //0-1024 bits --> X * 5000/1024 mV/bits
    ADCC_DischargeSampleCapacitor();
    amp_galgues_cable2 = ADCC_GetSingleConversion(channel_ANA3)*4.8828;

    //0-1024 bits --> X * 5000/1024 mV/bits
    ADCC_DischargeSampleCapacitor();
    amp_galgues_cable3 = ADCC_GetSingleConversion(channel_ANA4)*4.8828;

    if (amp_galgues_estructura > valor_max_galgues_estructura
            || amp_galgues_cable1 > valor_max_galgues_cable
            ||  amp_galgues_cable2 > valor_max_galgues_cable
            || amp_galgues_cable3 > valor_max_galgues_cable){
        estat_emergencia = ALARMA_GALGUES;
    }
    activar_lectura_galgues = false;
}
```

4.3. Monitorització del vent

Els efectes del vent es mesuraran amb l'anemòmetre que es veu a la figura 35. Disposa de 4 cables, dos d'alimentació i dos de dades.

Figura 35: Anemòmetre

Aquest transmetrà un senyal analògic per indicar la direcció del vent, a nivell intern s'obté la senyal d'un potenciòmetre de 20 kΩ. Així com les galgues aquest senyal es transmetrà a través d'un convertidor voltatge-intensitat fins el microcontrolador i es llegirà mitjançant l'ADC.

La direcció del vent es llegeix com un senyal de 10 bits, per tant té un rang de valors de 0 a 1023. Es fa un escalat d'aquesta senyal per obtenir el valor en graus, on 180º corresponen al sud geogràfic.

Per altra banda es transmetrà també un senyal digital indicant la velocitat del vent. El senyal s'enviarà utilitzant un mòdul UART/RS-485, els mateixos que s'utilitzen per comunicar-se amb el PLC via Modbus. La velocitat es calcularà seguint la fórmula de l'equació 7.

$$V = 0{,}44704 \cdot \left(P \cdot (2{,}25/T)\right) \qquad \text{(Eq. 7)}$$

On P són el nombre de polsos per període de mostreig i T és el temps de mostreig en segons. La velocitat resultant s'obté en m/s.

Per tal de llegir aquesta senyal digital s'habilitarà com a entrada de rellotge (T1CKI) i el propi hardware portarà a terme un comptatge constant dels polsos. Després,

mitjançant una interrupció per temporitzador de 1 segon (TMR0) es llegirà tant el senyal analògic com el valor del comptador de polsos i aplicant l'equació vista anteriorment obtenim les dues dades desitjades.

Quan el valor de velocitat superi els 20m/s s'activarà l'alarma de velocitat del vent elevada i s'activarà l'estat d'emergència, enviant els corresponents missatges pel bus I2C. Per altra banda quan la velocitat del vent sigui inferior a 5m/s es desactivarà la variable de generació, factor relacionat amb el mode de baix consum/sleep.

En el codi següent podem veure l'adquisició de dades de l'anemòmetre així com les accions de l'estat d'emergència i de l'estat de sleep, aquestes s'expliquen més en detall en l'apartat 5.1. I2C.

```
void lectura_anemometre(){ //TMR 0 - 1s
    ADCC_DischargeSampleCapacitor();
    uint16_t dir_anemometre_raw = ADCC_GetSingleConversion(channel_ANA7);
    dir_anemometre = dir_anemometre_raw * 360 / 1024;
    vel_anemometre = 0.44704 *(voltes_anemometre * (2.25 / 1)); // 0.44704
*(P*(2.25 /T)) m/s
    if((vel_anemometre<limit_baix_anemometre) && Generacio_ESCLAU2){
        Generacio_ESCLAU2 = false;
        canvi_generacio();
    }
    else if(vel_anemometre>limit_alt_anemometre){
        estat_emergencia = ALARMA_VENT;
    }
    else if (!Generacio_ESCLAU2){
        despertar_accelerometre();
        TMR2_StartTimer();
        TMR3_StartTimer();
        TMR4_StartTimer();
        Generacio_ESCLAU2 = true;
        canvi_generacio();
    }
    activar_lectura_anem = false;
}
```

Els valors de direcció i velocitat poden ser llegits en qualsevol moment pel PLC mitjançant el bus RS-485, s'explica més en el subapartat 5.2.2.

5. COMUNICACIONS DEL SISTEMA

En el sistema s'utilitzaran dos busos de comunicació, I2C i RS-485 (Modbus RTU).

L'I2C s'utilitzarà per la comunicació entre microcontroladors i també per configurar i obtenir dades de l'acceleròmetre. Aquest bus és molt utilitzat en la industria dels microcontroladors i es troba present en molts dispositius, té una velocitat de transmissió força elevada i requereix tant sols de dos cables. A diferència del bus SPI, també disponible en el mòdul de comunicació MSSP del PIC18, l'I2C disposa d'un control de flux per hardware per tal d'evitar col·lisions i és més robust gràcies a les senyals d'assentiment que s'envien durant la comunicació. Per tots aquests motius s'ha escollit l'I2C com a bus de comunicació intern de l'aerogenerador.

El protocol Modbus s'utilitzarà en cablejat RS-485 per la comunicació entre el PLC i els microcontroladors. Aquest bus ens permet enviar i rebre senyals a una llarga distància, de fins a 1200m utilitzant un parell trenat. Aquesta transmissió diferencial permet aïllar la senyal enviada del soroll elèctric de l'ambient i que la tensió necessària del bus sigui menor. Permet una comunicació robusta amb una estructura de missatges senzilla i és àmpliament utilitzat en la indústria per a la supervisió i control de processos automatitzats. Una altra característica important del protocol és la seva independència de plataforma, és a dir, permet la comunicació entre diferents equips i sistemes de diversos fabricants. Per totes aquestes raons s'ha escollit el Modbus com a protocol de comunicació extern de l'aerogenerador.

S'han realitzat varies proves experimentals d'ambdós busos per comprovar el seu correcte funcionament, en l'annex C.3. es poden veure els resultats d'aquestes proves. Consisteix en proves de comunicació bàsiques entre els dispositius i s'intenten ajustar el màxim possible a la comunicació real que es tindrà en la versió final de l'acceleròmetre.

5.1. I2C

És un bus de comunicació en sèrie síncron inventat per Philips Semiconductors. S'utilitza principalment per la comunicació entre diferents dispositius d'un circuit. Està dissenyat com un bus mestre-esclau, on el mestre es el que inicia sempre la transferència de dades i l'esclau reacciona segons correspongui. Tot i així, és possible tenir més d'un mestre mitjançant un mode multi-mestre. En aquest mode, dos mestres es poden comunicar entre ells quan un d'aquests actua com a esclau. En el nostre cas, els microcontroladors PIC18 disposen de dos mòduls MSSP, els dos es troben

connectats a la mateixa xarxa I2C, un d'ells treballant com a mestre i l'altre com a esclau.

I2C utilitza dues línies bidireccionals anomenades SDA i SCL, que funcionen a 5 V o a 3.3 V a través d'unes resistències de "pull-up". La línia SDA és l'encarregada de transmetre les dades en el bus mentre que la línia SCL envia el senyal de rellotge que marca la velocitat de transmissió. Usualment es fan servir dos modes de funcionament, estàndard a 100 kbits/s i ràpid a 400 kbits/s. En el projecte s'utilitzarà el bus de comunicació en mode ràpid per tal de tenir una càrrega de bus menor i disposar de més temps per poder enviar missatges d'alerta entre dispositius. En la figura 36 es pot veure un exemple de transmissió de dades en el bus I2C i els canvis de nivell en les dues línies de comunicació.

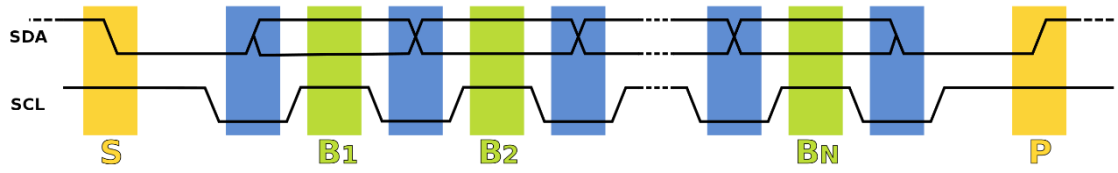

Figura 36: "Frame" de transmissió I2C

Les dues línies romanen a nivell alt fins l'inici d'una comunicació, on el bus de dades passa a nivell baix. Seguidament es transmet la informació, el bus del rellotge puja i baixa de forma periòdica a la freqüència marcada pel mode de funcionament, el bus de dades s'escriu quan el rellotge es troba a nivell baix i es llegeix quan el rellotge es troba a nivell alt. Finalment la comunicació s'atura quan el rellotge i les dades es queden a nivell alt.

El format de comunicació I2C varia segons les característiques de l'esclau i cal mirar amb deteniment la documentació abans de configurar la comunicació. Cada dispositiu requereix una seqüència concreta i uns temps concrets entre flancs per portar a terme la comunicació, tot i així hi han accions que solen ser iguals o molt semblants en tots els dispositius com podria ser la lectura i escriptura d'un byte. Aquestes seqüències es poden veure en les figures 37 i 38.

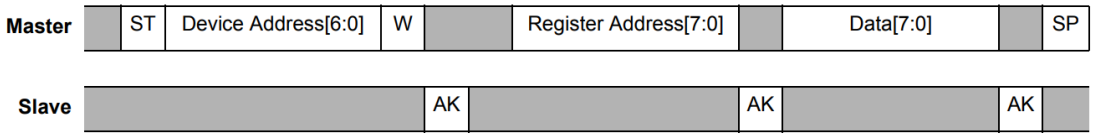

Figura 37: Escriptura d'un byte I2C

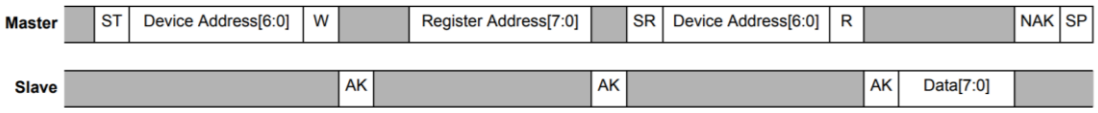

Figura 38: Lectura d'un byte I2C

Les dos comunicacions comencen de la mateixa forma, el mestre envia la condició d'inici vista anteriorment seguida de l'adreça de l'esclau en format de 7 bits més un bit d'escriptura (0 lògic) indicant que estem escrivint informació en el bus. Si tot va bé l'esclau ens retornarà un bit de "Acknowledge" (0 lògic) conforme ha identificat l'adreça que s'ha escrit. Seguidament s'envia l'adreça de 8 bits del registre sobre el qual es voldrà llegir o escriure la informació i l'esclau ens retornarà un altre bit AK conforme ha rebut correctament la informació.

En el cas de l'escriptura es tornaran a repetir les mateixes accions, el mestre envia 8 bits amb les dades i l'esclau respon amb un bit AK, finalment el mestre atura la comunicació.

En el cas de la lectura es torna a enviar un bit d'inici (s'anomena una condició de "Repeated Start" o SR) amb l'adreça del dispositiu esclau i un bit de lectura (1 lògic), l'esclau aquest cop respondrà amb un bit AK i les dades del registre que s'hagi demanat. Finalment el mestre envia un NAK (1 lògic) per aturar la comunicació.

Aquesta estructura, com ja s'ha comentat, pot variar entre dispositius, per exemple alguns de molt senzills no requereixen de registres i d'altres de més complexes ofereixen mètodes per llegir o escriure varis registres dins d'una sola comunicació.

Per tal de comunicar els microcontroladors entre si s'utilitza un extensor d'I2C degut a la distància entre dispositius. Aquests extensors utilitzen un bus I2C diferencial amb quatre conductors per tal d'evitar interferències electromagnètiques i proporcionar robustesa al sistema, permetent distàncies de fins a 1200m. S'incorporen connectors RJ45 per tal de realitzar les connexions entre extensors.

Tal i com s'ha vist en la placa utilitzada per sintonitzar el controlador del sistema de control del pitch de la figura 8, aquests mòduls es munten sobre els microcontroladors mitjançant el suport Mikrobus i s'alimenten a través d'ells. En el cas de que els dos suports dels que disposa el PIC estiguin ocupats es realitzen les connexions a través de la PCB i es col·loquen els mòduls sobre regletes. En les PCB es disposen aquestes regletes en els extrems per tal de que puguin manipular-se fàcilment.

En la figura 39 es pot veure el mòdul extensor de I2C.

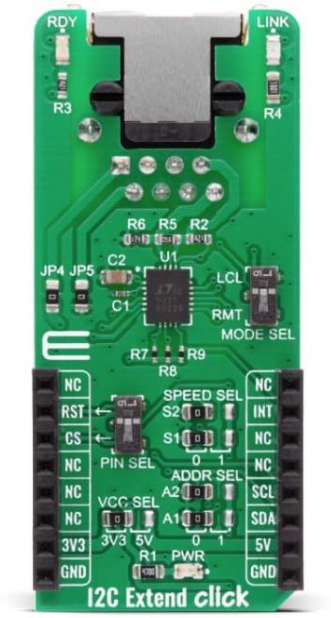

Figura 39: Extensor I2C

A més s'utilitzarà un HUB RJ45 com el de la figura 40 per tal d'interconnectar tots els cables procedents dels extensors.

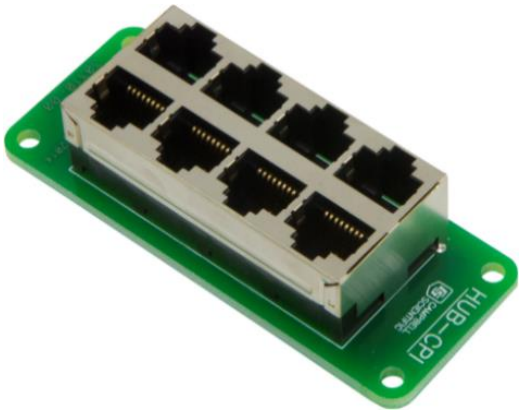

Figura 40: HUB RJ45

Els diferents sistemes de l'aerogenerador es troben connectats a aquesta xarxa I2C multi-mestre, d'aquesta manera poden treballar de forma autònoma, sense ordres del PLC. En casos d'emergència, d'error o de baixa generació els microcontroladors poden enviar els corresponents missatges a la xarxa I2C per tal de realitzar les accions pertinents de manera conjunta.

Per habilitar la comunicació dels microcontroladors a través de l'I2C, s'han de modificar directament els registres del mòdul MSSP. S'utilitza el mòdul MSSP1 com a mestre i el mòdul MSSP2 com a esclau en la xarxa multi-mestre. Lamentablement el

fabricant no proporciona llibreries de comunicació I2C específiques pel controlador PIC18. Per tant, s'ha de configurar la comunicació a nivell baix, creant funcions per enviar i rebre missatges tant en mode mestre com en mode esclau.

Posteriorment, s'han afegit les funcions als arxius i2c.c i i2c.h per millorar-ne la portabilitat i poder reutilitzar-les en altres projectes si fos necessari. La funció de recepció de dades en mode esclau necessita ser declarada en l'arxiu main.c per tal de que la interrupció que es genera al rebre dades funcioni correctament. La resta de funcions de comunicació (inicialització del mode mestre, inicialització del mode esclau, generació de condició d'espera, d'inici, d'inici repetit i d'aturada en mode mestre, lectura en mode mestre i escriptura en mode mestre) es declaren en l'arxiu i2c.h i es defineixen les seves accions en l'arxiu i2c.c. Totes les funcions del mode mestre i del mode esclau s'expliquen en detall en els subapartats 5.1.2. i 5.1.3. respectivament.

5.1.1. Protocol de comunicació del bus I2C

Per tal d'escriure i rebre missatges amb coherència s'ha creat un protocol de comunicació, en la taula 2 es pot veure l'adreça utilitzada pel microcontrolador de cada sistema.

Adreça	**Dispositiu**
0x08	Microcontrolador de control del pitch
0x10	Microcontrolador de monitorització de l'estructura mecànica i del vent
0x28	Microcontrolador de control del consum de potència

Taula 2: Adreces I2C dels microcontroladors de l'aerogenerador

Un microcontrolador en mode mestre utilitzarà aquestes adreces per comunicar-se amb els altres microcontroladors de l'aerogenerador. Aquestes adreces s'afegiran a la inicialització del mode esclau del MSSP2.

Cal tenir present que hi ha altres dispositius en el sistema, l'acceleròmetre i un microcontrolador ESP32 utilitzat en el sistema de control del consum de potència, i les seves adreces s'han tingut en compte per tal de no generar conflicte amb els microcontroladors.

En la taula 3 es poden veure tots els missatges que es transmeten en el bus multi-mestre i el seu significat.

Missatge	Significat
0x12	El sistema de control de pitch no detecta generació.
0x13	El sistema de control de pitch detecta generació.
0x1E	Missatge d'error del microcontrolador 1.
0x1F	El pitch es troba al seu valor màxim.
0x22	El sistema de monitorització no detecta generació.
0x23	El sistema de monitorització detecta generació.
0x2A	Vibracions excessives.
0x2B	Esforços excessius en els cables tensors o l'estructura.
0x2C	Velocitat del vent excessiva.
0x2E	Missatge d'error del microcontrolador 2.
0x3E	Missatge d'error del microcontrolador 3.
0xA3	El frenat elèctric es troba al màxim.
0xBB	Activació de mode test del bus I2C.
0xE1	"Heartbeat" del microcontrolador de control de pitch.
0xE2	"Heartbeat" del microcontrolador de monitorització.
0xE3	"Heartbeat" del microcontrolador de control del consum de potència.
0xF0	El sistema de control de potència no detecta generació.
0xF1	El sistema de control de potència detecta generació.

Taula 3: Missatges I2C dels microcontroladors de l'aerogenerador

Cada missatge s'utilitza en un mode de funcionament específic, en els subapartats 5.1.4., 5.1.5., 5.1.6. i 5.1.7. s'expliquen els diversos modes de funcionament i es relacionen amb els missatges vistos.

5.1.2. Funcions mode mestre

En mode mestre el microcontrolador es configura per poder transmetre i rebre dades, també controla la senyal de rellotge.

Primer de tot es crida la funció d'inicialització per tal d'escriure en els registres de MSSP1. Es configura el mòdul en mode mestre actuant sobre els registres de

configuració SSP1CON1, SSP1CON2, i es configura la velocitat del bus a 400kbits/s amb els registres SSP1ADD i SSP1STAT. També s'associa el canal de dades i el canal de rellotge a pins físics del microcontrolador utilitzant els registres SSP1CLKPPS i SSP1DATPPS i es desactiven temporalment amb el bit SSPEN del registre SSP1CON1 fins que es doni una condició d'inici. Finalment es prefixa a 300ns el temps entre que es produeix un flanc de baixada en el bus SCL i que es transmet la dada pel bus SDA amb el bit SDAHT del registre SSP1CON3.

```
void I2C_Master_Init(const unsigned long c)
{
    //Configuració del mòdul MSSP1 actuant com a  mestre de I2C
    SSP1CON1 = 0b00101000;
    SSP1CON2 = 0;
    //Definció del divisor del clock -> 400kbits/s
    SSP1ADD = (_XTAL_FREQ/(4*c))-1;
    //Es posa el "slew rate" en mode d'alta freqüència
    SSP1STAT = 0;

    //Definició dels pins del canals de dades i de clock.
    SSP1CLKPPS = 0x13;   //RC3->MSSP1:SCL1;
    RC3PPS = 0x0F;   //RC3->MSSP1:SCL1;
    RC4PPS = 0x10;   //RC4->MSSP1:SDA1;
    SSP1DATPPS = 0x14;   //RC4->MSSP1:SDA1;
    //Es desactiven els pins fins que es doni una seqüència de start
    SSP1CON1bits.SSPEN = 0;
    //Es prefixa el temps de hold del bus SDA.
    SSP1CON3bits.SDAHT = 1;
}
```

A partir d'aquí ja es poden activar les funcions de comunicació que realitzen accions de control en el bus. S'ha vist un exemple d'aplicació d'aquestes rutines en les figures 30 i 31.

```
void I2C_Master_Wait(void)
{
    //Funció que analitza si hi ha comunicació activa i si es el cas, espera
    while ((SSP1STAT & 0x04) || (SSP1CON2 & 0x1F));
}

void I2C_Master_Start(void)
{
    //Acció d'espera
    I2C_Master_Wait();
    //S'habiliten els pins de comunicació.
```

```
    SSP1CON1bits.SSPEN = 1;
    //S'inicia la seqüència de start.
    SSP1CON2bits.SEN = 1;
    //Quan acaba la seqüència aquest bit es posa a zero automàticament.
    while(SSP1CON2bits.SEN);
    //Fins que no es posi a zero la funció no acaba.
}

void I2C_Master_RepeatedStart(void)
{
    //Acció d'espera
    I2C_Master_Wait();
    //S'inicia la seqüència de repeated start.
    SSP1CON2bits.RSEN = 1;
    //Quan acaba la seqüència aquest bit es posa a zero automàticament.
    while (SSP1CON2bits.RSEN);
    //Fins que no es posi a zero la funció no acaba.
}

void I2C_Master_Stop(void)
{
    //Acció d'espera
    I2C_Master_Wait();
    //S'inicia la seqüència de stop.
    SSP1CON2bits.PEN = 1;
    //Quan acaba la seqüència aquest bit es posa a zero automàticament.
    while(SSP1CON2bits.PEN);
    //Fins que no es posi a zero la funció no acaba.
}
```

Les accions de "start", "repeated start" i "stop" son molt semblants entre elles. Habiliten el seu bit de registre corresponent i s'espera a que aquest s'esborri automàticament per "hardware" un cop l'acció s'ha realitzat.

La funció d'espera permet comprovar que no hi hagi una transmissió en progrés en el bus o que no hi hagin accions pendents per part del propi mestre.

L'escriptura i la lectura son les funcions principals del mestre i es mostren a continuació:

```
void I2C_Master_Write(unsigned d)
{
    //Acció d'espera
    I2C_Master_Wait();
    //Escriure la dada desitjada en el buffer.
```

```
    SSP1BUF = d;
    //Esperar que es buidi el buffer i que es rebi l'Acknowledge
    while(SSP1STATbits.R_nW);
}

unsigned short I2C_Master_Read(unsigned short a)
{
    unsigned short temp; //Variable de retorn
    I2C_Master_Wait(); //Acció d'espera
    SSP1CON2bits.RCEN = 1; //habilita el bit de recepció
    I2C_Master_Wait(); //Espera del bus per rebre dades.
    temp = SSP1BUF; //El valor del registre SSP1BUF es guarda a la variable.
    I2C_Master_Wait(); //Espera del bus
    SSP1CON2bits.ACKDT = (a)?0:1;/*Depenent del valor de la variable a, el
                                  * mestre retorna un ACK per continuar amb la
                                  * comunicació o un NACK per finalitzar-la.*/
    SSP1CON2bits.ACKEN = 1; //Habilita l'Acknowledge.
    return temp; //Retorna la variable.
}
```

Per fer una escriptura s'introdueixen les dades que es vulguin enviar en el buffer, després s'espera a que es buidi i es rebi l'ACK acabant amb la transmissió.

Per tal de fer una lectura s'habilita la recepció de dades i s'espera a que el buffer s'empleni, un cop el buffer està ple es guarda el seu valor en una variable. Finalment, segons el paràmetre que haguem passat a la funció, s'envia un ACK (1) per continuar amb la comunicació o un NACK (0) per finalitzar-la.

En el cas de que es produís un error de comunicació, per danys en els cables o soroll extern, el programa quedaria aturat eternament en la funció d'espera. Això provocaria el desbordament del "watchdog" i el programa es reiniciaria en mode error, en el subapartat 5.1.7. s'expliquen les accions que es porten a terme en aquest cas.

5.1.3. Funcions mode esclau

En mode esclau el microcontrolador es configura per poder respondre a les accions del mestre i funcionar a la freqüència requerida.

Primer de tot es crida la funció d'inicialització per tal d'escriure en els registres del MSSP2. Es configura el mòdul en mode esclau amb els registres SSP2CON1 i SSP2CON2 i se li associa una adreça de bus específica al registre SSP2ADD. També s'habilita el mode ràpid per tal de que pugui funcionar a 400kbits/s amb el registre SSP2STAT. Per últim, de la mateixa forma que s'havia fet amb el mestre, s'associa el canal de dades i el canal de rellotge a pins físics del microcontrolador amb els

registres SSP2DATPPS i SSP2CLKPPS. Per últim es neteja la "flag" d'interrupció (SSP2IF) i s'habiliten les interrupcions del mòdul (SSP2IE).

```
void I2C_Slave_Init(short address)
{
    //Activar "slew rate" a 400kbits/s
    SSP2STAT = 0x00;
    //Definir l'adreça de l'esclau
    SSP2ADD = address;
    //Configurar en mode esclau i habilitar els pins
    SSP2CON1 = 0x36;
    SSP2CON2 = 0x01;

    //Associar SDA i SCL a pins físics del micro
    SSP2DATPPS = 0x0A;   //RB2->MSSP2:SDA2;
    RB1PPS = 0x11;   //RB1->MSSP2:SCL2;
    RB2PPS = 0x12;   //RB2->MSSP2:SDA2;
    SSP2CLKPPS = 0x09;   //RB1->MSSP2:SCL2;
    SSP2IF = 0;        //Netejar la flag d'interrupció
    SSP2IE = 1;        //Habilitar interrupcions del mòdul SSP2
}
```

La funció de lectura a diferència del que s'ha vist en el mestre actua dins de la interrupció del mòdul MSSP2. És a dir, aquesta funció s'activa cada vegada que el buffer rep un byte d'adreça que casa amb l'adreça que s'ha configurat en la inicialització. Per tant, si es reben bytes que van destinats a un altre esclau, el dispositiu no fa res, però quan es rep un byte que correspon amb l'adreça inicialitzada, la "flag" es posa a nivell alt i fa la interrupció, executant el fragment de codi que es veu a continuació. A partir del moment en que s'activa, per a cada byte rebut es segueix produint una interrupció fins que no es detecta la condició d'aturada per part del mestre.

A continuació es pot veure el fragment de codi que gestiona les interrupcions de l'esclau.

```
void I2C_Slave_Read(void) {
    if(SSP2IF==1){
        //De la manera com marca el protocol l'esclau posa a zero el clock.
        SSP2CON1bits.CKP = 0;
         //El motiu és per garantir el temps de configuració de les dades.
        if ((SSP2CON1bits.SSPOV) || (SSP2CON1bits.WCOL))
        { //Cas d'overflow o col·lisió
            lecturaI2C = SSP2BUF; //Es borra el contingut del buffer.
            SSP2CON1bits.SSPOV = 0; // Es borra el bit d'OVERFLOW
```

```
            SSP2CON1bits.WCOL = 0;  // Es borra el bit de col·lisió.
            SSP2CON1bits.CKP = 1; //Es reinicialitza el rellotge.
        }

        if(!SSP2STATbits.D_nA && !SSP2STATbits.R_nW)
        { //El byte rebut coincideix amb l'adreça i està en mode escriptura.
            SSP2CON2bits.ACKEN = 1; //S'habilita l'acknowledge
            lecturaI2C = SSP2BUF; //Es llegeix el contingut del buffer.
            SSP2CON1bits.CKP = 1; //Es reinicialitza el rellotge.
            switch (lecturaI2C){ //Processament de la dada rebuda
                default:
                   break;
            }
        }
        else if(!SSP2STATbits.D_nA && SSP2STATbits.R_nW)
        {//El byte rebut coincideix amb l'adreça i està en mode lectura.
            SSP2CON2bits.ACKEN = 1; //S'habilita l'acknowledge
            lecturaI2C = SSP2BUF;//Es borra el buffer per ser escrit.
            /*Per evitar que es doni overflow, es desactiva el registre BF per
             *indicar que el buffer està preparat per enviar una altra dada.*/
            SSP2STATbits.BF = 0;
            SSP2BUF = escripturaI2C; //S'escriu el byte a enviar en el buffer.
            //Es reincialitza el rellotge.
            SSP2CON1bits.CKP = 1;
            //S'espera fins que s'hagi enviat el byte cap al mestre.
            while(SSP2STATbits.BF);
        }
        SSP2IF = 0; //Es borra el flag d'interrupció.
    }
}
```

Aquesta interrupció es pot produir per un "overflow" o col·lisió en el bus, en aquest cas es llegeixen les dades del buffer per tal de netejar-lo, es netegen les "flags" d'error i es reinicialitza el rellotge.

Per la resta de casos, a cada byte rebut, l'esclau envia l'Acknowledge per l'SDA, posant-lo a nivell baix, i també posa el rellotge a nivell baix per l'SCL. Un cop s'han llegit les dades del buffer, o s'han escrit dades en el buffer, s'habilita el rellotge, es permet la transmissió de dades per l'SDA i es borra el "flag" d'interrupció. En cas de missatge entrant, s'afegeix una estructura "switch case" per tal de tractar la dada rebuda. Les accions que es portin a terme dins d'aquesta estructura han de ser ràpides per tal de no afectar al flux del programa allargant la interrupció. En la taula 3 es poden veure tots els missatges de bus, cada microcontrolador realitza accions concretes segons el codi que rebi.

5.1.4. Mode test

Per tal d'assegurar el correcte funcionament de tots els sistemes del molí així com del bus de comunicacions s'implementa un mode test. Es tracta d'una interrupció temporitzada que envia un missatge des d'un microcontrolador a la resta de microcontroladors. Aquests responen a la petició amb un codi únic, anomenat "heartbeat". Això permet saber que tots els dispositius del sistema es troben en funcionament normal i que el bus I2C no presenta interferències ni mals contactes. "Heartbeat" és un terme utilitzat habitualment en programació de microcontroladors, fa referència a un missatge enviat a un període constant per tal de comprovar l'estat, tant del dispositiu, com del bus de comunicacions.

Aquest mode test s'implementa en els tres microcontroladors per tal d'assegurar que totes les funcions referents a l'I2C (escriptura de mestre, lectura de mestre, escriptura d'esclau i lectura d'esclau) funcionen amb normalitat.

A partir d'una interrupció de temporitzador de 4 segons s'activa el mode test, s'escriu l'activació a la resta de microcontroladors (0xBB) i es llegeix la seva resposta (0xE1, 0xE2 o 0xE3 segons correspongui).

En el següent fragment de codi podem veure l'activació del mode test del programa de l'esclau 2 (monitorització de l'estructura i del vent).

```
void TMR3_compareInterrupt(void){ //4s - MODE TEST I2C
    //Activació del mode test - Sistema de control del pitch.
    I2C_Master_Start();         //Start condition
    I2C_Master_Write(adreca_controlador_pitch_escriptura);
    I2C_Master_Write(codi_start_test);
    I2C_Master_Stop();

    //Activació del mode test - Sistema de control de potència.
    I2C_Master_Start();         //Start condition
    I2C_Master_Write(adreca_controlador_potencia_escriptura);
    I2C_Master_Write(codi_start_test);
    I2C_Master_Stop();

    //Lectura del mode test - Sistema de control del pitch.
    I2C_Master_Start();         //Start condition
    I2C_Master_Write(adreca_controlador_pitch_lectura);
    Test_ESCLAU1 = I2C_Master_Read(0);
    I2C_Master_Stop();

    //Lectura del mode test - Sistema de control de potència.
    I2C_Master_Start();         //Start condition
    I2C_Master_Write(adreca_controlador_potencia_lectura);
```

```
    Test_ESCLAU3 = I2C_Master_Read(0);
    I2C_Master_Stop();

    recepcio_mode_test = true;
}
```

En cas de no rebre resposta, el microcontrolador es quedaria aturat en la funció d'espera de l'I2C, fent saltar el "watchdog" i reiniciant el programa en mode error.

En cas contrari, un cop llegides les dades es comprova que els valors siguin els correctes i es porta un comptatge dels errors de comunicació. En cas de que les dades rebudes difereixin de les esperades durant tres cicles de comptador, 12 segons en total, s'habilita la "flag" d'error del mode test i es procedeix a la maniobra pertinent, veure subapartat 5.1.7. Estat d'error.

En el següent fragment de codi es pot veure la funció de comprovació d'error del mode test.

```
if(recepcio_mode_test){
    if(Test_ESCLAU1 != 0xE1 || Test_ESCLAU3 != 0xE3){
        comptador_test++;
        if (comptador_test == 3){
            comptador_test = 0;
            estat_error = Error_mode_test;
        }
    }else{
        comptador_test = 0;
    }
    recepcio_mode_test = false;
}
```

5.1.5. Mode sleep

El mode sleep o de baix consum permet que els microcontroladors de l'aerogenerador "s'adormin" per tal d'estalviar energia en situacions on el vent no sigui suficient com per generar electricitat.

Es farà un seguiment de l'estat de les condicions ambientals utilitzant tres variables booleanes globals per a determinar si és necessari activar o no la generació. Aquestes variables estaran definides en els tres microcontroladors.

El sistema de control de pitch disposa de l'encoder incremental situat a l'eix del rotor per tal de conèixer la velocitat de rotació de les pales, es desactivarà la seva variable

global quan el pitch es trobi en la posició més perpendicular al vent i es detecti una velocitat de rotació inferior a 12rpm (aquest valor pot ser modificat pel PLC mitjançant la comunicació Modbus, vegeu subapartat 5.2.1.).

El sistema de monitorització utilitza l'anemòmetre per tal d'obtenir dades de la velocitat del vent, si aquesta és inferior al límit establert es desactiva la variable de generació de l'esclau 2.

Per part del sistema de control de potència la seva variable global anirà lligada a un càlcul de potències on es determina si la potència generada pel molí supera les pèrdues elèctriques.

Quan es produeixi un canvi en qualsevol d'aquestes variables globals s'enviarà el corresponent missatge pel bus I2C; 0x12 o 0x13 en el cas de l'esclau 1, 0x22 o 0x23 en el cas de l'esclau 2, 0xF0 o 0xF1 en el cas de l'esclau 3. D'aquesta manera les tres variables presentaran en tot moment els mateixos valors en tots els microcontroladors. En aquest fragment de codi que pertany a l'esclau 2 podem veure la lectura de les variables globals de generació.

```
switch (lecturaI2C){
    case 0x12:
        Generacio_ESCLAU1 = false;
        break;

    case 0x13:
        Generacio_ESCLAU1 = true;
        break;

    case 0xF0:
        Generacio_ESCLAU3 = false;
        break;

    case 0xF1:
        Generacio_ESCLAU3 = true;
        break;
}
```

Finalment quan les tres variables es trobin en estat de no generació s'activarà el mode sleep en els tres PIC, adormint-se fins que torni a detectar-se generació. Durant aquest període es desactiva el "watchdog" per evitar que salti de forma inintencionada.

El sistema es despertarà del mode sleep mitjançant les interrupcions dels temporitzadors 0 o 1, aquests temporitzadors estan lligats a l'adquisició de dades necessària per determinar si torna a detectar-se generació o no. Una interrupció de

comunicació, tant I2C com Modbus, també permet que el microcontrolador es desperti per tal de realitzar les accions pertinents.

D'aquesta manera quan l'aerogenerador es trobi en estat d'estalvi d'energia i algun dels sistemes detecti generació s'enviarà un missatge a la resta de microcontroladors despertant-los i activant el booleà corresponent per tal de que no es tornin a dormir.

La resta de temporitzadors així com dispositius externs (acceleròmetre) s'aturen també durant el mode sleep per tal d'estalviar recursos i evitar que s'executin altres parts del programa que no siguin necessàries. Aquests temporitzadors i dispositius es reactivaran quan el sistema en qüestió torni a detectar generació. En el següent fragment de codi es pot veure la condició que activa l'estat de baix consum.

```
if(!Generacio_ESCLAU1 && !Generacio_ESCLAU2 && !Generacio_ESCLAU3
&& !activar_lectura_anem){
    dormir_accelerometre();
    WDTCON0bits.SEN=0; //Desactivar watchdog perque no salti en sleep
    TMR2_StopTimer();
    TMR3_StopTimer();
    TMR4_StopTimer();
    SLEEP();
}
```

5.1.6. Estat d'emergència

La maniobra d'emergència permet aturar la generació del molí de forma segura i controlada quan algun dels elements de l'aerogenerador es troba en perill. Es porta a terme majoritàriament des del microcontrolador de control de potència ja que aquest té accés al sistema elèctric del molí per tal de curtcircuitar el generador. Tot i així, l'activació d'aquest estat pot donar-se des de qualsevol dels tres sistemes de molí, quan es doni algun dels casos de perill s'envia l'avís a través del bus I2C per tal d'alertar a la resta de microcontroladors.

El sistema de monitorització activa aquesta maniobra quan es superen els límits de vibracions, esforços tensors o velocitat del vent, enviant els missatges 0x2A, 0x2B o 0x2C respectivament. El sistema de control de pitch obté dades de la velocitat de rotació de les pales mitjançant un encoder, quan aquesta velocitat sigui massa elevada com per ser compensada amb el moviment de pitch s'enviarà el missatge 0x1F, informant de que el pitch es troba en la seva posició màxima. Per últim el sistema de

control de potència pot activar aquest estat d'emergència quan la generació sigui excessiva, activant el frenat elèctric al màxim i enviant el missatge 0xA3.

Quan algun d'aquests missatges s'envia al bus I2C es porta a terme l'inici de la maniobra.

El sistema de control de pitch inclina les pales fins la seva posició més paral·lela al vent, si no s'hi troba actualment, ignorant la velocitat de rotació actual del molí. Quan s'assoleixi la posició desitjada s'enviarà el missatge 0x1F com a confirmació.

El sistema de control de potència augmenta la demanda d'energia connectant totes les càrregues disponibles, activant d'aquesta forma el frenat elèctric, en cas de que encara no estigués en funcionament. S'envia també el missatge 0xA3 per confirmar que el frenat s'ha portat a terme.

Un cop han actuat els dos mecanismes de frenat, per tal d'immobilitzar completament la turbina es curtcircuiten els pols de l'alternador i la maniobra es dona per finalitzada.

D'aquesta manera, quan s'ha produït la frenada d'emergència, les aspes de la turbina queden en paral·lel a la direcció del vent, i l'eix de la turbina bloquejat pels efectes electromagnètics del curtcircuit de l'alternador. Per tal de tornar al règim de funcionament normal, l'aerogenerador necessita ser rearmat des del PLC. En cas de que es presentin possibles danys en els relés del generador el rearmament no serà suficient i caldrà portar a terme una revisió del sistema per part d'un tècnic especialitzat.

La lògica que segueix el programa del sistema de control de potència no és objecte d'aquest projecte i per tant s'explica tant sols superficialment com a context general.

La maniobra d'emergència predomina per sobre de qualsevol altre estat de l'aerogenerador ja que es considera que les accions a realitzar son crítiques i no fer-les pot provocar danys greus en la instal·lació.

5.1.7. Estat d'error

La maniobra d'error permet aturar la generació en cas de que alguna part del programa falli. Es pot activar per un dispar del "watchdog", quan el mode test de l'I2C llegeix valors erronis durant 12 segons o quan l'acceleròmetre no s'inicialitza correctament. Comparteix moltes similituds amb l'estat d'emergència amb la diferència

que aquest es porta a terme per tal de que l'aerogenerador sigui revisat i no per evitar que algun element quedi malmès.

En el cas de que el mode test o l'acceleròmetre activin aquest estat indica que s'han de revisar les connexions del bus I2C de tots els dispositius connectats en el bus, en canvi si salta el "watchdog" es revisarà el microcontrolador en qüestió on hagi saltat.

Tal i com passa en l'estat d'emergència, el sistema de control de pitch inclina les pales fins la seva posició més paral·lela al vent, frenant així la rotació de l'aerogenerador. Quan s'assoleix la posició desitjada també s'envia el missatge 0x1F com a confirmació.

El sistema de control de potència actua també de forma similar a com ho feia en l'estat d'emergència, activant el frenat elèctric i confirmant l'acció amb el missatge 0xA3 a través del bus.

L'aerogenerador es queda en aquest estat fins que és revisat per un tècnic i rearmat des del PLC.

Un altre punt a destacar és que com que aquesta maniobra té una prioritat inferior a l'estat d'emergència, els sistemes de l'aerogenerador poden seguir funcionant parcialment per tal de detectar les condicions d'emergència i activar-la si fos necessari.

5.2. RS-485 (Modbus)

Modbus és un protocol de comunicació per xarxes de comunicacions industrials desenvolupat per l'empresa Modicon. És de tipologia mestre/esclau amb una estructura de bus lineal on només existeix un mestre, que controla l'accés al medi i monitoritza el funcionament de la xarxa. La resta de dispositius programables actuen com esclaus, que responen i procedeixen segons la sol·licitud del mestre. El mestre pot adreçar-se a esclaus individualment o pot iniciar un missatge de difusió a tots els esclaus.

Modbus conté dos modes de comunicació sèrie coneguts com a ASCII i RTU, l'aerogenerador utilitzarà el mode RTU, que correspon a la comunicació sèrie asíncrona.

En la figura 41 es pot veure un exemple de transmissió Modbus on es pot veure l'estructura dels missatges.

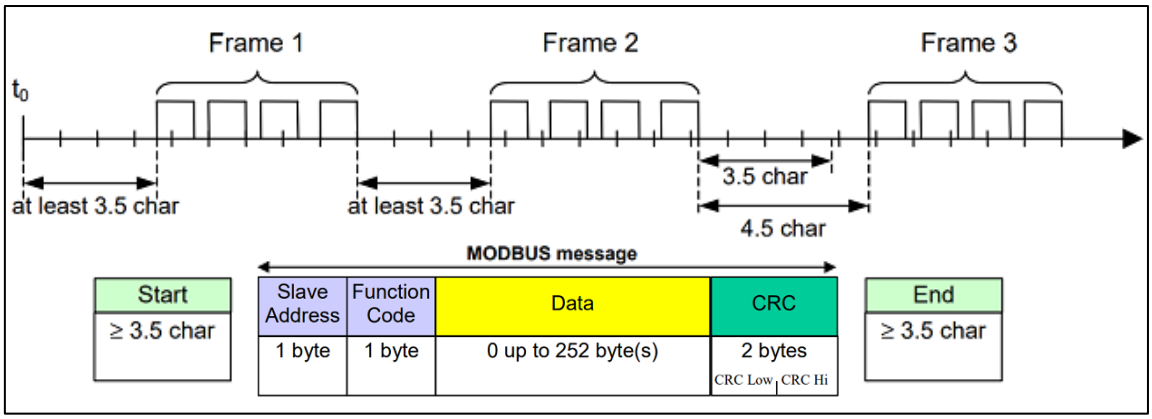

Figura 41: Transmissió Modbus RTU

En mode RTU, els missatges comencen amb un interval de silenci d'almenys 3,5 vegades el temps de caràcter (8 bits) seguit per un "frame" de bytes en format hexadecimal separat per camps.

El primer camp transmès és l'adreça del dispositiu (1 byte). Un cop enviat, cada dispositiu el descodifica per detectar si es tracta d'un missatge dirigit a ell, en cas contrari continua a l'espera del següent missatge. Després de l'adreça s'envien la resta de camps, un codi de funció que defineix l'acció sol·licitada (1 byte), les dades, de 0 fins a 252 bytes, i dos bytes de comprovació d'errors CRC.

Les funcions utilitzades pel sistema de l'aerogenerador son la lectura d'un registre (0x04), i l'escriptura de múltiples registres (0x10). Un registre correspon a dos bytes de memòria del dispositiu. La lectura permet adquirir dos bytes de dades d'un microcontrolador, l'escriptura en canvi envia quatre bytes de dades per tal de sobreescriure les variables corresponents. En els subapartats 5.2.1. i 5.2.2. es poden veure les variables que es guarden a cada registre.

Després de la transmissió de l'últim caràcter, es genera un interval de silenci d'almenys 3,5 vegades el temps de caràcter per marcar el final del missatge. Després d'aquest interval pot començar la transmissió d'un nou missatge. Si no es respecta el temps de silenci mínim, els dispositius no diferenciaran dos missatges rebuts consecutius.

Tota la trama del missatge ha de ser transmesa com a un flux continu, deixant un temps entre bytes no superior a 1,5 vegades el temps de caràcters, com es pot veure en la figura 42.

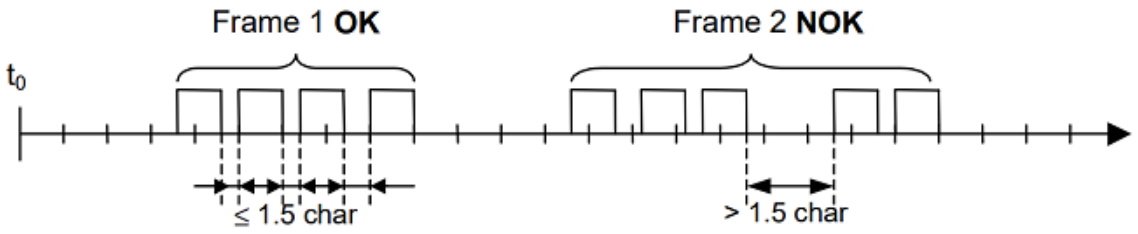

Figura 42: Espai entre bytes Modbus RTU

El Modbus seguirà l'estàndard RS-485 que segueix la norma ANSI/TIA/EIA-485-A-1998. La comunicació realitzada és "Half Dúplex", de forma que el mestre pot enviar o rebre dades però no simultàniament. El cablejat consisteix en un parell de fils de coure trenats sobre els quals es transmet el senyal diferencial per enviar bits de dades. Aquest, és bastant immune a les interferències i admet distàncies llargues quan s'utilitza a velocitat baixa.

En la figura 43 es poden veure les connexions del bus RS-485 necessaries per portar a terme la comunicació Modbus.

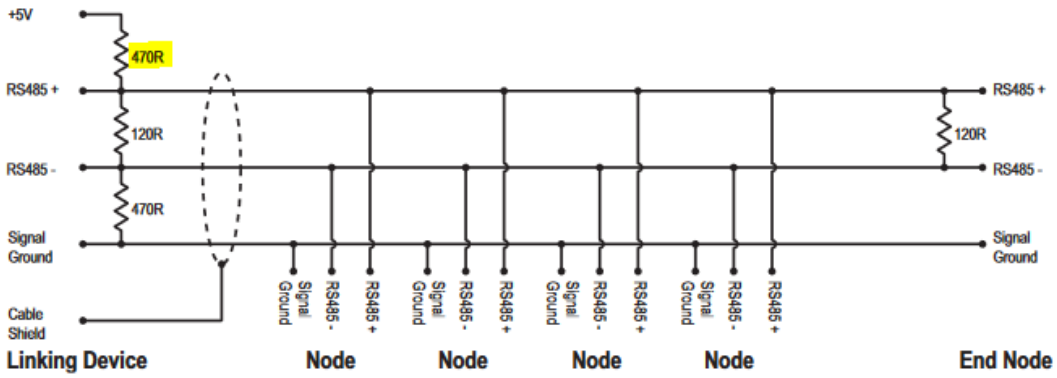

Figura 43: Connexió Modbus (RS485)

El PLC, mestre de la comunicació, enviarà els missatges a través del bus i aquests arribaran als microcontroladors, esclaus, a través d'un mòdul de comunicacions RS-485 a UART, veure figura 44. Aquests mòduls es munten sobre els microcontroladors mitjançant el suport Mikrobus tal i com s'ha vist en l'extensor del bus I2C. En el cas de que els dos suports dels que disposa el PIC estiguin ocupats es realitzen les connexions a través de la PCB i es col·loquen els mòduls sobre regletes.

Figura 44: Clickboard UART/RS-485

Mestre i esclaus es troben situats a una distància propera a la màxima suportada pel bus, per tal d'evitar interferències degudes a la mala qualitat del senyal es vol utilitzar una passarel·la Modbus-ràdio com la de la figura 45.

Figura 45: Passarel·la Modbus/radio

Respecte la comunicació entre el PLC i els PIC18, en aquest projecte s'han integrat en el software dels dos microcontroladors les llibreries i funcions necessàries, adaptant-se a les variables. A nivell intern del programa l'EUSART es configura a 9600 bauds i s'activen les interrupcions per tal de que la transmissió funcioni de manera simultània amb la resta del codi, funcionant de la mateixa manera que la comunicació I2C. A diferència d'aquest, no hi ha un mecanisme de detecció d'adreça i per suplir-ho s'aplica un filtratge als missatges entrants.

En els següents subapartats, en les taules 4, 5, 6 i 7, es mostren les variables de lectura i escriptura del sistema de control de pitch i del sistema de monitorització de l'estructura mecànica i del vent.

Els canvis realitzats en les variables mitjançant l'escriptura per Modbus (exceptuant les variables d'error i d'emergència) son provisionals i poden servir per tal de testejar

optimitzacions o millores del sistema. Un reinici complet del sistema podria retornar les variables als seus valors per defecte, si es volgués que les modificacions fossin permanents caldria actualitzar el programa del microcontrolador en qüestió utilitzant l'entorn de treball MPLABX i una connexió USB. Teòricament aquest reinici complet no s'hauria de produir mai de forma inintencionada, tot i així cal tenir-ho en compte.

5.2.1. Variables del sistema de control de pitch

Variable	Descripció	Unitat
estat_error	Enumeració que indica l'error en que es troba el sistema.	-
estat_emergencia	Variable booleana que indica que el microcontrolador es troba en estat d'emergència.	-
alarma_vel_elevada	Booleà que indica que s'ha superat la velocitat màxima de rotació de l'aerogenerador.	-
Generacio_ESCLAU1	Variable booleana que indica si el microcontrolador detecta generació o no.	-
temperatura	Variable en coma flotant que representa la temperatura detectada pel sensor LM35.	ºC
posicio_cm	Variable en coma flotant que representa la posició actual del pitch.	cm
pos_consigna	Variable en coma flotant que representa la consigna de posició del pitch.	cm
vel_real	Variable en coma flotant que representa la velocitat de moviment actual del pitch.	mm/s
valor_PWM	Enter que representa el valor de "duty cycle" que s'ha enviat per últim cop a l'entrada PWM del relé.	%
velocitat_encoder	Variable en coma flotant que representa la velocitat de rotació actual de les pales de l'aerogenerador.	rpm

Taula 4: Dades de lectura per Modbus del sistema de control de pitch

Aquestes variables poden ser llegides mitjançant el codi 0x04.

Les variables booleanes, les enumeracions i els enters s'uneixen sempre que es pugui en un mateix registre per estalviar espai i millorar l'eficiència de la comunicació. Per

altra banda, cadascuna de les variables en coma flotant s'envia dividida en dos registres diferents, ja que ocupen 4 bytes cadascuna. Posteriorment, aquestes son reconstruïdes en el PLC.

Variable	**Descripció**	**Unitat**	**Valor per defecte**
estat_error	Enumeració que permet rearmar el sistema de control de pitch en cas d'error.	-	0
estat_emergencia	Variable booleana que permet rearmar el sistema quan es troba en emergència.	-	0
constant_temp	Variable en coma flotant utilitzada per linealitzar la resposta del sensor de posició.	cm/ºC	0
enable_PI	Booleà que permet habilitar o inhabilitar l'acció de control PI i el moviment del pitch.	-	1
limit_inferior_encoder	Variable entera que marca el límit inferior de velocitat de rotació de l'aerogenerador.	rpm	12
limit_superior_encoder	Variable entera que marca el límit superior de velocitat de rotació de l'aerogenerador.	rpm	20
vel_baixa	Variable en coma flotant que determina la velocitat baixa de moviment del pitch.	mm/s	18,5
vel_mitja	Variable en coma flotant que determina la velocitat mitja de moviment del pitch.	mm/s	25
vel_alta	Variable en coma flotant que determina la velocitat alta de moviment del pitch.	mm/s	35

Taula 5: Dades d'escriptura per Modbus del sistema de control de pitch, part 1

Variable	Descripció	Unitat	Valor per defecte
kp_vel_baixa	Variable en coma flotant del terme proporcional del controlador a velocitat baixa.	-	9,693100
ti_vel_baixa	Variable en coma flotant del terme integral del controlador a velocitat baixa.	-	0,148394
kp_vel_mitja	Variable en coma flotant del terme proporcional del controlador a velocitat mitja.	-	9,773600
ti_vel_mitja	Variable en coma flotant del terme integral del controlador a velocitat mitja.	-	0,062671
kp_vel_alta	Variable en coma flotant del terme proporcional del controlador a velocitat alta.	-	10,07970
ti_vel_alta	Variable en coma flotant del terme integral del controlador a velocitat alta.	-	0,225096

Taula 6: Dades d'escriptura per Modbus del sistema de control de pitch, part 2

Les variables booleanes, les enteres i les enumeracions s'uneixen per tal d'estalviar espai en els registres i millorar la comunicació. A diferència de l'acció de lectura, l'escriptura actua sobre dos registres (4 bytes) permetent enviar les variables en coma flotant en una sola comunicació.

Les variables estat_error i estat_emergencia poden ser llegides per saber si el microcontrolador es troba en estat d'error o d'emergència, respectivament. En cas afirmatiu també poden ser escrites per tal de retornar-les al seu estat inicial (0), rearmant el sistema.

Amb el valor de la variable alarma_vel_elevada es pot determinar si el sistema ha entrat en mode d'emergència per una comunicació I2C externa (0) o la condició s'ha detectat des del propi microcontrolador (1).

La variable constant_temp permet modificar la linealització del valor de posició, en funció de la temperatura obtinguda a través del sensor LM35. Per defecte aquest valor es troba a 0 ja que es desconeix la variació de precisió respecte la temperatura que presenta el sensor de posició lineal. Mitjançant les dades obtingudes des del PLC es podrà determinar aquesta variació i modificar el valor de la constant en un estudi posterior, quan l'aerogenerador ja es trobi operatiu.

Es pot inhabilitar l'acció del PI mitjançant el booleà enable_PI per tal d'aturar el moviment de les pales a voluntat.

Els límits de l'encoder determinen a partir de quin valor de rotació lineal es considera que el rotor gira massa lent o massa ràpid. Aquests límits afecten a l'activació de l'estat d'emergència i del mode sleep.

Es poden modificar les consignes de velocitat de moviment del pitch així com les variables del terme proporcional i integral de cada velocitat per tal de variar la resposta del pitch.

5.2.2. Variables del sistema de monitorització

Variable	Descripció	Unitat
estat_error	Enumeració que indica l'error en que es troba el sistema.	-
estat_emergencia	Enumeració que indica l'emergència en que es troba el sistema.	-
Generacio_ESCLAU2	Variable booleana que indica si el microcontrolador detecta generació o no.	-
X	Variable en coma flotant que correspon al valor d'acceleració en l'eix X.	g
Y	Variable en coma flotant que correspon al valor d'acceleració en l'eix Y.	g
Z	Variable en coma flotant que correspon al valor d'acceleració en l'eix Z.	g
amp_galgues_estructura	Variable en coma flotant, representa el valor de tensió de l'amplificador del pont de galgues situat en l'estructura.	mV

Taula 7: Dades de lectura per Modbus del sistema de monitorització, part 1

Variable	Descripció	Unitat
amp_galgues_cable1	Variable en coma flotant, representa el valor de tensió de l'amplificador del pont de galgues situat en el primer cable tensor.	mV
amp_galgues_cable2	Variable en coma flotant, representa el valor de tensió de l'amplificador del pont de galgues situat en el segon cable tensor.	mV
amp_galgues_cable3	Variable en coma flotant, representa el valor de tensió de l'amplificador del pont de galgues situat en el tercer cable tensor.	mV
vel_anemometre	Variable en coma flotant que correspon a la velocitat del vent detectada per l'anemòmetre.	m/s
dir_anemometre	Variable en coma flotant que correspon a la direcció del vent detectada per l'anemòmetre.	º

Taula 8: Dades de lectura per Modbus del sistema de monitorització, part 2

De la mateixa manera que en la comunicació del sistema de control de pitch, les variables booleanes i les enumeracions s'uneixen sempre que es pugui en un mateix registre per estalviar espai i millorar l'eficiència de la comunicació.

Les variables en coma flotant s'envien dividides en dos registres diferents, ja que ocupen 4 bytes cadascuna. Posteriorment, aquestes son reconstruïdes en el PLC.

Variable	Descripció	Unitat	Valor per defecte
estat_error	Enumeració que permet rearmar el sistema en cas d'error.	-	0
estat_emergencia	Enumeració que permet rearmar el sistema en cas d'emergència.	-	0
Accelerometre_reinit	Booleà que habilita la reinicialització de l'acceleròmtre.	-	0

Taula 9: Dades d'escriptura per Modbus del sistema de monitorització, part 1

Variable	Descripció	Unitat	Valor per defecte
escala_max	Enter que indica el valor d'escala màxim de les dades d'acceleració obtingudes.	-	0 (±2g)
filtre_pas_alt	Booleà que activa o desactiva el filtre pas alt incorporat en el propi acceleròmetre.	-	0 (desactivat)
acceleracio_max	Variable en coma flotant, marca el nivell d'acceleració màxim que activa la interrupció.	g	2
Offset_X	Variable entera que representa l'offset de l'eix de coordenades X.	mg	0
Offset_Y	Variable entera que representa l'offset de l'eix de coordenades Y.	mg	0
Offset_Z	Variable entera que representa l'offset de l'eix de coordenades Z.	mg	0
valor_max_galgues_estructura	Enter que marca el límit de tensió a partir del qual es considera esforç elevat en l'estructura.	mV	2600
valor_max_galgues_cable	Enter que marca el límit de tensió a partir del qual es considera esforç elevat en els cables.	mV	2800

Taula 10: Dades d'escriptura per Modbus del sistema de monitorització, part 2

Variable	Descripció	Unitat	Valor per defecte
limit_baix_anemometre	Variable entera que marca el límit inferior de velocitat del vent.	m/s	5
limit_alt_anemometre	Variable entera que marca el límit superior de velocitat del vent.	m/s	20

Taula 11: Dades d'escriptura per Modbus del sistema de monitorització, part 3

Com s'ha vist en la comunicació del control de pitch, les variables booleanes, les enteres i les enumeracions s'uneixen per tal d'estalviar espai en els registres i millorar la comunicació.

A diferència de l'acció de lectura, l'escriptura actua sobre dos registres (4 bytes) permetent enviar les variables en coma flotant en una sola comunicació.

Igual que en el sistema de control de pitch, les variables estat_error i estat_emergencia poden ser llegides per saber si el microcontrolador es troba en estat d'error o d'emergència, respectivament. En cas afirmatiu poden ser escrites per tal de retornar-les al seu estat inicial (0), rearmant el sistema.

La unitat g fa referència a l'acceleració produïda per la força de la gravetat, el seu valor correspon a 9,807 m/s^2.

La configuració de l'acceleròmetre es pot modificar enviant ordres d'escriptura a través del Modbus. Es pot seleccionar el valor d'escala màxim d'entre 3 valors predefinits, un 0 correspon a ±2g, un 1 correspon a ±4g i un 2 correspon a ±8g. Cal tenir en compte que com més elevat sigui el valor d'escala, menys resolució tenen les dades d'acceleració obtingudes. Es pot activar o desactivar el filtratge de senyals que incorpora el propi acceleròmetre mitjançant la variable filtre_pas_alt. El nivell d'acceleració màxim que activa l'estat d'emergència pot tenir qualsevol valor entre 0g i 8g, s'envia en format coma flotant. De la mateixa manera es poden modificar els valors d'offset de cadascun dels eixos de coordenades dins del rang ±255mg, aquest s'envia en format enter de 16 bits amb signe (int16_t).

Quan es modifiqui qualsevol dels valors de configuració de l'acceleròmetre s'haurà d'habilitar la variable Accelerometre_reinit, inicialitzant de nou l'accelerometre amb els

canvis aplicats. Fins que no es realitzi aquesta reinicialització els valors obtinguts de l'acceleròmetre correspondran a la última configuració aplicada.

Mitjançant l'escriptura per Modbus es poden modificar els límits dels valors obtinguts dels esforços tant del cables tensors com de l'estructura, afectant directament a l'activació de l'estat d'emergència. Cal tenir present que el valor màxim de tensió del pont de galgues de l'estructura ha de ser inferior al valor màxim de tensió dels ponts de galgues dels cables ja que l'estructura principal presenta una menor deformació enfront als esforços del vent.

Es poden retocar també els límits de l'anemòmetre, que determinen els valors mínim i màxim de la velocitat del vent. Aquests límits afecten a l'activació del mode sleep i de l'estat d'emergència respectivament.

6. PROVES EXPERIMENTALS

Per tal de comprovar el correcte funcionament dels sistemes creats, s'han realitzat experiments de certes parts dels programes. Concretament s'han realitzat proves del sistema de control de pitch, de l'adquisició i tractament de dades de l'acceleròmetre i finalment de les comunicacions del sistema (entre microcontroladors amb el bus I2C i entre microcontrolador i PLC utilitzant el Modbus).

En l'annex C es poden veure els procediments seguits a cada experiment i els resultats obtinguts amb les seves corresponents conclusions.

Amb tots aquests sistemes funcionant correctament es redueix la càrrega de treball de la posada en marxa, on es provarà el sistema complet i s'ajustaran els paràmetres necessaris.

7. RESUM DEL PRESSUPOST

El conjunt de totes les parts que componen el pressupost tenen un cost total de vint-i-tres mil vuit-cents euros amb quinze cèntims, sense IVA.

8. CONCLUSIONS

Una vegada dissenyats i creats el control de pitch i la monitorització de l'estructura mecànica i del vent en el present document, es confirma que funcionen correctament amb el hardware i software implementat.

En conseqüència s'han assolit els objectius que es perseguien inicialment. S'ha establert un sistema en l'aerogenerador per tal de modificar l'angle de pitch de manera fiable. S'ha aconseguit monitoritzar la informació dels sensors, obtenint dades de l'entorn de l'aerogenerador i de la pròpia estructura. També s'ha creat el sistema de comunicació I2C entre els microcontroladors i de Modbus RTU amb el PLC. Cal destacar el sistema multitasca implementat en els microcontroladors que permet realitzar totes les accions assignades a partir d'interrupcions. També s'han afegit sistemes de seguretat amb autodiagnòstics (mode test, estat d'emergència i estat d'error) i un sistema d'estalvi d'energia (mode sleep), millorant així el funcionament global de l'aerogenerador i la seva eficiència. Finalment cal esmentar que el treball portat a terme deixa pràcticament finalitzada la seva part corresponent del projecte Llavor.

Les PCBs es troben en producció i la posada en marxa de l'aerogenerador es realitzarà durant els pròxims mesos, en ella s'instal·laran els dispositius i es modificaran els valors d'alguns paràmetres si fos necessari. El sistema es deixa obert a futures optimitzacions o millores que es puguin portar a terme.

POL CARRERAS VIÑAS - DNI 41576344A

Firmado digitalmente por POL CARRERAS VIÑAS - DNI 41576344A
Fecha: 2023.06.02 10:24:48 +02'00'

L'enginyer elèctric, Pol Carreras i Viñas.

Graduat en Enginyeria Electrònica Industrial i Automàtica.

Girona, 2 de juny de 2023.

9. RELACIÓ DE DOCUMENTS

El projecte en qüestió està format pel conjunt de cinc documents: Memòria, Plànols, Plec de condicions, Estat d'amidaments i Pressupost.

10. BIBLIOGRAFIA

Afzal, S. I^2C Primer: What is I^2C? (https://www.analog.com/en/technical-articles/i2c-primer-what-is-i2c-part-1.html , 5 de desembre 2022)

Castaño, S.A. Control PID con microcontrolador PIC. (https://controlautomaticoeducacion.com/microcontroladores-pic/control-pid/ , 10 de desembre 2022)

Castaño, S.A. Controladores PID Discreto. (https://controlautomaticoeducacion.com/control-realimentado/controladores-pid-discreto/ , 3 d'abril de 2023)

Gil, L.A. & Rincón, J.L. CONTROL PID PARA EL CONTROL DE VELOCIDAD DE UN MOTOR DC. (https://repositorio.utp.edu.co/server/api/core/bitstreams/1a4fb4e4-bc53-4e89-ab3b-cd028b2e8da6/content , 19 de març de 2023)

Guemisa. ENCODER INCREMENTAL. (https://www.guemisa.com/sicod/docus/ENCODER-TEC.pdf , 18 de febrer de 2023)

Hoffmann, K. Applying the Wheatstone Bridge Circuit. (https://www.hbm.com/fileadmin/mediapool/hbmdoc/technical/T01569.pdf , 15 de maig 2023)

Ljung, L. System Identification Toolbox™ User's Guide. (https://es.mathworks.com/help/pdf_doc/ident/ident_ug.pdf , 5 de maig 2023)

Matlab. Control System Toolbox™ User's Guide. (https://es.mathworks.com/help/pdf_doc/control/control_ug.pdf , 5 de maig 2023)

Microchip. PIC18F27/47Q10 Datasheet. (https://ww1.microchip.com/downloads/aemDocuments/documents/MCU08/ProductDocuments/DataSheets/PIC18F27-47Q10-Data-Sheet-40002043E.pdf , 5 de desembre 2022)

Nelson, V.P. C programming for embedded microcontroller systems. (https://www.eng.auburn.edu/~nelson/courses/elec3040_3050/C%20programming%20for%20embedded%20system%20applications.pdf , 5 de desembre de 2022)

Ozeki LTD. Modbus RTU. (https://ozeki.hu/p_5854-modbus-rtu.html , 5 de desembre 2022)

Valenti, C. Implementing a PID Controller Using a PIC18 MCU. (https://ww1.microchip.com/downloads/en/Appnotes/00937a.pdf , 10 de desembre de 2022)

11. GLOSSARI

ADC: Analog-to-Digital Converter.

CCP: Capture/Compare/PWM.

CLC: Configurable Logic Cell.

CMP: Comparator.

CMR: Common Mode Rejection.

CPU: Central Processing Unit

CRC: Cyclic Redundancy Check.

CVD: Capacitive Voltage Divider.

CWG: Complementary Waveform Generator.

DSM: Data Signal Modulator.

EUSART: Enhanced Universal Synchronous Asynchronous Receiver Transmitter.

FIFO: First In First Out.

FVR: Fixed Voltage Reference.

GPIO: General Purpose Input Output.

HLT: Hardware Limit Timer.

HPC: High Pin Count.

ICD: In-Circuit Debug.

ICSP: In-Circuit Serial Programming.

IDE: Integrated Development Environment.

I2C: Inter-Integrated Circuit.

LIN: Local Interconnect Network.

MCC: MPLAB Code Configurator.

MOSFET: Metal Oxide Semiconductor Field-Effect Transistor.

MSSP: Master Synchronous Serial Port.

ODR: Output Data Rate.

PCB: Printed Circuit Board.

PI: Proporcional Integrador.

PLC: Programmable Logic Controller.

PuTTY: Port Unique TeleTYpe.

PWM: Pulse-Width Modulators.

RISC: Reduced Instruction Set Computer.

RTU: Remote Terminal Unit.

SCL: Serial Clock.

SDA: Serial Data.

SPI: Serial Peripheral Interface.

UART: Universal Asynchronous Receiver Transmitter.

ZCD: Zero-Cross Detect.

A. PROGRAMARI CONTROL DE PITCH

En aquest annex es mostren i es comenten les configuracions de MCC utilitzades així com els fitxers de programació .c i .h que conté el programa del microcontrolador.

A.1. Configuració MCC

En la configuració de l'MCC s'ha d'habilitar el mòdul PWM3 i associar-lo a un temporitzador, en aquest cas al TMR2, habilitar la interrupció del temporitzador 0 (TMR0) i configurar-la a 1s, habilitar el temporitzador 1 (TMR1) i el seu pin corresponent per fer la lectura dels polsos de l'encoder, habilitar la interrupció del temporitzador 3 (TMR3) i configurar-lo a 100ms, habilitar la interrupció del temporitzador 4 (TMR4) i configurar-lo a 4s, habilitar l'ADC, habilitar els mòduls MSSP i EUSART per les comunicacions i per últim, configurar els pins per tal de que coincideixin amb les connexions fetes a la PCB.

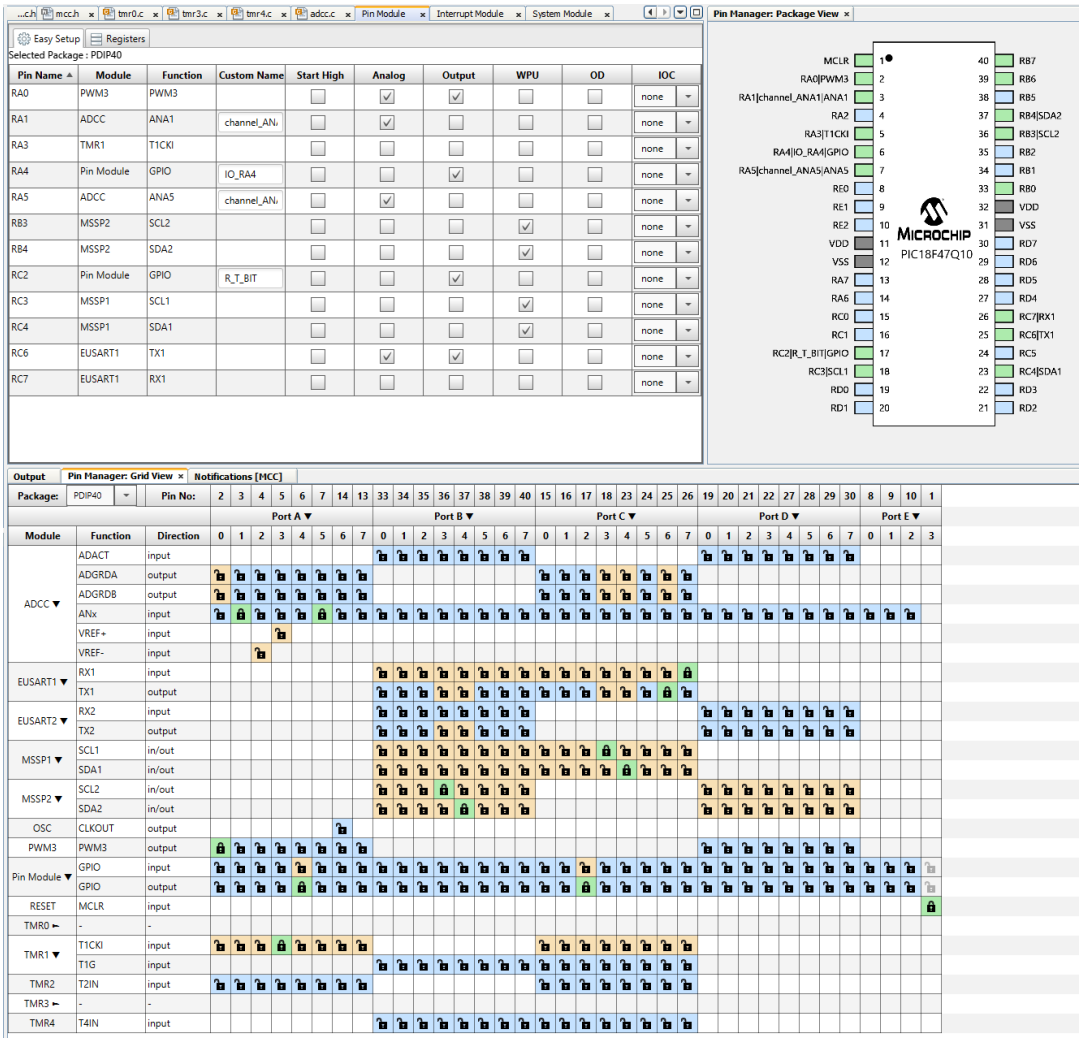

Figura 46: Configuració dels pins del control de pitch

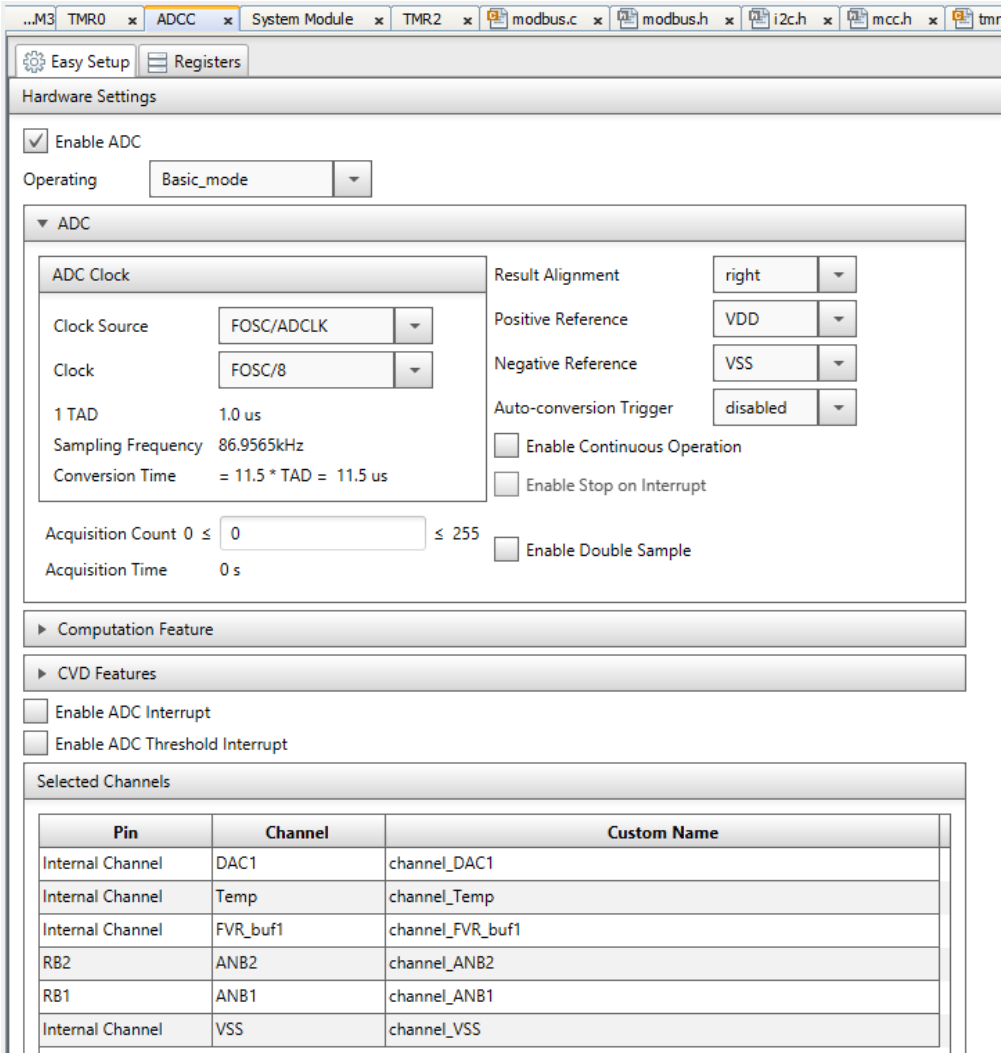

Figura 47: Configuració de l'ADC del control de pitch

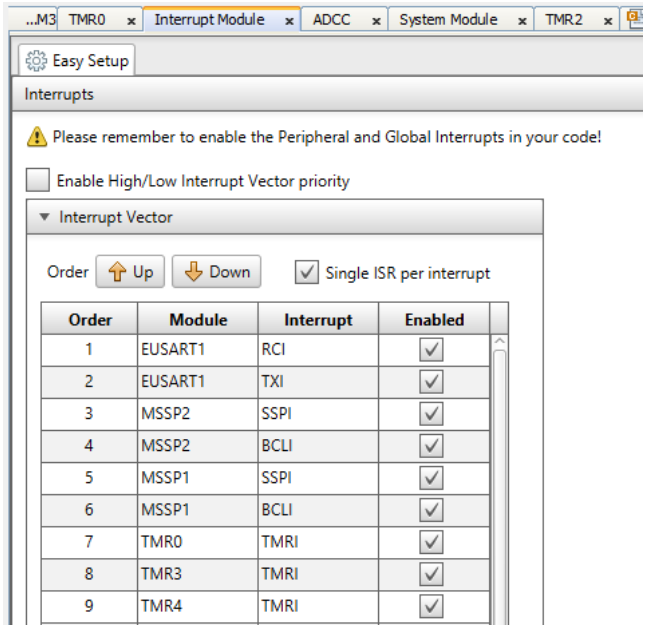

Figura 48: Configuració del vector d'interrupció del control de pitch

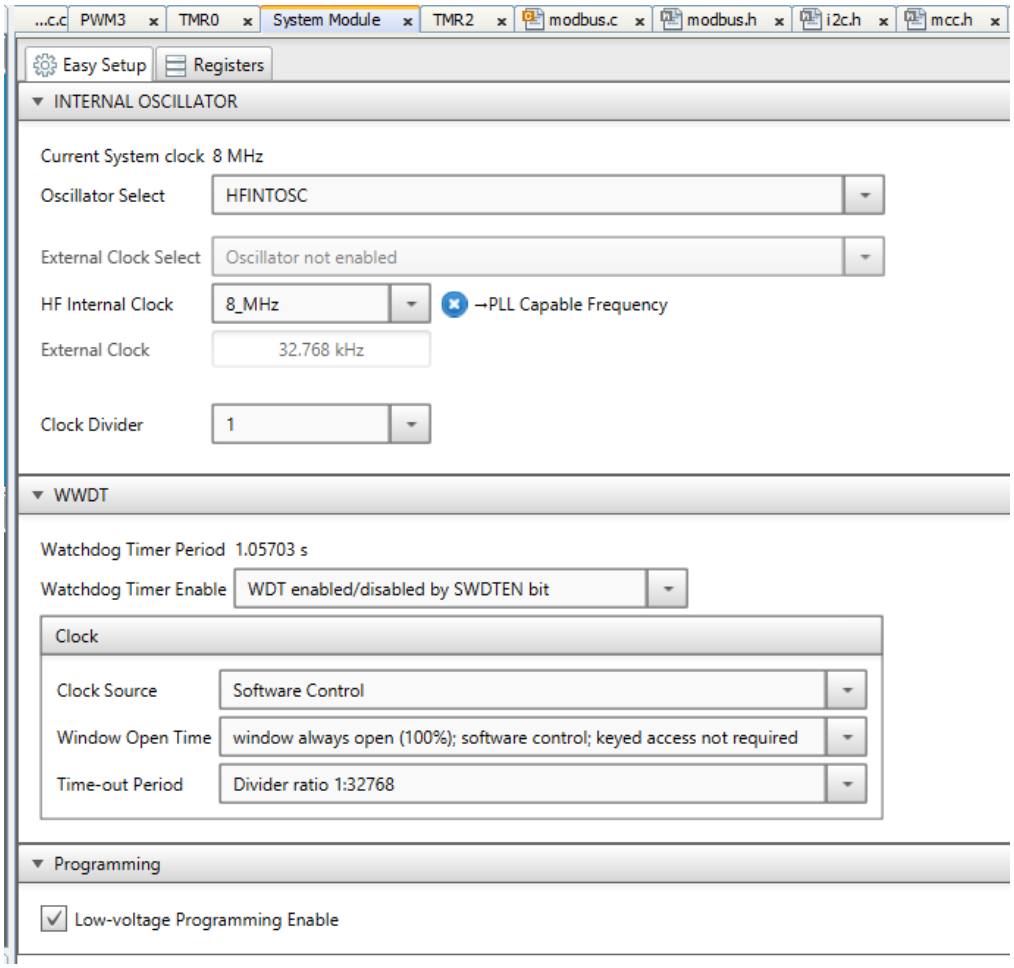

Figura 49: Configuració del mòdul de sistema del control de pitch

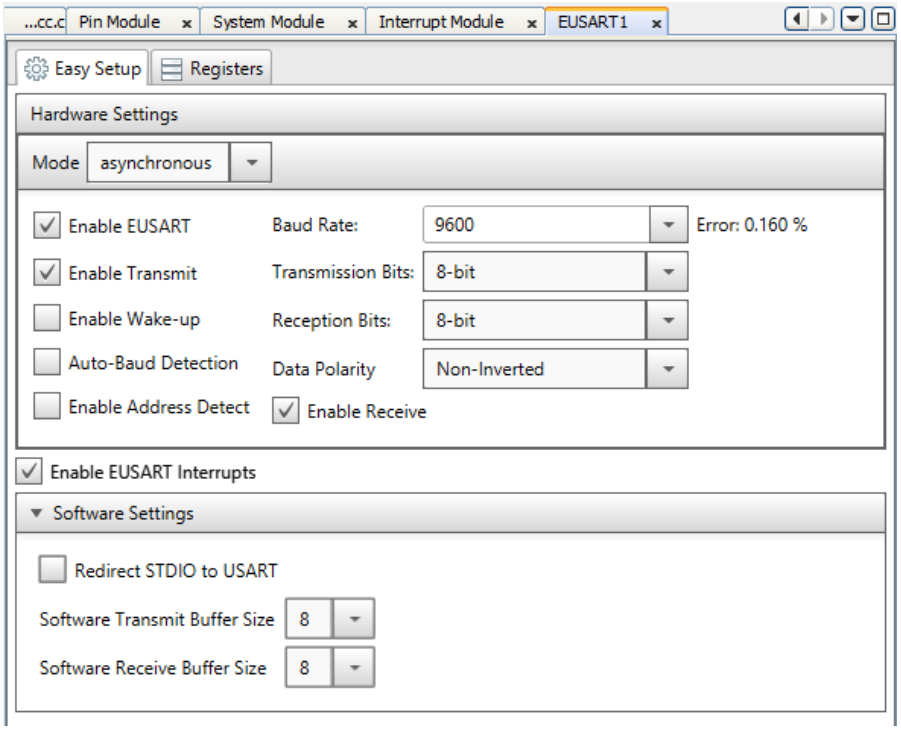

Figura 50: Configuració de l'EUSART1 del control de pitch

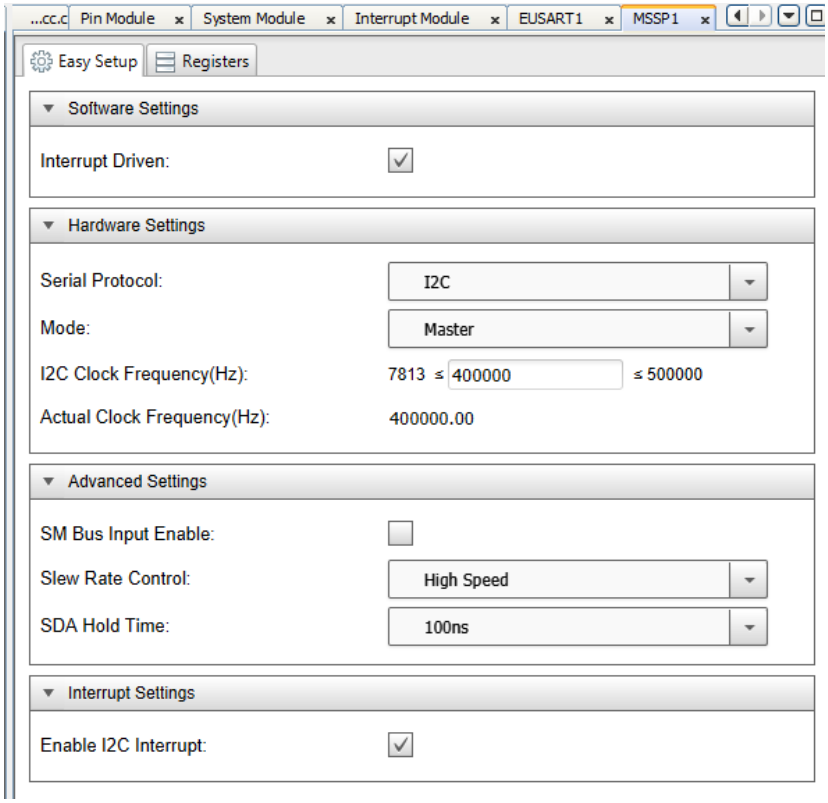

Figura 51: Configuració del MSSP1 del control de pitch

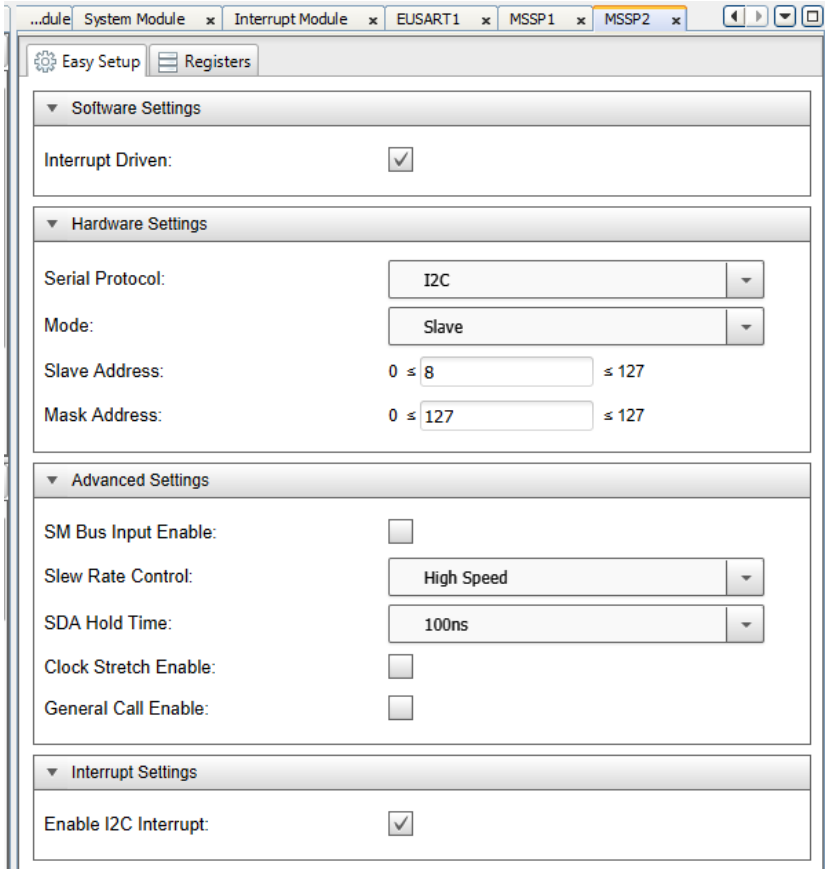

Figura 52: Configuració del MSSP2 del control de pitch

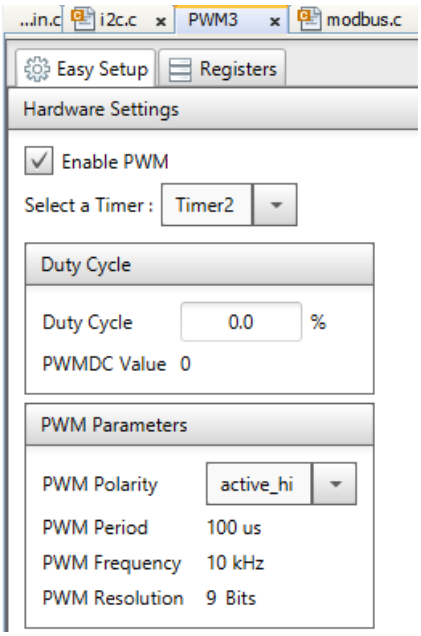

Figura 53: Configuració del PWM3 del control de pitch

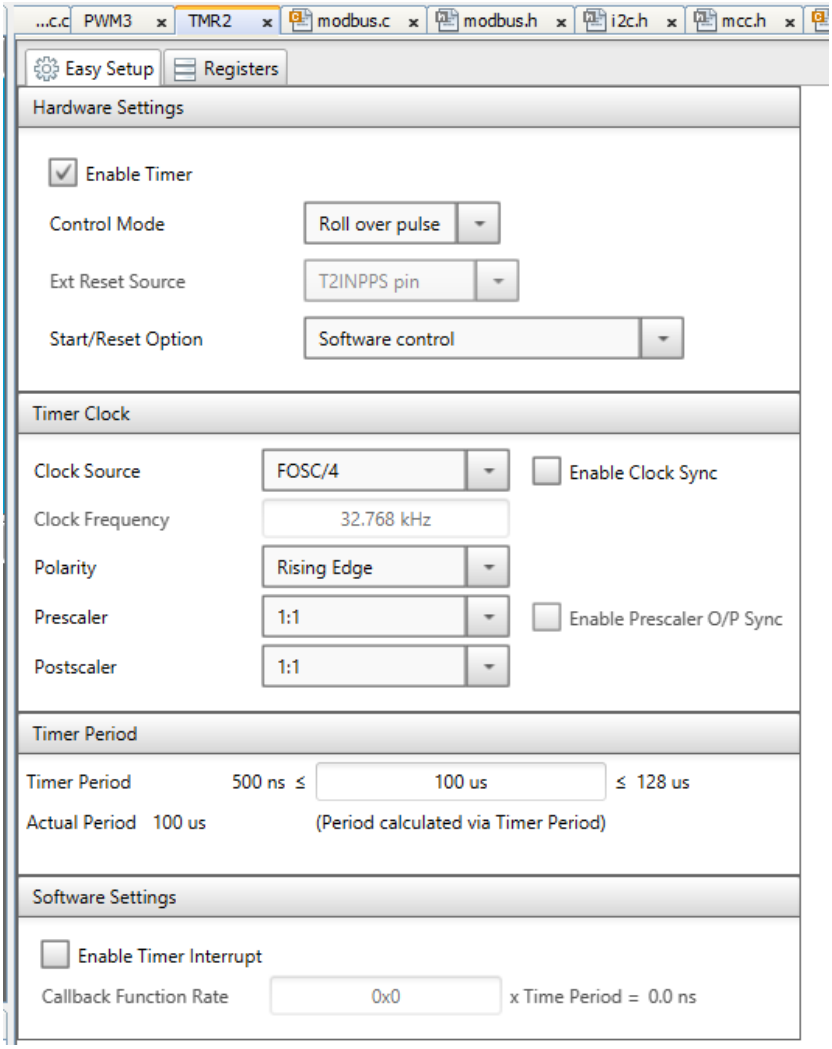

Figura 54: Configuració del TMR2 del control de pitch

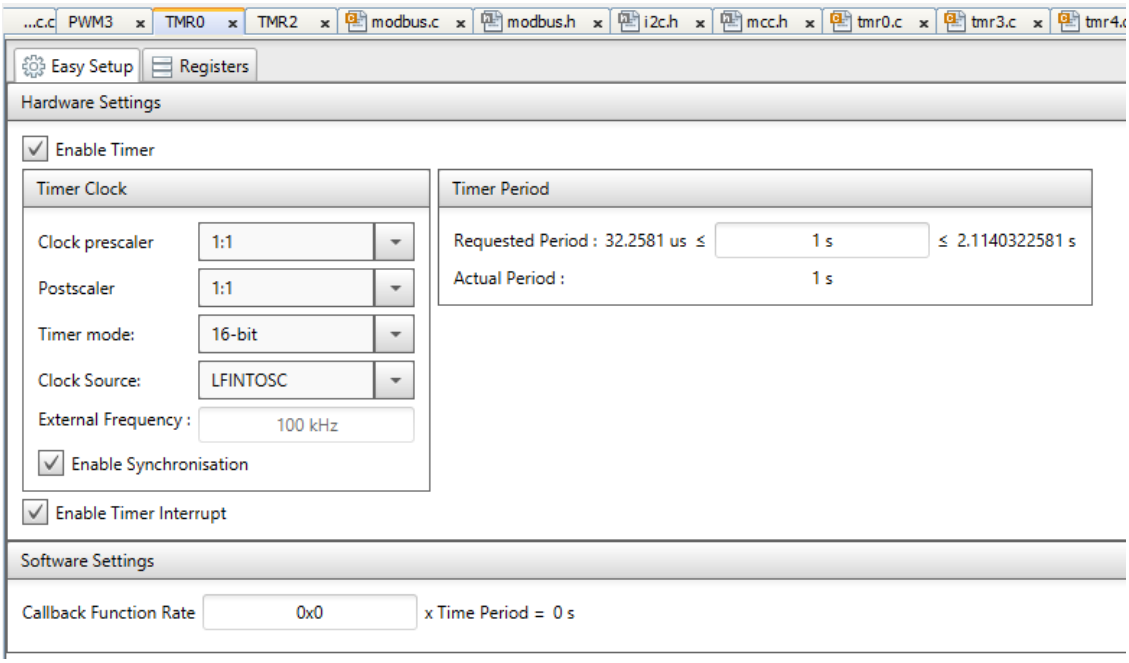

Figura 55: Configuració del TMR0 del control de pitch

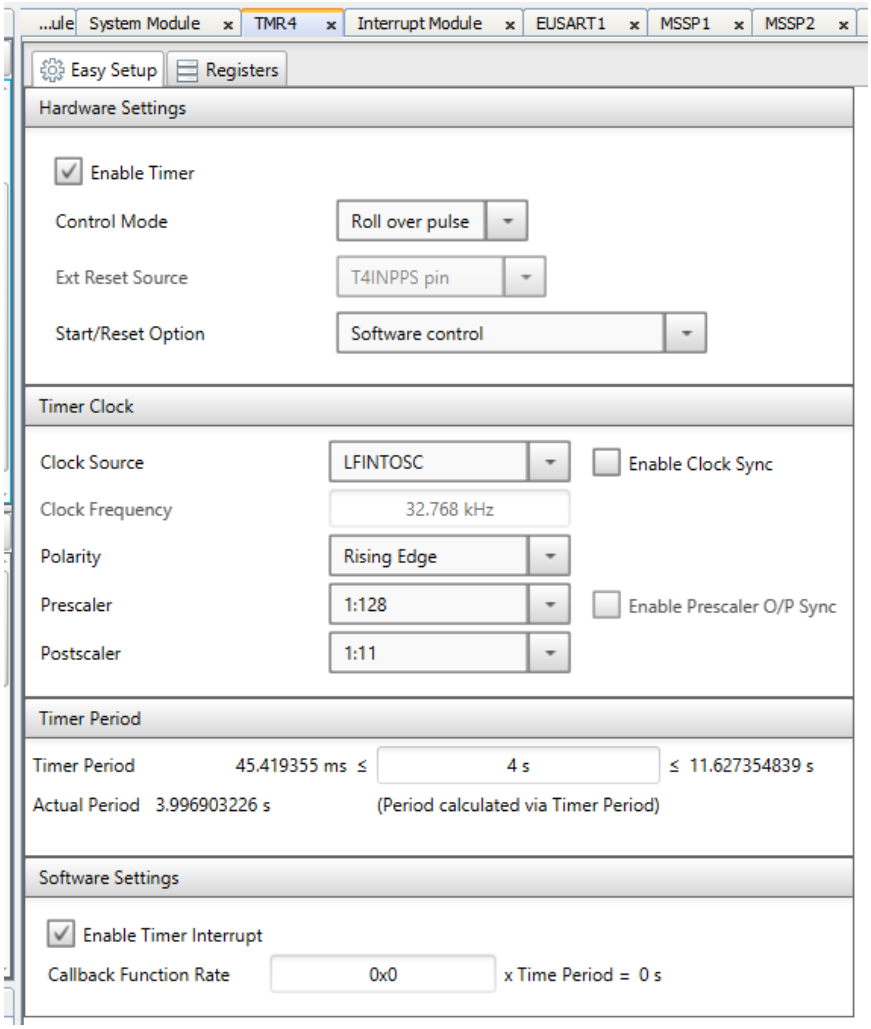

Figura 56: Configuració del TMR4 del control de pitch

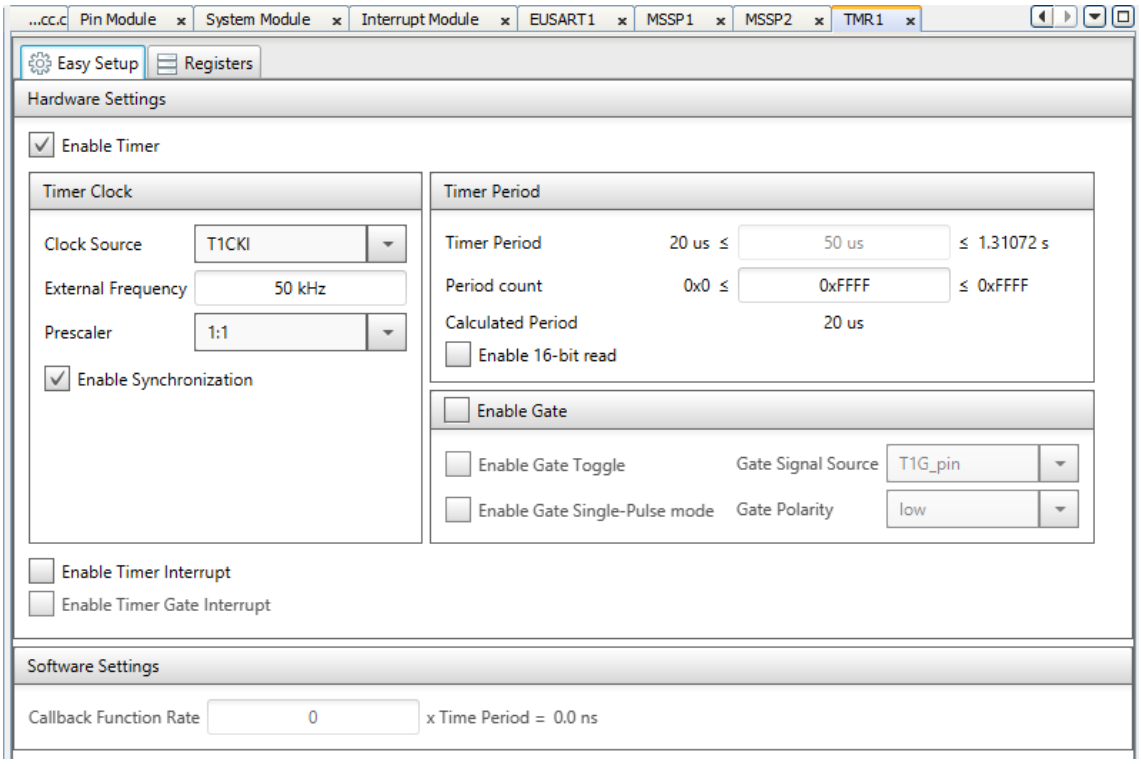

Figura 57: Configuració del TMR1 del control de pitch

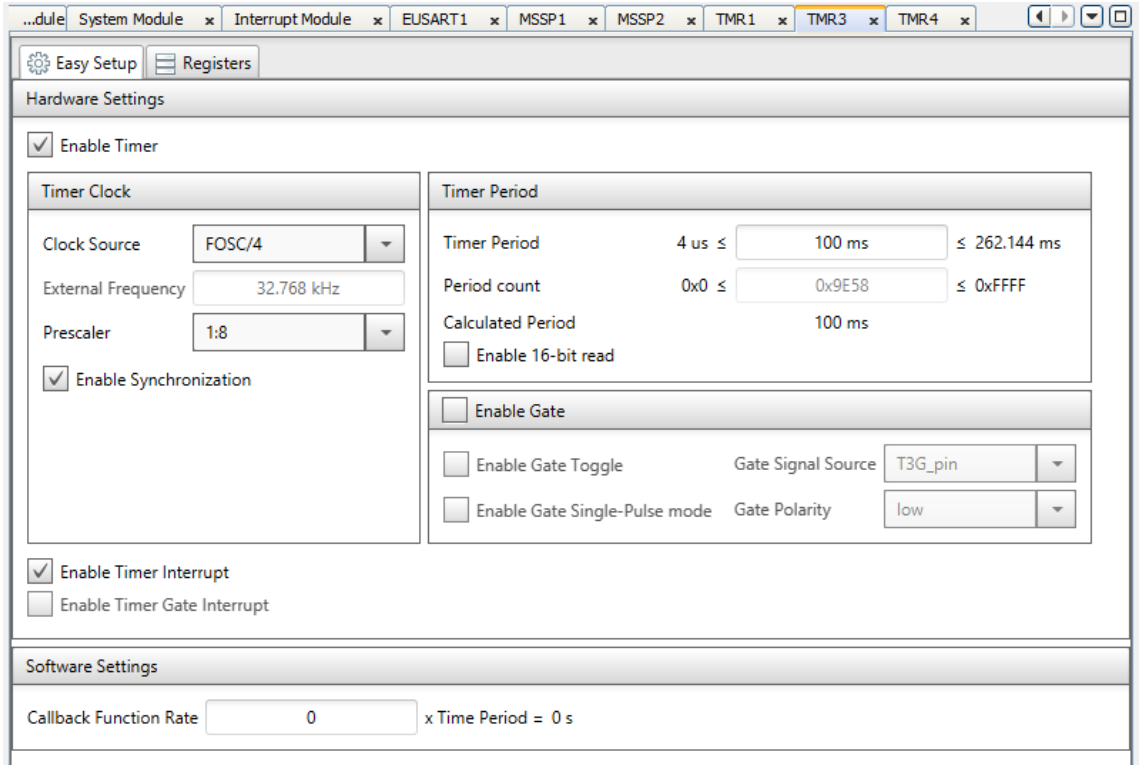

Figura 58: Configuració del TMR3 del control de pitch

Un cop generada la configuració els arxius creats s'agrupen dins la carpeta mcc_generated_files.

A.2. Arxiu main.c

```
/*******************************************************************************
* Projecte              :   Control de pitch d'un aerogenerador
* Nom de l'arxiu        :   main.c
* Autor                 :   PC
* Data inicial          :   01/05/2023
* Versió                :   1.0.0
* Compilador            :   Microchip XC8
* Microprocessador      :   PIC18F47Q10
* Notes                 :   Cap
*******************************************************************************/

/*************** MODULE REVISION LOG ***************************************
*
*    Data    Versió de software  Inicials   Descripció
*  01/05/2023    1.0.0           PC         Creació del main
*
*******************************************************************************/
/** \file main.c
 * \brief Aquest programa realitza les funcions de moviment del pitch
 *  de l'aerogenerador del projecte llavor - UdG
 */



/*******************************************************************************
 * Includes
*******************************************************************************/

#include "mcc_generated_files/mcc.h"
#include "i2c.h"
#include "modbus.h"

// PIC18F47Q10 Configuration Bit Settings
// 'C' source line config statements
#include <xc.h>
#include <pic18f47q10.h>
#include <stdio.h>


/*******************************************************************************
* Defines
*******************************************************************************/

#define ADRECA_CONTROLADOR_MONITORITZACIO 0x10
#define ADRECA_CONTROLADOR_POTENCIA 0x28
```

```
/*****************************************************************************
* Global variables
*****************************************************************************/
//Variables MODBUS
uint8_t c_control=0;
uint8_t RxBuffer[20];
uint8_t rcv;

//Variables estat d'error
enum ESTAT_ERROR_t estat_error = 0;
const uint8_t codi_error = 0x1E;

//Variables estat emergència
const uint8_t codi_posicio_maxima = 0x1F;
bool estat_emergencia;
bool posicio_max;
bool alarma_vel_elevada;

// CONFIG I2C
const       uint8_t       adreca_controlador_monitoritzacio_escriptura       =
ADRECA_CONTROLADOR_MONITORITZACIO << 1;
const        uint8_t        adreca_controlador_monitoritzacio_lectura        =
(ADRECA_CONTROLADOR_MONITORITZACIO << 1) | 0x01;
const         uint8_t         adreca_controlador_potencia_escriptura         =
ADRECA_CONTROLADOR_POTENCIA << 1;
const          uint8_t          adreca_controlador_potencia_lectura          =
(ADRECA_CONTROLADOR_POTENCIA << 1) | 0x01;
uint8_t lecturaI2C;
uint8_t escripturaI2C;

//Variables mode test
const uint8_t codi_start_test = 0xBB;
uint8_t Test_ESCLAU2=0xE2;
uint8_t Test_ESCLAU3=0xE3;
_Bool recepcio_mode_test;
uint8_t comptador_test;

//Variables mode sleep
_Bool Generacio_ESCLAU1;
_Bool Generacio_ESCLAU2;
_Bool Generacio_ESCLAU3;
const uint8_t codi_base_generacio = 0x12;

//Variables sensor de posició
float temperatura;
uint16_t posicio_raw, volt_lm35;
float pos_consigna, posicio_cm, posicio_cm1;
float constant_temp;
_Bool do_adquirir_valors;
```

```
//Variables encoder
float velocitat_encoder = 0;
uint8_t p_llistat_posicions;
_Bool do_canvi_consigna;
uint8_t limit_inferior_encoder = 12; //rpm
uint8_t limit_superior_encoder = 20; //rpm

//Variables PI
float Cn, Cn_1;
float error_p;
float error_v, error_v1;
float vel_consigna, vel_real;
float ti, kp;
float kp_vel_baixa = 1.9154;
float ti_vel_baixa = 0.1656; //Ti = Kp/Ki
float kp_vel_mitja = 0.9992;
float ti_vel_mitja = 0.10186; //Ti = Kp/Ki
float kp_vel_alta = 1.6919;
float ti_vel_alta = 0.1476; //Ti = Kp/Ki
float q0, q1;
const float T = 0.1;
const float limit_baix = 1, limit_alt = 2, zona_morta = 0.5;
float vel_baixa = 18.5, vel_mitja = 25, vel_alta = 35;
_Bool do_PI;
_Bool enable_PI;

//Variables PWM
uint8_t valor_PWM;



/*****************************************************************************
* Functions
*****************************************************************************/

void DADES_REBUDES_EUSART(void){
    R_T_BIT_PORT = 0; //Deshabilita la direcció de transmissió del PIC
    rcv = EUSART1_Read();//Escriu la variable de recive.
    //Emmagatzema les dades en el buffer.

    switch (c_control){
        case 0:
            if(rcv == SlaveAddress) {
                //ADREÇA
                RxBuffer[c_control] = rcv;
                c_control++;
            }
            break;
        case 1:
```

```
        if(rcv==0x04||rcv==0x10){
            //FUNCIÓ
            RxBuffer[c_control] = rcv;
            c_control++;
        }else c_control=0;
        break;
    case 2:
        RxBuffer[c_control] = rcv;
        c_control++;
        break;
    case 3:
        RxBuffer[c_control] = rcv;
        c_control++;
        break;
    case 4:
        RxBuffer[c_control] = rcv;
        c_control++;
        break;
    case 5:
        RxBuffer[c_control] = rcv;
        c_control++;
        break;
    case 6:
        RxBuffer[c_control] = rcv;
        c_control++;
        break;
    case 7:
        RxBuffer[c_control] = rcv;
        c_control++;
        if(RxBuffer[1]==0x04){
            modbus_function();
        }
        break;
    case 8:
        RxBuffer[c_control] = rcv;
        c_control++;
        break;
    case 9:
        RxBuffer[c_control] = rcv;
        c_control++;
        break;
    case 10:
        RxBuffer[c_control] = rcv;
        c_control++;
        break;
    case 11:
        RxBuffer[c_control] = rcv;
        c_control++;
        break;
```

```
        case 12:
            RxBuffer[c_control] = rcv;
            c_control++;
            break;
        case 13:
            RxBuffer[c_control] = rcv;
            c_control++;
            break;
        case 14:
            RxBuffer[c_control] = rcv;
            c_control++;
            break;
        case 15:
            RxBuffer[c_control] = rcv;
            modbus_function();
            break;
    }
}

void I2C_Slave_Read(void) {
    if(SSP2IF==1){

        //De la manera com marca el protocol l'esclau posa a zero el clock.
        SSP2CON1bits.CKP = 0;
         //El motiu és per garantir el temps de configuració de les dades.
        if ((SSP2CON1bits.SSPOV) || (SSP2CON1bits.WCOL))
        { //Overflow o col·lisió
            lecturaI2C = SSP2BUF; //Es borra el contingut del buffer.
            SSP2CON1bits.SSPOV = 0; // Es borra el bit d'OVERFLOW
            SSP2CON1bits.WCOL = 0;  // Es borra el bit de col·lisió.
            SSP2CON1bits.CKP = 1; //Es reinicialitza el rellotge.
        }

        if(!SSP2STATbits.D_nA && !SSP2STATbits.R_nW)
        { //El primer byte rebut coincideix amb l'adreça i està en mode
escriptura.
            SSP2CON2bits.ACKEN = 1; //S'habilita l'acknowledge
            lecturaI2C = SSP2BUF; //Es llegeix el contingut del buffer.
            SSP2CON1bits.CKP = 1; //Es reinicialitza el rellotge.
            switch (lecturaI2C){
                case 0x22:
                    Generacio_ESCLAU2 = false;
                    break;

                case 0x23:
                    Generacio_ESCLAU2 = true;
                    break;

                case 0x2A:
```

```
                case 0x2B:
                case 0x2C:
                case 0xA3:
                    estat_emergencia = true;
                    break;

                case 0x2E:
                case 0x3E:
                    estat_error = Error_extern;
                    break;

                case 0xBB:
                    escripturaI2C = 0xE1;
                    break;

                case 0xF0:
                    Generacio_ESCLAU3 = false;
                    break;

                case 0xF1:
                    Generacio_ESCLAU3 = true;
                    break;

                default:
                    break;

            }
        }
        else if(!SSP2STATbits.D_nA && SSP2STATbits.R_nW)
        {//El primer byte rebut coincideix amb l'adreça i està en mode
lectura.
            SSP2CON2bits.ACKEN = 1; //S'habilita l'acknowledge
            lecturaI2C = SSP2BUF;//Es borra el buffer per ser escit.
            /*Per evitar que es doni l'overflow, s'habilita el regitre BF per
             *   indicar que el buffer està preparat per enviar una altra
dada.*/
            SSP2STATbits.BF = 0;
            SSP2BUF = escripturaI2C; //S'escriu el byte a enviar en el buffer.
            //Es reincialitza el rellotge.
            SSP2CON1bits.CKP = 1;
            //S'espera fins que s'hagi enviat el byte cap al mestre.
            while(SSP2STATbits.BF);
        }
        SSP2IF = 0; //Es borra el flag d'interrupció.
    }
}

void Init();
void PI();
void canvi_consigna();
```

```
void adquirir_valors();
void canvi_generacio();
void enviar_error();
void TMR0_compareInterrupt();
void TMR3_compareInterrupt();
void TMR4_compareInterrupt();
/*****************************************************************************
* Main application
*****************************************************************************/


void main()
{
    // Initialize the device
    SYSTEM_Initialize();

    // Enable the Global Interrupts
    INTERRUPT_GlobalInterruptEnable();

    // Enable the Peripheral Interrupts
    INTERRUPT_PeripheralInterruptEnable();


    if(!STATUSbits.TO && !PCON0bits.RWDT){
        estat_error = Error_WATCHDOG;
    }

      Init();                                          //Inicialitzar          el
microcontrolador
    TMR0_SetInterruptHandler(TMR0_compareInterrupt); //interrupció de 1s per
llegir la velocitat del rotor i decidir si canvia la consigna
    TMR3_SetInterruptHandler(TMR3_compareInterrupt); //interrupcio de 100ms
per tal de recollir les dades d'error i activar el PI
    TMR4_SetInterruptHandler(TMR4_compareInterrupt); //interrupció de 4s pel
mode test de l'I2C
    EUSART1_SetRxInterruptHandler(DADES_REBUDES_EUSART);
    I2C2_SlaveSetIsrHandler(I2C_Slave_Read);
    I2C_Slave_Init(0x08); //Adreça d'esclau de la placa
    I2C_Master_Init(400000); //Inicialitzar I2C Master a 400KHz
    WDTCON0bits.SEN = 1; //Habilitar el watchdog

    while(1)                                  //Loop infinit
      {
        CLRWDT(); //Reseteig del watchdog
        if(do_canvi_consigna) canvi_consigna(); //S'activa el canvi de valor
de consigna quan ho demana la interrupció del timer 0
        if(alarma_vel_elevada && posicio_max){
            I2C_Master_Start();         //Start condition
            I2C_Master_Write(adreca_controlador_potencia_escriptura);      //7
bit address + Write
            I2C_Master_Write(codi_posicio_maxima);
```

```
            I2C_Master_Stop();
            valor_PWM = 0;          //anul·lar moviment
            PWM3_LoadDutyValue(valor_PWM);
            WDTCON0bits.SEN=0;  //Deshabilitar  el  watchdog  fins  que  no  es
rearmi el sistema
            while (estat_emergencia || estat_error){}
            WDTCON0bits.SEN=1; //Habilitar el watchdog un cop rearmat
        }
             if (do_adquirir_valors) adquirir_valors();
        if(enable_PI && do_PI) PI();                            //S'activa el PI
només quan ho demana la interrupció del timer 3
        else if(!enable_PI) {
            valor_PWM = 0;     //anul·lar moviment
            PWM3_LoadDutyValue(valor_PWM);
        }

        if(recepcio_mode_test){
            if(Test_ESCLAU2 != 0xE2 || Test_ESCLAU3 != 0xE3){
                comptador_test++;
                if (comptador_test == 3){
                    comptador_test = 0;
                    estat_error = Error_mode_test;
                }
            }else{
                comptador_test = 0;
            }
            recepcio_mode_test = false;
        }

        if(!Generacio_ESCLAU1  &&  !Generacio_ESCLAU2  &&  !Generacio_ESCLAU3  &&
!do_canvi_consigna){
            valor_PWM = 0;          //anul·lar moviment
            PWM3_LoadDutyValue(valor_PWM);
            WDTCON0bits.SEN=0; //Deshabilitar el watchdog perque no salti en
sleep
            TMR3_StopTimer();
            TMR4_StopTimer();
            SLEEP();
        } else {
            TMR3_StartTimer();
            TMR4_StartTimer();
        }

        if(estat_error   !=   NO_ERROR   &&   estat_error   !=   Error_extern)
enviar_error();
       }
}


void Init()
{
```

```
      posicio_raw = 0;
    TMR1 = 0;
    do_adquirir_valors = true;
    do_canvi_consigna = true;
    WDTCON0bits.SEN=1; //Habilitar el watchdog
}

void TMR4_compareInterrupt(){ //4s - MODE TEST I2C
    //Activació del mode test - Sistema de monitorització.
    I2C_Master_Start();         //Start condition
    I2C_Master_Write(adreca_controlador_monitoritzacio_escriptura);         //7
bit address + Write
    I2C_Master_Write(codi_start_test);
    I2C_Master_Stop();

    //Activació del mode test - Sistema de control de potència.
    I2C_Master_Start();         //Start condition
    I2C_Master_Write(adreca_controlador_potencia_escriptura);         //7   bit
address + Write
    I2C_Master_Write(codi_start_test);
    I2C_Master_Stop();

    //Lectura del mode test - Sistema de monitorització.
    I2C_Master_Start();         //Start condition
    I2C_Master_Write(adreca_controlador_monitoritzacio_lectura);       //7 bit
address + Read
    Test_ESCLAU2 = I2C_Master_Read(0);
    I2C_Master_Stop();

    //Lectura del mode test - Sistema de control de potència.
    I2C_Master_Start();         //Start condition
    I2C_Master_Write(adreca_controlador_potencia_lectura);      //7 bit address
+ Read
    Test_ESCLAU3 = I2C_Master_Read(0);
    I2C_Master_Stop();

    recepcio_mode_test = true;
}



void TMR0_compareInterrupt(){ // 1s - LECTURA ENCODER I CANVI DE POSICIÓ
    velocitat_encoder = TMR1/10; // X/1 P/s * 1/600 r/P * 60 s/min --> Y rpm
(P:polsos, r:revolucions, s:segons)
    TMR1 = 0;
    do_canvi_consigna = true;
    WDTCON0bits.SEN=1; //Habilitar el watchdog en cas de wake up from sleep.
}

void TMR3_compareInterrupt(){ // 100ms - LECCTURA SENSOR DE POSICIÓ I CONTROL
PI
    do_adquirir_valors = true;
```

```
}

void canvi_consigna(){
    if(estat_emergencia || estat_error){
        pos_consigna = 9;
        alarma_vel_elevada = true;
        do_canvi_consigna = false;
        return;
    }
    if (velocitat_encoder>limit_superior_encoder){
        if(p_llistat_posicions<5) p_llistat_posicions++;
        else estat_emergencia = true;
    } else if (velocitat_encoder<limit_inferior_encoder){
        if(p_llistat_posicions>0) p_llistat_posicions--;
        else if (Generacio_ESCLAU1){
            Generacio_ESCLAU1 = false;
            canvi_generacio();
        }
    } else{
        alarma_vel_elevada = false;
        if(!Generacio_ESCLAU1){
            Generacio_ESCLAU1 = true;
            canvi_generacio();
        }
    }
    switch (p_llistat_posicions) {
        case 0:
            pos_consigna = 0;
            break;
        case 1:
            pos_consigna = 1.8;
            break;
        case 2:
            pos_consigna = 3.6;
            break;
        case 3:
            pos_consigna = 5.4;
            break;
        case 4:
            pos_consigna = 7.2;
            break;
        case 5:
            pos_consigna = 9;
            break;
    }
    do_canvi_consigna = false;
}
```

```
void adquirir_valors(){
   ADCC_DischargeSampleCapacitor();
   posicio_raw = ADCC_GetSingleConversion(channel_ANA1); //recull la posició
   ADCC_DischargeSampleCapacitor();
   volt_lm35 = ADCC_GetSingleConversion(channel_ANA5); //recull el voltatge
del sensor de temperatura 10 bits
   temperatura = (((uint32_t)volt_lm35)*500/1024); //ajustar els rangs de
voltatge i escalar
   posicio_raw = posicio_raw - (constant_temp*temperatura); //acabar de
quadrar l'ajustament de temperatura
   posicio_cm = ((float) posicio_raw - 39.745) / 102.88;
   error_p = pos_consigna - posicio_cm;
   vel_real = (posicio_cm - posicio_cm1)*100; //dividir per 0.1s i multiplicar
per 10 per obtenir la velocitat real en mm/s
   posicio_cm1 = posicio_cm;
   if (posicio_cm > 8.4) posicio_max = true;
   else posicio_max = false;
   do_adquirir_valors = false;
   do_PI = true;
}

void PI()
{
    if (error_p < 0) error_p = -error_p;
    if (error_p <= zona_morta){
        vel_consigna = 0;
        kp = 0;
        ti = 1000000;
        Cn_1 = 0;
    }else if (error_p > zona_morta && error_p <= limit_baix){
        vel_consigna = vel_baixa;
        kp = kp_vel_baixa;
        ti = ti_vel_baixa;
    } else if (error_p > limit_baix && error_p <= limit_alt){
        vel_consigna = vel_mitja;
        kp = kp_vel_mitja;
        ti = ti_vel_mitja;
    }else{
        vel_consigna = vel_alta;
        kp = kp_vel_alta;
        ti = ti_vel_alta;
    }

      error_v = vel_consigna - vel_real;
    q0=kp*(1+T/(2*ti));
    q1=-kp*(1-T/(2*ti));
    Cn = Cn_1 + q0*error_v + q1*error_v1; //llei del controlador PI discret

    if(Cn >= 100) Cn = 100;            // Limitador de duty cycle per evitar
warnings
```

```
      if(Cn <= 5)  Cn = 0;                    // Zona morta de duty cycle

    error_v1 = error_v;
    Cn_1 = Cn;

    valor_PWM = Cn;

    if(valor_PWM == 0) PWM3_LoadDutyValue(valor_PWM);

      else if(pos_consigna > posicio_cm){     //Si cal anar endavant
       if (IO_RA4_PORT == 1){
           valor_PWM = 0;     //anul·lar moviment
           PWM3_LoadDutyValue(valor_PWM);
           IO_RA4_PORT = 0;    //Canvi de sentit
       }else{
           PWM3_LoadDutyValue(valor_PWM);  //Carregar  valor  de  sortida  del
controlador
       }
    }

      else if (pos_consigna < posicio_cm){       //Si cal anar endarrere
       if(IO_RA4_PORT == 0){
           valor_PWM = 0;         //anul·lar moviment
           PWM3_LoadDutyValue(valor_PWM);
           IO_RA4_PORT = 1;    //Canvi de sentit
       } else{
           PWM3_LoadDutyValue(valor_PWM);  //Carregar  valor  de  sortida  del
controlador
       }
    }

      do_PI = false;

      return;
}

void canvi_generacio(){
    //Enviament  variable  generacio_ESCLAU1  mode  sleep  -  Sistema  de
monitorització.
    I2C_Master_Start();         //Start condition
    I2C_Master_Write(adreca_controlador_monitoritzacio_escriptura);        //7
bit address + Write
    I2C_Master_Write(codi_base_generacio | Generacio_ESCLAU1);
    I2C_Master_Stop();

    //Enviament variable generacio_ESCLAU1 mode sleep - Sistema de control de
potència.
    I2C_Master_Start();         //Start condition
    I2C_Master_Write(adreca_controlador_potencia_escriptura);          //7  bit
address + Write
    I2C_Master_Write(codi_base_generacio | Generacio_ESCLAU1);
```

```
    I2C_Master_Stop();
}

void enviar_error(){
    //Activació del mode error - Sistema de control de potència.
    I2C_Master_Start();         //Start condition
    I2C_Master_Write(adreca_controlador_potencia_escriptura);          //7  bit
address + Write
    I2C_Master_Write(codi_error);
    I2C_Master_Stop();
}
```

A.3. Arxiu i2c.c

```
/*****************************************************************************
* Títol                 :   Llibreria I2C
* Nom de l'arxiu        :   i2c.c
* Autors                :   AQ & PC
* Data inicial          :   08/04/2023
* Versió                :   1.0.0
* Compilador            :   Microchip XC8
* Microprocessador      :   PIC18F47Q10
* Notes                 :   Cap
*****************************************************************************/

/*************** MODULE REVISION LOG ***************************************
*
*    Data    Versió de software  Inicials   Descripció
*  08/04/2023    1.0.0           AQ & PC    Creació de les funcions
*
*****************************************************************************/
/** \file i2c.c
 * \brief Aquesta llibreria conté les funcions necessaries per la comunicació
en
 *  mode mestre i en mode esclau mitjançant el mòdul MSSP del microcontrolador
 */

/*****************************************************************************
 * Includes

*****************************************************************************/
#include "mcc_generated_files/mcc.h"
#include "mcc_generated_files/pin_manager.h"

#include "i2c.h"
#include <xc.h>
#include <pic18f47q10.h>
```

```
/*******************************************************************************
 * Constants i definicions
*******************************************************************************/

/*******************************************************************************
 * Macros
*******************************************************************************/

/*******************************************************************************
 * Typedefs
*******************************************************************************/

/*******************************************************************************
 * Variables
*******************************************************************************/

/*******************************************************************************
 * Definició de funcions
*******************************************************************************/

/*******************************************************************************
 * Funció : I2C_Master_Init
 *
 * \section Descripció:
 *
 *  Aquesta funció inicialitza la comunicació en mode mestre.
 *
 * \param           c --> velocitat del rellotge del bus en kbits/s.
 *
 * \return          Cap.
*******************************************************************************/
void I2C_Master_Init(const unsigned long c)
{
    //Configuració del mòdul MSSP1 actuant com a  mestre de I2C
    SSP1CON1 = 0b00101000;
    SSP1CON2 = 0;
    //Definció del divisor del clock -> 400kbits/s
    SSP1ADD = (_XTAL_FREQ / (4 * c)) - 1;
    //Es posa el "slew rate" en mode d'alta freqüència
    SSP1STAT = 0;
```

```
    //Definició dels pins del canals de dades i de clock.
    SSP1CLKPPS = 0x13;   //RC3->MSSP1:SCL1;
    RC3PPS = 0x0F;   //RC3->MSSP1:SCL1;
    RC4PPS = 0x10;   //RC4->MSSP1:SDA1;
    SSP1DATPPS = 0x14;   //RC4->MSSP1:SDA1;
    //Es desactiven els pins fins que es doni una seqüència de start
    SSP1CON1bits.SSPEN = 0;
    //Es prefixa el temps de hold del bus SDA.
    SSP1CON3bits.SDAHT = 1;
}

/*****************************************************************************
* Funció : I2C_Master_Wait
*
* \section Descripció:
*
*  Aquesta funció d'espera serveix per comprovar que no hi ha accions pendents
 * en el bus.
*
* \param           Cap.
*
* \return          Cap.
*****************************************************************************/

void I2C_Master_Wait(void)
{
    //Funció que analitza si hi ha comunicació activa i si es el cas, espera
    while ((SSP1STAT & 0x04) || (SSP1CON2 & 0x1F));
}

/*****************************************************************************
* Funció : I2C_Master_Start
*
* \section Descripció:
*
*  Aquesta funció envia la condició de "start" en el bus.
*
* \param           Cap.
*
* \return          Cap.
*****************************************************************************/

void I2C_Master_Start(void)
{
    //Rutina d'espera
    I2C_Master_Wait();
    //S'habiliten els pins.
    SSP1CON1bits.SSPEN = 1;
    //S'incina la seqüència del bit start.
```

```
    SSP1CON2bits.SEN = 1;
    //Quan acaba la seqüència aquest bit es posa a zero.
    while (SSP1CON2bits.SEN);
    //Fins que no es posi a zero la funció no acaba.
}

/*****************************************************************************
* Funció : I2C_Master_RepeatedStart
*
* \section Descripció:
*
*  Aquesta funció envia la condició de "repeated start" en el bus.
*
* \param           Cap.
*
* \return          Cap.
*****************************************************************************/

void I2C_Master_RepeatedStart(void)
{
    //Rutina d'espera
    I2C_Master_Wait();
    //S'incina la seqüència del bit de repeated start.
    SSP1CON2bits.RSEN = 1;
    //Quan acaba la seqüència aquest bit es posa a zero.
    while (SSP1CON2bits.RSEN);
    //Fins que no es posi a zero la funció no acaba.
}

/*****************************************************************************
* Funció : I2C_Master_Stop
*
* \section Descripció:
*
*  Aquesta funció envia la condició de "stop" en el bus.
*
* \param           Cap.
*
* \return          Cap.
*****************************************************************************/

void I2C_Master_Stop(void)
{
    //Rutina d'espera
    I2C_Master_Wait();
    //S'incina la seqüència del bit stop.
    SSP1CON2bits.PEN = 1;
    //Quan acaba la seqüència aquest bit es posa a zero.
    while (SSP1CON2bits.PEN);
```

```
    //Fins que no es posi a zero la funció no acaba.
}

/*****************************************************************************
* Funció : I2C_Master_Read
*
* \section Descripció:
*
*  Aquesta funció realitza una lectura del mestre a l'esclau
*
* \param            a --> Al finalitzar la lectura es transmet un ACK (1)
                       * o un NACK (0) des del mestre.
*
* \return           temp --> dades rebudes en el buffer (unsigned short).
*****************************************************************************/

unsigned short I2C_Master_Read(unsigned short a)
{
    unsigned short temp; //Variable de retorn
    I2C_Master_Wait(); //Espera del bus
    SSP1CON2bits.RCEN = 1; //habilita el bit de recepció
    I2C_Master_Wait(); //Espera que el bus rebi les dades.
    temp = SSP1BUF; //El valor del buffer queda volcat a la variable.
    I2C_Master_Wait(); //Espera del bus
    SSP1CON2bits.ACKDT = (a) ? 0 : 1;/*Depenent del valor de la variable a, el
                                     * mestre retorna un ACK per continuar amb la
                                     * comunicació o un NACK per finalitzar-la.*/
    SSP1CON2bits.ACKEN = 1; //Habilita l'Acknowledge.
    return temp; //Retorna la variable.
}

/*****************************************************************************
* Funció : I2C_Master_Write
*
* \section Descripció:
*
*  Aquesta funció realitza una escriptura del mestre a l'esclau
*
* \param            d --> Dades a transmetre en el buffer.
*
* \return           Cap.
*****************************************************************************/

void I2C_Master_Write(unsigned d)
{
    //Acció d'espera
    I2C_Master_Wait();
    //Escriure en el buffer.
    SSP1BUF = d;
```

```
    //Esperar que es buidi el buffer i de rebre l'Acknowledge
    while (SSP1STATbits.R_nW);
}

/*****************************************************************************
* Funció : I2C1_MasterIsNack
*
* \section Descripció:
*
*  Aquesta funció permet saber si el mestre es troba en ACK o NACK
*
* \param            Cap.
*
* \return           Estat del mestre (bool).
*****************************************************************************/

bool I2C1_MasterIsNack(void)
{
    return SSP1CON2bits.ACKSTAT;
}

/*****************************************************************************
* Funció : I2C_Slave_Init
*
* \section Descripció:
*
*  Aquesta funció inicialitza la comunicació en mode esclau.
*
* \param            address --> Configura el mòdul I2C amb l'adreça
                               *d'esclau seleccionada.
*
* \return           Cap.
*****************************************************************************/

void I2C_Slave_Init(short address)
{
    //Activar "slew rate" a 400kbits/s
    SSP2STAT = 0x00;
    //Definir l'adreça de l'esclau
    SSP2ADD = address;
    //Configurar en mode esclau i habilitar els pins
    SSP2CON1 = 0x36;
    SSP2CON2 = 0x01;

    //Associar SDA i SCL a pins físics del micro
    SSP2DATPPS = 0x0C;   //RB4->MSSP2:SDA2;
    RB3PPS = 0x11;   //RB3->MSSP2:SCL2;
    RB4PPS = 0x12;   //RB4->MSSP2:SDA2;
    SSP2CLKPPS = 0x0B;   //RB3->MSSP2:SCL2;
```

```
    SSP2IF = 0;         //Netejar la flag d'interrupcions
    SSP2IE = 1;         //Habilitar interrupcions del port síncron
}


/************************** END OF FUNCTIONS ******************************/
```

A.4. Arxiu i2c.h

```
/*******************************************************************************
* Títol                 :   Llibreria I2C
* Nom de l'arxiu        :   i2c.h
* Autors                :   AQ & PC
* Data inicial          :   08/04/2023
* Versió                :   1.0.0
* Compilador            :   Microchip XC8
* Microprocessador      :   PIC18F47Q10
* Notes                 :   Cap
*******************************************************************************/
/*************** INTERFACE CHANGE LIST ************************************
*
*    Data    Versió de software  Inicials   Descripció
*  08/04/2023    1.0.0           AQ & PC    Creació de la llibreria
*
*******************************************************************************/
/** \file i2c.h
 *   \brief Aquest mòdul conté les funcions de l'I2C a implementar en el
programa.
 */
#ifndef I2C_H
#define I2C_H

/*******************************************************************************
* Includes
*******************************************************************************/
#include "mcc_generated_files/mcc.h"
#include "mcc_generated_files/pin_manager.h"

#include <xc.h>
#include <pic18f47q10.h>

/*******************************************************************************
* Constants de preprocessador
*******************************************************************************/


/*******************************************************************************
* Constants de configuració
*******************************************************************************/
```

```
/*****************************************************************************
* Macros
*****************************************************************************/


/*****************************************************************************
* Typedefs
*****************************************************************************/


/*****************************************************************************
* Variables
*****************************************************************************/


/*****************************************************************************
* Prototips de funcions
*****************************************************************************/
#ifdef __cplusplus
extern "C" {
#endif


void I2C_Master_Init(const unsigned long c);
void I2C_Master_Wait(void);
void I2C_Master_Start(void);
void I2C_Master_RepeatedStart(void);
void I2C_Master_Stop(void);
unsigned short I2C_Master_Read(unsigned short a);
void I2C_Master_Write(unsigned d);
static inline bool I2C1_MasterIsNack(void);
void I2C_Slave_Init(short address);

#endif /* I2C_H */

/*** End of File *************************************************************/
```

La llibreria en qüestió conté totes les funcions necessàries per la comunicació I2C exceptuant la interrupció d'esclau que per la seva naturalesa ha de ser inclosa en el main.c.

A.5. Arxiu modbus.c

```
/*
 * File:   modbus.c
 *
 * Created on February 23, 2022, 10:53 PM
```

```
 */


#include "mcc_generated_files/mcc.h"
#include "modbus.h"
#include <stdio.h>
#include <pic18f47q10.h>
#include <xc.h>




bool LECTURA_DEL_uint16_auto[16];
bool LECTURA_DEL_uint16_test[16];


//Gobal variables
uint8_t response[20];
uint8_t error;
bool frame;
uint16_t comprovacio=0;
//Funcions Modbus
typedef enum functions{READ_COILS = 0x01, //mode d'accés a bit
READ_INPUT_REGISTERS = 0x04,
WRITE_MULTIPLE_REGISTERS = 0x10,
}function;
//Definim les diferents excepcions que hi ha en el protocol
typedef enum exceptions{
    ILLEGAL_FUNCTION = 0x01,
    ILLEGAL_DATA_ADDRESS = 0x02,
    ILLEGAL_DATA_VALUE = 0x03,
}exception;
//LUT (Look Up Table) CRC sol·licitud
static uint16_t MODBUS_CRC16_RX(uint8_t len)
{
    static const uint16_t table[256] = {
    0x0000, 0xC0C1, 0xC181, 0x0140, 0xC301, 0x03C0, 0x0280, 0xC241,
    0xC601, 0x06C0, 0x0780, 0xC741, 0x0500, 0xC5C1, 0xC481, 0x0440,
    0xCC01, 0x0CC0, 0x0D80, 0xCD41, 0x0F00, 0xCFC1, 0xCE81, 0x0E40,
    0x0A00, 0xCAC1, 0xCB81, 0x0B40, 0xC901, 0x09C0, 0x0880, 0xC841,
    0xD801, 0x18C0, 0x1980, 0xD941, 0x1B00, 0xDBC1, 0xDA81, 0x1A40,
    0x1E00, 0xDEC1, 0xDF81, 0x1F40, 0xDD01, 0x1DC0, 0x1C80, 0xDC41,
    0x1400, 0xD4C1, 0xD581, 0x1540, 0xD701, 0x17C0, 0x1680, 0xD641,
    0xD201, 0x12C0, 0x1380, 0xD341, 0x1100, 0xD1C1, 0xD081, 0x1040,
    0xF001, 0x30C0, 0x3180, 0xF141, 0x3300, 0xF3C1, 0xF281, 0x3240,
    0x3600, 0xF6C1, 0xF781, 0x3740, 0xF501, 0x35C0, 0x3480, 0xF441,
    0x3C00, 0xFCC1, 0xFD81, 0x3D40, 0xFF01, 0x3FC0, 0x3E80, 0xFE41,
    0xFA01, 0x3AC0, 0x3B80, 0xFB41, 0x3900, 0xF9C1, 0xF881, 0x3840,
    0x2800, 0xE8C1, 0xE981, 0x2940, 0xEB01, 0x2BC0, 0x2A80, 0xEA41,
```

```
    0xEE01, 0x2EC0, 0x2F80, 0xEF41, 0x2D00, 0xEDC1, 0xEC81, 0x2C40,
    0xE401, 0x24C0, 0x2580, 0xE541, 0x2700, 0xE7C1, 0xE681, 0x2640,
    0x2200, 0xE2C1, 0xE381, 0x2340, 0xE101, 0x21C0, 0x2080, 0xE041,
    0xA001, 0x60C0, 0x6180, 0xA141, 0x6300, 0xA3C1, 0xA281, 0x6240,
    0x6600, 0xA6C1, 0xA781, 0x6740, 0xA501, 0x65C0, 0x6480, 0xA441,
    0x6C00, 0xACC1, 0xAD81, 0x6D40, 0xAF01, 0x6FC0, 0x6E80, 0xAE41,
    0xAA01, 0x6AC0, 0x6B80, 0xAB41, 0x6900, 0xA9C1, 0xA881, 0x6840,
    0x7800, 0xB8C1, 0xB981, 0x7940, 0xBB01, 0x7BC0, 0x7A80, 0xBA41,
    0xBE01, 0x7EC0, 0x7F80, 0xBF41, 0x7D00, 0xBDC1, 0xBC81, 0x7C40,
    0xB401, 0x74C0, 0x7580, 0xB541, 0x7700, 0xB7C1, 0xB681, 0x7640,
    0x7200, 0xB2C1, 0xB381, 0x7340, 0xB101, 0x71C0, 0x7080, 0xB041,
    0x5000, 0x90C1, 0x9181, 0x5140, 0x9301, 0x53C0, 0x5280, 0x9241,
    0x9601, 0x56C0, 0x5780, 0x9741, 0x5500, 0x95C1, 0x9481, 0x5440,
    0x9C01, 0x5CC0, 0x5D80, 0x9D41, 0x5F00, 0x9FC1, 0x9E81, 0x5E40,
    0x5A00, 0x9AC1, 0x9B81, 0x5B40, 0x9901, 0x59C0, 0x5880, 0x9841,
    0x8801, 0x48C0, 0x4980, 0x8941, 0x4B00, 0x8BC1, 0x8A81, 0x4A40,
    0x4E00, 0x8EC1, 0x8F81, 0x4F40, 0x8D01, 0x4DC0, 0x4C80, 0x8C41,
    0x4400, 0x84C1, 0x8581, 0x4540, 0x8701, 0x47C0, 0x4680, 0x8641,
    0x8201, 0x42C0, 0x4380, 0x8341, 0x4100, 0x81C1, 0x8081, 0x4040};
    uint8_t index = 0;
    uint16_t crc = 0xFFFF;
    for(uint8_t j = 0; j < len; j++){
        index = (RxBuffer[j]) ^ crc;
        crc >>= 8;
        crc ^= table[index];
    }
    return crc;
}
//LUT (Look Up Table) CRC resposta
static uint16_t MODBUS_CRC16_TX(uint8_t len){
    static const uint16_t table[256] = {
    0x0000, 0xC0C1, 0xC181, 0x0140, 0xC301, 0x03C0, 0x0280, 0xC241,
    0xC601, 0x06C0, 0x0780, 0xC741, 0x0500, 0xC5C1, 0xC481, 0x0440,
    0xCC01, 0x0CC0, 0x0D80, 0xCD41, 0x0F00, 0xCFC1, 0xCE81, 0x0E40,
    0x0A00, 0xCAC1, 0xCB81, 0x0B40, 0xC901, 0x09C0, 0x0880, 0xC841,
    0xD801, 0x18C0, 0x1980, 0xD941, 0x1B00, 0xDBC1, 0xDA81, 0x1A40,
    0x1E00, 0xDEC1, 0xDF81, 0x1F40, 0xDD01, 0x1DC0, 0x1C80, 0xDC41,
    0x1400, 0xD4C1, 0xD581, 0x1540, 0xD701, 0x17C0, 0x1680, 0xD641,
    0xD201, 0x12C0, 0x1380, 0xD341, 0x1100, 0xD1C1, 0xD081, 0x1040,
    0xF001, 0x30C0, 0x3180, 0xF141, 0x3300, 0xF3C1, 0xF281, 0x3240,
    0x3600, 0xF6C1, 0xF781, 0x3740, 0xF501, 0x35C0, 0x3480, 0xF441,
    0x3C00, 0xFCC1, 0xFD81, 0x3D40, 0xFF01, 0x3FC0, 0x3E80, 0xFE41,
    0xFA01, 0x3AC0, 0x3B80, 0xFB41, 0x3900, 0xF9C1, 0xF881, 0x3840,
    0x2800, 0xE8C1, 0xE981, 0x2940, 0xEB01, 0x2BC0, 0x2A80, 0xEA41,
    0xEE01, 0x2EC0, 0x2F80, 0xEF41, 0x2D00, 0xEDC1, 0xEC81, 0x2C40,
    0xE401, 0x24C0, 0x2580, 0xE541, 0x2700, 0xE7C1, 0xE681, 0x2640,
    0x2200, 0xE2C1, 0xE381, 0x2340, 0xE101, 0x21C0, 0x2080, 0xE041,
    0xA001, 0x60C0, 0x6180, 0xA141, 0x6300, 0xA3C1, 0xA281, 0x6240,
    0x6600, 0xA6C1, 0xA781, 0x6740, 0xA501, 0x65C0, 0x6480, 0xA441,
```

```
    0x6C00, 0xACC1, 0xAD81, 0x6D40, 0xAF01, 0x6FC0, 0x6E80, 0xAE41,
    0xAA01, 0x6AC0, 0x6B80, 0xAB41, 0x6900, 0xA9C1, 0xA881, 0x6840,
    0x7800, 0xB8C1, 0xB981, 0x7940, 0xBB01, 0x7BC0, 0x7A80, 0xBA41,
    0xBE01, 0x7EC0, 0x7F80, 0xBF41, 0x7D00, 0xBDC1, 0xBC81, 0x7C40,
    0xB401, 0x74C0, 0x7580, 0xB541, 0x7700, 0xB7C1, 0xB681, 0x7640,
    0x7200, 0xB2C1, 0xB381, 0x7340, 0xB101, 0x71C0, 0x7080, 0xB041,
    0x5000, 0x90C1, 0x9181, 0x5140, 0x9301, 0x53C0, 0x5280, 0x9241,
    0x9601, 0x56C0, 0x5780, 0x9741, 0x5500, 0x95C1, 0x9481, 0x5440,
    0x9C01, 0x5CC0, 0x5D80, 0x9D41, 0x5F00, 0x9FC1, 0x9E81, 0x5E40,
    0x5A00, 0x9AC1, 0x9B81, 0x5B40, 0x9901, 0x59C0, 0x5880, 0x9841,
    0x8801, 0x48C0, 0x4980, 0x8941, 0x4B00, 0x8BC1, 0x8A81, 0x4A40,
    0x4E00, 0x8EC1, 0x8F81, 0x4F40, 0x8D01, 0x4DC0, 0x4C80, 0x8C41,
    0x4400, 0x84C1, 0x8581, 0x4540, 0x8701, 0x47C0, 0x4680, 0x8641,
    0x8201, 0x42C0, 0x4380, 0x8341, 0x4100, 0x81C1, 0x8081, 0x4040 };
    uint8_t index = 0;
    uint16_t crc = 0xFFFF;
    for(uint8_t j = 0; j < len; j++){
        index = (response[j]) ^ crc;
        crc >>= 8;
        crc ^= table[index];
    }
    return crc;
}
//CRC 8 bits low byte
uint8_t CRC8_Lo (uint16_t crc16){
    uint8_t crc_low = crc16 | 0;
    return crc_low;
}
//CRC 8 bits high byte
uint8_t CRC8_Hi(uint16_t crc16){
    uint8_t crc_high = (crc16 - CRC8_Lo(crc16)) >> 8;
    return crc_high;
}
//Comprovació que la trama llegida i calculada de les dades que rebem són
iguals
bool checkCRC(void){
    if(RxBuffer[1] == READ_COILS || RxBuffer[1] == READ_INPUT_REGISTERS){
        uint16_t crc16;
        uint8_t crc_Hi, crc_Lo;
        crc16 = MODBUS_CRC16_RX(6);
        crc_Hi = CRC8_Hi(crc16);
        crc_Lo = CRC8_Lo(crc16);
        if(RxBuffer[7] == crc_Hi && RxBuffer[6] == crc_Lo){
            comprovacio=1;
            return true;
        }else{
            return false;
        }
    }
```

```
    else if(RxBuffer[1] == WRITE_MULTIPLE_REGISTERS){
        uint16_t crc16;
        uint8_t crc_Hi, crc_Lo;
        crc16 = MODBUS_CRC16_RX(13);
        crc_Hi = CRC8_Hi(crc16);
        crc_Lo = CRC8_Lo(crc16);
        if(RxBuffer[14] == crc_Hi && RxBuffer[13] == crc_Lo){
            return true;
        }else{
            return false;
        }
    }
}

void error_function(void){
    if(RxBuffer[1] == READ_COILS){
        uint16_t crc16;
        crc16 = MODBUS_CRC16_TX(3);
        response[0] = SlaveAddress;
        response[1] = 0x80 + READ_COILS;
        response[2] = error;
        response[4] = CRC8_Hi(crc16);
        response[3] = CRC8_Lo(crc16);
        for(uint8_t j = 0; j < 5; j++){
            EUSART1_Write(response[j]);
        }
    }
    else if(RxBuffer[1] == READ_INPUT_REGISTERS){
        uint16_t crc16;
        crc16 = MODBUS_CRC16_TX(3);
        response[0] = SlaveAddress;
        response[1] = 0x80 + READ_INPUT_REGISTERS;
        response[2] = error;
        response[4] = CRC8_Hi(crc16);
        response[3] = CRC8_Lo(crc16);
        for(uint8_t j = 0; j < 5; j++){
            EUSART1_Write(response[j]);
        }
    }
    else if(RxBuffer == WRITE_MULTIPLE_REGISTERS){
        uint16_t crc16;
        crc16 = MODBUS_CRC16_TX(3);
        response[0] = SlaveAddress;
        response[1] = 0x80 + WRITE_MULTIPLE_REGISTERS;
        response[2] = error;
        response[4] = CRC8_Hi(crc16);
        response[3] = CRC8_Lo(crc16);
        for(uint8_t j = 0; j < 5; j++){
            EUSART1_Write(response[j]);
```

```
        }
    }
}
void modbus_read_input_registers(uint16_t reg_num){
    uint8_t starting_address_Lo, starting_address_Hi, reg_num_Hi, reg_num_Lo,
    CRC_read_Hi, CRC_read_Lo,
    j, k = 0;
    //LA VARIABLE INPUTREGISTER ES LA QUE ES TRANSMET
    uint16_t crc16, adc_result, starting_address;
    uint16_t inputregister = 0;
    frame = 1;
    starting_address_Hi = RxBuffer[2];
    starting_address_Lo = RxBuffer[3];
    reg_num_Hi = RxBuffer[4];
    reg_num_Lo = RxBuffer[5];
    CRC_read_Hi = RxBuffer[6];
    CRC_read_Lo = RxBuffer[7];
    if(checkCRC() == true){
        //Passem els 2 bytes de 8 bits en un numero de 16 bits
        starting_address = starting_address_Hi;
        starting_address = starting_address << 8;
        starting_address = starting_address | starting_address_Lo;
        reg_num = reg_num_Hi;
        reg_num = reg_num << 8;
        reg_num = reg_num | reg_num_Lo;
        //Tot aixo diria que no cal
        /*if(reg_num != 1){
        error = ILLEGAL_DATA_VALUE;
        frame = 0;
        }
        if(starting_address != 0){
        error = ILLEGAL_DATA_ADDRESS;
        frame = 0;
        } */
        //processament de la trama quan hi ha un error
        if(frame == 0){
            error_function();
        }
        //processament de la trama quan és correcte
        if(frame == 1){
            switch (reg_num){
                // Declarem aqui tots els registres
                case 1:
                    // inputregister = la teva variable
                    inputregister = estat_error<<8 | valor_PWM;
                    break;
                case 2:
                    // inputregister = la teva variable
                    inputregister = estat_emergencia<<2
```

```
                    | alarma_vel_elevada<<1 | Generacio_ESCLAU1;
                    break;
                case 3:
                    // inputregister = la teva variable
                    inputregister = *(((uint16_t *) &temperatura)+1);
                    break;
                case 4:
                    // inputregister = la teva variable
                    inputregister = *(uint16_t *) &temperatura;
                    break;
                case 5:
                    // inputregister = la teva variable
                    inputregister = *(((uint16_t *) &posicio_cm)+1);
                    break;
                case 6:
                    // inputregister = la teva variable
                    inputregister = *(uint16_t *) &posicio_cm;
                    break;
                case 7:
                    // inputregister = la teva variable
                    inputregister = *(((uint16_t *) &pos_consigna)+1);
                    break;
                case 8:
                    // inputregister = la teva variable
                    inputregister = *(uint16_t *) &pos_consigna;
                    break;
                case 9:
                    // inputregister = la teva variable
                    inputregister = *(((uint16_t *) &vel_real)+1);
                    break;
                case 10:
                    inputregister = *(uint16_t *) &vel_real;
                    break;
                case 11:
                    inputregister = *(((uint16_t *) &velocitat_encoder)+1);
                    break;
                case 12:
                    inputregister = *(uint16_t *) &velocitat_encoder;
                    break;
                default:
                    inputregister = 0;
            }
            uint8_t inputregisterpart1 = (inputregister >> 8) & 0xFF; // Agafa
els 8 bits més significatius
            uint8_t inputregisterpart2 = inputregister & 0xFF; // Agafa els 8
bits menys significatius
            R_T_BIT_PORT=1; //Activem el bit de transmissió de dades
            response[0] = SlaveAddress;
            response[1] = READ_INPUT_REGISTERS;
            response[2] = 2 * reg_num; //2*reg_num_Lo
```

```
            //response[3] = 0;
            response[3] = inputregisterpart1;  //canviat de notació big endian
a mid-little endian
            response[4] = inputregisterpart2; //ull
            crc16 = MODBUS_CRC16_TX(5);
            response[6] = CRC8_Hi(crc16); //crc em surt 0
            response[5] = CRC8_Lo(crc16);
            for (uint8_t jk = 0; jk <= 6; jk++){
                EUSART1_Write(response[jk]);
            }
            __delay_ms(10);
            R_T_BIT_PORT=0; //Desactivem el bit de transmissió de dades
        }
    }
}
// NOMES S'ESCRIUEN DOS REGISTRES
void modbus_write_multiple_registers(void){
    uint8_t starting_address_Hi, starting_address_Lo, reg_num_Hi, reg_num_Lo,
    byte_count, auto_data_Hi, auto_data_Lo, test_data_Hi, test_data_Lo,
    test_value_data_Hi, test_value_data_Lo, CRC_read_Hi, CRC_read_Lo;
    uint16_t crc16, starting_address, reg_num, auto_data, test_data;
    frame = 1; //inicialitzem com a trama correcta
    starting_address_Hi= RxBuffer[2];
    starting_address_Lo = RxBuffer[3];
    reg_num_Hi = RxBuffer[4];
    reg_num_Lo = RxBuffer[5];
    byte_count = RxBuffer[6];
    auto_data_Hi= RxBuffer[7];
    auto_data_Lo= RxBuffer[8];
    test_data_Hi = RxBuffer[9];
    test_data_Lo = RxBuffer[10];
    test_value_data_Hi = RxBuffer[11];
    test_value_data_Lo = RxBuffer[12];
    CRC_read_Hi = RxBuffer[13];
    CRC_read_Lo = RxBuffer[14];
    if(checkCRC() == true){
        //passem a 16 bits
        starting_address = starting_address_Hi;
        starting_address = starting_address << 8;
        starting_address = starting_address | starting_address_Lo;
        reg_num = reg_num_Hi;
        reg_num = reg_num << 8;
        reg_num = reg_num | reg_num_Lo;
        auto_data = auto_data_Hi;
        auto_data = auto_data << 8;
        auto_data = auto_data | auto_data_Lo;
        test_data = test_data_Hi;
        test_data = test_data << 8;
        test_data = test_data | test_data_Lo;
```

```
    if(starting_address < 0){ //sempre adreça inici de zero
        frame = 0;
        error = ILLEGAL_DATA_ADDRESS;
    }
    /*
    if(reg_num != 3 && reg_num * 2 != byte_count){
    frame = 0;
    error = ILLEGAL_DATA_VALUE;
    } */
    //processament de la trama quan hi ha un error
    if (frame == 0){
        error_function();
    }
    //processament de la trama quan és correcte
    if(frame == 1){//trama correcta
        R_T_BIT_PORT=1;
        response[0] = SlaveAddress;
        response[1] = WRITE_MULTIPLE_REGISTERS;
        response[2] = starting_address_Hi;
        response[3] = starting_address_Lo;
        response[4] = reg_num_Hi;
        response[5] = reg_num_Lo;
        crc16 = MODBUS_CRC16_TX(6);
        response[7] = CRC8_Hi(crc16);
        response[6] = CRC8_Lo(crc16);
        for(uint8_t j = 0; j <= 7; j++){
            EUSART1_Write(response[j]);
        }
        __delay_ms(5);
        R_T_BIT_PORT=0;
    }
}
switch (reg_num){
    // Declaració de tots els registres
    case 3:
        estat_error = (auto_data>>8)&0xFF;
        estat_emergencia = (auto_data>>1) &0x01;
        enable_PI = auto_data&0x01;
        limit_inferior_encoder = (test_data>>8)&0xFF;
        limit_superior_encoder = test_data & 0xFF;
        break;
    case 4:
        float_conversion.i[0] = auto_data;
        float_conversion.i[1] = test_data;
        constant_temp = float_conversion.f;
    case 5:
        float_conversion.i[0] = auto_data;
        float_conversion.i[1] = test_data;
        vel_baixa = float_conversion.f;
```

```
        case 6:
            float_conversion.i[0] = auto_data;
            float_conversion.i[1] = test_data;
            vel_mitja = float_conversion.f;
        case 7:
            float_conversion.i[0] = auto_data;
            float_conversion.i[1] = test_data;
            vel_alta = float_conversion.f;
        case 8:
            float_conversion.i[0] = auto_data;
            float_conversion.i[1] = test_data;
            kp_vel_baixa = float_conversion.f;
        case 9:
            float_conversion.i[0] = auto_data;
            float_conversion.i[1] = test_data;
            ti_vel_baixa = float_conversion.f;
        case 10:
            float_conversion.i[0] = auto_data;
            float_conversion.i[1] = test_data;
            kp_vel_mitja = float_conversion.f;
        case 11:
            float_conversion.i[0] = auto_data;
            float_conversion.i[1] = test_data;
            ti_vel_mitja = float_conversion.f;
        case 12:
            float_conversion.i[0] = auto_data;
            float_conversion.i[1] = test_data;
            kp_vel_alta = float_conversion.f;
        case 13:
            float_conversion.i[0] = auto_data;
            float_conversion.i[1] = test_data;
            ti_vel_alta = float_conversion.f;
    }
}
//Funcions modbus programades
void modbus_function (void){
    if(RxBuffer[0] == SlaveAddress){//COMPARAR ADREÇA
        switch (RxBuffer[1]){//COMPARAR FUNCIÓ
            case READ_INPUT_REGISTERS:
                modbus_read_input_registers(RxBuffer[4]);//Dada  a  llegir  pel
PLC
                break;
            case WRITE_MULTIPLE_REGISTERS://Escriu en el PIC
                modbus_write_multiple_registers();
                break;
            default:
                error = ILLEGAL_FUNCTION;
                frame = 0;
                break;
```

```
        }

    }
    c_control=0;
}
```

A.6. Arxiu modbus.h

```
#ifndef MODBUS_H
#define MODBUS_H

#include <xc.h>

#define SlaveAddress 1

#ifdef __cplusplus
extern "C" {
#endif

//FUNCIONS DEL MODBUS

static uint16_t MODBUS_CRC16_RX(uint8_t len);
static uint16_t MODBUS_CRC16_TX(uint8_t len);
uint8_t CRC8_Lo (uint16_t crc16);
uint8_t CRC8_Hi(uint16_t crc16);
bool checkCRC(void);
void initial_leds_state(void);
void error_function(void);
void modbus_read_input_registers(uint16_t reg_num);
void modbus_write_multiple_registers(void);
void modbus_function (void);


union float_conversion_t {
   uint16_t i [2];
   float f;
} float_conversion;

extern uint8_t c_control;
extern uint8_t RxBuffer[];

//VARIABLES DEL MICROCONTROLADOR 1
//Read and write variables
extern enum ESTAT_ERROR_t
{
    NO_ERROR = 0,
    Error_WATCHDOG,
    Error_mode_test,
    Error_extern
} estat_error;
```

```
extern _Bool estat_emergencia;

//Read variables
extern _Bool alarma_vel_elevada;
extern _Bool Generacio_ESCLAU1;
extern float temperatura;
extern float posicio_cm;
extern float pos_consigna;
extern float vel_real;
extern uint8_t valor_PWM;
extern float velocitat_encoder;

//Write variables
extern float constant_temp;
extern _Bool enable_PI;
extern uint8_t limit_inferior_encoder; //rpm
extern uint8_t limit_superior_encoder; //rpm
extern float vel_baixa;
extern float vel_mitja;
extern float vel_alta;
extern float kp_vel_baixa;
extern float ti_vel_baixa;
extern float kp_vel_mitja;
extern float ti_vel_mitja;
extern float kp_vel_alta;
extern float ti_vel_alta;

#endif
```

Igual que en la llibreria d'I2C la interrupció generada quan es reben dades a través de la EUSART està inclosa en el main.c.

B. PROGRAMARI MONITORITZACIÓ DE L'ESTRUCTURA I DEL VENT

En aquest annex, de la mateixa manera que en l'anterior, es mostren i es comenten les configuracions de MCC utilitzades així com els fitxers de programació .c i .h que contenen el programa del microcontrolador.

B.1. Configuració MCC

En la configuració de l'MCC s'han d'habilitar els mòduls de comunicació MSSP, per l'I2C, l'EUSART pel Modbus, associant-los als pins corresponents del Mikro-bus, habilitar la interrupció del temporitzador 0 (TMR0) i configurar-la a 1s, habilitar la interrupció del temporitzador 2 (TMR2) i configurar-la a 500ms, habilitar la interrupció del temporitzador 3 (TMR3) i configurar-la a 4s, habilitar la interrupció del temporitzador 4 (TMR4) i configurar-la a 2,5ms, habilitar l'ADC i per últim configurar els pins per tal de que coincideixin amb les connexions fetes a la PCB.

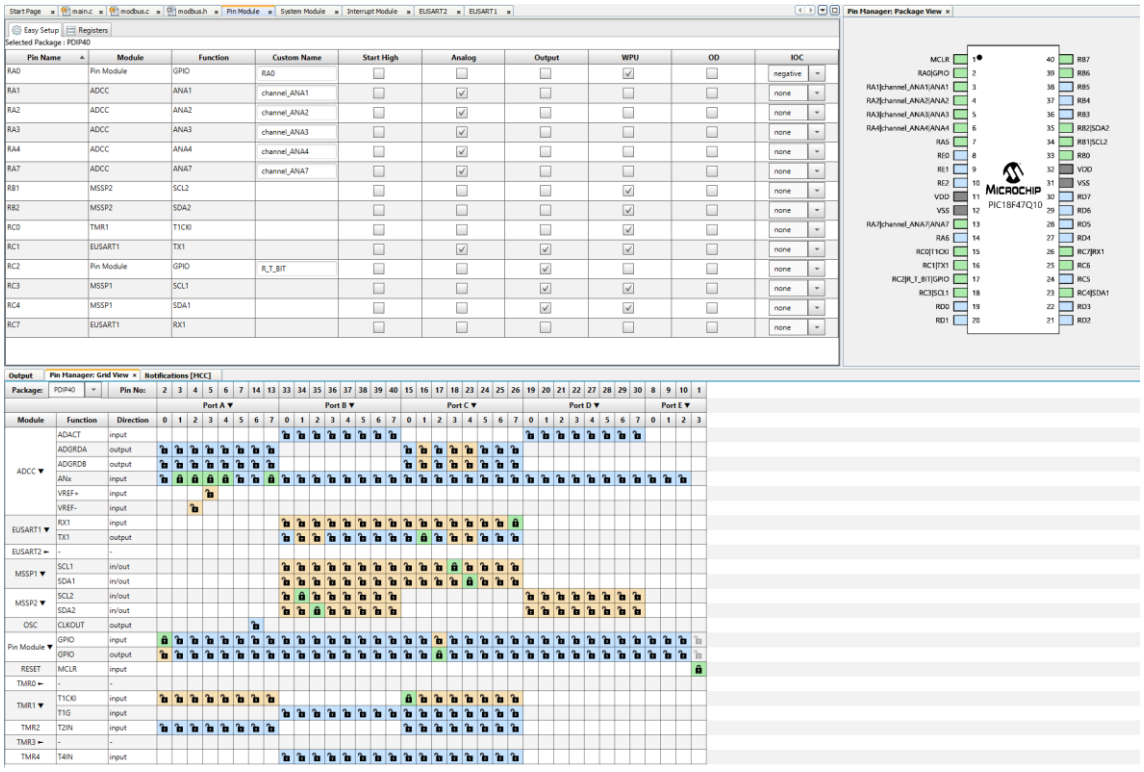

Figura 59: Configuració dels pins de la monitorització de l'estructura mecànica i del vent

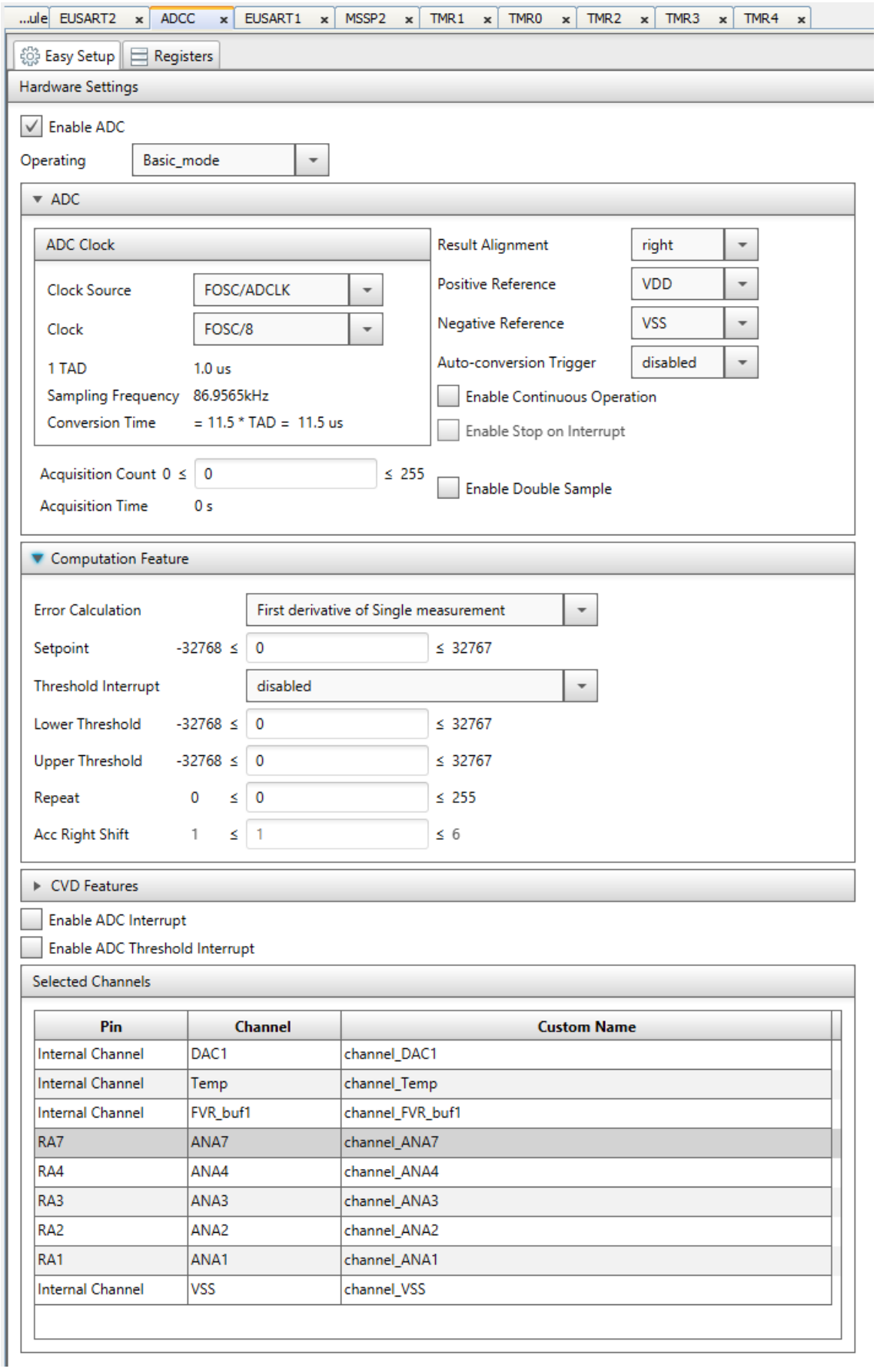

Figura 60: Configuració de l'ADC de la monitorització de l'estructura mecànica i del vent

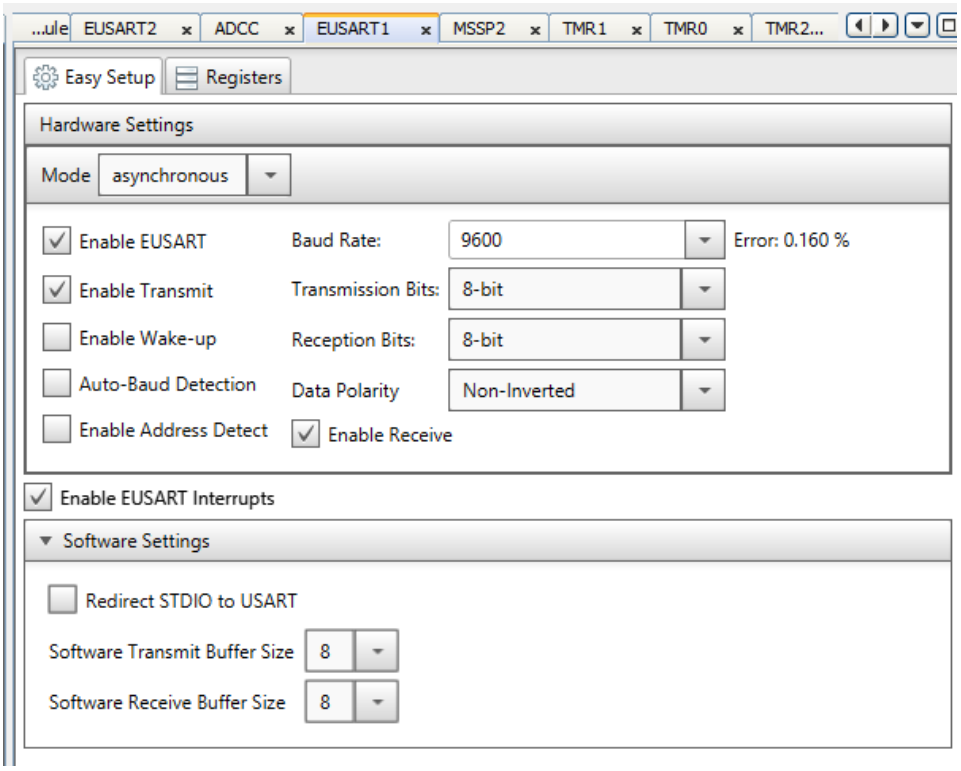

Figura 61: Configuració de l'EUSART1 de la monitorització de l'estructura mecànica i del vent

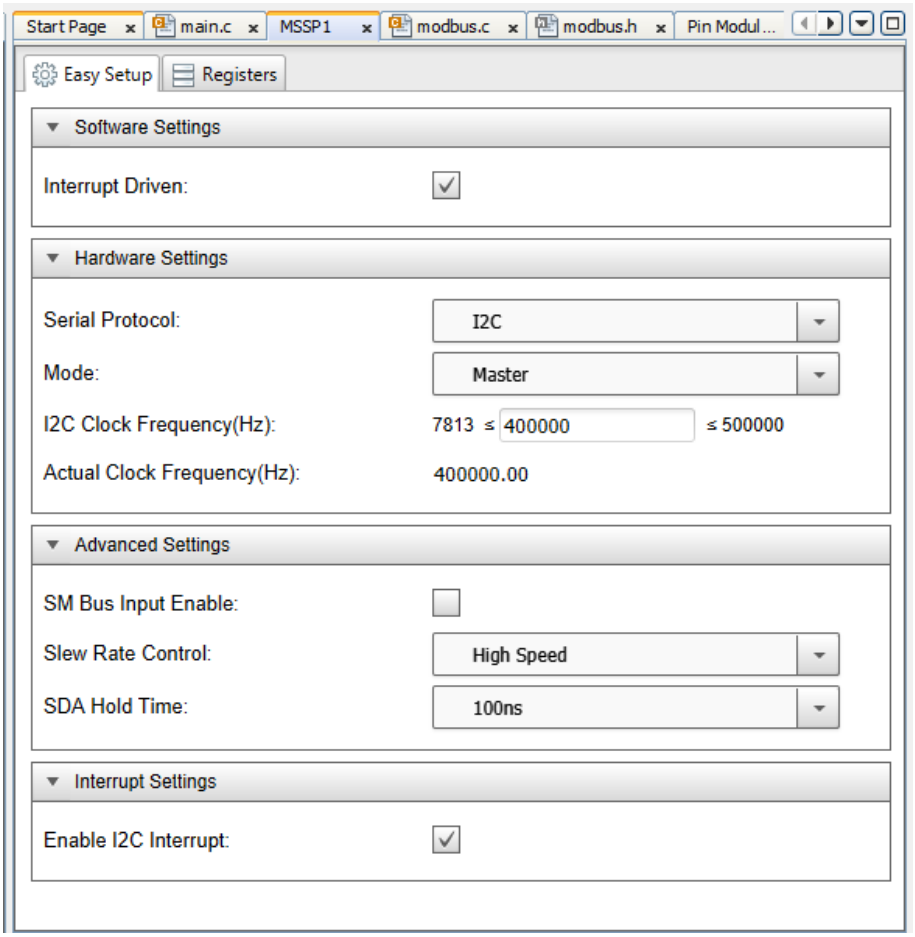

Figura 62: Configuració del MSSP1 de la monitorització de l'estructura mecànica i del vent

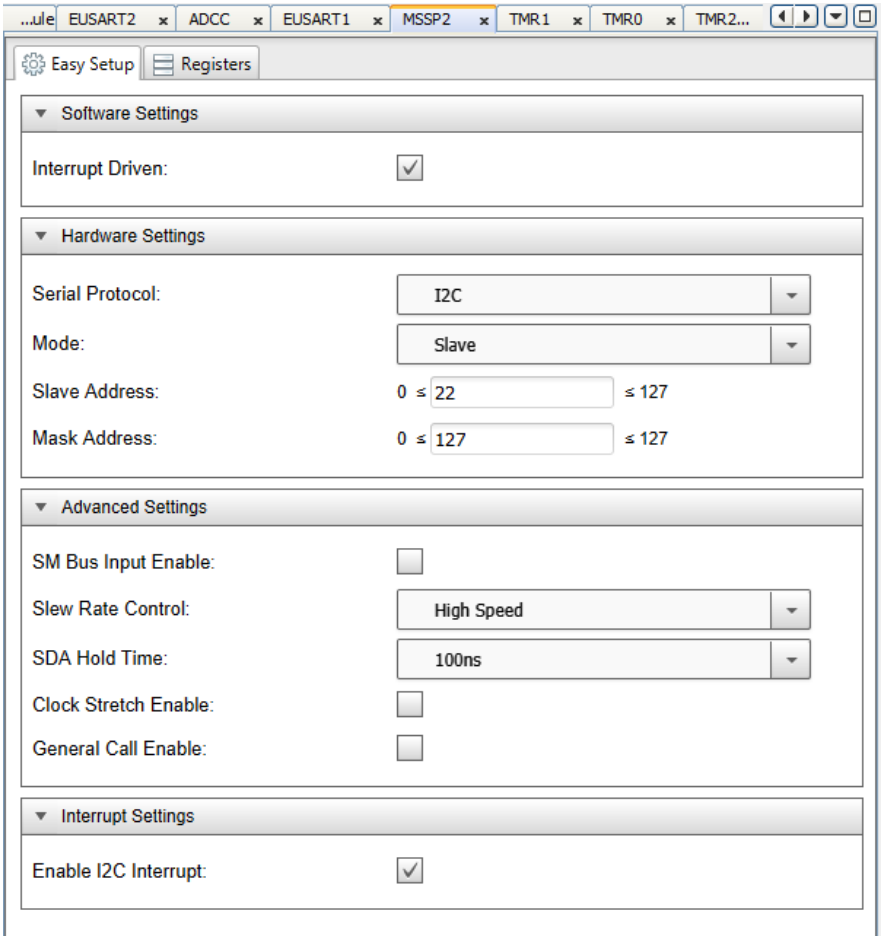

Figura 63: Configuració del MSSP2 de la monitorització de l'estructura mecànica i del vent

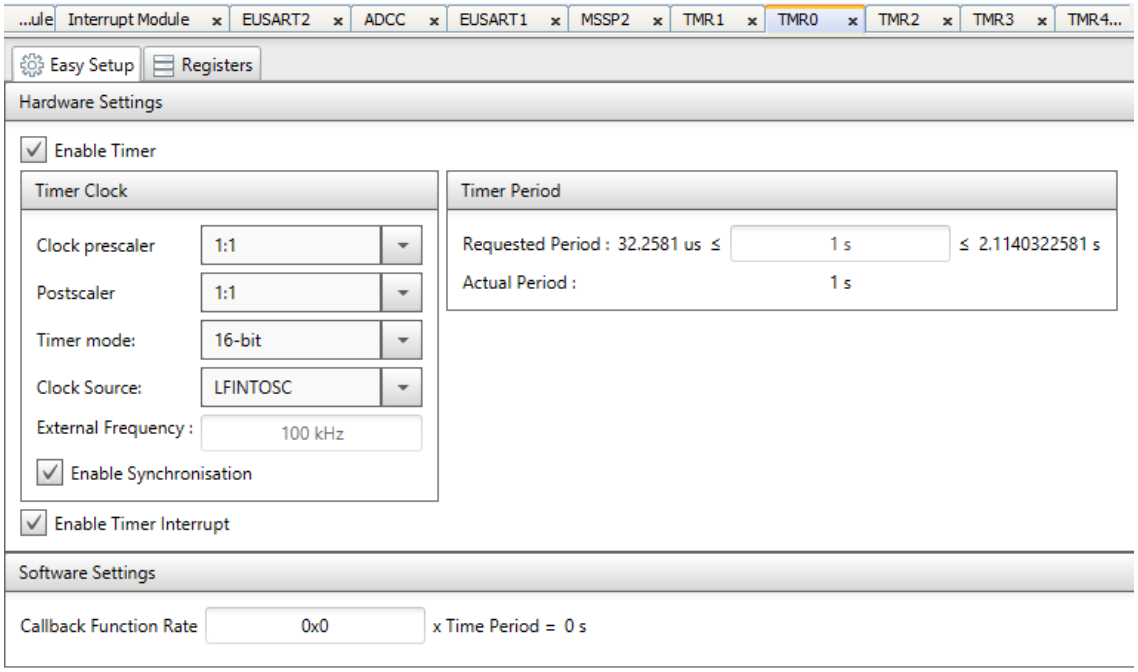

Figura 64: Configuració del TMR0 de la monitorització de l'estructura mecànica i del vent

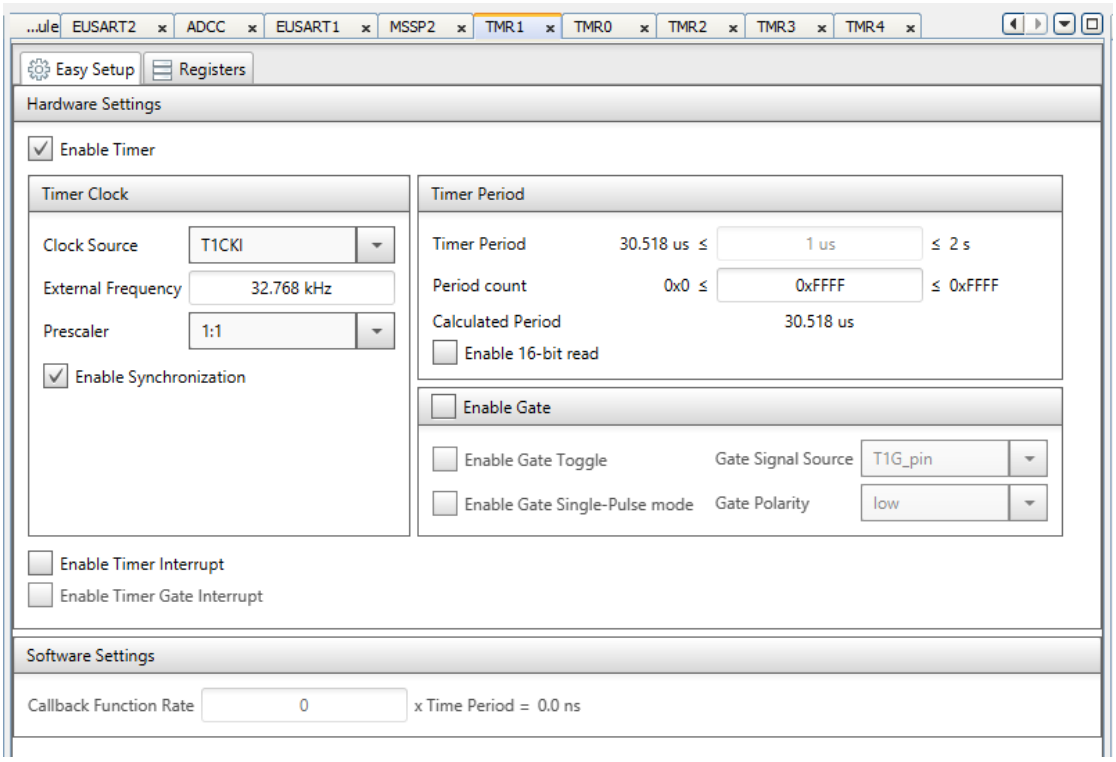

Figura 65: Configuració del TMR1 de la monitorització de l'estructura mecànica i del vent

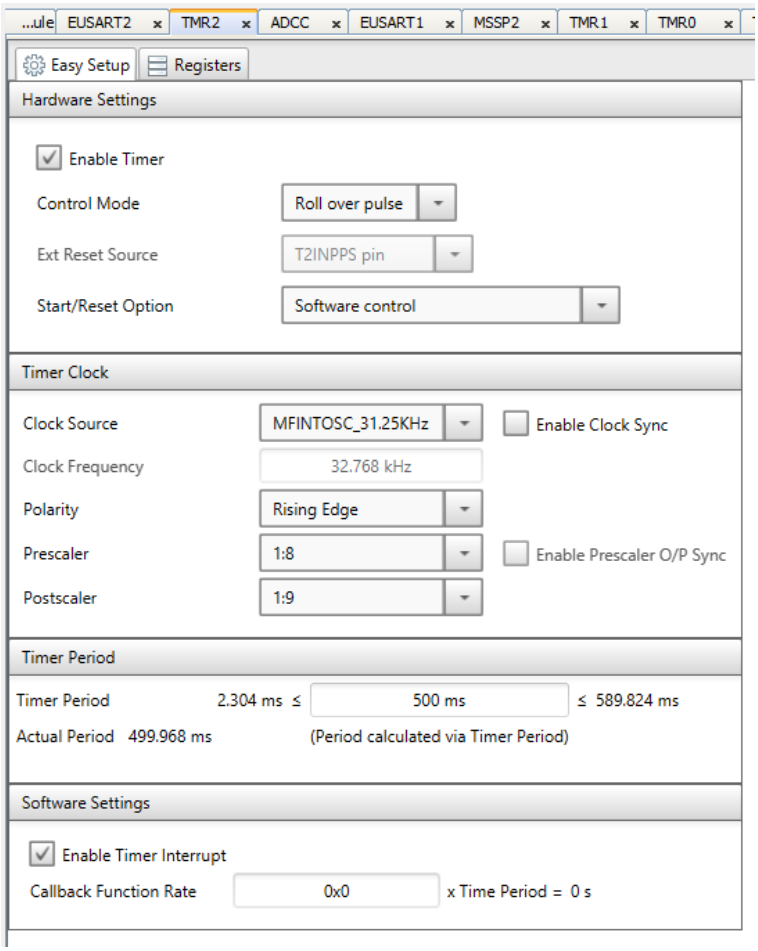

Figura 66: Configuració del TMR2 de la monitorització de l'estructura mecànica i del vent

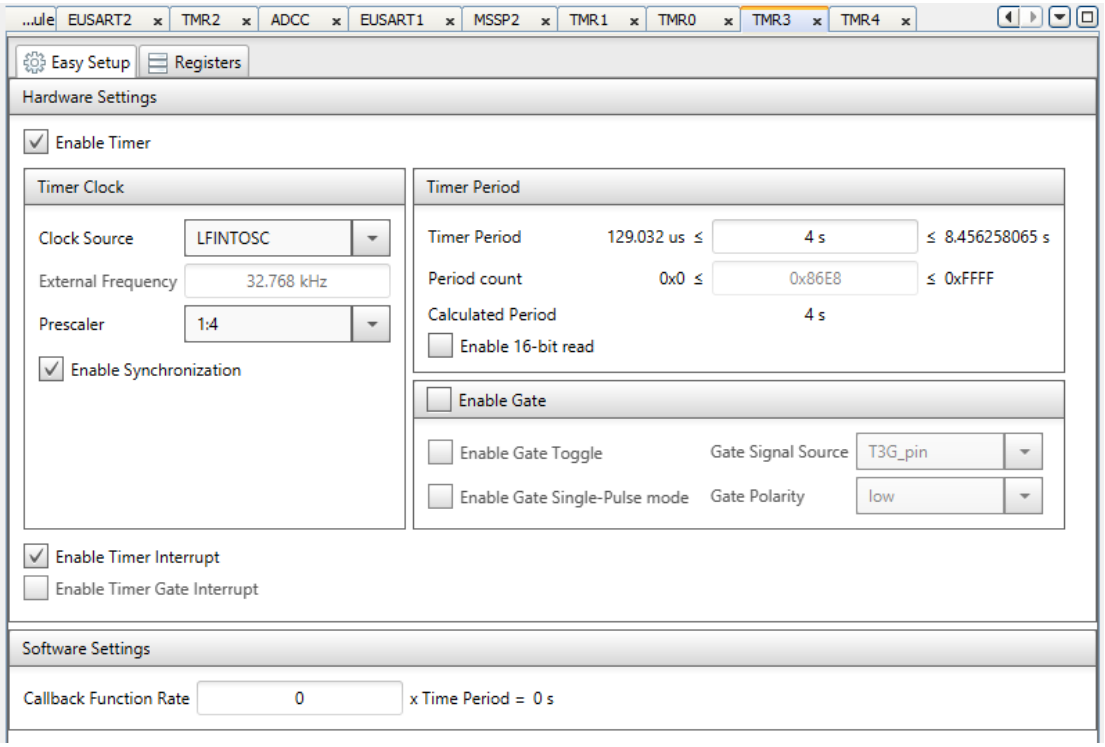

Figura 67: Configuració del TMR3 de la monitorització de l'estructura mecànica i del vent

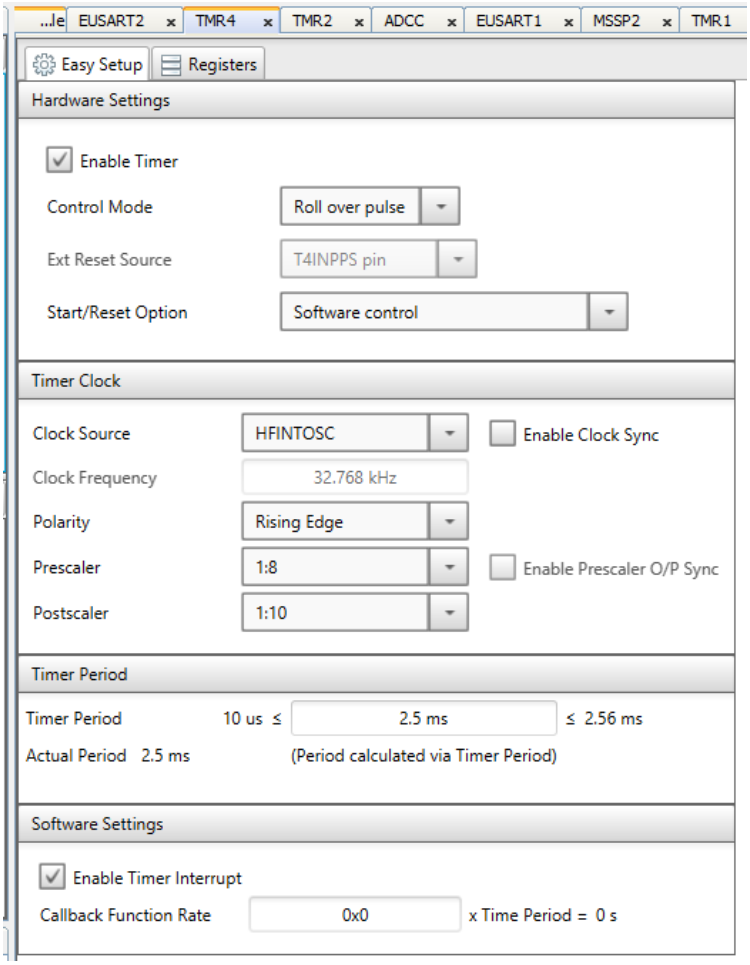

Figura 68: Configuració del TMR4 de la monitorització de l'estructura mecànica i del vent

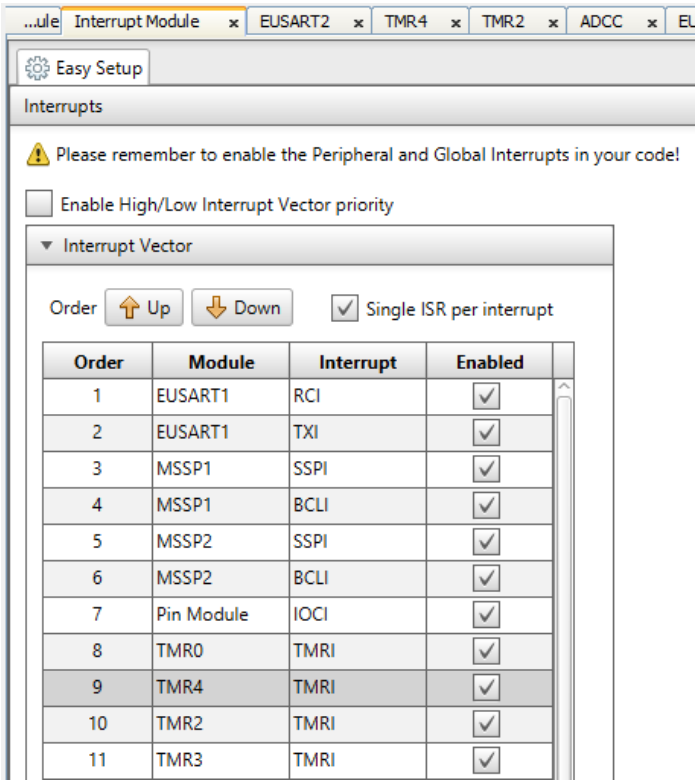

Figura 69: Configuració del vector d'interrupcions de la monitorització de l'estructura mecànica i del vent

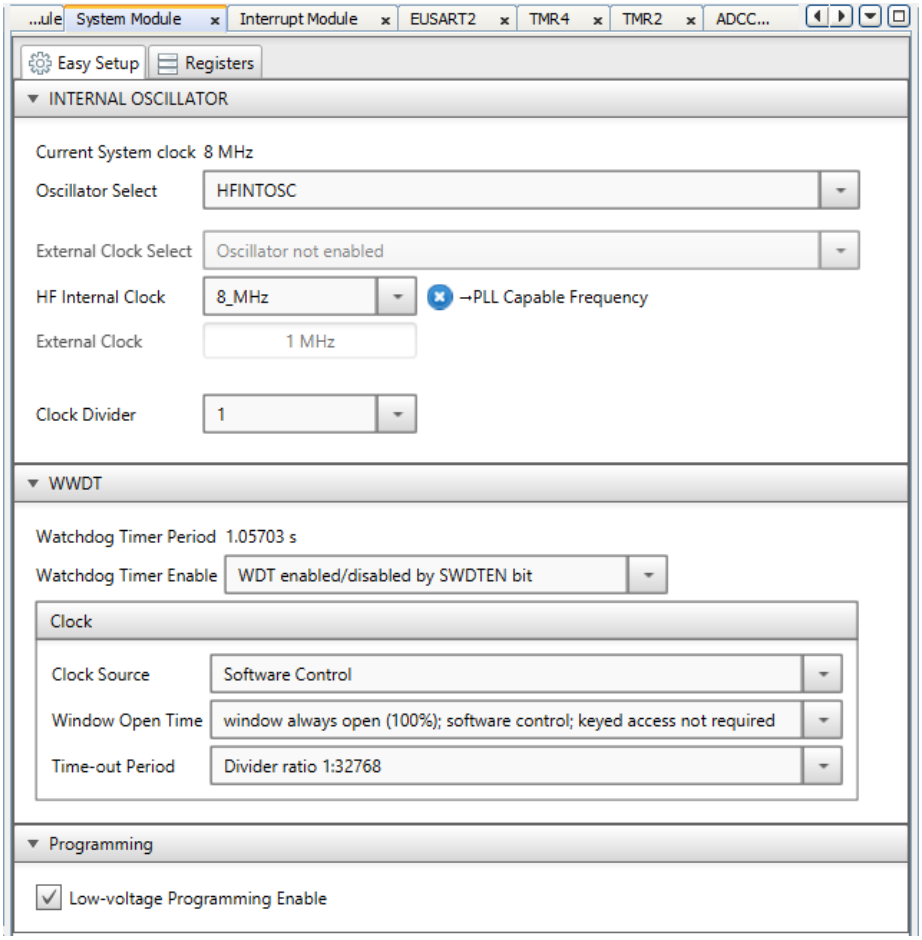

Figura 70: Configuració del mòdul del sistema de la monitorització de l'estructura mecànica i del vent

Un cop generada la configuració els arxius creats s'agrupen dins la carpeta mcc_generated_files.

B.2. Arxiu main.c

```
/*****************************************************************************
* Projecte              :   Monitorització de l'estructura mecànica i del vent
* Nom de l'arxiu        :   main.c
* Autor                 :   PC
* Data inicial          :   01/05/2023
* Versió                :   1.0.0
* Compilador            :   Microchip XC8
* Microprocessador      :   PIC18F47Q10
* Notes                 :   Cap
*****************************************************************************/

/*************** MODULE REVISION LOG ***************************************
*
*    Data    Versió de software  Inicials   Descripció
*  01/05/2023    1.0.0           PC         Creació del main
*
*****************************************************************************/

/** \file main.c
 * \brief Aquest programa realitza les funcions de monitorització de
 * l'estructura mecànica i del vent de l'aerogenerador del projecte llavor -
UdG
 */

/*****************************************************************************
* Includes
*****************************************************************************/

#include "mcc_generated_files/mcc.h"
#include "mcc_generated_files/pin_manager.h"
#include "i2c.h"
#include "modbus.h"


// PIC18F47Q10 Configuration Bit Settings
// 'C' source line config statements
#include <xc.h>
#include <pic18f47q10.h>
#include <stdio.h>


/*****************************************************************************
* Defines
```

```
*****************************************************************************/
#define ADRECA_ACCELEROMETRE 0x1C
#define ADRECA_CONTROLADOR_PITCH 0x08
#define ADRECA_CONTROLADOR_POTENCIA 0x28


/*****************************************************************************
* Global variables
*****************************************************************************/

//Variables MODBUS
uint8_t c_control=0;
uint8_t RxBuffer[20];
uint8_t rcv;

//Variables estat d'error
bool error_notificat;
const uint8_t codi_error = 0x2E;
enum ESTAT_ERROR_t estat_error;

//Variables estat emergència
enum ESTAT_EMERGENCIA_t estat_emergencia;
const uint8_t codi_alarma_vibracions = 0x2A;
const uint8_t codi_alarma_galgues = 0x2B;
const uint8_t codi_alarma_vent = 0x2C;

// CONFIG I2C
const uint8_t adreca_controlador_pitch_escriptura = ADRECA_CONTROLADOR_PITCH
<< 1;
const uint8_t adreca_controlador_pitch_lectura = (ADRECA_CONTROLADOR_PITCH <<
1) | 0x01;
const uint8_t adreca_controlador_potencia_escriptura =
ADRECA_CONTROLADOR_POTENCIA << 1;
const uint8_t adreca_controlador_potencia_lectura =
(ADRECA_CONTROLADOR_POTENCIA << 1) | 0x01;
uint8_t lecturaI2C;
uint8_t escripturaI2C;

//Variables mode test
const uint8_t codi_start_test = 0xBB;
uint8_t Test_ESCLAU1=0xE1;
uint8_t Test_ESCLAU3=0xE3;
_Bool recepcio_mode_test;
uint8_t comptador_test;


//Variables mode sleep
_Bool Generacio_ESCLAU1;
_Bool Generacio_ESCLAU2;
_Bool Generacio_ESCLAU3;
```

```
const uint8_t codi_base_generacio = 0x22;

//Variables acceleròmetre
const uint8_t adreca_acc_lectura = (ADRECA_ACCELEROMETRE << 1) | 0x01;
const uint8_t adreca_acc_escriptura = ADRECA_ACCELEROMETRE << 1;

int8_t LecturaX, LecturaY, LecturaZ;
int16_t SumaX, SumaY, SumaZ;
int8_t MitjanaX, MitjanaY, MitjanaZ;
float X, Y, Z;
uint8_t contador;
const uint8_t num_mostres = 4;
_Bool activar_lectura_acc;

_Bool Accelerometre_reinit;
uint8_t escala_max = 0; //0 --> 2g, 1 --> 4g , 2 --> 8g
_Bool filtre_pas_alt = false;
float acceleracio_max = 2; //nivell d'acceleració a partir del qual salta la
interrupció
int16_t Offset_X = 0; // en mg
int16_t Offset_Y = 0; // en mg
int16_t Offset_Z = 0; // en mg

//Variables galgues
float amp_galgues_estructura;  // en mV
float amp_galgues_cable1;  // en mV
float amp_galgues_cable2;  // en mV
float amp_galgues_cable3;  // en mV
uint16_t valor_max_galgues_estructura = 2600; // en mV
uint16_t valor_max_galgues_cable = 2800; // en mV
_Bool activar_lectura_galgues;

//Variables anemometre
uint16_t voltes_anemometre;
float vel_anemometre; // en m/s
float dir_anemometre; // en °
uint8_t limit_baix_anemometre = 5; // en m/s
uint8_t limit_alt_anemometre = 20; // en m/s
_Bool activar_lectura_anem;



/*****************************************************************************
* Functions
*****************************************************************************/
void DADES_REBUDES_EUSART(void){
    R_T_BIT_PORT = 0; //Deshabilita la direcció de transmissió del PIC
    rcv = EUSART1_Read();//Escriu la variable de recive.
    //Emmagatzema les dades en el buffer.
```

```
switch (c_control){
    case 0:
        if(rcv == SlaveAddress) {
            //ADREÇA
            RxBuffer[c_control] = rcv;
            c_control++;
        }
        break;
    case 1:
        if(rcv==0x04||rcv==0x10){
            //FUNCIÓ
            RxBuffer[c_control] = rcv;
            c_control++;
        }else c_control=0;
        break;
    case 2:
        RxBuffer[c_control] = rcv;
        c_control++;
        break;
    case 3:
        RxBuffer[c_control] = rcv;
        c_control++;
        break;
    case 4:
        RxBuffer[c_control] = rcv;
        c_control++;
        break;
    case 5:
        RxBuffer[c_control] = rcv;
        c_control++;
        break;
    case 6:
        RxBuffer[c_control] = rcv;
        c_control++;
        break;
    case 7:
        RxBuffer[c_control] = rcv;
        c_control++;
        if(RxBuffer[1]==0x04){
            modbus_function();
        }
        break;
    case 8:
        RxBuffer[c_control] = rcv;
        c_control++;
        break;
    case 9:
        RxBuffer[c_control] = rcv;
        c_control++;
```

```
            break;
        case 10:
            RxBuffer[c_control] = rcv;
            c_control++;
            break;
        case 11:
            RxBuffer[c_control] = rcv;
            c_control++;
            break;
        case 12:
            RxBuffer[c_control] = rcv;
            c_control++;
            break;
        case 13:
            RxBuffer[c_control] = rcv;
            c_control++;
            break;
        case 14:
            RxBuffer[c_control] = rcv;
            c_control++;
            break;
        case 15:
            RxBuffer[c_control] = rcv;
            modbus_function();
            break;
    }
}

void I2C_Slave_Read(void) {
    if(SSP2IF==1){

        //De la manera com marca el protocol l'esclau posa a zero el clock.
        SSP2CON1bits.CKP = 0;
         //El motiu és per garantir el temps de configuració de les dades.
        if ((SSP2CON1bits.SSPOV) || (SSP2CON1bits.WCOL))
        { //Overflow o col·lisió
            lecturaI2C = SSP2BUF; //Es borra el contingut del buffer.
            SSP2CON1bits.SSPOV = 0; // Es borra el bit d'OVERFLOW
            SSP2CON1bits.WCOL = 0;  // Es borra el bit de col·lisió.
            SSP2CON1bits.CKP = 1; //Es reinicialitza el rellotge.
        }

        if(!SSP2STATbits.D_nA && !SSP2STATbits.R_nW)
        { //El primer byte rebut coincideix amb l'adreça i està en mode
escriptura.
            SSP2CON2bits.ACKEN = 1; //S'habilita l'acknowledge
            lecturaI2C = SSP2BUF; //Es llegeix el contingut del buffer.
            SSP2CON1bits.CKP = 1; //Es reinicialitza el rellotge.
            switch (lecturaI2C){
```

```
                case 0x12:
                    Generacio_ESCLAU1 = false;
                    break;

                case 0x13:
                    Generacio_ESCLAU1 = true;
                    break;

                case 0xBB:
                    escripturaI2C=0xE2;
                    break;

                case 0xF0:
                    Generacio_ESCLAU3 = false;
                    break;

                case 0xF1:
                    Generacio_ESCLAU3 = true;
                    break;
            }
        }
        else if(!SSP2STATbits.D_nA && SSP2STATbits.R_nW)
        {//El primer byte rebut coincideix amb l'adreça i està en mode
lectura.
            SSP2CON2bits.ACKEN = 1; //S'habilita l'acknowledge
            lecturaI2C = SSP2BUF;//Es borra el buffer per ser escrit.
            /*Per evitar que es doni l'overflow, s'habilita el regitre BF per
             *   indicar que el buffer està preparat per enviar una altra
dada.*/
            SSP2STATbits.BF = 0;
            SSP2BUF = escripturaI2C; //S'escriu el byte a enviar en el buffer.
            //Es reincialitza el rellotge.
            SSP2CON1bits.CKP = 1;
            //S'espera fins que s'hagi enviat el byte cap al mestre.
            while(SSP2STATbits.BF);
        }
        SSP2IF = 0; //Es borra el flag d'interrupció.
    }
}

void TMR0_compareInterrupt(void);
void TMR2_compareInterrupt(void);
void TMR3_compareInterrupt(void);
void TMR4_compareInterrupt(void);
void IOC0_ISR(void);
_Bool Accelerometre_Init(void);
void lectura_accelerometre(void);
void lectura_galgues(void);
void lectura_anemometre(void);
void dormir_accelerometre(void);
```

```
void despertar_accelerometre(void);
void enviar_emergencia(void);
void enviar_error(void);
void canvi_generacio(void);

/******************************************************************************
* Main loop
******************************************************************************/

void main() {
    // Initialize the device
    SYSTEM_Initialize();

    // Enable the Global Interrupts
    INTERRUPT_GlobalInterruptEnable();

    // Enable the Peripheral Interrupts
    INTERRUPT_PeripheralInterruptEnable();

    if(!STATUSbits.TO && !PCON0bits.RWDT){
        estat_error = Error_WATCHDOG;
    }

    TMR1 = 0;
    TMR0_SetInterruptHandler(TMR0_compareInterrupt);
    TMR2_SetInterruptHandler(TMR2_compareInterrupt);
    TMR3_SetInterruptHandler(TMR3_compareInterrupt);
    TMR4_SetInterruptHandler(TMR4_compareInterrupt);
    IOCAF0_SetInterruptHandler(IOC0_ISR);
    EUSART1_SetRxInterruptHandler(DADES_REBUDES_EUSART);
    I2C2_SlaveSetIsrHandler(I2C_Slave_Read);
    I2C_Slave_Init(0x10); //Adreça d'esclau de la placa
    I2C_Master_Init(400000); //Inicialitzar I2C Master a 400KHz
    if (!Accelerometre_Init()) estat_error = Error_accelerometre;
    WDTCON0bits.SEN=1; //Habilitar el watchdog

    while (1) {
        CLRWDT(); //Reseteig periodic del watchdog
        if(Accelerometre_reinit){
            if (!Accelerometre_Init()) estat_error = Error_accelerometre;
            Accelerometre_reinit = false;
        }
        if (activar_lectura_acc && estat_error != Error_accelerometre)
lectura_accelerometre();
        if (activar_lectura_galgues) lectura_galgues();
        if (activar_lectura_anem) lectura_anemometre();
        if(estat_emergencia) enviar_emergencia();

        if(recepcio_mode_test){
```

```
            if(Test_ESCLAU1 != 0xE1 || Test_ESCLAU3 != 0xE3){
                comptador_test++;
                if (comptador_test == 3){
                    comptador_test = 0;
                    estat_error = Error_mode_test;
                }
            }else{
                comptador_test = 0;
            }
            recepcio_mode_test = false;
        }
        if(estat_error && !error_notificat) enviar_error();
        else if (!estat_error) error_notificat = false;

        if(!Generacio_ESCLAU1 && !Generacio_ESCLAU2 && !Generacio_ESCLAU3 &&
!activar_lectura_anem){
            dormir_accelerometre();
            WDTCON0bits.SEN=0; //Deshabilitar el watchdog perque no salti en
sleep
            TMR2_StopTimer();
            TMR3_StopTimer();
            TMR4_StopTimer();
            SLEEP();
        }
    }
}

void TMR0_compareInterrupt(void) { //1 s - LECTURA ANEMÒMETRE
    voltes_anemometre = TMR1;
    TMR1 = 0;
    activar_lectura_anem = true;
    WDTCON0bits.SEN=1; //Habilitar el watchdog en cas de wake up from sleep.
}

void IOC0_ISR(void) { //Interrupció per valor massa elevat d'acceleració
    if (IOCAFbits.IOCAF0 == 1) {
        estat_emergencia = ALARMA_VIBRACIONS;
    }
}

void TMR2_compareInterrupt(void) { //500 ms - LECTURA GALGUES
    activar_lectura_galgues = true;
}

void TMR3_compareInterrupt(void){ //4s - MODE TEST I2C
    //Activació del mode test - Sistema de control del pitch.
    I2C_Master_Start();         //Start condition
    I2C_Master_Write(adreca_controlador_pitch_escriptura);      //7 bit address
+ Write
    I2C_Master_Write(codi_start_test);
```

```
    I2C_Master_Stop();

    //Activació del mode test - Sistema de control de potència.
    I2C_Master_Start();         //Start condition
    I2C_Master_Write(adreca_controlador_potencia_escriptura);          //7  bit
address + Write
    I2C_Master_Write(codi_start_test);
    I2C_Master_Stop();

    //Lectura del mode test - Sistema de control del pitch.
    I2C_Master_Start();         //Start condition
    I2C_Master_Write(adreca_controlador_pitch_lectura);      //7 bit address +
Read
    Test_ESCLAU1 = I2C_Master_Read(0);
    I2C_Master_Stop();

    //Lectura del mode test - Sistema de control de potència.
    I2C_Master_Start();         //Start condition
    I2C_Master_Write(adreca_controlador_potencia_lectura);      //7 bit address
+ Read
    Test_ESCLAU3 = I2C_Master_Read(0);
    I2C_Master_Stop();

    recepcio_mode_test = true;
}

void TMR4_compareInterrupt(void) { //2.5 ms - LECTURA ACCELEROMETRE
        activar_lectura_acc = true;
}

_Bool Accelerometre_Init(void){  // Inicialització acceleròmetre
    //Apaguem accelerometre
    I2C_Master_Start(); //Start condition
    I2C_Master_Write(adreca_acc_escriptura);
    I2C_Master_Write(0x2A); //Adreça del registre CTRL_REG1
    I2C_Master_Write(0x00); //ASLP_RATE[1:0]--DR[2:0]--LNOISE--F_READ--ACTIVE
    I2C_Master_Stop(); //Stop condition

    //Desactivar o activar high-pass filter i configurar a 2g el max scale
    uint8_t XYZ_DATA_CFG = 0;
    if (filtre_pas_alt) XYZ_DATA_CFG|=0x10;
    if (escala_max < 3) XYZ_DATA_CFG|=escala_max;
    I2C_Master_Start(); //Start condition
    I2C_Master_Write(adreca_acc_escriptura);
    I2C_Master_Write(0x0E); //Adreça del registre XYZ_DATA_CFG Register
    I2C_Master_Write(XYZ_DATA_CFG); //000--HPF_OUT--00--FS[1:0]
    I2C_Master_Stop(); //Stop condition

    //Permet activar una flag quan es supera cert valor d'acceleració marcat
pel registre 0x17 (FF_MT_THS)
```

```
    I2C_Master_Start(); //Start condition
    I2C_Master_Write(adreca_acc_escriptura);
    I2C_Master_Write(0x15); //FF_MT_CFG Freefall/Motion Configuration Register
    I2C_Master_Write(0b11111000); //ELE--OAE--ZEFE--YEFE--XEFE--000
    I2C_Master_Stop(); //Stop condition

    //Activar interrupció motion CTRL_REG4
    I2C_Master_Start(); //Start condition
    I2C_Master_Write(adreca_acc_escriptura);
    I2C_Master_Write(0x2D); //Adreça del registre CTRL_REG4
    I2C_Master_Write(0b00000100); //INT_EN_FF_MT
    I2C_Master_Stop(); //Stop condition

    //Associar interrupció a pin INT1 CTRL_REG5
    I2C_Master_Start(); //Start condition
    I2C_Master_Write(adreca_acc_escriptura);
    I2C_Master_Write(0x2E); //Adreça del registre CTRL_REG5
    I2C_Master_Write(0b00000100); //INT_CFG_FF_MT
    I2C_Master_Stop(); //Stop condition


    //Engegar acceleròmetre, dades d'un byte i freq de 800hz CTRL_REG1
    I2C_Master_Start(); //Start condition
    I2C_Master_Write(adreca_acc_escriptura);
    I2C_Master_Write(0x2A); //Adreça del registre CTRL_REG1
    I2C_Master_Write(0x03); //ASLP_RATE[1:0]--DR[2:0]--LNOISE--F_READ--ACTIVE
    I2C_Master_Stop(); //Stop condition


    //Activar FIFO circular F_SETUP FIFO Setup Register
    I2C_Master_Start(); //Start condition
    I2C_Master_Write(adreca_acc_escriptura);
    I2C_Master_Write(0x09); //Adreça del registre F_SETUP FIFO Setup Register
    I2C_Master_Write(0b01000000); //F_MODE[1:0]--F_WMRK[5:0]
    I2C_Master_Stop(); //Stop condition

    //Configurar a 2g el nivell d'acceleració que activa el flag
    uint8_t motion_threshold;
    if(acceleracio_max<8){
        motion_threshold = acceleracio_max * 15.873; //resolucio=0.063g
    }
    I2C_Master_Start(); //Start condition
    I2C_Master_Write(adreca_acc_escriptura);
    I2C_Master_Write(0x17); //FF_MT_THS Freefall and Motion Threshold Register
    I2C_Master_Write(motion_threshold);//DBCNTM--THS[6:0]             default:
0b00100000
    I2C_Master_Stop(); //Stop condition

    //Configurar a 10ms el temps per el qual es supera el valor abans
d'activar la flag
```

```
    I2C_Master_Start(); //Start condition
    I2C_Master_Write(adreca_acc_escriptura);
    I2C_Master_Write(0x18);   //Adreça   del   registre   FF_MT_COUNT   Debounce
Register
    I2C_Master_Write(0b00000001); //D[7:0] (resolucio=10ms)
    I2C_Master_Stop(); //Stop condition

    //AJUSTAMENT D'OFFSET
    //Offset X
    int8_t correccio_offset_X;
    correccio_offset_X = Offset_X>>1; //resolucio 2mg
    I2C_Master_Start();  //Start condition
    I2C_Master_Write(adreca_acc_escriptura);
    I2C_Master_Write(0x2F); //OFF_X Offset Correction X Register
    I2C_Master_Write(correccio_offset_X); //D[7:0]  default: 0b00000000
    I2C_Master_Stop();   //Stop condition

    //Offset Y
    int8_t correccio_offset_Y;
    correccio_offset_Y = Offset_Y>>1; //resolucio 2mg
    I2C_Master_Start();         //Start condition
    I2C_Master_Write(adreca_acc_escriptura);
    I2C_Master_Write(0x30);     //OFF_Y Offset Correction Y Register
    I2C_Master_Write(correccio_offset_Y);     //D[7:0]  default: 0b00000000
    I2C_Master_Stop();          //Stop condition

    //Offset Z
    int8_t correccio_offset_Z;
    correccio_offset_Z = Offset_Z>>1; //resolucio 2mg
    I2C_Master_Start();         //Start condition
    I2C_Master_Write(adreca_acc_escriptura);
    I2C_Master_Write(0x31);     //OFF_Z Offset Correction Z Register
    I2C_Master_Write(correccio_offset_Z);     //D[7:0]  default: 0b00000000
    I2C_Master_Stop();          //Stop condition

    //Comprovació de comunicació
    I2C_Master_Start();
    I2C_Master_Write(adreca_acc_escriptura); //7 bit address + Write
    I2C_Master_Write(0x0D); //Registre WHO_AM_I
    I2C_Master_RepeatedStart();
    I2C_Master_Write(adreca_acc_lectura); //7 bit address + Read
    uint8_t ID = I2C_Master_Read(0); //Read + NotAcknowledge ID=0x1A?
    I2C_Master_Stop(); //Stop condition

    if (ID != 0x1A) return false;
    else return true;
}

void lectura_accelerometre(){ //TMR4 - 2.5ms
```

```
    I2C_Master_Start();
    I2C_Master_Write(adreca_acc_escriptura); //7 bit address + Write
    I2C_Master_Write(0x01); //Registre OUT_X_MSB
    I2C_Master_RepeatedStart();
    I2C_Master_Write(adreca_acc_lectura); //7 bit address + Read
    LecturaX = I2C_Master_Read(1); //Read + Acknowledge
    LecturaY = I2C_Master_Read(1); //Read + Acknowledge
    LecturaZ = I2C_Master_Read(0); //Read + NotAcknowledge
    I2C_Master_Stop(); //Stop condition

    if (contador < num_mostres) {
        SumaX += LecturaX;
        SumaY += LecturaY;
        SumaZ += LecturaZ;
        contador++;
    } else {
        MitjanaX = SumaX >> 2;
        MitjanaY = SumaY >> 2;
        MitjanaZ = SumaZ >> 2;
        switch (escala_max){
            case 0:
               X = MitjanaX * 15.6; //valor en mg
               Y = MitjanaY * 15.6; //valor en mg
               Z = MitjanaZ * 15.6; //valor en mg
               break;
            case 1:
               X = MitjanaX * 31.25; //valor en mg
               Y = MitjanaY * 31.25; //valor en mg
               Z = MitjanaZ * 31.25; //valor en mg
               break;
            case 2:
               X = MitjanaX * 62.5; //valor en mg
               Y = MitjanaY * 62.5; //valor en mg
               Z = MitjanaZ * 62.5; //valor en mg
               break;
        }
        SumaX = SumaY = SumaZ = contador = 0;
    }
    activar_lectura_acc = false;
}

void lectura_galgues(){ //TMR 2 - 500ms
    //0-1024 bits --> X * 5000/1024 mV/bits
    ADCC_DischargeSampleCapacitor();
    amp_galgues_estructura = ADCC_GetSingleConversion(channel_ANA1)*4.8828;
    //0-1024 bits --> X * 5000/1024 mV/bits
    ADCC_DischargeSampleCapacitor();
    amp_galgues_cable1 = ADCC_GetSingleConversion(channel_ANA2)*4.8828;
    //0-1024 bits --> X * 5000/1024 mV/bits
```

```
    ADCC_DischargeSampleCapacitor();
    amp_galgues_cable2 = ADCC_GetSingleConversion(channel_ANA3)*4.8828;
    //0-1024 bits --> X * 5000/1024 mV/bits
    ADCC_DischargeSampleCapacitor();
    amp_galgues_cable3 = ADCC_GetSingleConversion(channel_ANA4)*4.8828;
    if (amp_galgues_estructura > valor_max_galgues_estructura
            || amp_galgues_cable1 > valor_max_galgues_cable
            ||  amp_galgues_cable2 > valor_max_galgues_cable
            || amp_galgues_cable3 > valor_max_galgues_cable){
        estat_emergencia = ALARMA_GALGUES;
    }
    activar_lectura_galgues = false;
}

void lectura_anemometre(){ //TMR 0 - 1s
    ADCC_DischargeSampleCapacitor();
    uint16_t dir_anemometre_raw = ADCC_GetSingleConversion(channel_ANA7);
    dir_anemometre = dir_anemometre_raw * 360 / 1024;
    vel_anemometre = 0.44704 *(voltes_anemometre * (2.25 / 1)); // 0.44704
*(P*(2.25 /T)) m/s
    if((vel_anemometre<limit_baix_anemometre) && Generacio_ESCLAU2){
        Generacio_ESCLAU2 = false;
        canvi_generacio();
    }
    else if(vel_anemometre>limit_alt_anemometre){
        estat_emergencia = ALARMA_VENT;
    }
    else if (!Generacio_ESCLAU2){
        despertar_accelerometre();
        TMR2_StartTimer();
        TMR3_StartTimer();
        TMR4_StartTimer();
        Generacio_ESCLAU2 = true;
        canvi_generacio();
    }
    activar_lectura_anem = false;
}

void canvi_generacio(){
    //Enviament variable generacio_ESCLAU1 mode sleep - Sistema de
monitorització.
    I2C_Master_Start();         //Start condition
    I2C_Master_Write(adreca_controlador_pitch_escriptura);      //7 bit address
+ Write
    I2C_Master_Write(codi_base_generacio | Generacio_ESCLAU2);
    I2C_Master_Stop();
    //Enviament variable generacio_ESCLAU1 mode sleep - Sistema de control de
potència.
    I2C_Master_Start();         //Start condition
    I2C_Master_Write(adreca_controlador_potencia_escriptura);          //7  bit
address + Write
```

```
    I2C_Master_Write(codi_base_generacio | Generacio_ESCLAU2);
    I2C_Master_Stop();
}

void enviar_error(){
    //Activació del mode error - Sistema de control de potència.
    I2C_Master_Start();         //Start condition
    I2C_Master_Write(adreca_controlador_potencia_escriptura);          //7  bit
address + Write
    I2C_Master_Write(codi_error);
    I2C_Master_Stop();
    //Activació del mode error - Sistema de control de pitch.
    I2C_Master_Start(); //Start condition
    I2C_Master_Write(adreca_controlador_pitch_escriptura);  //7  bit  address  +
write
    I2C_Master_Write(codi_error);
    I2C_Master_Stop(); //Stop condition
    error_notificat = true;
}

void enviar_emergencia(){
    switch (estat_emergencia){
        case ALARMA_VIBRACIONS:
            I2C_Master_Start(); //Start condition
            I2C_Master_Write(adreca_controlador_potencia_escriptura);  //7  bit
address + write (desplaçar a l'esquerra i afegir un 0)
            I2C_Master_Write(codi_alarma_vibracions);
            I2C_Master_Stop(); //Stop condition
            I2C_Master_Start(); //Start condition
            I2C_Master_Write(adreca_controlador_pitch_escriptura);    //7    bit
address + write (desplaçar a l'esquerra i afegir un 0)
            I2C_Master_Write(codi_alarma_vibracions);
            I2C_Master_Stop(); //Stop condition
            break;

        case ALARMA_GALGUES:
            I2C_Master_Start(); //Start condition
            I2C_Master_Write(adreca_controlador_potencia_escriptura);  //7  bit
address + write (desplaçar a l'esquerra i afegir un 0)
            I2C_Master_Write(codi_alarma_galgues);
            I2C_Master_Stop(); //Stop condition
            I2C_Master_Start(); //Start condition
            I2C_Master_Write(adreca_controlador_pitch_escriptura);    //7    bit
address + write (desplaçar a l'esquerra i afegir un 0)
            I2C_Master_Write(codi_alarma_galgues);
            I2C_Master_Stop(); //Stop condition
            break;

        case ALARMA_VENT:
            I2C_Master_Start(); //Start condition
```

```
            I2C_Master_Write(adreca_controlador_potencia_escriptura);  //7  bit
address + write (desplaçar a l'esquerra i afegir un 0)
            I2C_Master_Write(codi_alarma_vent);
            I2C_Master_Stop(); //Stop condition
            I2C_Master_Start(); //Start condition
            I2C_Master_Write(adreca_controlador_pitch_escriptura);    //7    bit
address + write (desplaçar a l'esquerra i afegir un 0)
            I2C_Master_Write(codi_alarma_vent);
            I2C_Master_Stop(); //Stop condition
            break;
    }
    WDTCON0bits.SEN=0; //Deshabilitar el watchdog fins que no es rearmi el
sistema
    while (estat_emergencia){} //Bucle infinit fins que es rearmi el sistema a
través del modbus
    WDTCON0bits.SEN=1; //Habilitar el watchdog un cop rearmat

    //Maniobra per desactivar la interrupció d'acceleració elevada
    I2C_Master_Start();
    I2C_Master_Write(adreca_acc_escriptura); //7 bit address + Write
    I2C_Master_Write(0x16);  //Registre    FF_MT_SRC  Freefall/Motion  Source
Register
    I2C_Master_RepeatedStart();
    I2C_Master_Write(adreca_acc_lectura); //7 bit address + Read
    uint8_t InfoMax = I2C_Master_Read(0); //Read + NotAcknowledge
    I2C_Master_Stop();          //Stop condition
}

void dormir_accelerometre(){
    I2C_Master_Start(); //Start condition
    I2C_Master_Write(adreca_acc_escriptura);   //7   bit   address   +   write
(desplaçar a l'esquerra i afegir un 0)
    I2C_Master_Write(0x2A); //Adreça del registre CTRL_REG1
    I2C_Master_Write(0b00000000);   //ASLP_RATE[1:0]--DR[2:0]--LNOISE--F_READ--
ACTIVE
    I2C_Master_Stop(); //Stop condition
}

void despertar_accelerometre(){
    //Engegar accelerometre
    I2C_Master_Start(); //Start condition
    I2C_Master_Write(adreca_acc_escriptura);   //7   bit   address   +   write
(desplaçar a l'esquerra i afegir un 0)
    I2C_Master_Write(0x2A); //Adreça del registre CTRL_REG1
    I2C_Master_Write(0b00000011);   //ASLP_RATE[1:0]--DR[2:0]--LNOISE--F_READ--
ACTIVE
    I2C_Master_Stop(); //Stop condition
}
```

B.3. Arxiu i2c.c

```
/*******************************************************************************
* Títol                 :   Llibreria I2C
* Nom de l'arxiu        :   i2c.c
* Autors                :   AQ & PC
* Data inicial          :   08/04/2023
* Versió                :   1.0.0
* Compilador            :   Microchip XC8
* Microprocessador      :   PIC18F47Q10
* Notes                 :   Cap
*******************************************************************************/

/*************** MODULE REVISION LOG ***************************************
*
*    Data    Versió de software  Inicials   Descripció
*  08/04/2023    1.0.0           AQ & PC    Creació de les funcions
*
*******************************************************************************/
/** \file i2c.c
 * \brief Aquesta llibreria conté les funcions necessaries per la comunicació
en
 *  mode mestre i en mode esclau mitjançant el mòdul MSSP del microcontrolador
 */
/*******************************************************************************
* Includes
*******************************************************************************/
#include "mcc_generated_files/mcc.h"
#include "mcc_generated_files/pin_manager.h"

#include "i2c.h"
#include <xc.h>
#include <pic18f47q10.h>

/*******************************************************************************
* Constants i definicions
*******************************************************************************/
```

```
/*******************************************************************************
* Macros
*******************************************************************************/

/*******************************************************************************
* Typedefs
*******************************************************************************/

/*******************************************************************************
* Variables
*******************************************************************************/

/*******************************************************************************
* Definició de funcions
*******************************************************************************/

/*******************************************************************************
* Funció : I2C_Master_Init
*
* \section Descripció:
*
*  Aquesta funció inicialitza la comunicació en mode mestre.
*
* \param            c --> velocitat del rellotge del bus en kbits/s.
*
* \return           Cap.
*******************************************************************************/
void I2C_Master_Init(const unsigned long c)
{
    //Configuració del mòdul MSSP1 actuant com a  mestre de I2C
    SSP1CON1 = 0b00101000;
    SSP1CON2 = 0;
    //Definció del divisor del clock -> 400kbits/s
    SSP1ADD = (_XTAL_FREQ/(4*c))-1;
    //Es posa el ?slew rate? en mode d?alta freqüència
    SSP1STAT = 0;
```

```
    //Definició dels pins del canals de dades i de clock.
    SSP1CLKPPS = 0x13;   //RC3->MSSP1:SCL1;
    RC3PPS = 0x0F;   //RC3->MSSP1:SCL1;
    RC4PPS = 0x10;   //RC4->MSSP1:SDA1;
    SSP1DATPPS = 0x14;   //RC4->MSSP1:SDA1;
    //Es desactiven els pins fins que es doni una seqüència de start
    SSP1CON1bits.SSPEN = 0;
    //Es prefixa el temps de hold del bus SDA.
    SSP1CON3bits.SDAHT = 1;
}

/**************************************************************************
* Funció : I2C_Master_Wait
*
* \section Descripció:
*
*  Aquesta funció d'espera serveix per comprovar que no hi ha accions pendents
 * en el bus.
*
* \param            Cap.
*
* \return           Cap.
**************************************************************************/

void I2C_Master_Wait(void)
{
    //Funció que analitza si hi ha comunicació activa i si es el cas, espera
    while ((SSP1STAT & 0x04) || (SSP1CON2 & 0x1F));
}

/**************************************************************************
* Funció : I2C_Master_Start
*
* \section Descripció:
*
*  Aquesta funció envia la condició de "start" en el bus.
```

```
*
* \param            Cap.
*
* \return           Cap.
*******************************************************************************/

void I2C_Master_Start(void)
{
    //Rutina d'espera
    I2C_Master_Wait();
    //S'habiliten els pins.
    SSP1CON1bits.SSPEN = 1;
    //S'incina la seqüència del bit start.
    SSP1CON2bits.SEN = 1;
    //Quan acaba la seqüència aquest bit es posa a zero.
    while(SSP1CON2bits.SEN);
    //Fins que no es posi a zero la funció no acaba.
}

/*******************************************************************************
* Funció : I2C_Master_RepeatedStart
*
* \section Descripció:
*
*  Aquesta funció envia la condició de "repeated start" en el bus.
*
* \param            Cap.
*
* \return           Cap.
*******************************************************************************/

void I2C_Master_RepeatedStart(void)
{
    //Rutina d'espera
    I2C_Master_Wait();
    //S'incina la seqüència del bit de repeated start.
```

```
    SSP1CON2bits.RSEN = 1;
    //Quan acaba la seqüència aquest bit es posa a zero.
    while (SSP1CON2bits.RSEN);
    //Fins que no es posi a zero la funció no acaba.
}

/*****************************************************************************
* Funció : I2C_Master_Stop
*
* \section Descripció:
*
*  Aquesta funció envia la condició de "stop" en el bus.
*
* \param           Cap.
*
* \return          Cap.
*****************************************************************************/

void I2C_Master_Stop(void)
{
    //Rutina d'espera
    I2C_Master_Wait();
    //S'incina la seqüència del bit stop.
    SSP1CON2bits.PEN = 1;
    //Quan acaba la seqüència aquest bit es posa a zero.
    while(SSP1CON2bits.PEN);
    //Fins que no es posi a zero la funció no acaba.
}

/*****************************************************************************
* Funció : I2C_Master_Read
*
* \section Descripció:
*
*  Aquesta funció realitza una lectura del mestre a l'esclau
*
```

```
* \param            a --> Al finalitzar la lectura es transmet un ACK (1)
                       * o un NACK (0) des del mestre.
*
* \return           temp --> dades rebudes en el buffer (unsigned short).
*******************************************************************************/

unsigned short I2C_Master_Read(unsigned short a)
{
    unsigned short temp; //Variable de retorn
    I2C_Master_Wait(); //Espera del bus
    SSP1CON2bits.RCEN = 1; //habilita el bit de recepció
    I2C_Master_Wait(); //Espera que el bus rebi les dades.
    temp = SSP1BUF; //El valor del buffer queda volcat a la variable.
    I2C_Master_Wait(); //Espera del bus
    SSP1CON2bits.ACKDT = (a)?0:1;/*Depenent del valor de la variable a, el
                                  * mestre retorna un ACK per continuar amb la
                                  * comunicació o un NACK per finalitzar-la.*/
    SSP1CON2bits.ACKEN = 1; //Habilita l'Acknowledge.
    return temp; //Retorna la variable.
}

/*******************************************************************************
* Funció : I2C_Master_Write
*
* \section Descripció:
*
*  Aquesta funció realitza una escriptura del mestre a l'esclau
*
* \param            d --> Dades a transmetre en el buffer.
*
* \return           Cap.
*******************************************************************************/

void I2C_Master_Write(unsigned d)
{
    //Acció d'espera
```

```
    I2C_Master_Wait();
    //Escriure en el buffer.
    SSP1BUF = d;
    //Esperar que es buidi el buffer i de rebre l'Acknowledge
    while(SSP1STATbits.R_nW);
}

/*****************************************************************************
* Funció : I2C1_MasterIsNack
*
* \section Descripció:
*
*  Aquesta funció permet saber si el mestre es troba en ACK o NACK
*
* \param           Cap.
*
* \return          Estat del mestre (bool).
*****************************************************************************/

bool I2C1_MasterIsNack(void)
{
    return SSP1CON2bits.ACKSTAT;
}

/*****************************************************************************
* Funció : I2C_Slave_Init
*
* \section Descripció:
*
*  Aquesta funció inicialitza la comunicació en mode esclau.
*
* \param           address --> Configura el mòdul I2C amb l'adreça
                              *d'esclau seleccionada.
*
* \return          Cap.
*****************************************************************************/
```

```
void I2C_Slave_Init(short address)
{
    //Activar "slew rate" a 400kbits/s
    SSP2STAT = 0x00;
    //Definir l'adreça de l'esclau
    SSP2ADD = address;
    //Configurar en mode esclau i habilitar els pins
    SSP2CON1 = 0x36;
    SSP2CON2 = 0x01;

    //Associar SDA i SCL a pins físics del micro
    SSP2DATPPS = 0x0A;   //RB2->MSSP2:SDA2;
    RB1PPS = 0x11;   //RB1->MSSP2:SCL2;
    RB2PPS = 0x12;   //RB2->MSSP2:SDA2;
    SSP2CLKPPS = 0x09;   //RB1->MSSP2:SCL2;
    SSP2IF = 0;        //Netejar la flag d'interrupcions
    SSP2IE = 1;        //Habilitar interrupcions del port síncron
}


/***************************** END OF FUNCTIONS *****************************/
```

B.4. Arxiu i2c.h

```
/*******************************************************************************
* Títol                 :   Llibreria I2C
* Nom de l'arxiu        :   i2c.h
* Autors                :   AQ & PC
* Data inicial          :   08/04/2023
* Versió                :   1.0.0
* Compilador            :   Microchip XC8
* Microprocessador      :   PIC18F47Q10
* Notes                 :   Cap
*******************************************************************************/
/*************** INTERFACE CHANGE LIST ************************************
*
*    Data    Versió de software  Inicials   Descripció
*  08/04/2023    1.0.0           AQ & PC    Creació de la llibreria
*
*******************************************************************************/
/** \file i2c.h
 *   \brief Aquest mòdul conté les funcions de l'I2C a implementar en el
programa.
```

```
 */
#ifndef I2C_H
#define I2C_H

/*******************************************************************************
* Includes
*******************************************************************************/
#include "mcc_generated_files/mcc.h"
#include "mcc_generated_files/pin_manager.h"

#include <xc.h>
#include <pic18f47q10.h>

/*******************************************************************************
* Constants de preprocessador
*******************************************************************************/


/*******************************************************************************
* Constants de configuració
*******************************************************************************/


/*******************************************************************************
* Macros
*******************************************************************************/


/*******************************************************************************
* Typedefs
*******************************************************************************/


/*******************************************************************************
* Variables
*******************************************************************************/


/*******************************************************************************
* Prototips de funcions
*******************************************************************************/
#ifdef __cplusplus
extern "C" {
#endif


void I2C_Master_Init(const unsigned long c);
void I2C_Master_Wait(void);
void I2C_Master_Start(void);
```

```
void I2C_Master_RepeatedStart(void);
void I2C_Master_Stop(void);
unsigned short I2C_Master_Read(unsigned short a);
void I2C_Master_Write(unsigned d);
static inline bool I2C1_MasterIsNack(void);
void I2C_Slave_Init(short address);

#endif /* I2C_H */

/*** End of File **************************************************************/
```

B.5. Arxiu modbus.c

```
/*
 * File:   modbus.c
 *
 * Created on February 23, 2022, 10:53 PM
 */


#include "mcc_generated_files/mcc.h"
#include "modbus.h"
#include <stdio.h>
#include <pic18f47q10.h>
#include <xc.h>


bool LECTURA_DEL_uint16_auto[16];
bool LECTURA_DEL_uint16_test[16];


//Gobal variables
uint8_t response[20];
uint8_t error;
bool frame;
uint16_t comprovacio=0;
//Funcions Modbus
typedef enum functions{READ_COILS = 0x01, //mode d'accés a bit
READ_INPUT_REGISTERS = 0x04,
WRITE_MULTIPLE_REGISTERS = 0x10,
}function;
//Definim les diferents excepcions que hi ha en el protocol
typedef enum exceptions{
    ILLEGAL_FUNCTION = 0x01,
    ILLEGAL_DATA_ADDRESS = 0x02,
    ILLEGAL_DATA_VALUE = 0x03,
}exception;
//LUT (Look Up Table) CRC sol·licitud
```

```
static uint16_t MODBUS_CRC16_RX(uint8_t len)
{
    static const uint16_t table[256] = {
    0x0000, 0xC0C1, 0xC181, 0x0140, 0xC301, 0x03C0, 0x0280, 0xC241,
    0xC601, 0x06C0, 0x0780, 0xC741, 0x0500, 0xC5C1, 0xC481, 0x0440,
    0xCC01, 0x0CC0, 0x0D80, 0xCD41, 0x0F00, 0xCFC1, 0xCE81, 0x0E40,
    0x0A00, 0xCAC1, 0xCB81, 0x0B40, 0xC901, 0x09C0, 0x0880, 0xC841,
    0xD801, 0x18C0, 0x1980, 0xD941, 0x1B00, 0xDBC1, 0xDA81, 0x1A40,
    0x1E00, 0xDEC1, 0xDF81, 0x1F40, 0xDD01, 0x1DC0, 0x1C80, 0xDC41,
    0x1400, 0xD4C1, 0xD581, 0x1540, 0xD701, 0x17C0, 0x1680, 0xD641,
    0xD201, 0x12C0, 0x1380, 0xD341, 0x1100, 0xD1C1, 0xD081, 0x1040,
    0xF001, 0x30C0, 0x3180, 0xF141, 0x3300, 0xF3C1, 0xF281, 0x3240,
    0x3600, 0xF6C1, 0xF781, 0x3740, 0xF501, 0x35C0, 0x3480, 0xF441,
    0x3C00, 0xFCC1, 0xFD81, 0x3D40, 0xFF01, 0x3FC0, 0x3E80, 0xFE41,
    0xFA01, 0x3AC0, 0x3B80, 0xFB41, 0x3900, 0xF9C1, 0xF881, 0x3840,
    0x2800, 0xE8C1, 0xE981, 0x2940, 0xEB01, 0x2BC0, 0x2A80, 0xEA41,
    0xEE01, 0x2EC0, 0x2F80, 0xEF41, 0x2D00, 0xEDC1, 0xEC81, 0x2C40,
    0xE401, 0x24C0, 0x2580, 0xE541, 0x2700, 0xE7C1, 0xE681, 0x2640,
    0x2200, 0xE2C1, 0xE381, 0x2340, 0xE101, 0x21C0, 0x2080, 0xE041,
    0xA001, 0x60C0, 0x6180, 0xA141, 0x6300, 0xA3C1, 0xA281, 0x6240,
    0x6600, 0xA6C1, 0xA781, 0x6740, 0xA501, 0x65C0, 0x6480, 0xA441,
    0x6C00, 0xACC1, 0xAD81, 0x6D40, 0xAF01, 0x6FC0, 0x6E80, 0xAE41,
    0xAA01, 0x6AC0, 0x6B80, 0xAB41, 0x6900, 0xA9C1, 0xA881, 0x6840,
    0x7800, 0xB8C1, 0xB981, 0x7940, 0xBB01, 0x7BC0, 0x7A80, 0xBA41,
    0xBE01, 0x7EC0, 0x7F80, 0xBF41, 0x7D00, 0xBDC1, 0xBC81, 0x7C40,
    0xB401, 0x74C0, 0x7580, 0xB541, 0x7700, 0xB7C1, 0xB681, 0x7640,
    0x7200, 0xB2C1, 0xB381, 0x7340, 0xB101, 0x71C0, 0x7080, 0xB041,
    0x5000, 0x90C1, 0x9181, 0x5140, 0x9301, 0x53C0, 0x5280, 0x9241,
    0x9601, 0x56C0, 0x5780, 0x9741, 0x5500, 0x95C1, 0x9481, 0x5440,
    0x9C01, 0x5CC0, 0x5D80, 0x9D41, 0x5F00, 0x9FC1, 0x9E81, 0x5E40,
    0x5A00, 0x9AC1, 0x9B81, 0x5B40, 0x9901, 0x59C0, 0x5880, 0x9841,
    0x8801, 0x48C0, 0x4980, 0x8941, 0x4B00, 0x8BC1, 0x8A81, 0x4A40,
    0x4E00, 0x8EC1, 0x8F81, 0x4F40, 0x8D01, 0x4DC0, 0x4C80, 0x8C41,
    0x4400, 0x84C1, 0x8581, 0x4540, 0x8701, 0x47C0, 0x4680, 0x8641,
    0x8201, 0x42C0, 0x4380, 0x8341, 0x4100, 0x81C1, 0x8081, 0x4040};
    uint8_t index = 0;
    uint16_t crc = 0xFFFF;
    for(uint8_t j = 0; j < len; j++){
        index = (RxBuffer[j]) ^ crc;
        crc >>= 8;
        crc ^= table[index];
    }
    return crc;
}
//LUT (Look Up Table) CRC resposta
static uint16_t MODBUS_CRC16_TX(uint8_t len){
    static const uint16_t table[256] = {
    0x0000, 0xC0C1, 0xC181, 0x0140, 0xC301, 0x03C0, 0x0280, 0xC241,
    0xC601, 0x06C0, 0x0780, 0xC741, 0x0500, 0xC5C1, 0xC481, 0x0440,
```

```
    0xCC01, 0x0CC0, 0x0D80, 0xCD41, 0x0F00, 0xCFC1, 0xCE81, 0x0E40,
    0x0A00, 0xCAC1, 0xCB81, 0x0B40, 0xC901, 0x09C0, 0x0880, 0xC841,
    0xD801, 0x18C0, 0x1980, 0xD941, 0x1B00, 0xDBC1, 0xDA81, 0x1A40,
    0x1E00, 0xDEC1, 0xDF81, 0x1F40, 0xDD01, 0x1DC0, 0x1C80, 0xDC41,
    0x1400, 0xD4C1, 0xD581, 0x1540, 0xD701, 0x17C0, 0x1680, 0xD641,
    0xD201, 0x12C0, 0x1380, 0xD341, 0x1100, 0xD1C1, 0xD081, 0x1040,
    0xF001, 0x30C0, 0x3180, 0xF141, 0x3300, 0xF3C1, 0xF281, 0x3240,
    0x3600, 0xF6C1, 0xF781, 0x3740, 0xF501, 0x35C0, 0x3480, 0xF441,
    0x3C00, 0xFCC1, 0xFD81, 0x3D40, 0xFF01, 0x3FC0, 0x3E80, 0xFE41,
    0xFA01, 0x3AC0, 0x3B80, 0xFB41, 0x3900, 0xF9C1, 0xF881, 0x3840,
    0x2800, 0xE8C1, 0xE981, 0x2940, 0xEB01, 0x2BC0, 0x2A80, 0xEA41,
    0xEE01, 0x2EC0, 0x2F80, 0xEF41, 0x2D00, 0xEDC1, 0xEC81, 0x2C40,
    0xE401, 0x24C0, 0x2580, 0xE541, 0x2700, 0xE7C1, 0xE681, 0x2640,
    0x2200, 0xE2C1, 0xE381, 0x2340, 0xE101, 0x21C0, 0x2080, 0xE041,
    0xA001, 0x60C0, 0x6180, 0xA141, 0x6300, 0xA3C1, 0xA281, 0x6240,
    0x6600, 0xA6C1, 0xA781, 0x6740, 0xA501, 0x65C0, 0x6480, 0xA441,
    0x6C00, 0xACC1, 0xAD81, 0x6D40, 0xAF01, 0x6FC0, 0x6E80, 0xAE41,
    0xAA01, 0x6AC0, 0x6B80, 0xAB41, 0x6900, 0xA9C1, 0xA881, 0x6840,
    0x7800, 0xB8C1, 0xB981, 0x7940, 0xBB01, 0x7BC0, 0x7A80, 0xBA41,
    0xBE01, 0x7EC0, 0x7F80, 0xBF41, 0x7D00, 0xBDC1, 0xBC81, 0x7C40,
    0xB401, 0x74C0, 0x7580, 0xB541, 0x7700, 0xB7C1, 0xB681, 0x7640,
    0x7200, 0xB2C1, 0xB381, 0x7340, 0xB101, 0x71C0, 0x7080, 0xB041,
    0x5000, 0x90C1, 0x9181, 0x5140, 0x9301, 0x53C0, 0x5280, 0x9241,
    0x9601, 0x56C0, 0x5780, 0x9741, 0x5500, 0x95C1, 0x9481, 0x5440,
    0x9C01, 0x5CC0, 0x5D80, 0x9D41, 0x5F00, 0x9FC1, 0x9E81, 0x5E40,
    0x5A00, 0x9AC1, 0x9B81, 0x5B40, 0x9901, 0x59C0, 0x5880, 0x9841,
    0x8801, 0x48C0, 0x4980, 0x8941, 0x4B00, 0x8BC1, 0x8A81, 0x4A40,
    0x4E00, 0x8EC1, 0x8F81, 0x4F40, 0x8D01, 0x4DC0, 0x4C80, 0x8C41,
    0x4400, 0x84C1, 0x8581, 0x4540, 0x8701, 0x47C0, 0x4680, 0x8641,
    0x8201, 0x42C0, 0x4380, 0x8341, 0x4100, 0x81C1, 0x8081, 0x4040 };
    uint8_t index = 0;
    uint16_t crc = 0xFFFF;
    for(uint8_t j = 0; j < len; j++){
        index = (response[j]) ^ crc;
        crc >>= 8;
        crc ^= table[index];
    }
    return crc;
}
//CRC 8 bits low byte
uint8_t CRC8_Lo (uint16_t crc16){
    uint8_t crc_low = crc16 | 0;
    return crc_low;
}
//CRC 8 bits high byte
uint8_t CRC8_Hi(uint16_t crc16){
    uint8_t crc_high = (crc16 - CRC8_Lo(crc16)) >> 8;
    return crc_high;
}
```

```
//Comprovació que la trama llegida i calculada de les dades que rebem són
iguals
bool checkCRC(void){
    if(RxBuffer[1] == READ_COILS || RxBuffer[1] == READ_INPUT_REGISTERS){
        uint16_t crc16;
        uint8_t crc_Hi, crc_Lo;
        crc16 = MODBUS_CRC16_RX(6);
        crc_Hi = CRC8_Hi(crc16);
        crc_Lo = CRC8_Lo(crc16);
        if(RxBuffer[7] == crc_Hi && RxBuffer[6] == crc_Lo){
            comprovacio=1;
            return true;
        }else{
            return false;
        }
    }
    else if(RxBuffer[1] == WRITE_MULTIPLE_REGISTERS){
        uint16_t crc16;
        uint8_t crc_Hi, crc_Lo;
        crc16 = MODBUS_CRC16_RX(13);
        crc_Hi = CRC8_Hi(crc16);
        crc_Lo = CRC8_Lo(crc16);
        if(RxBuffer[14] == crc_Hi && RxBuffer[13] == crc_Lo){
            return true;
        }else{
            return false;
        }
    }
}

void error_function(void){
    if(RxBuffer[1] == READ_COILS){
        uint16_t crc16;
        crc16 = MODBUS_CRC16_TX(3);
        response[0] = SlaveAddress;
        response[1] = 0x80 + READ_COILS;
        response[2] = error;
        response[4] = CRC8_Hi(crc16);
        response[3] = CRC8_Lo(crc16);
        for(uint8_t j = 0; j < 5; j++){
            EUSART1_Write(response[j]);
        }
    }
    else if(RxBuffer[1] == READ_INPUT_REGISTERS){
        uint16_t crc16;
        crc16 = MODBUS_CRC16_TX(3);
        response[0] = SlaveAddress;
        response[1] = 0x80 + READ_INPUT_REGISTERS;
        response[2] = error;
```

```
        response[4] = CRC8_Hi(crc16);
        response[3] = CRC8_Lo(crc16);
        for(uint8_t j = 0; j < 5; j++){
            EUSART1_Write(response[j]);
        }
    }
    else if(RxBuffer == WRITE_MULTIPLE_REGISTERS){
        uint16_t crc16;
        crc16 = MODBUS_CRC16_TX(3);
        response[0] = SlaveAddress;
        response[1] = 0x80 + WRITE_MULTIPLE_REGISTERS;
        response[2] = error;
        response[4] = CRC8_Hi(crc16);
        response[3] = CRC8_Lo(crc16);
        for(uint8_t j = 0; j < 5; j++){
            EUSART1_Write(response[j]);
        }
    }
}
void modbus_read_input_registers(uint16_t reg_num){
    uint8_t starting_address_Lo, starting_address_Hi, reg_num_Hi, reg_num_Lo,
    CRC_read_Hi, CRC_read_Lo,
    j, k = 0;
    //LA VARIABLE INPUTREGISTER ES LA QUE ES TRANSMET
    uint16_t crc16, adc_result, starting_address;
    uint16_t inputregister = 0;
    frame = 1;
    starting_address_Hi = RxBuffer[2];
    starting_address_Lo = RxBuffer[3];
    reg_num_Hi = RxBuffer[4];
    reg_num_Lo = RxBuffer[5];
    CRC_read_Hi = RxBuffer[6];
    CRC_read_Lo = RxBuffer[7];
    if(checkCRC() == true){
        //Passem els 2 bytes de 8 bits en un numero de 16 bits
        starting_address = starting_address_Hi;
        starting_address = starting_address << 8;
        starting_address = starting_address | starting_address_Lo;
        reg_num = reg_num_Hi;
        reg_num = reg_num << 8;
        reg_num = reg_num | reg_num_Lo;
        //Tot aixo diria que no cal
        /*if(reg_num != 1){
        error = ILLEGAL_DATA_VALUE;
        frame = 0;
        }
        if(starting_address != 0){
        error = ILLEGAL_DATA_ADDRESS;
        frame = 0;
```

```
        } */
        //processament de la trama quan hi ha un error
        if(frame == 0){
            error_function();
        }
        //processament de la trama quan és correcte
        if(frame == 1){
            switch (reg_num){
                // Declarem aqui tots els registres
                case 1:
                    // inputregister = la teva variable
                    inputregister = estat_error<<8 | estat_emergencia;
                    break;
                case 2:
                    // inputregister = la teva variable
                    inputregister = Generacio_ESCLAU2;
                    break;
                case 3:
                    // inputregister = la teva variable
                    inputregister = *(((uint16_t *) &X)+1);
                    break;
                case 4:
                    // inputregister = la teva variable
                    inputregister = *(uint16_t *) &X;
                    break;
                case 5:
                    // inputregister = la teva variable
                    inputregister = *(((uint16_t *) &Y)+1);
                    break;
                case 6:
                    // inputregister = la teva variable
                    inputregister = *(uint16_t *) &Y;
                    break;
                case 7:
                    // inputregister = la teva variable
                    inputregister = *(((uint16_t *) &Z)+1);
                    break;
                case 8:
                    // inputregister = la teva variable
                    inputregister = *(uint16_t *) &Z;
                    break;
                case 9:
                    // inputregister = la teva variable
                    inputregister          =          *(((uint16_t          *)
&amp_galgues_estructura)+1);
                    break;
                case 10:
                    inputregister = *(uint16_t *) &amp_galgues_estructura;
                    break;
```

```
                case 11:
                    inputregister = *(((uint16_t *) &amp_galgues_cable1)+1);
                    break;
                case 12:
                    inputregister = *(uint16_t *) &amp_galgues_cable1;
                    break;
                case 13:
                    inputregister = *(((uint16_t *) &amp_galgues_cable2)+1);
                    break;
                case 14:
                    inputregister = *(uint16_t *) &amp_galgues_cable2;
                    break;
                case 15:
                    inputregister = *(((uint16_t *) &amp_galgues_cable3)+1);
                    break;
                case 16:
                    inputregister = *(uint16_t *) &amp_galgues_cable3;
                    break;
                case 17:
                    inputregister = *(((uint16_t *) &vel_anemometre)+1);
                    break;
                case 18:
                    inputregister = *(uint16_t *) &vel_anemometre;
                    break;
                case 19:
                    inputregister = *(((uint16_t *) &dir_anemometre)+1);
                    break;
                case 20:
                    inputregister = *(uint16_t *) &dir_anemometre;
                    break;
                default:
                    inputregister = 0;
                    break;
            }
            uint8_t inputregisterpart1 = (inputregister >> 8) & 0xFF; // Agafa
els 8 bits més significatius
            uint8_t inputregisterpart2 = inputregister & 0xFF; // Agafa els 8
bits menys significatius
            R_T_BIT_PORT = 1; //Activem el bit de transmissió de dades
            response[0] = SlaveAddress;
            response[1] = READ_INPUT_REGISTERS;
            response[2] = 2 * reg_num; //2*reg_num_Lo
            //response[3] = 0;
            response[3] = inputregisterpart1;  //canviat de notació big endian
a mid-little endian
            response[4] = inputregisterpart2; //ull
            crc16 = MODBUS_CRC16_TX(5);
            response[6] = CRC8_Hi(crc16); //crc em surt 0
            response[5] = CRC8_Lo(crc16);
            for (uint8_t jk = 0; jk <= 6; jk++){
```

```
                EUSART1_Write(response[jk]);
            }
            __delay_ms(10);
            R_T_BIT_PORT = 0; //Desactivem el bit de transmissió de dades
        }
    }
}

// NOMES S'ESCRIUEN DOS REGISTRES
void modbus_write_multiple_registers(void){
    uint8_t starting_address_Hi, starting_address_Lo, reg_num_Hi, reg_num_Lo,
    byte_count, auto_data_Hi, auto_data_Lo, test_data_Hi, test_data_Lo,
    test_value_data_Hi, test_value_data_Lo, CRC_read_Hi, CRC_read_Lo;
    uint16_t crc16, starting_address, reg_num, auto_data, test_data;
    frame = 1; //inicialitzem com a trama correcta
    starting_address_Hi= RxBuffer[2];
    starting_address_Lo = RxBuffer[3];
    reg_num_Hi = RxBuffer[4];
    reg_num_Lo = RxBuffer[5];
    byte_count = RxBuffer[6];
    auto_data_Hi= RxBuffer[7];
    auto_data_Lo= RxBuffer[8];
    test_data_Hi = RxBuffer[9];
    test_data_Lo = RxBuffer[10];
    test_value_data_Hi = RxBuffer[11];
    test_value_data_Lo = RxBuffer[12];
    CRC_read_Hi = RxBuffer[13];
    CRC_read_Lo = RxBuffer[14];
    if(checkCRC() == true){
        //passem a 16 bits
        starting_address = starting_address_Hi;
        starting_address = starting_address << 8;
        starting_address = starting_address | starting_address_Lo;
        reg_num = reg_num_Hi;
        reg_num = reg_num << 8;
        reg_num = reg_num | reg_num_Lo;
        auto_data = auto_data_Hi;
        auto_data = auto_data << 8;
        auto_data = auto_data | auto_data_Lo;
        test_data = test_data_Hi;
        test_data = test_data << 8;
        test_data = test_data | test_data_Lo;
        if(starting_address < 0){ //sempre adreça inici de zero
            frame = 0;
            error = ILLEGAL_DATA_ADDRESS;
        }
        /*
        if(reg_num != 3 && reg_num * 2 != byte_count){
        frame = 0;
```

```
        error = ILLEGAL_DATA_VALUE;
        } */
        //processament de la trama quan hi ha un error
        if (frame == 0){
            error_function();
        }
        //processament de la trama quan és correcte
        if(frame == 1){//trama correcta
            R_T_BIT_PORT=1; //Activem el bit de transmissió de dades
            response[0] = SlaveAddress;
            response[1] = WRITE_MULTIPLE_REGISTERS;
            response[2] = starting_address_Hi;
            response[3] = starting_address_Lo;
            response[4] = reg_num_Hi;
            response[5] = reg_num_Lo;
            crc16 = MODBUS_CRC16_TX(6);
            response[7] = CRC8_Hi(crc16);
            response[6] = CRC8_Lo(crc16);
            for(uint8_t j = 0; j <= 7; j++){
                EUSART1_Write(response[j]);
            }
            __delay_ms(5);
            R_T_BIT_PORT=0;
        }
    }
    switch (reg_num){
        // Declaració de tots els registres
        case 3:
            estat_error = (auto_data>>8)&0xFF;
            estat_emergencia = auto_data & 0xFF;
            escala_max = (test_data>>8)&0xFF;
            Accelerometre_reinit = (test_data >> 1) & 0x01;
            filtre_pas_alt = test_data & 0x01;
            break;
        case 4:
            float_conversion.i[0] = auto_data;
            float_conversion.i[1] = test_data;
            acceleracio_max = float_conversion.f;
            break;
        case 5:
            Offset_X = auto_data;
            Offset_Y = test_data;
            break;
        case 6:
            Offset_Z = auto_data;
            valor_max_galgues_estructura = test_data;
            break;
        case 7:
            valor_max_galgues_cable = auto_data;
```

```
            limit_baix_anemometre = (test_data>>8)&0xFF;
            limit_alt_anemometre = test_data & 0xFF;
            break;
    }
}


//Funcions modbus programades
void modbus_function (void){
    if(RxBuffer[0] == SlaveAddress){//COMPARAR ADREÇA
        switch (RxBuffer[1]){//COMPARAR FUNCIÓ
            case READ_INPUT_REGISTERS:
                modbus_read_input_registers(RxBuffer[4]);//Dada  a  llegir  pel
PLC
                break;
            case WRITE_MULTIPLE_REGISTERS://Escriu en el PIC
                modbus_write_multiple_registers();
                break;
            default:
                error = ILLEGAL_FUNCTION;
                frame = 0;
                break;
        }

    }
    c_control=0;
}
```

B.6. Arxiu modbus.h

```
#ifndef MODBUS_H
#define MODBUS_H

#include <xc.h>
#define SlaveAddress 1

#ifdef __cplusplus
extern "C" {
#endif

//FUNCIONS DEL MODBUS

static uint16_t MODBUS_CRC16_RX(uint8_t len);
static uint16_t MODBUS_CRC16_TX(uint8_t len);
uint8_t CRC8_Lo (uint16_t crc16);
uint8_t CRC8_Hi(uint16_t crc16);
bool checkCRC(void);
void initial_leds_state(void);
void error_function(void);
```

```
void modbus_read_input_registers(uint16_t reg_num);
void modbus_write_multiple_registers(void);
void modbus_function (void);


//Conversió de uint16 a float
union float_conversion_t {
   uint16_t i [2];
   float f;
} float_conversion;

//Gobal variables
extern uint8_t c_control;
extern uint8_t RxBuffer[];

//VARIABLES DEL MICROCONTROLADOR 1
//Read and write variables
extern enum ESTAT_ERROR_t
{
    NO_ERROR = 0,
    Error_WATCHDOG,
    Error_accelerometre,
    Error_mode_test
} estat_error;

extern enum ESTAT_EMERGENCIA_t
{
    NO_EMERGENCIA = 0,
    ALARMA_VIBRACIONS,
    ALARMA_GALGUES,
    ALARMA_VENT
} estat_emergencia;

//Read variables
extern _Bool Generacio_ESCLAU2;
extern float X; //g
extern float Y; //g
extern float Z; //g
extern float amp_galgues_estructura;  //mV
extern float amp_galgues_cable1;  //mV
extern float amp_galgues_cable2;  //mV
extern float amp_galgues_cable3; //mV
extern float vel_anemometre; // m/s
extern float dir_anemometre; // °

//Write variables
extern _Bool Accelerometre_reinit;
extern uint8_t escala_max;
extern _Bool filtre_pas_alt;
```

```
extern float acceleracio_max; //nivell d'acceleració a partir del qual salta
la interrupció
extern int16_t Offset_X;
extern int16_t Offset_Y;
extern int16_t Offset_Z;
extern uint16_t valor_max_galgues_estructura; // en mV
extern uint16_t valor_max_galgues_cable; // en mV
extern uint8_t limit_baix_anemometre; // en m/s
extern uint8_t limit_alt_anemometre; // en m/s

#endif
```

C. RESULTATS EXPERIMENTALS

Com ja s'ha comentat en la memòria, s'han realitzat diversos experiments als laboratoris de la UdG per tal de verificar el correcte funcionament dels sistemes. Els experiments son: el control de pitch que realitza el motor lineal, l'adquisició de dades de l'acceleròmetre i les comunicacions I2C i Modbus.

C.1. Control de pitch

S'ha realitzat una prova en el taller de mecànica per tal de verificar el moviment del motor.

Anteriorment a la versió definitiva del programa del microcontrolador, es va comprovar el rang del sensor de posició i la seva linealitat, utilitzant un regle per comparar les dades obtingudes en el microcontrolador amb les dades reals. Es van introduir després les dades en un excel, obtenint els resultats de la figura 71.

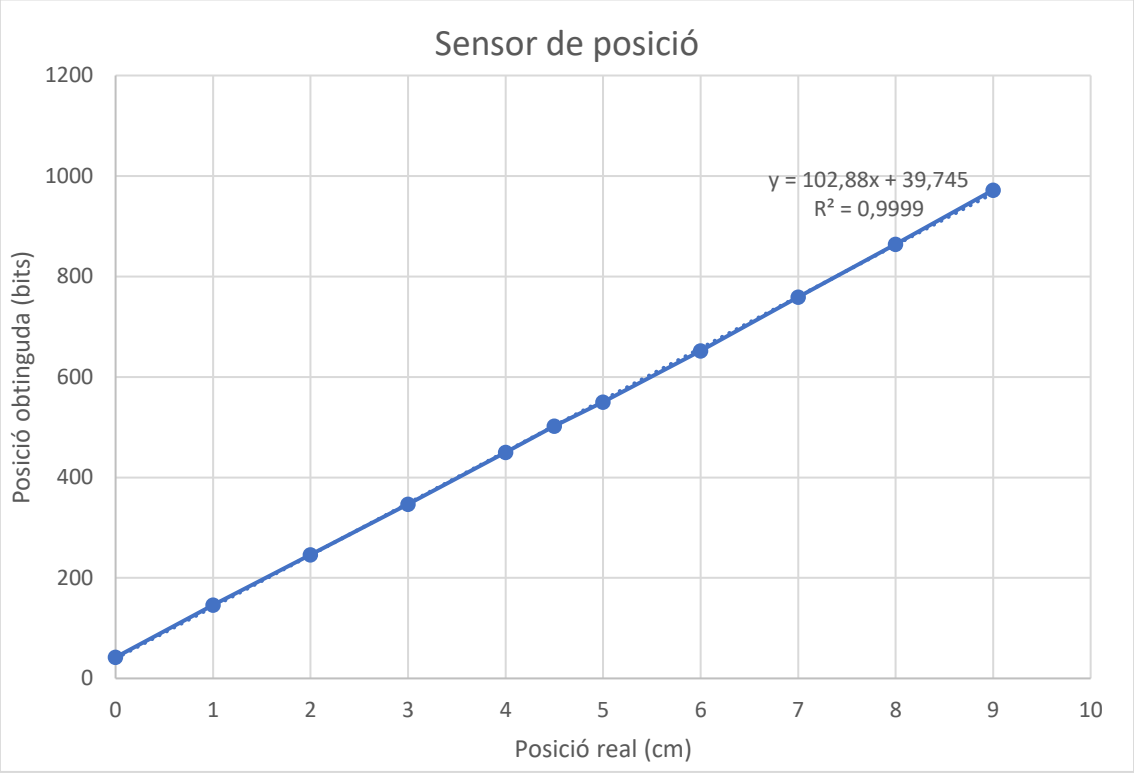

Figura 71: Dades obtingudes del sensor lineal

Les dades obtingudes del sensor confirmen el seu comportament lineal, amb un coeficient de determinació de 0,9999. La recta descrita per l'excel és introduïda en el

programari del microcontrolador, per tal de transformar les dades obtingudes de bits a cm (pendent = 102,88 i ordenada a l'origen = 39,745).

Amb les dades de posició correctament obtingudes s'aplica el sistema complet del controlador PI per tal de controlar el motor. La prova consisteix en moure el motor a unes posicions específiques en uns temps concrets, similar al que s'ha vist en la simulació del subapartat 3.2.2.

El motor parteix de la posició 0cm. Mitjançant una interrupció temporitzada s'introdueix que vagi fins a la posició 5cm al cap de 5 segons i retorni a la posició 4cm al cap de 10 segons. D'aquesta manera es comprova tant el moviment ràpid i lent, com el canvi de sentit del sistema.

En la figura 72 es poden veure les connexions realitzades entre el microcontrolador i els relés de control del motor que corresponen a les de l'esquema de la figura 5. El motor es troba unit mecànicament al sensor lineal mitjançant una volandera i dos cargols, es troba a més situat sobre el regle per poder comprovar el desplaçament real.

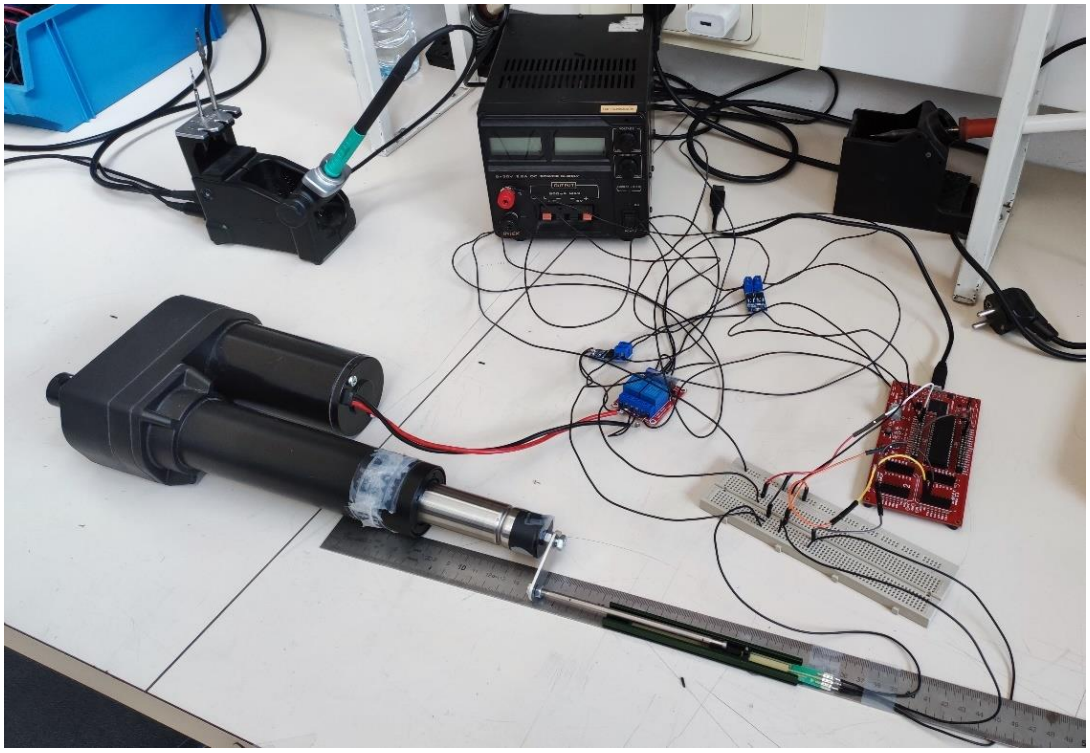

Figura 72: Proves de laboratori del sistema de control per PI

Mitjançant aquest experiment es demostra que el motor realitza l'acció demanada de forma correcta i ràpida.

Finalment s'utilitza un comptador enllaçat a una interrupció temporitzada de 1ms per comprovar el temps d'execució de la rutina de control del PI. La llei de control discreta del PI requereix d'una càrrega elevada de processament ja que treballa amb moltes variables en coma flotant simultàniament, per tant resulta interessant conèixer aquest temps d'execució per determinar si pot presentar un problema en el programa complet. Els resultats obtinguts indiquen un temps d'execució aproximat de 7ms, per tant no es preveu cap problema en la càrrega de treball del microcontrolador.

C.2. Acceleròmetre

Per tal de realitzar proves de funcionament de l'acceleròmetre, aquest s'ha instal·lat en un prototip d'aerogenerador més petit de 5 pales situat en el laboratori d'energies.

S'ha utilitzat el túnel de vent per simular condicions reals de vibracions. En la figura 73 es pot observar aquest petit aerogenerador situat dins el túnel de vent i en la figura 74 es pot veure la posició de l'acceleròmetre encastat en el prototip.

Figura 73: Prototip d'aerogenerador en el túnel de vent

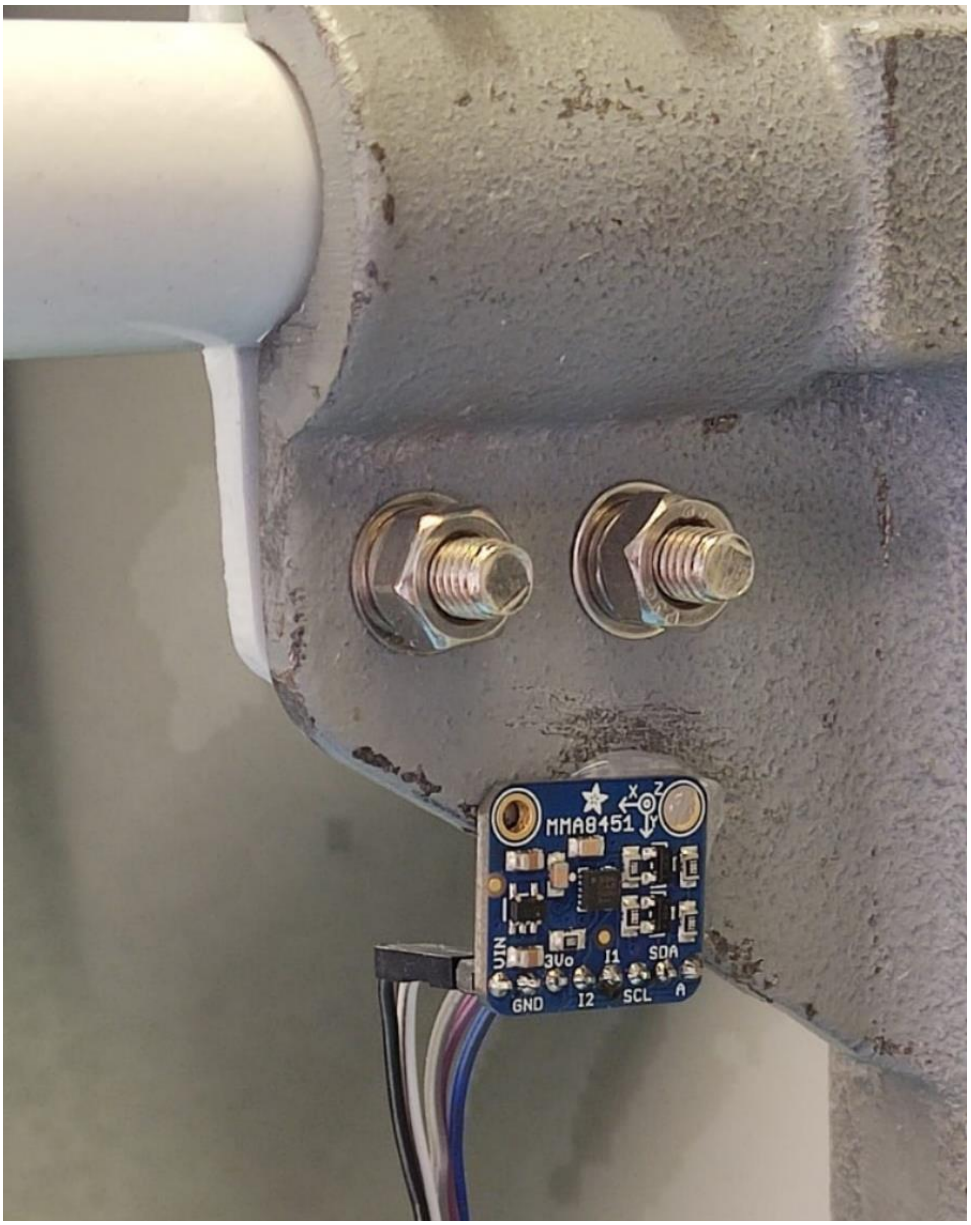

Figura 74: Col·locació de la placa d'acceleròmetre en el prototip

Partint del programa de monitorització de l'estructura mecànica i del vent, s'ha utilitzat un mòdul de comunicacions UART/USB com el que s'ha fet servir per sintonitzar el controlador del sistema de control de pitch, veure figura 8. S'ha habilitat la comunicació a través de la UART de les dades d'acceleració en l'eix X i l'eix Z. Els experiments es realitzen primer en un eix i després en l'altre ja que la comunicació a través de la UART és lenta i es volen obtenir les dades a la màxima freqüència possible per tal d'observar bé la forma de les vibracions. L'eix Y no s'analitza ja que pràcticament no es produeixen vibracions verticals.

Utilitzant el mateix sistema que s'ha vist en la sintonització del controlador en l'apartat 3.2., s'envien les dades obtingudes de l'acceleròmetre a través d'USB i es llegeixen mitjançant el programari PuTTY.

Per cada eix s'obtenen dades en dos situacions: amb l'aerogenerador funcionant en condicions normals i amb l'aerogenerador funcionant amb 4 pales enlloc de 5, simulant que una pala s'ha trencat.

Les dades obtingudes es processen i s'importen en un excel per tal de representar gràficament les vibracions, veure figura 75 i 76.

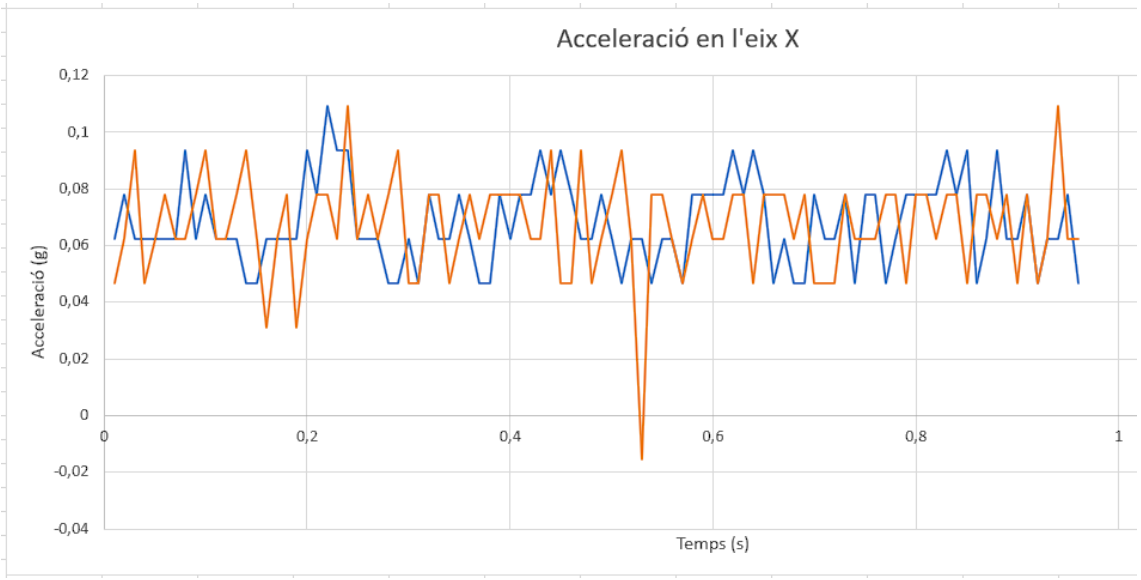

Figura 75: Valors d'acceleració en l'eix X en funció del temps

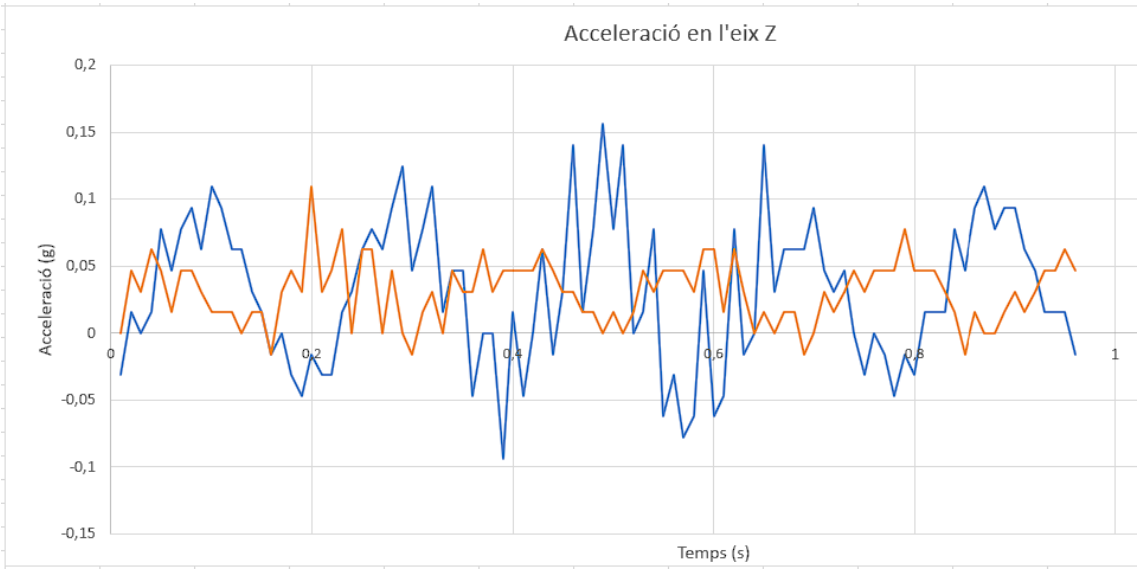

Figura 76: Valors d'acceleració en l'eix Z en funció del temps

Els valors d'acceleració de les gràfiques es representen en "g" que correspon a l'acceleració produïda per la gravetat 9,806m/s^2. El color taronja representa els valors d'acceleració de l'aerogenerador amb 5 pales (equilibrat) i el color blau representa els valors amb 4 pales (desequilibrat). Tant en l'eix X com en l'eix Z es pot observar una petita desviació del valor mitjà, aquesta podria ser compensada ajustant l'offset en les opcions de configuració de l'acceleròmetre.

En les gràfiques es pot observar que les vibracions afecten molt poc a l'eix X, independentment de si el molí està en equilibri o desequilibri. Les dades en l'eix Z, en canvi, presenten oscil·lacions pròpies de la vibració i aquestes s'accentuen quan es treu una pala a l'aerogenerador.

A partir de l'oscil·lació en la gràfica de l'eix Z es pot calcular la freqüència de ressonància o freqüència fonamental de l'estructura de l'aerogenerador. Amb les dades obtingudes, s'ha determinat una freqüència de 5,7Hz quan el molí es troba en equilibri (5 pales) i 5Hz quan el molí es troba en desequilibri (4 pales), demostrant també que la freqüència fonamental varia si es modifica l'estructura.

Per altra banda també s'ha realitzat una petita prova per comprovar la interrupció generada per l'acceleròmetre quan es supera cert valor de vibracions. En la prova, es redueix el valor de dispar des de la configuració de l'acceleròmetre dins dels marges observats en la figura 76, assegurant que, un cop iniciat el programa, efectivament el microcontrolador entra en estat d'emergència.

C.3. Comunicacions

Els experiments de comunicació I2C i Modbus s'han portat a terme en el laboratori de microprocessadors.

Per tal de realitzar les pertinents comprovacions de l'I2C es connecten dos microcontroladors entre ells, un en mode mestre i l'altre en mode esclau. En el microcontrolador esclau es connecta un potenciòmetre a una entrada ADC i es guarda el seu valor en una variable global, el mestre per la seva banda llegeix el valor d'aquesta variable de forma periòdica. Quan aquesta variable supera un cert valor, el mestre encén un LED de confirmació.

En la figura 77 es pot veure el muntatge de la prova de comunicacions.

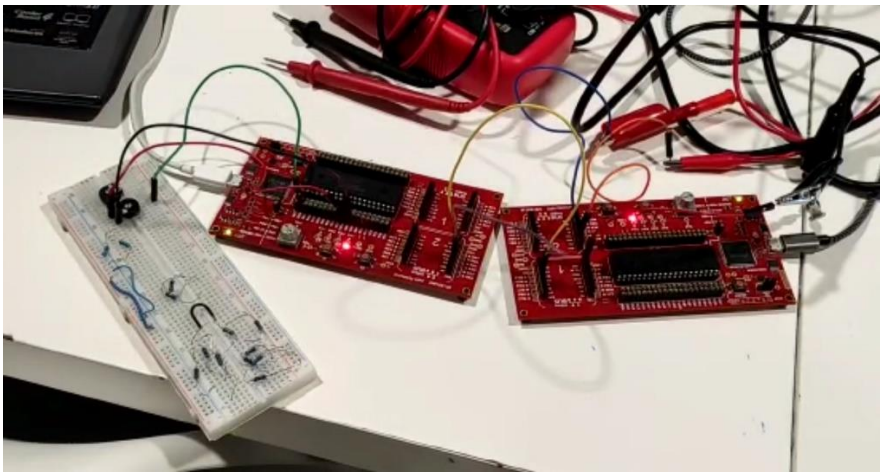

Figura 77: Connexió entre els microcontroladors - prova de comunicacions I2C

Movent el potenciòmetre s'aconsegueix activar i desactivar el LED del mestre a voluntat, per tant aquesta prova es dona per vàlida i es demostra el correcte funcionament de la comunicació I2C.

Per la prova de comunicació del Modbus s'utilitza el mòdul UART/RS485 vist en l'apartat 5.2, figura 44. Es modifica el programa del PIC18 per tal d'incloure tant sols la comunicació Modbus amb les variables d'escriptura i lectura i es realitzen les connexions entre el PLC i el microcontrolador com es pot veure en la figura 78.

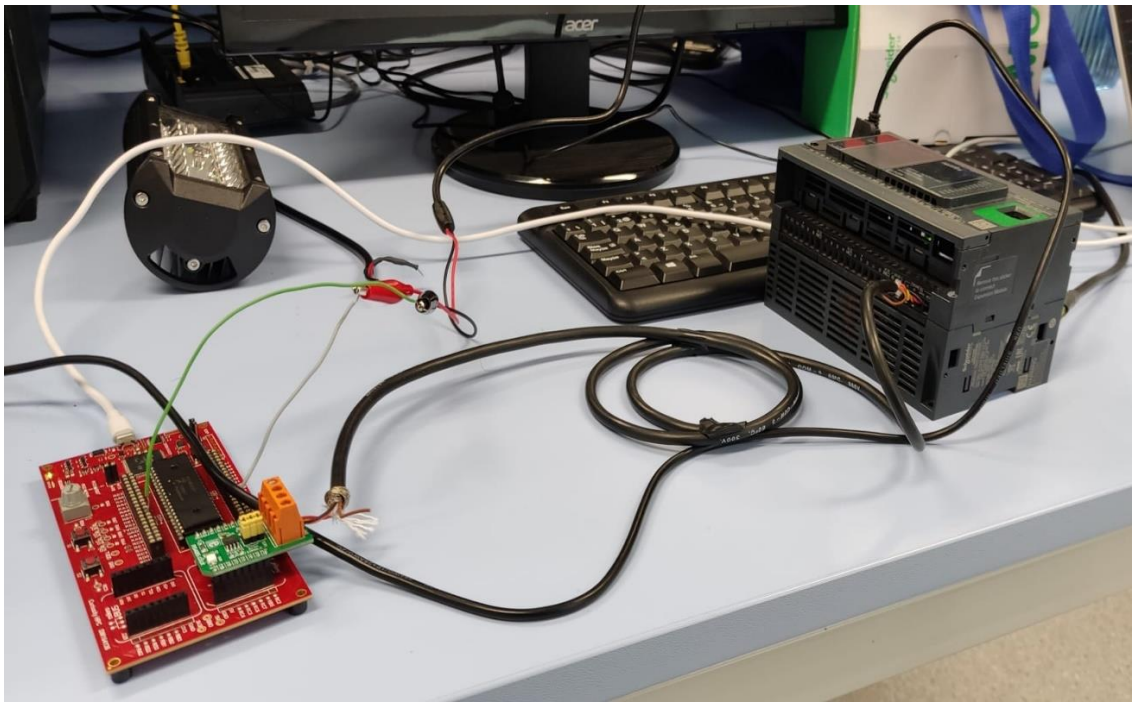

Figura 78: Connexió entre el PLC i el microcontrolador - prova de comunicacions Modbus

Des del programa del PLC es realitza una lectura de totes les variables vistes en els subapartats 5.2.1. i 5.2.2. Registre a registre, es comprova que els valors obtinguts corresponguin amb els guardats en la memòria del microcontrolador. Per aquesta prova s'introdueixen valors ficticis a les variables per poder comparar-los correctament.

En les figures 79 i 80 podem apreciar les trames de comunicació Modbus del PLC i del microcontrolador respectivament.

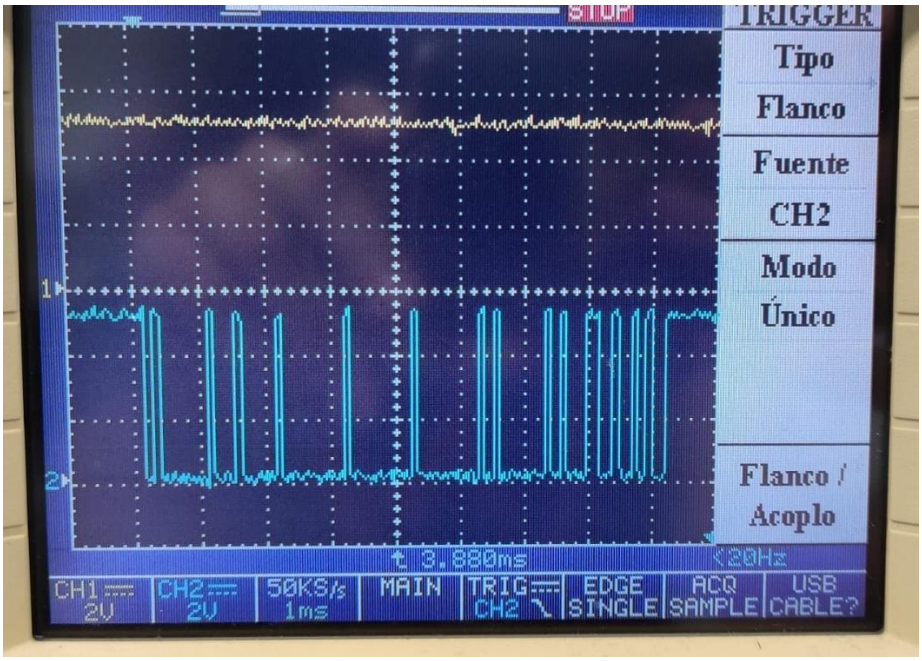

Figura 79: Trames de petició de lectura del PLC

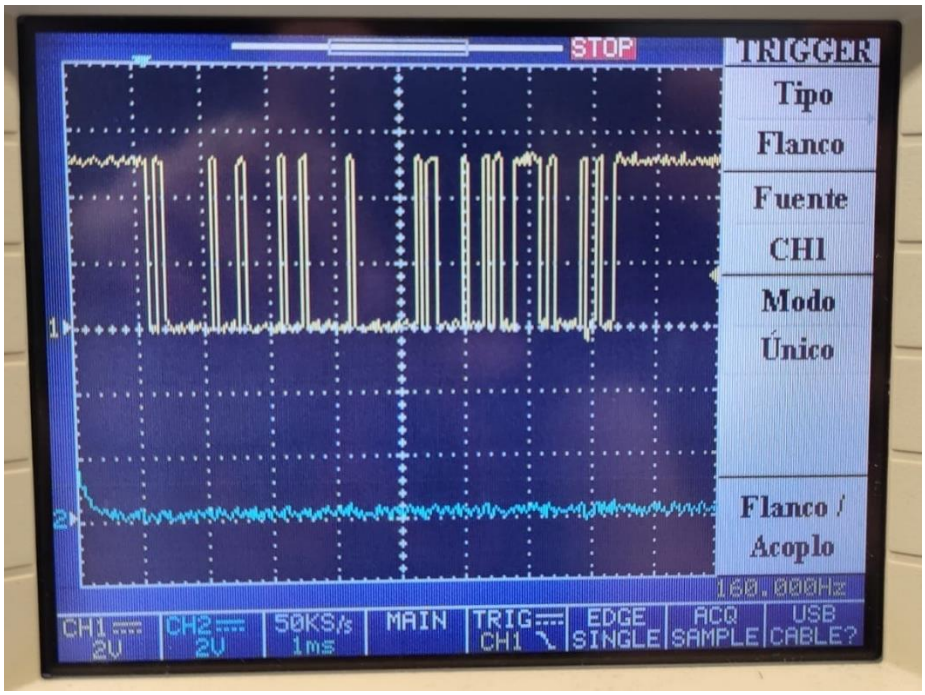

Figura 80: Trames de resposta del microcontrolador

També es fa la comprovació de l'escriptura seguint el mateix procediment, escrivint registre a registre en totes les variables i comprovant que els valors en ambdós dispositius coincideixin.

Aquesta prova es realitza amb els programes dels tres microcontroladors del projecte Llavor amb resultats satisfactoris, per tant, les comunicacions Modbus es donen per validades.