

Treball final de grau

Estudi: Grau en Enginyeria en Tecnologies Industrials

Títol: Estudi i implementació d'un protocol Modbus de comunicació Master-Slave per un aerogenerador

Document: 1. Memòria

Alumne: Sergi Riera Toran

Tutor: Inès Ferrer Mallorquí

Departament: Enginyeria elèctrica, electrònica i automàtica

Àrea: Enginyeria de sistemes i automàtica

Tutor: Lluís Pacheco Valls

Departament: Arquitectura i tecnologia de computadors

Àrea: Arquitectura i tecnologia de computadors

Convocatòria (mes/any): Setembre 2021

Agraïments

Agraeixo a la Universitat per l'oportunitat que em van donar en fer aquest projecte i pel material subministrat.

També agrair a la família que no només m'ha donat suport moral en el projecte sinó que també des del primer dia que vaig començar el grau. Als meus pares, per donar-me els recursos necessaris i estar al meu costat donant-me suport i aconsellant-me sempre.

ÍNDEX DE CONTINGUTS

| | |
|---|----|
| Índex de continguts | i |
| Índex de figures..... | iv |
| Índex de taules..... | vi |
| 1 Introducció | 1 |
| 1.1 Antecedents | 1 |
| 1.2 Objecte..... | 1 |
| 1.3 Abast..... | 2 |
| 1.4 Peticionari | 2 |
| 2 Aerogenerador | 3 |
| 3 Xarxa de comunicació de dades..... | 5 |
| 3.1 Introducció | 5 |
| 3.2 Xarxes industrials..... | 5 |
| 3.2.1 Nivell de gestió..... | 6 |
| 3.2.2 Nivell de control o planta | 6 |
| 3.2.3 Nivell de camp i procés | 6 |
| 3.2.4 Nivell d'entrada – sortida (E/S)..... | 7 |
| 4 Estudi del protocol de comunicació Modbus RTU | 9 |
| 4.1 Introducció | 9 |
| 4.2 Transmissió de dades | 9 |
| 4.3 Bits necessaris per caràcter a transmetre | 10 |
| 4.4 Comprovació d'errors: CRC | 11 |
| 4.5 Casos durant la sol·licitud del dispositiu mestre | 11 |
| 4.6 Referència Modbus | 12 |
| 4.7 Funcions Modbus..... | 12 |
| 4.8 Codis d'excepció Modbus | 14 |

| | | |
|-------|---|----|
| 4.8.1 | Camp de codi de funcions..... | 14 |
| 4.8.2 | Camp de dades..... | 14 |
| 4.8.3 | Llistat de codis d'excepció..... | 14 |
| 4.9 | Interfície RS485 | 15 |
| 5 | Implementació del protocol Modbus RTU | 17 |
| 5.1 | Introducció | 17 |
| 5.2 | Analogia | 17 |
| 5.3 | Dispositius..... | 18 |
| 5.3.1 | PLC..... | 18 |
| 5.3.2 | Microcontrolador | 19 |
| 5.3.3 | Transceptor RS485 | 19 |
| 5.3.4 | Arduino Uno | 21 |
| 5.4 | MPLAB..... | 22 |
| 5.4.1 | Recurs ADCC | 24 |
| 5.5 | Mapa de memòria Modbus..... | 25 |
| 5.6 | Funcions Modbus utilitzades | 26 |
| 5.6.1 | Read Coils (0x01) | 26 |
| 5.6.2 | Read Input Registers (0x04)..... | 27 |
| 5.6.3 | Write Multiple Registers (0x10) | 28 |
| 5.7 | Codis d'excepció Modbus utilitzats..... | 29 |
| 5.7.1 | Illegal function (0x01) | 30 |
| 5.7.2 | Illegal data address (0x02) | 31 |
| 5.7.3 | Illegal data value (0x03) | 31 |
| 5.8 | CRC Implementat..... | 31 |
| 5.9 | Funcionament general del protocol implementat | 32 |
| 6 | Proves de comunicació | 35 |
| 6.1 | Mode test | 35 |
| 6.1.1 | Orientació 30° | 37 |
| 6.1.2 | Orientació 45° | 38 |

| | | |
|---------|---|----|
| 6.1.3 | Orientació 60° | 39 |
| 6.1.4 | Orientació 90° | 40 |
| 6.2 | Mode automàtic..... | 41 |
| 6.2.1 | Posició 1 del potenciòmetre | 43 |
| 6.2.2 | Posició 2 del potenciòmetre | 44 |
| 7 | Resum del pressupost..... | 47 |
| 8 | Conclusions..... | 49 |
| 9 | Relació de documents..... | 51 |
| 10 | Referències | 53 |
| 11 | Glossari | 55 |
| Annex A | Configuracions inicials dels recursos a MPLAB | 57 |
| A.1.1 | Configuració recurs EUSART | 57 |
| A.1.2 | Configuració recurs ADCC | 60 |
| Annex B | Programari microcontrolador..... | 63 |
| B.1 | Source files: Main..... | 63 |
| B.2 | Source files: Modbus..... | 63 |
| B.3 | Source files: MCC generated files | 78 |
| B.3.1 | ADCC..... | 78 |
| B.3.2 | Device configuration..... | 79 |
| B.3.3 | Eusart | 81 |
| B.3.4 | Mcc | 82 |
| B.3.5 | Pin manager..... | 83 |
| B.4 | Header files: Modbus | 85 |
| B.5 | Header files: MCC generated files..... | 86 |
| B.5.1 | ADCC..... | 86 |
| B.5.2 | Device configuration..... | 86 |
| B.5.3 | Eusart | 86 |
| B.5.4 | Mcc | 87 |
| B.5.5 | Pin manager..... | 88 |

| | | |
|---------|---|----|
| Annex C | Programari PLC | 91 |
| C.1 | Variables | 91 |
| C.2 | CFC | 92 |
| C.3 | Descripció dels blocs de funcions..... | 92 |
| C.3.1 | Bloc ADDM | 92 |
| C.3.2 | Bloc WRITE_VAR | 93 |
| C.3.3 | Bloc READ_VAR..... | 94 |
| C.3.4 | Bloc BLINK..... | 95 |
| C.3.5 | Bloc MOVE | 95 |
| Annex D | Petit programari per visualitzar les trames amb Arduino | 97 |
| Annex E | Esquemes addicionals..... | 99 |

ÍNDIX DE FIGURES

| | | |
|------------|--|----|
| Figura 1: | Aerogenerador real d'Ordís | 3 |
| Figura 2: | Piràmide CIM de comunicació | 5 |
| Figura 3: | Sistema de cablejat convencional versus bus de camp | 6 |
| Figura 4: | Cicle de Sol·licitud-Resposta..... | 9 |
| Figura 5: | Topologia xarxa multipunt RS485 Half-Duplex | 15 |
| Figura 6: | PLC TM241C24R Schneider Electrics..... | 18 |
| Figura 7: | Placa de desenvolupament Curiosity HPC amb el microcontrolador | 19 |
| Figura 8: | Transceptor RS422/485 | 20 |
| Figura 9: | Connexió del transceptor RS485 amb la placa de desenvolupament | 21 |
| Figura 10: | Placa Arduino..... | 22 |
| Figura 11: | Estructura del codi..... | 23 |
| Figura 12: | Gestor de pins del microcontrolador | 23 |
| Figura 13: | Pin module | 24 |
| Figura 14: | Vista dels pins usats en el microcontrolador | 24 |

| | |
|---|----|
| Figura 15: Procés d'enviar resposta al PLC..... | 30 |
| Figura 16: Calculadora CRC online | 31 |
| Figura 17: Diagrama de flux del funcionament del protocol implementat | 33 |
| Figura 18: Exemple forçar LED EA3 en mode test | 36 |
| Figura 19: Visualització del mode de treball test en el microcontrolador..... | 36 |
| Figura 20: Registre per forçar una orientació de 30°..... | 37 |
| Figura 21: Visualització dels LEDs en el microcontrolador per 30° | 37 |
| Figura 22: Valor LED des del SoMachine amb orientació 30°..... | 38 |
| Figura 23: Registre per forçar una orientació de 45°..... | 38 |
| Figura 24: Visualització dels LEDs en el microcontrolador per 45° | 38 |
| Figura 25: Valor LED des del SoMachine amb orientació 45°..... | 39 |
| Figura 26: Registre per forçar una orientació de 60°..... | 39 |
| Figura 27: Visualització dels LEDs en el microcontrolador per 60° | 40 |
| Figura 28: Valor LED des del SoMachine amb orientació 60°..... | 40 |
| Figura 29: Registre per forçar una orientació de 90°..... | 40 |
| Figura 30: Visualització dels LEDs en el microcontrolador per 90° | 41 |
| Figura 31: Valor LED des del SoMachine amb orientació 90°..... | 41 |
| Figura 32: Registre en mode automàtic..... | 42 |
| Figura 33: Rang de valors del vent i angle de les pales..... | 42 |
| Figura 34: Visualització del mode de treball automàtic en el microcontrolador | 43 |
| Figura 35: Posició 1 del potenciòmetre..... | 43 |
| Figura 36: Valor del LED i potenciòmetre a SoMachine per la posició 1 | 44 |
| Figura 37: Posició 2 del potenciòmetre..... | 44 |
| Figura 38: Valor del LED i potenciòmetre a SoMachine per la posició 2..... | 45 |
| Figura 39: Configuració general del recurs EUSART..... | 57 |
| Figura 40: Configuració dels registres | 58 |
| Figura 41: Configuració general del recurs ADCC..... | 60 |
| Figura 42: Configuració dels registres ADCC | 61 |
| Figura 43: Interfície i diagrames de blocs de SoMachine..... | 92 |

| | |
|--------------------------------|----|
| Figura 44: Bloc ADDM..... | 92 |
| Figura 45: Bloc WRITE_VAR..... | 93 |
| Figura 46: Bloc READ_VAR | 94 |
| Figura 47: Bloc BLINK..... | 95 |
| Figura 48: Bloc MOVE..... | 95 |

ÍNDEX DE TAULES

| | |
|---|----|
| Taula 1: Format del missatge RTU sol·licitat pel mestre..... | 10 |
| Taula 2: Bits necessaris per caràcter a transmetre sense paritat..... | 11 |
| Taula 3: Dades en un dispositiu de xarxa Modbus | 12 |
| Taula 4: Funcions Modbus | 13 |
| Taula 5: Relació binària-decimal-hexadecimal | 14 |
| Taula 6: Llistat codis d'excepció Modbus..... | 15 |
| Taula 7: Analogia aerogenerador - microcontrolador..... | 18 |
| Taula 8: Distribució de pins del transceptor RS485 | 20 |
| Taula 9: Configuracions i indicadors del transceptor RS485..... | 21 |
| Taula 10: Mapa de memòria Modbus | 25 |
| Taula 11: Resum funcions Modbus implementades | 26 |
| Taula 12: Sol·licitud Read Coils des del PLC | 26 |
| Taula 13: Respostes Read Coils del microcontrolador | 27 |
| Taula 14: Equivalència resposta amb el LED | 27 |
| Taula 15: Sol·licitud Read Input Registers des del PLC..... | 27 |
| Taula 16: Resposta Read Input Registers del microcontrolador | 28 |
| Taula 17: Sol·licitud Write Multiple Registers des del PLC - mode automàtic | 28 |
| Taula 18: Sol·licitud Write Multiple Registers des del PLC - mode test..... | 29 |
| Taula 19: Resposta Write Multiple Registers del microcontrolador | 29 |

| | |
|---|----|
| Taula 20: Camp de funció i dades en una resposta amb excepció | 30 |
| Taula 21: Valors que pot pendre el bloc VALOR_TEST | 36 |
| Taula 22: Configuració registre BAUD1CON | 58 |
| Taula 23: Configuració registre RC1STA | 59 |
| Taula 24: Configuració registre TX1STA | 59 |
| Taula 25: Configuració registre ADCON0 | 61 |
| Taula 26: Configuració registre ADCON1 | 62 |
| Taula 27: Paràmetres bloc ADDM | 93 |
| Taula 28: Paràmetres bloc WRITE_VAR | 94 |
| Taula 29: Paràmetres bloc READ_VAR | 94 |
| Taula 30: Paràmetres bloc BLINK | 95 |

1 INTRODUCCIÓ

El present projecte s'emmarca dins el projecte "Llavor" concedit per la Generalitat de Catalunya a la Universitat de Girona. En concret 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.

El projecte es desenvolupa per ser utilitzat a l'Aula Ecogranja Vilà S.L., empresa especialitzada en el camp de la generació i subministrament d'energia elèctrica obtinguda de fonts renovables, dins l'acord de cooperació científica signat entre aquesta i la Universitat de Girona.

1.1 Antecedents

Es vol posar en marxa l'automatització de l'aerogenerador instal·lat al poble d'Ordis. La idea és poder fer el comandament i la recollida de dades de l'aerogenerador des d'un edifici situat a 300 metres d'aquest.

Un aerogenerador és una màquina que aprofita l'energia cinètica del vent per convertir-la en energia elèctrica amb l'ajut d'unes pales, un rotor i un alternador, en termes generals.

El vent és una de les variables meteorològiques més destacades de l'Alt Empordà. A més, actualment i cada vegada més, s'ha de reduir en la mesura del possible les accions que poden comportar una acceleració del canvi climàtic. Per tant, posar en funcionament un petit aerogenerador per abastir el subministrament elèctric de la granja del poble és una bona elecció.

1.2 Objecte

La finalitat del projecte és estudiar i implementar un protocol de comunicació Modbus mestre-esclau. Al ser una fase inicial del projecte final, es pretén establir la comunicació entre un PLC que actuarà com a mestre i un conjunt de components en una placa amb un microcontrolador que actuarà com a esclau, simulant l'angle de les pales i la velocitat del vent.

1.3 Abast

Es pretén implementar el protocol de comunicació que ha de servir de model per muntar la xarxa de comunicació mestre-esclau real de l'aerogenerador d'Ordis per tal de controlar i monitorar paràmetres com són el mecanisme de Pitch, l'accionament de relés i lectura de dades per anomenar alguns. La feina es basa a fer la tria del PLC, la placa a adquirir i el cablejat a utilitzar. A més, en aquesta fase inicial es farà el desenvolupament del protocol de comunicació en un esclau, un accionament d'elements remots ubicats a la placa microcontrolada i finalment, les proves de comunicació.

1.4 Peticionari

El peticionari del projecte és la persona que té l'aerogenerador al poble d'Ordis.

2 AEROGENERADOR

L'energia eòlica consisteix a 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. En alguns països ja cobreix una fracció notable del subministrament. Segons alguns estudis, aquest recurs podria arribar a satisfer la demanda energètica global. No obstant això, el seu principal problema 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. 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 el qual es vol fer l'automatització amb la finalitat de recollir dades i poder fer un comandament es troba al poble d'Ordis i és d'eix horitzontal. Veure *Figura 1: Aerogenerador real d'Ordis*.



Figura 1: Aerogenerador real d'Ordis

Es pretén que en un futur projecte es pugui fer un seguiment de les dades recollides del vent, de la velocitat del rotor, de la tensió i intensitat de les bateries, per anomenar alguns. A més, s'ha de poder fer un control de l'angle de Pitch.

Per afegir, l'aerogenerador servirà de model pel disseny i dimensionament de nous petits aerogeneradors ja que s'incorporaran galgues amb la finalitat de pendre mesures dels esforços sotmesos sobre l'estructura. A més, s'incorporarà un acceleròmetre que permetrà recollir les vibracions en tres direccions.

Finalment, l'aerogenerador haurà de subministrar energia per alimentar la granja del poble en la mesura que sigui possible i alimentar unes bombes hidràuliques per emplenar dipòsits.

3 XARXA DE COMUNICACIÓ DE DADES

3.1 Introducció

En els últims anys el desenvolupament dels sistemes informàtics ha crescut ràpidament, de tal manera que podem trobar computadors en pràcticament tots els àmbits de la vida quotidiana. A més, els computadors s'han incorporat en l'àmbit industrial pel control de plantes, monitoratge de processos productius, gestió integrada de diferents etapes de fabricació, control de màquines, robots, manipuladors, per anomenar alguns. A causa de les característiques i necessitats de l'entorn industrial, es requereixen equips robustos i d'alt rendiment per executar tasques de control i monitoratge automàtica com són els PLC i els microcontroladors.

En moltes ocasions aquests computadors o equips de control no fan operacions aïllades, sinó que necessiten intercanviar dades amb altres equips per realitzar les seves funcions.

3.2 Xarxes industrials

Entenem una xarxa industrial com una xarxa de comunicació, que consisteix en un conjunt de terminals que s'enllacen per l'intercanvi d'informació on aquest es constitueix per automatismes d'entorn industrial, és a dir, sensors, actuadors, blocs d'entrada i sortida, controladors o PLCs d'acord amb la piràmide CIM (*Computer Integrated Manufacturing*) que es mostra a la *Figura 2: Piràmide CIM de comunicació*. (EL MODELO CIM Y JERARQUÍA DE REDES DE COMUNICACIÓN EN LA INDUSTRIA, n.d.)



Figura 2: Piràmide CIM de comunicació

3.2.1 Nivell de gestió

El volum d'informació intercanviada és molt elevada i els temps de resposta no són crítics. S'utilitza per xarxes d'oficina, comptabilitat i administració, vendes, gestió de comandes, entre altres.

3.2.2 Nivell de control o planta

Interconnexió de mòduls de fabricació entre si i amb departaments com disseny o planificació. Se sol emprar per a l'enllaç entre les funcions d'enginyeria i planificació amb les de control de producció de planta i seqüències d'operacions.

3.2.3 Nivell de camp i procés

El denominat nivell de camp i procés és aquell nivell on ja incorporen automatismes per tenir un control sobre els dispositius de més baix nivell, com poden ser actuadors i sensors. En aquest nivell s'utilitza el concepte de bus de camp.

3.2.3.1 Bus de camp

Un bus de camp és un sistema de transmissió de dades que simplifica la instal·lació i operació de màquines i equipaments industrials. Típicament són xarxes digitals, bidireccionals i multipunt sobre un bus sèrie. L'objectiu d'aquests busos és substituir les connexions punt a punt entre els elements del procés, tal com mostra la *Figura 3: Sistema de cablejat convencional versus bus de camp*.

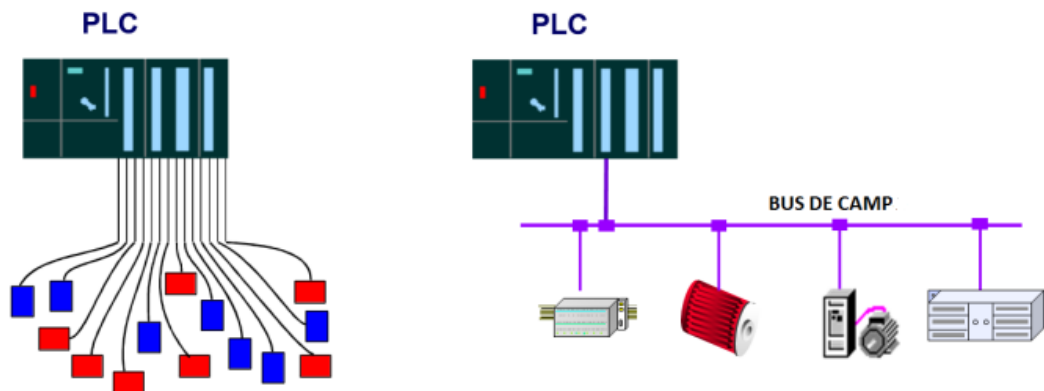


Figura 3: Sistema de cablejat convencional versus bus de camp

L'objectiu de substituir el sistema convencional punt a punt per a busos de camp permet millorar la qualitat del producte, reduir els costos i millorar l'eficiència. El principal avantatge que ofereixen els busos de camp és la reducció de costos, que principalment prové de tres fonts: estalvi en el cost d'instal·lació, estalvi en el cost de manteniment i estalvi de millores contínues del funcionament del sistema. Una de les característiques més visuals dels busos de camps és la

reducció de cable pel control d'una instal·lació. Cada component només requereix un cable per la connexió dels diferents nodes.

Cada dispositiu de camp incorpora certa capacitat de procés de dades, com executar funcions simples de control i diagnòstic, la qual el converteix en un dispositiu intel·ligent, mantenint sempre un cost baix.

També s'ha de tenir en compte que les prestacions del sistema milloren amb l'ús de la tecnologia de busos de camp a causa de la simplificació en la forma d'obtenir informació de planta des de diferents sensors. Aquesta simplificació en l'obtenció de dades permet sistemes de control més eficients.(Fernández, n.d.)

3.2.4 Nivell d'entrada – sortida (E/S)

És el nivell més senzill i per això es troba a la base de la piràmide CIM. Està format per elements simples com sensors i actuadors que interactuen directament amb el procés.

4 ESTUDI DEL PROTOCOL DE COMUNICACIÓ MODBUS RTU

4.1 Introducció

Modbus és un protocol de comunicació per xarxes de comunicacions industrials i desenvolupat per Modicon. La seva especificació oberta permet la comunicació entre qualsevol dispositiu que compleixi amb el protocol.

La seva tipologia és Mestre/Esclau amb una estructura de bus lineal on només existeix un mestre, qui controla l'accés al medi i monitora el funcionament de la xarxa, i un o més dispositius programables actuen com esclaus, que responen i procedeixen segons la sol·licitud del mestre.

Modbus conté dos modes de comunicació sèrie coneguts com a ASCII i RTU, però el projecte se centra en el mode RTU.

4.2 Transmissió de dades

Els dispositius es comuniquen mitjançant una tècnica mestre/esclau, en què només hi ha un dispositiu que treballa com a mestre i que pot iniciar una consulta. La resta de dispositius actuen com esclaus subministrant la resposta de les dades sol·licitades. El mestre pot adreçar-se a esclaus individualment o pot iniciar un missatge de difusió a tots els esclaus. Veure *Figura 4: Cicle de Sol·licitud-Resposta*.

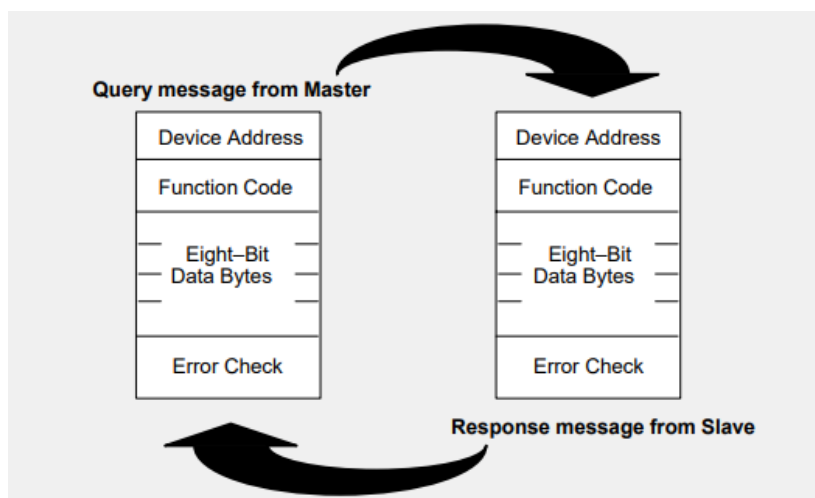


Figura 4: Cicle de Sol·licitud-Resposta

La transmissió de dades té perfectament definits els punts d'inici i final del missatge. Els dispositius receptors reconeixen el començament del missatge, llegint la direcció de l'esclau per determinar si estan sent cridats i saber quan s'acaba el missatge amb la finalitat de saber si es poden utilitzar els bytes de comprovació d'error per confirmar la integritat del missatge.

En mode RTU, els missatges comencen amb un interval de silenci d'almenys 3,5 vegades els temps per caràcter. Aleshores, el primer camp transmès és la direcció del dispositiu, on els caràcters permesos van de 09 a AF en hexadecimal. Quan es rep el primer camp, cada dispositiu el descodifica per detectar si es tracta d'un missatge dirigit a ell i, en cas contrari continua l'espera del següent missatge. 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 diferenciarien dos missatges rebuts consecutius.

Tota la trama del missatge ha de ser transmesa com a un flux continu. Si un nou missatge comença abans del temps equivalent d'almenys 3,5 vegades el temps de caràcter (TC), el dispositiu encarregat de llegir no tindrà constància que es tracta d'un nou missatge.

El protocol Modbus estableix el format per a la consulta del mestre mitjançant la col·locació de l'adreça del dispositiu, un codi de funció que defineix l'acció sol·licitada, les dades que calen enviar des d'una posició i un camp de comprovació d'errors. Tal com es pot veure a la *Taula 1: Format del missatge RTU sol·licitat pel mestre*.

| Inici | Direcció Esclau | Funció | Dades | CRC | Final |
|---------|-----------------|--------|---------|---------|---------|
| >3.5 TC | 1 byte | 1 byte | N bytes | 2 bytes | >3.5 TC |

Taula 1: Format del missatge RTU sol·licitat pel mestre

El missatge de resposta de l'esclau també es construeix mitjançant el protocol Modbus. Tant el missatge sol·licitat com el de resposta, dependran del tipus d'acció desitjat sobre els esclaus.

4.3 Bits necessaris per caràcter a transmetre

Els caràcters pel mode de transmissió RTU poden tenir fins a una longitud d'onze bits. Aquests bits es distribueixen de la següent manera:

- 1 bit d'inici
- 8 bits de dades, on el bit menys significatiu s'envia primer
- 1 bit de paritat, en el cas d'utilitzar paritat
- 1 bit de parada, en el cas que no s'utilitzi el bit de paritat es poden utilitzar 2 bits de parada

En el corresponent projecte, s'ha optat no utilitzar el bit de paritat i només fer ús d'un bit de parada. Per tant, els bits necessaris per caràcter a transmetre seran deu. Tal com es mostra a la *Taula 2: Bits necessaris per caràcter a transmetre sense paritat.*

| Transmissió per caràcter | | | | | | | | | |
|--------------------------|---|---|---|---|---|---|---|---|--------|
| Inici | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | Parada |

Taula 2: Bits necessaris per caràcter a transmetre sense paritat

4.4 Comprovació d'errors: CRC

Els mètodes de detecció i correcció d'errors són molt usats dins les comunicacions. Aquests errors poden ser deguts al soroll o radiacions externes. Si aquests errors no es detecten, poden causar un mal funcionament del sistema. Les xarxes sèrie estàndard Modbus utilitzen la Comprovació de Redundància Cíclica (CRC).

Els bytes de comprovació d'error en els missatges contenen un valor CRC que s'utilitza per comprovar el contingut de tot el missatge. Els dos bytes del CRC sempre han d'estar presents per complir el protocol, sense opció a deshabilitar-los.

Abans de tractar les dades d'una sol·licitud per part del mestre, sempre es comprova que el CRC que llegeix l'esclau sigui el mateix que el que aquest calcula amb la trama que rep. Si el CRC llegit i calculat són iguals, es tracten les dades i s'envia una resposta en el mestre. Per contra, si no coincideixen els valors, les dades no es tracten i no s'envia cap resposta.

4.5 Casos durant la sol·licitud del dispositiu mestre

Quan un dispositiu mestre envia una sol·licitud a un dispositiu esclau es poden produir un dels següents quatre esdeveniments:

- El dispositiu esclau rep la sol·licitud sense cap error de comunicació i gestiona la consulta enviant una resposta normal.
- El dispositiu esclau no rep la sol·licitud a causa d'un error de comunicació i no es retorna cap resposta. En aquest cas, el mestre processarà una condició de temps d'espera per la sol·licitud.
- El dispositiu esclau rep la sol·licitud, però detecta un error de comunicació com pot ser en el CRC. En aquest cas, tampoc es retorna cap resposta i el dispositiu mestre processarà una condició de temps d'espera per la sol·licitud.
- El dispositiu esclau rep la petició sense cap error de comunicació, però no la pot gestionar. Per exemple, si la sol·licitud consisteix a llegir una sortida o registre inexistent. En aquest

cas, l'esclau retorna una resposta d'excepció informant el mestre de la naturalesa de l'error.

4.6 Referència Modbus

Modbus maneja principalment dos tipus de dades: bits individuals i paraules de 16 bits. Els bits individuals corresponen a entrades i sortides discretes amb estat ON/OFF i les paraules a registres d'entrada o sortida el qual el seu estat indica un valor analògic. *Tal com es mostra Taula 3: Dades en un dispositiu de xarxa Modbus.*

| Sector | Format | Tipus d'accés | Referència |
|----------------------|---------------------|---------------------|------------|
| Sortides discretes | Bits individuals | Lectura – escritura | 0X |
| Entrades discretes | Bits individuals | Lectura | 1X |
| Registres d'entrada | Paraules de 16 bits | Lectura | 3X |
| Registres de sortida | Paraules de 16 bits | Lectura – escritura | 4X |

Taula 3: Dades en un dispositiu de xarxa Modbus

A la *Taula 3* es mostren els quatre tipus de dades que poden estar presents en els controladors que tenen Modbus com a protocol de comunicació, el format en el qual es troben dins el dispositiu, el tipus d'accés i com poden ser modificats o transformats. És evident que les entrades no tenen la possibilitat de ser canviades per un software d'aplicació, ja que aquestes fan referència a estats externs dels controladors. A més, segons el protocol Modbus, cada tipus de dades presents en els controladors té associat una referència.

4.7 Funcions Modbus

El protocol Modbus té associat una sèrie de funcions que permeten accedir a diferents tipus de dades poden ser l'accés a bits i l'accés a 16 bits. Aquestes funcions permeten tant escriure les diferents dades com llegir-les i estan dirigides amb un codi de funció. Tal com es mostra a la *Taula 4: Funcions Modbus.*

| | | | Funció | Codi (hex.) | Codi (decimal) |
|----------------|------------------------------|-----------------------------|---|----------------|-------------------|
| Accés de dades | Accés a bits | Entrades discretes físiques | <i>Read Discrete Inputs</i> | 02 | 02 |
| | | Bits interns | <i>Read Coils</i> | 01 | 01 |
| | | | <i>Write Single Coil</i> | 05 | 05 |
| | | | <i>Write Multiple Coils</i> | 0F | 15 |
| | Accés a 16 bits | Registres d'entrada física | <i>Read Input Register</i> | 04 | 04 |
| | | Registres interns | <i>Read Holding Registers</i> | 03 | 03 |
| | | | <i>Write Single Register</i> | 06 | 06 |
| | | | <i>Write Multiple Registers</i> | 10 | 16 |
| | | | <i>Read/Write Multiple Registers</i> | 17 | 23 |
| | | | <i>Mask Write Register</i> | 16 | 22 |
| | | <i>Read FIFO Queue</i> | 18 | 24 | |
| | Accés a registres de fitxers | | <i>Read File Record</i> | 14 | 20 |
| | | | <i>Write File Record</i> | 15 | 21 |
| Diagnòstics | | | <i>Read Exception Status</i> | 07 | 07 |
| | | | <i>Diagnostic</i> | 08 | 08 |
| | | | <i>Get Com Event Couner</i> | 0B | 11 |
| | | | <i>Get Com Event Log</i> | 0C | 12 |
| | | | <i>Report Slave ID</i> | 11 | 17 |
| | | | <i>Read Device Identification</i> | 2B | 43 |
| Altres | | | <i>Encapsulated Interface Transport</i> | 2B | 43 |

Taula 4: Funcions Modbus

Encara que Modbus tingui l'opció de programar moltes funcions, només se centrarà en aquelles funcions que requereix el projecte.

4.8 Codis d'excepció Modbus

Els codis d'excepció s'utilitzen quan l'esclau es troba en l'últim cas del punt 4.5 *Casos durant la sol·licitud del dispositiu mestre (p.11)*. Els missatges de resposta d'excepció tenen dos camps que el diferencien d'una resposta normal, el camp de codi de funcions i el camp de dades.

4.8.1 Camp de codi de funcions

En una resposta d'excepció, el dispositiu esclau estableix el bit més significatiu del codi de funció a 1. Això fa que el codi de funció a la resposta d'excepció sigui exactament un valor de 80 hexadecimal superior al valor que seria en una resposta normal. Veure *Taula 5: Relació binària-decimal-hexadecimal*.

| Byte en codi funció d'excepció | Valor en decimal | Valor en hexadecimal |
|--------------------------------|------------------|----------------------|
| 1000000 | 128 | 0x80 |

Taula 5: Relació binària-decimal-hexadecimal

4.8.2 Camp de dades

En una resposta d'excepció, l'esclau retorna un codi d'excepció en el camp de dades. Això defineix la condició que ha provocat no obtenir una resposta normal.

4.8.3 Llistat de codis d'excepció

Cada codi d'excepció té un nom i un significat tal com es pot veure a la *Taula 6: Llistat codis d'excepció Modbus*.

| Codi (hexadecimal) | Nom |
|--------------------|---|
| 01 | Illegal function |
| 02 | Illegal data address |
| 03 | Illegal data value |
| 04 | Server device failure |
| 05 | Acknowledge |
| 06 | Server device busy |
| 08 | Memory parity error |
| 0A | Gateway path unavailable |
| 0B | Gateway target device failed to respond |

Taula 6: Llistat codis d'excepció Modbus

4.9 Interfície RS485

Per dur a terme la implementació del protocol, s'ha optat per l'estàndard RS485, ja que defineix un bus per la transmissió sèrie multipunt, on en un instant, pot haver-hi un equip transmetent dades i varis rebent-les, fins a un total de 32 dispositius. La comunicació realitzada és Half-Duplex, de forma que un equip pot enviar o rebre dades però no simultàniament. El cablejat consisteix en un parell de fils de coure trenats sobre els quals es transmet un senyal diferencial per enviar bits de dades. Aquest, és bastant immune a les interferències i admet distàncies llargues fins a 1200 metres. Veure *Figura 5: Topologia xarxa multipunt RS485 Half-Duplex*.

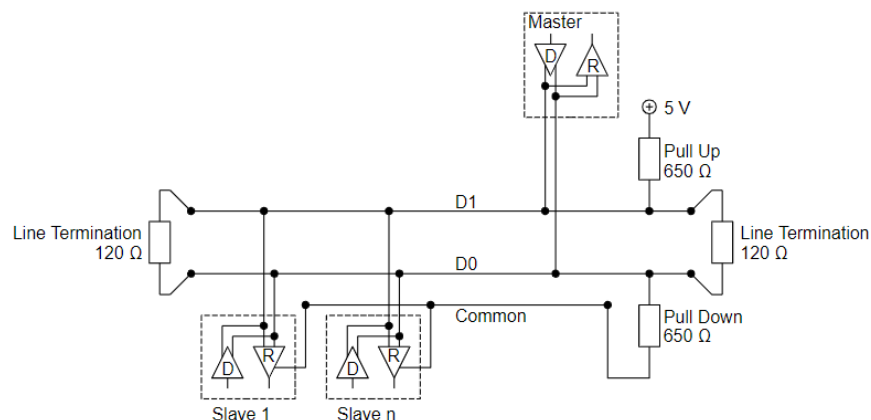


Figura 5: Topologia xarxa multipunt RS485 Half-Duplex

Si el dispositiu mestre del bus no té polarització de línia pròpia, cal connectar un parell de resistències als cables equilibrats RS485:

- Una resistència Pull-Up a una tensió de 5 V al cable D1 (+)
- Una resistència Pull-Down a terra al cable D0 (-)

Com a norma general, les resistències de polarització han de tenir un valor tal que la tensió entre les dues línies del bus sigui superior a 200 mV quan no hi hagi activitat. D'aquesta manera, tot el soroll que es pugui generar al voltant, no superarà aquesta diferència i cap participant tendirà a comunicar. (Telecommunications Industry Association, 2000)

A més, al principi i al final de la xarxa multipunt hi ha d'haver una resistència de terminació de 120 Ω , de tal manera que la resistència de tota la xarxa sigui de 60 Ω .(Marais, 2008)

5 IMPLEMENTACIÓ DEL PROTOCOL MODBUS RTU

En aquest punt es descriuen tots els aspectes necessaris per dur a terme la implementació del protocol.

5.1 Introducció

És convenient programar dos modes de treball per l'aerogenerador. Un primer mode automàtic que ha de permetre que segons la velocitat del vent a l'alçada de les pales, aquestes es posicionin en una orientació predefinida de tal manera que puguin absorbir més l'energia cinètica o pel contrari, que sigui un petit mecanisme de frenada.

Un segon mode test, l'usuari ha de permetre girar les pales en unes orientacions predefinides independentment de la velocitat del vent. Aquest mode és útil per fer proves i per forçar una orientació quan es vulgui.

5.2 Analogia

Com ja s'ha esmentat anteriorment, en un futur projecte es vol fer una completa automatització d'un petit aerogenerador. No obstant això, en aquest projecte, que correspon en una fase inicial, no tenim equips de control com poden ser sensor i actuadors ni accés a l'aerogenerador. Per aquest motiu en aquest punt s'explica la relació entre els equips reals i els senyals que es poden generar o rebre en el microcontrolador.

Per simular la velocitat del vent s'ha utilitzat el potenciòmetre que conté la placa de desenvolupament Curiosity HPC. Aquesta velocitat del vent, s'ha parametritzat de tal manera que pot prendre valors des de zero fins a 25 m/s, ja que per velocitats superiors és convenient frenar l'aerogenerador.

S'ha interpretat que les pales poden prendre quatre orientacions respecte a un punt de referència. Aquestes orientacions corresponen a 30°, 45°, 60° i 90°. Per saber l'angle de les pales, s'han configurat els quatre LEDs que incorpora la placa de desenvolupament Curiosity HPC.

Finalment, s'han incorporat dos LEDs addicionals que permeten saber en quin mode s'està treballant en temps real. Un LED de color verd que indica que el mode de treball és automàtic mentre que un LED vermell indica que s'està treballant en mode test.

El resum de l'analogia es pot observar a la *Taula 7: Analogia aerogenerador - microcontrolador*.

| Pin PIC | Descripció | Nomenclatura plànol | Analogia |
|---------|---------------|---------------------|--------------------------|
| RA0 | Potenciòmetre | RA1+RA2 | Velocitat del vent (m/s) |
| RA7 | LED | EA5 | Angle pales 30° |
| RA6 | LED | EA4 | Angle pales 45° |
| RA5 | LED | EA3 | Angle pales 60° |
| RA4 | LED | EA2 | Angle pales 90° |

Taula 7: Analogia aerogenerador - microcontrolador

5.3 Dispositius

A continuació, es descriuen els diferents dispositius necessaris i escollits per dur a terme el projecte.

5.3.1 PLC

S'ha optat per un autòmat programable que treballarà com a mestre i serà l'encarregat de donar ordres a tots els dispositius esclaus que faran possible el monitoratge i recollida de dades de l'aerogenerador en un futur projecte. El PLC escollit ha estat del fabricant Schneider Electric, concretament el model Modicon TM241C24R, ja que aquests controladors estan preparats per treballar amb el protocol Modbus. Es mostra el PLC a la *Figura 6: PLC TM241C24R Schneider Electrics*.



Figura 6: PLC TM241C24R Schneider Electrics

La part de la programació del PLC es fa càrrec un altre estudiant de la Universitat i es programa amb el SoMachine. De totes maneres, s'inclou la seva programació a l'*Annex C (p.91)*.

5.3.2 Microcontrolador

Els dispositius que treballen com a esclaus, dins del protocol, són uns microcontroladors PIC16F18875 que estan incorporats en una placa de desenvolupament Curiosity High Pin Count (HPC). S'ha escollit aquesta placa, ja que incorpora quatre LEDs, dos pulsadors, un potenciòmetre i diversos pins d'entrada i/o sortida. Tal com es mostra a la *Figura 7: Placa de desenvolupament Curiosity HPC amb el microcontrolador*. D'aquesta manera, es permet fer totes les proves i simulacions sense haver de crear una placa de circuit imprès.

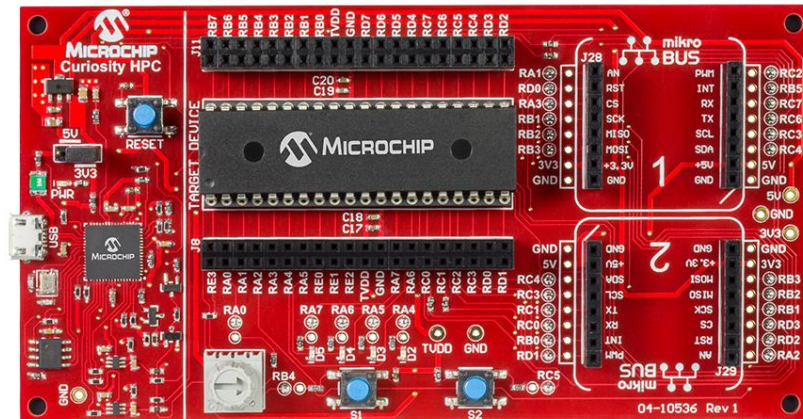


Figura 7: Placa de desenvolupament Curiosity HPC amb el microcontrolador

Tot i que en una fase més avançada del projecte es tindran 3 microcontroladors esclaus que permetran el correcte monitoratge i recollida de dades de l'aerogenerador, en aquesta fase inicial del projecte només es farà ús d'un microcontrolador per tal d'establir la comunicació.

5.3.3 Transceptor RS485

Els microcontroladors treballen amb un dispositiu que controla els ports i els dispositius sèrie, anomenat UART (Universal Asynchronous Receiver – Transmitter). Com la comunicació es fa sobre l'estàndard RS485 i és Half-Dúplex, s'ha hagut d'implementar un transceptor RS485 per establir la comunicació. Aquest, compta amb una capacitat de comunicació Half-Dúplex, detecció d'obertures i curtcircuits, aturada tèrmica i molt més. És molt adequat per transmetre paquets de dades a llargues distàncies i zones sorolloses, mitjançant el bus de cable trenat, que ofereix una bona immunitat contra les interferències electromagnètiques.

El transceptor consta de quatre terminals de caragol, un per l'alimentació a 5V DC, un altre per la connexió a terra, i els altres dos pel senyal parell diferencial, el positiu i negatiu. (MIKROE, 2021)

El transceptor és el que es mostra a la *Figura 8: Transceptor RS422/485* i va connectat als pins superiors de la dreta de la placa de desenvolupament de la *Figura 7: Placa de desenvolupament Curiosity HPC amb el microcontrolador*.

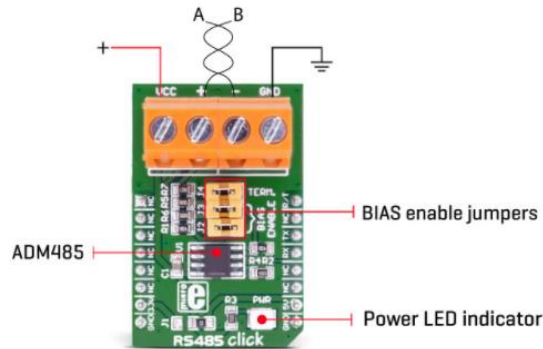


Figura 8: Transceptor RS422/485

Com s'ha esmentat prèviament, el transceptor RS485 pot anar encaixat en una part de la placa de desenvolupament Curiosity HPC. Per aquest motiu, pot ser útil tenir la relació de pins del transceptor. Veure la *Taula 8: Distribució de pins del transceptor RS485*.

| Descripció | Pin | | | Pin | Descripció |
|------------|-----|---|----|-----|----------------------|
| - | NC | 1 | 16 | R/W | Recepció/Transmissió |
| - | NC | 2 | 15 | NC | - |
| - | NC | 3 | 14 | TX | UART transmissió |
| - | NC | 4 | 13 | RX | UART recepció |
| - | NC | 5 | 12 | NC | - |
| - | NC | 6 | 11 | NC | - |
| - | NC | 7 | 10 | 5V | Alimentació |
| Ground | GND | 8 | 9 | GND | Ground |

Taula 8: Distribució de pins del transceptor RS485

És important tenir en compte si s'han de fer servir les resistències de final de línia i les de polarització tal com s'ha esmentat en el capítol 4.9 *Interfície RS485* (p.15). Aquestes resistències són causades per uns petits sòcols que poden ser extrets amb facilitat.

Cada sòcol correspon amb un tipus de resistència. Tal com es mostra a la Taula 9: Configuracions i indicadors del transceptor RS485.

| Etiqueta | Nom | Descripció |
|----------|-------------|-------------------------------|
| LD1 | PWR | Indicador LED d'alimentació |
| JP2 | BIAS ENABLE | Resistència Pull-Up |
| JP3 | BIAS ENABLE | Resistència Pull-Down |
| JP4 | TERM | Resistència de final de línia |

Taula 9: Configuracions i indicadors del transceptor RS485

Com que la xarxa multipunt connecta el PLC amb només un microcontrolador PIC16F18875, cal tenir present que s'han d'incorporar les diferents resistències. Per tant, els petits sòcols JP2, JP3 i JP4 han d'estar posats.

Com s'ha esmentat, el transceptor RS485 va connectat directament als pins de la placa de desenvolupament Cusiosity HPC. Veure *Figura 9: Connexió del transceptor RS485 amb la placa de desenvolupament*.

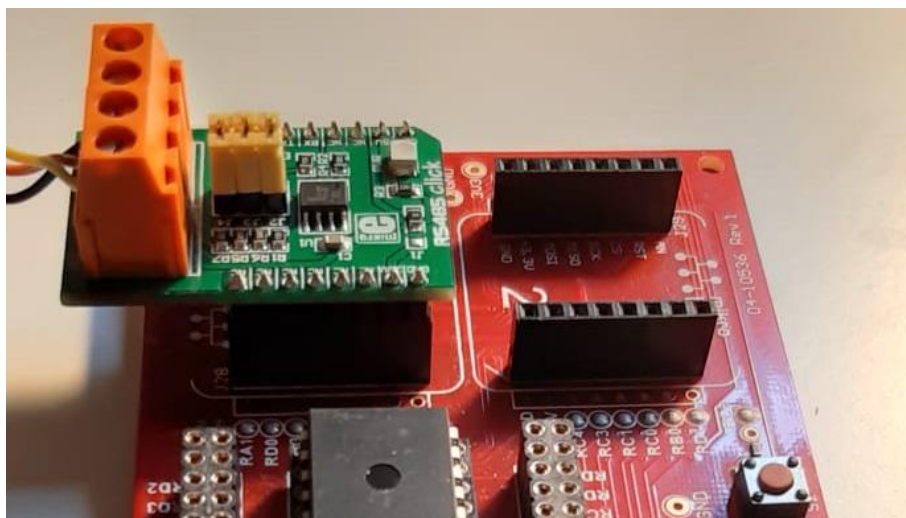


Figura 9: Connexió del transceptor RS485 amb la placa de desenvolupament

5.3.4 Arduino Uno

S'ha utilitzat un Arduino Uno, que és una placa de circuit imprès simple. El principal ús d'aquest dispositiu ha estat per saber la trama que es rebia del PLC i s'enviava cap aquest, ja que des del PIC16F18875 no es poden visualitzar les dades rebudes ni transmeses. Veure *Figura 10: Placa Arduino*.



Figura 10: Placa Arduino

Aleshores, l'arduino ha estat un element clau, ja que hagués sigut totalment impossible programar el microcontrolador PIC16F18875 sense saber com s'enviaven les dades. A més, gràcies a aquest, s'ha pogut veure el CRC que feia servir el PLC.

Inicialment, quan es va programar per primera vegada el microcontrolador PIC16F18875, no va ser possible la comunicació amb el PLC. El principal motiu era que tot i saber com s'havien d'enviar les dades a partir de llegir documents, la realitat era diferent. A més, el CRC inicial implementat no feia servir el mateix polinomi que en el PLC i per molt que s'intentés establir la comunicació no es tenia cap resposta de cap dels dos dispositius.

Per tant, tot i la importància de la petita programació que ha comportat l'Arduino ha estat indirectament relacionada amb el projecte. No obstant això, es pot veure el codi a l'*Annex D* (p. 97).

5.4 MPLAB

Per la programació del microcontrolador s'ha utilitzat el programa MPLAB. També s'ha usat un compilador XC8, ja que proporciona reduccions de mida de codi i millores de velocitat. El codi elaborat que permet la comunicació és estructurat i s'han utilitzat llibreries que el mateix programa ofereix. Tal com es pot veure a la *Figura 11: Estructura del codi*.

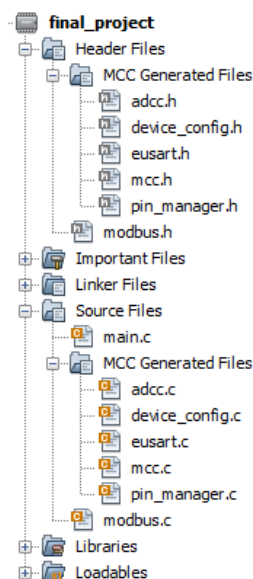


Figura 11: Estructura del codi

El programa permet generar automàticament una sèrie de llibreries dins una carpeta anomenada *MCC Generated Files* tal com es pot veure a la Figura 11. Aquestes llibreries existeixen en el programa en forma de recursos, on es poden seleccionar els recursos necessaris. En el projecte, només s'han utilitzat dos recursos: EUSART (Enhanced Universal Synchronous Asynchronous Receiver Transmitter) i ADCC (Analog to Digital Conversion with Computation). Els altres recursos es generen automàticament quan se seleccionen els pins corresponents a la programació. Tal com es pot veure a la Figura 12: Gestor de pins del microcontrolador.

| Package: PDIP40 | | | Pin No: | | | 2 | 3 | 4 | 5 | 6 | 7 | 14 | 13 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 15 | 16 | 17 | 18 | 23 | 24 | 25 | 26 | 19 | 20 | 21 | 22 | 27 | 28 | 29 | 30 | 8 | 9 | 10 | 1 |
|-----------------|----------|-----------|----------|---|---|---|---|---|---|----------|---|----|----|----|----|----|----------|----|----|----|----|----|----|----------|----|----|----|----|----|----|----------|----|----|----|----|----|----|---|---|----|---|
| | | | Port A ▼ | | | | | | | Port B ▼ | | | | | | | Port C ▼ | | | | | | | Port D ▼ | | | | | | | Port E ▼ | | | | | | | | | | |
| Module | Function | Direction | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | | | |
| ADCC ▼ | ADCACT | input | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | ADGRDA | output | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | ADGRDB | output | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | ANx | input | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | VREF+ | input | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | VREF- | input | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| EUSART ▼ | RX | input | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | TX | output | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| OSC | CLKOUT | output | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Pin Module ▼ | GPIO | input | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | GPIO | output | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RESET | MCLR | input | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Figura 12: Gestor de pins del microcontrolador

Per tant, en el gestor de pins que es mostra a la Figura 12, s'han seleccionat tots els pins necessaris per dur a terme el projecte. Alguns pins s'han declarat com a entrades mentre que d'altres com a sortides.

El programa també incorpora una finestra on es poden canviar el nom de les variables dels pins seleccionats. Tal com es mostra a la Figura 13: Pin module.

| Easy Setup Registers | | | | | | |
|---------------------------|------------|----------|--------------------|--------------------------|-------------------------------------|-------------------------------------|
| Selected Package : PDIP40 | | | | | | |
| Pin Name | Module | Function | Custom Name | Start High | Analog | Output |
| RA0 | ADCC | ANA0 | pote | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| RA4 | Pin Module | GPIO | angle_90 | <input type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> |
| RA5 | Pin Module | GPIO | angle_60 | <input type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> |
| RA6 | Pin Module | GPIO | angle_45 | <input type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> |
| RA7 | Pin Module | GPIO | angle_30 | <input type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> |
| RC2 | Pin Module | GPIO | RS485_TXEN | <input type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> |
| RC6 | EUSART | TX | | <input type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> |
| RC7 | EUSART | RX | | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| RD6 | Pin Module | GPIO | automatic_mode_led | <input type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> |
| RD7 | Pin Module | GPIO | test_mode_led | <input type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> |

Figura 13: Pin module

A més, dóna l'opció d'escollir que el valor inicial del pin estigui activat i declarar les dades del pin com a analògiques o digital.

Una opció molt còmoda que ofereix el programa és que es pot veure quins pins estan ocupats i en quina posició del microcontrolador. Tal com es pot observar a la *Figura 14: Vista dels pins usats en el microcontrolador*.

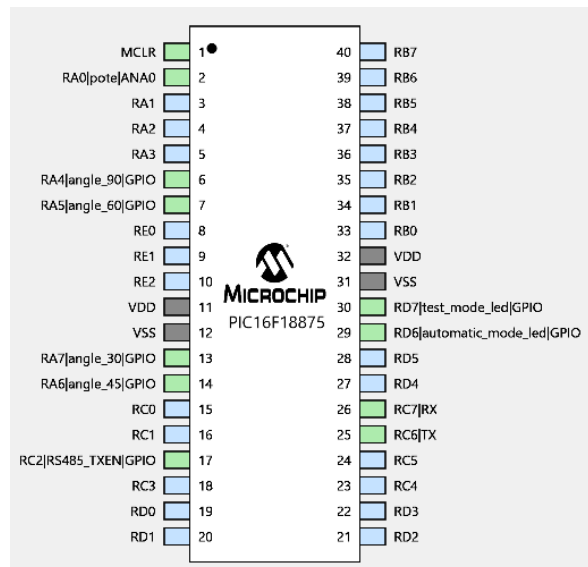


Figura 14: Vista dels pins usats en el microcontrolador

Per veure les configuracions inicials dels recursos EUSART i ADCC veure *Annex A (p.57)*

5.4.1 Recurs ADCC

El recurs ADCC permet la conversió analògica – digital d'un senyal d'entrada binària de 10 bits.

Pel corresponent projecte, es programa de tal manera que el potènciometre prengui valors dins un rang entre 0 i 25. D'aquesta manera es simularà la velocitat del vent en m/s. A continuació s'adjunta el codi que permet fer la conversió:

```
adc_result = ADCC_GetSingleConversion(pote);  
wind_conversion = (adc_result * max_wind/1023);
```

La variable `adc_result` guarda la conversió feta pel potènciometre. Aquesta conversió és de 10 bits i per tant, s'obté un rang de valors que va des de 0 a 1023. Aleshores, la variable `wind_conversion` fa una operació per escalar els 1023 valors en el rang que va de 0 a 25.

5.5 Mapa de memòria Modbus

A continuació, es detalla el mapa de memòria, ja que és important per monitorar, configurar i controlar el mòdul d'entrades i sortides. Es mostra a la *Taula 10: Mapa de memòria Modbus*.

| Paràmetre | Registre | Adreça | Descripció |
|-----------------------|----------|--------|--|
| Potenciòmetre RA1+RA2 | 3X | 0000 | Simula la velocitat del vent en m/s |
| LED EA5 | 0X | 0000 | Informa que l'angle de les pales es de 30° |
| LED EA4 | | 0001 | Informa que l'angle de les pales es de 45° |
| LED EA3 | | 0002 | Informa que l'angle de les pales es de 60° |
| LED EA2 | | 0003 | Informa que l'angle de les pales es de 90° |
| Automàtic | 4X | 0000 | Escriu el mode de treball: automàtic o test |
| Test | | 0001 | |
| Valor test | | 0002 | Escriu el valor de l'angle de les pales en mode test |

Taula 10: Mapa de memòria Modbus

Quan el paràmetre *Automàtic* és u i el paràmetre *Test* és zero, el mode de treball és automàtic. En canvi, quan el mode *Automàtic* val zero i el mode *Test* val u, el mode de treball és test.

5.6 Funcions Modbus utilitzades

S'ha vist que el protocol Modbus incorpora varies funcions com les de la *Taula 4*. No obstant, només es farà ús de tres funcions: Read Coils (0x01), Read Input Registers (0x04) i Write Multiple Registers (0x10).

Aquestes funcions s'han programat amb un adreçament i una determinada longitud, tal com es mostra a *Taula 11: Resum funcions Modbus implementades*.

| Funció | Adreça inicial | Longitud |
|--------------------------|----------------|----------|
| Read Coils | 0000 | 4 |
| Read Input Registers | 0000 | 1 |
| Write Multiple Registers | 0000 | 3 |

Taula 11: Resum funcions Modbus implementades

5.6.1 Read Coils (0x01)

Aquesta funció permet al mestre indagar per l'estat de les sortides discretes (coils) de l'esclau. Aquestes sortides tenen un estat binari ON/OFF i una referència Modbus 0X. L'ús principal d'aquesta funció en el projecte és saber l'angle de les pales sempre que se sol·liciti des del PLC.

Per tant, abans de saber la informació de les pales, des del PLC s'ha d'enviar una sol·licitud tal com es pot veure a la *Taula 12: Sol·licitud Read Coils des del PLC*.

| | Esclau | Funció | Adreça d'inici | | Nombre de coils | | CRC | |
|-------|--------|--------|----------------|---|-----------------|---|-----|-----|
| Byte | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Trama | 1 | 1 | 0 | 0 | 0 | 4 | 61 | 201 |

Taula 12: Sol·licitud Read Coils des del PLC

Segons la *Taula 12*, la sol·licitud s'està dirigint a l'esclau número 1 amb la funció Read Coils on concretament vol saber l'estat de 4 coils amb una adreça inicial de zero.

Finalment, després de la sol·licitud del PLC cal una resposta del microcontrolador. Aquesta resposta dependrà del valor dels coils. Per tant, com que es sol·licita l'estat de quatre coils, hi haurà com a molt quatre respostes diferents. Tal com es mostra a la *Taula 13: Respostes Read Coils del microcontrolador*.

| | Esclau | Funció | Nombre de bytes | Data | CRC | |
|---------|--------|--------|-----------------|------|-----|----|
| Byte | 0 | 1 | 2 | 3 | 4 | 5 |
| Trama 1 | 1 | 1 | 1 | 1 | 144 | 72 |
| Trama 2 | 1 | 1 | 1 | 2 | 208 | 73 |
| Trama 3 | 1 | 1 | 1 | 4 | 80 | 75 |
| Trama 4 | 1 | 1 | 1 | 8 | 80 | 78 |

Taula 13: Respostes Read Coils del microcontrolador

Cada valor de resposta obtingut fa referència al fet que un dels quatre LEDs està encès. Per exemple, quan la resposta és de valor 4, el LED EA3 està encès, com es mostra a la Taula 14.

| Data | Binari | | | | Decimal |
|---------|---------|---------|---------|---------|---------|
| - | LED EA2 | LED EA3 | LED EA4 | LED EA5 | - |
| Trama 1 | 0 | 0 | 0 | 1 | 1 |
| Trama 2 | 0 | 0 | 1 | 0 | 2 |
| Trama 3 | 0 | 1 | 0 | 0 | 4 |
| Trama 4 | 1 | 0 | 0 | 0 | 8 |

Taula 14: Equivalència resposta amb el LED

5.6.2 Read Input Registers (0x04)

Aquesta funció permet llegir l'estat dels registres d'entrada de l'esclau. Els registres d'entrada tenen una longitud de 16 bits, tenen una referència 3X i es comencen a dirigir a partir de l'adreça zero.

En el respectiu projecte, aquesta funció fa referència a la velocitat del vent en m/s. Aleshores, la trama de sol·licitud que envia el PLC es mostra a la Taula 15: Sol·licitud Read Input Registers des del PLC.

| | Esclau | Funció | Adreça d'inici | | Nombre de coils | | CRC | |
|-------|--------|--------|----------------|---|-----------------|---|-----|-----|
| Byte | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Trama | 1 | 4 | 0 | 0 | 0 | 1 | 49 | 202 |

Taula 15: Sol·licitud Read Input Registers des del PLC

La resposta proporcionada del microcontrolador serà causada pel valor del potenciòmetre. No obstant això, el format de la trama vindrà donada tal com es mostra a la *Taula 16: Resposta Read Input Registers del microcontrolador*.

| | Esclau | Funció | Nombre de bytes | Data | | CRC | |
|---------|--------|--------|-----------------|------|---|-----|---|
| Byte | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Trama 1 | 1 | 4 | 2 | A | B | C | D |

Taula 16: Resposta Read Input Registers del microcontrolador

On els valors A i B de la *Taula 16* depenen de la posició del potenciòmetre i els valors C i D depenen directament dels valors A i B.

5.6.3 Write Multiple Registers (0x10)

Aquesta funció és una seqüència de registres de sortida amb referència 4X. Com en altres funcions, aquesta s'inicia amb una adreça de zero.

En el respectiu projecte, aquesta funció escriu en el mode en què es vol treballar, automàtic o test, i en el cas de treballar en mode test, el valor del coil que es vol forçar. Per tant, podem diferenciar dues trames diferents.

En primer lloc, quan es vol treballar en mode automàtic la sol·licitud del PLC serà causada per la trama que es mostra a la *Taula 17: Sol·licitud Write Multiple Registers des del PLC - mode automàtic*.

| | Esclau | Funció | Adreça inicial | | Quantitat | | Nº bytes | Data 1 | | Data 2 | | Data 3 | | CRC | |
|------|--------|--------|----------------|---|-----------|---|----------|--------|---|--------|----|--------|----|-----|----|
| Byte | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| T | 1 | 16 | 0 | 0 | 0 | 3 | 6 | 0 | 1 | 0 | 0 | 0 | A | B | C |

Taula 17: Sol·licitud Write Multiple Registers des del PLC - mode automàtic

On el valor de A de la *Taula 17* pot ser qualsevol valor que no queda afectat, ja que estem en mode automàtic i no forcem cap coil.

En segon lloc, quan es vol treballar en mode test, es vol forçar un dels quatre coils que representa l'angle de les pales de l'aerogenerador. Per tant, segons el coil que es vol forçar tindrem quatre trames diferents de sol·licitud del PLC, tal com es mostra a la *Taula 18: Sol·licitud Write Multiple Registers des del PLC - mode test*.

| | Esclau | Funció | Adreça inicial | | Quant. | | Nº bytes | Data 1 | | Data 2 | | Data 3 | | CRC | |
|--------|--------|--------|----------------|---|--------|---|----------|--------|---|--------|----|--------|----|-----|-----|
| B | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| T 1 | 1 | 16 | 0 | 0 | 0 | 3 | 6 | 0 | 0 | 0 | 1 | 0 | 0 | 183 | 64 |
| T 2 | 1 | 16 | 0 | 0 | 0 | 3 | 6 | 0 | 0 | 0 | 1 | 0 | 1 | 118 | 128 |
| T 3 | 1 | 16 | 0 | 0 | 0 | 3 | 6 | 0 | 0 | 0 | 1 | 0 | 2 | 54 | 128 |
| T 4 | 1 | 16 | 0 | 0 | 0 | 3 | 6 | 0 | 0 | 0 | 1 | 0 | 3 | 247 | 65 |

Taula 18: Sol·licitud Write Multiple Registers des del PLC - mode test

La resposta proporcionada pel microcontrolador serà la mateixa independentment del mode de treball. Aquesta resposta només inclou el número d'esclau, la funció utilitzada, l'adreça de començament, la quantitat de registres i el CRC. Tal com es pot veure a la *Taula 19: Resposta Write Multiple Registers del microcontrolador*.

| | Esclau | Funció | Adreça d'inici | | Quantitat de registres | | CRC | |
|-------|--------|--------|----------------|---|------------------------|---|-----|---|
| Byte | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Trama | 1 | 16 | 0 | 0 | 0 | 3 | 128 | 8 |

Taula 19: Resposta Write Multiple Registers del microcontrolador

5.7 Codis d'excepció Modbus utilitzats

S'ha vist que quan el dispositiu esclau no pot gestionar la sol·licitud rebuda, genera una resposta amb un codi d'excepció. Per la implementació del protocol només s'utilitzen tres codis d'excepció: el 0x01, 0x02 i 0x03.

El procés del microcontrolador per gestionar una resposta normal o amb el codi d'excepció es mostra a la *Figura 15: Procés d'enviar resposta al PLC*.

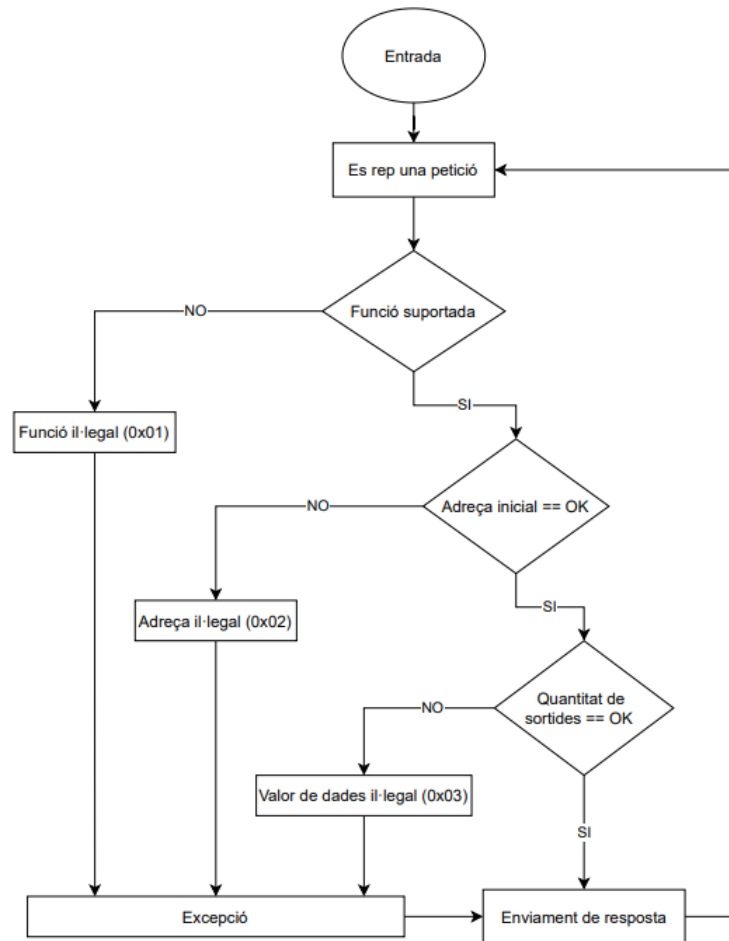


Figura 15: Procés d'enviar resposta al PLC

Per saber que es tracta d'una resposta amb codi d'excepció, en el camp de funció s'escriu el valor de funció més 80 en base hexadecimal i en el camp de dades, s'escriu el codi d'excepció. Veure *Taula 20: Camp de funció i dades en una resposta amb excepció*.

| | 0x01 | 0x02 | 0x03 |
|----------------------|------|------|------|
| Funció (hex.) | 81 | 82 | 83 |
| Codi excepció (hex.) | 01 | 02 | 03 |

Taula 20: Camp de funció i dades en una resposta amb excepció

5.7.1 Illegal function (0x01)

La funció rebuda en el microcontrolador no està permesa. Qualsevol funció sol·licitada diferent de les programades ha de retornar aquesta excepció. El codi d'excepció és 01 en base hexadecimal.

5.7.2 Illegal data address (0x02)

L'adreça de dades rebuda de la sol·licitud no és una adreça permesa pel microcontrolador. En altres paraules, si l'adreça d'inici en una sol·licitud és diferent de la que el microcontrolador espera, el codi d'excepció serà 02 en base hexadecimal. Concretament, les tres funcions programades, tenen una adreça inicial de zero.

5.7.3 Illegal data value (0x03)

Un valor contingut al camp de referència de dades de la sol·licitud no és un valor permès pel microcontrolador. Això indica un error d'estructura, com ara que la longitud implícita és incorrecte. Cada funció programada té una longitud determinada, tal com es mostra a la *Taula 11*. Per tant, si la longitud de la petició és diferent de la longitud programada, el codi d'excepció serà 03 en base hexadecimal.

5.8 CRC Implementat

Schneider Electric utilitza 16 bits pel càlcul del CRC que empaqueta en 2 bytes de 8 bits. Aleshores, les trames contenen el byte més significatiu (MSB) i el menys significatiu (LSB), on envia primer el byte menys significatiu seguit del més significatiu.

Per saber el polinomi del CRC que fa servir el PLC, he calgut una calculadora online que feia càlculs de CRC. Veure *Figura 16: Calculadora CRC online*.

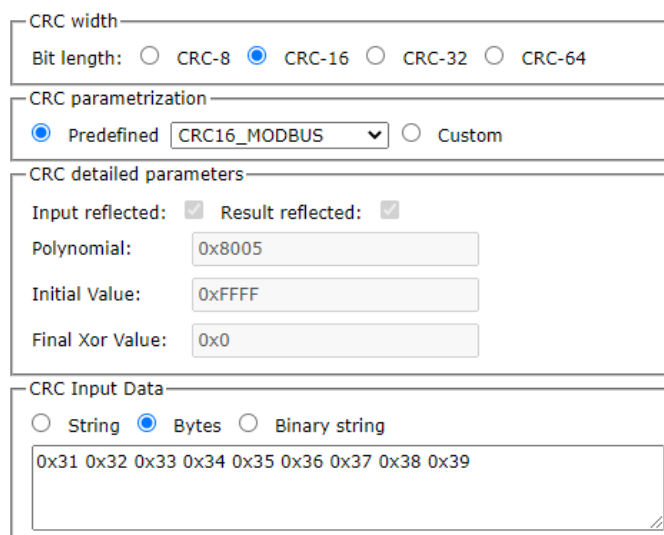


Figura 16: Calculadora CRC online

S'ha comparat el valor del CRC del PLC amb el valor que proporcionava la calculadora online i es pot afirmar que en tots els casos han coincidit. Per tant, s'ha programat un CRC en el

microcontrolador de tal manera que proporcioni el mateix CRC que el PLC, utilitzant el polinomi 0x8005 amb un valor inicial de 0xFFFF.

S'ha optat per programar el CRC del microcontrolador a partir d'un mètode anomenat *Look Up Table* (LUT). Aquest mètode necessita anar a buscar el valor del CRC en una taula predefinida i que varia segons el polinomi utilitzat.

5.9 Funcionament general del protocol implementat

En aquest punt es descriu el funcionament gernal del protocol implementat. El microcontrolador sempre està pendent si arriba una sol·licitud per part del PLC. Quan aquesta arriba, el microcontrolador llegeix els dos primers bytes, que corresponen al byte del número d'esclau i el byte amb la funció sol·licitada. Si el número d'esclau no coincideix, vol dir que la sol·licitud no va dirigit en aquest esclau, però en el present projecte només consta d'un esclau i això no passarà

Segons la funció que es llegeix, el microcontrolador seguirà llegint un nombre determinat de bytes, ja que la longitud depèn de la funció programada. Per tant, si es tracta de la funció 0x01 o 0x04, el microcontrolador llegirà sis bytes més, obtenint un total de longitud de trama de 8 bytes. Per contra, si es tracta de la funció 0x10, el microcontrolador llegirà 13 bytes més, obtenint un total de longitud de trama de 15 bytes. Veure *Figura 17: Diagrama de flux del funcionament del protocol implementat*.

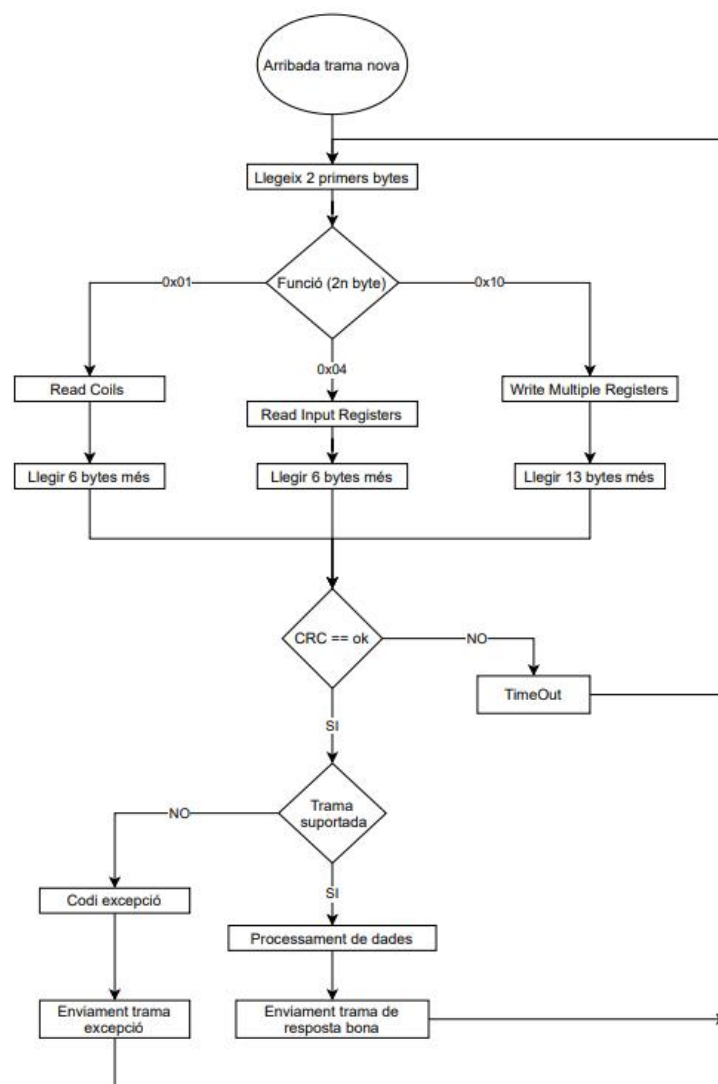


Figura 17: Diagrama de flux del funcionament del protocol implementat

Una vegada el microcontrolador té tota la trama guardada en una posició de memòria en forma de vector, comprova si el CRC rebut en els últims dos bytes de la trama correspon amb el CRC que calcula. Si els dos valors coincideixen, es procedeix a fer el processament de les dades. En cas que no coincideixi, el microcontrolador no farà cap acció i el temps d'espera de la resposta del PLC s'esgotarà i es tornarà a enviar una petició nova.

Quan els dos valors coincideixen, es comprova que la trama és suportada pel microcontrolador, concretament es comprova la funció, la direcció i la longitud. Si algun d'aquests tres paràmetres no és suportat, es crea una resposta d'excepció. En cas que tots tres paràmetres siguin suportats, es procedeix a fer les accions sobre el microcontrolador i a donar resposta al PLC.

El funcionament es repeteix sempre que el PLC envii una sol·licitud.

6 PROVES DE COMUNICACIÓ

En aquest capítol es mostren les proves de comunicació així com el funcionament del protocol entre l'autòmat programable i el microcontrolador en els dos modes de treball.

L'autòmat s'ha programat amb el programa SoMachine d'Schneider Electric. Aquest, incorpora blocs de funcions.

En el programa s'han implementat 5 blocs de funcions diferents:

- Bloc ADDM
- Bloc BLINK
- Bloc WRITE_VAR
- Bloc READ_VAR
- Bloc MOVE

Veure *C.3 Descripció dels blocs de funcions (p.92)* per entrar en detall amb les funcions dels diferents blocs.

6.1 Mode test

El mode de treball test permet forçar l'orientació de les pales en unes posicions predefinides des del programa SoMachine. Aquestes orientacions es veuen reflectides en l'estat dels quatre LEDs.

En primer lloc, s'ha d'escriure aquest mode de treball escrivint un registre en els blocs de funcions MOVE amb els valors:

- AUTO = 0
- TEST = 1

També cal afegir quina orientació predefinida es vol forçar, escrivint el valor corresponent en el bloc VALOR_TEST on el valor a escriure serà un dels quatre que es mostra a la *Taula 21:Valors que pot pendre el bloc VALOR_TEST* i depedrà segons quina orientació es vol.

| Valor | Visualització PIC | Pin | Descripció |
|-------|-------------------|-----|----------------------------------|
| 0 | LED EA5 == 1 | RA7 | Representa una orientació de 30° |
| 1 | LED EA4 == 1 | RA6 | Representa una orientació de 45° |
| 2 | LED EA3 == 1 | RA5 | Representa una orientació de 60° |
| 3 | LED EA2 == 1 | RA4 | Representa una orientació de 90° |

Taula 21: Valors que pot pendre el bloc VALOR_TEST

A continuació es mostra un exemple on es força el LED EA3 que representa una orientació de 60°. Veure *Figura 18: Exemple forçar LED EA3 en mode test.*

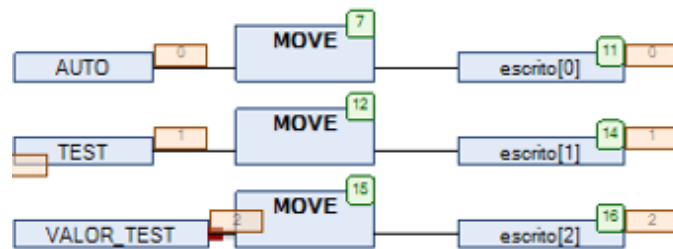


Figura 18: Exemple forçar LED EA3 en mode test

Quan ja s'ha forçat el valor del registre que conté el mode de treball i el valor de l'orientació en el mode test, s'encén el LED de color vermell que es troba fora la placa de desenvolupament Curiosity HPC i indica el mode de treball. D'aquesta manera, una persona externa pot saber el mode de treball a temps real. Veure *Figura 19: Visualització del mode de treball test en el microcontrolador.*

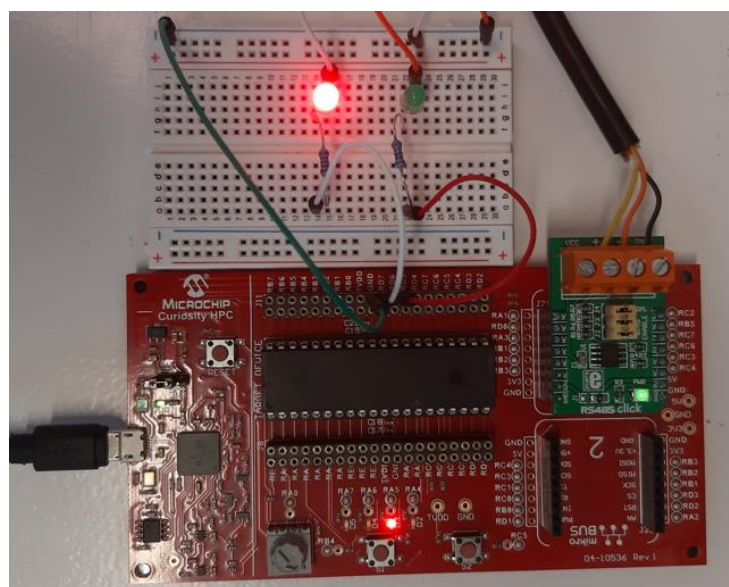


Figura 19: Visualització del mode de treball test en el microcontrolador

6.1.1 Orientació 30°

En aquest cas es força l'orientació de 30°, escrivint el bloc VALOR_TEST un valor zero. Tal com es mostra a la *Figura 20: Registre per forçar una orientació de 30°*.

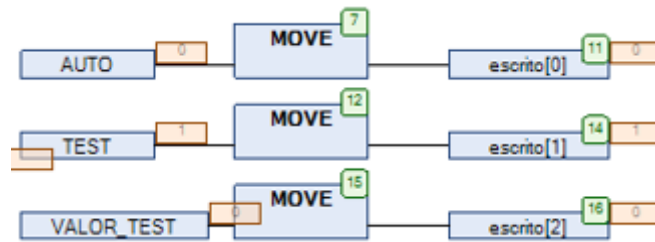


Figura 20: Registre per forçar una orientació de 30°

El microcontrolador executa les dades rebudes en forma de trama. El resultat de l'execució és que s'encén el LED EA5 tal com es mostra a la *Figura 21: Visualització dels LEDs en el microcontrolador per 30°*.

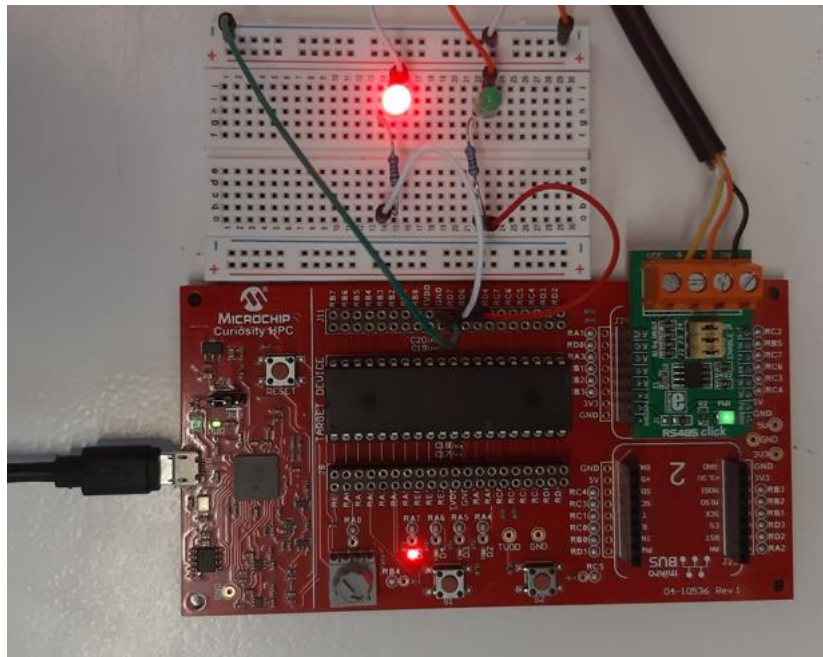


Figura 21: Visualització dels LEDs en el microcontrolador per 30°

A continuació es mostra el valor del LED encès des del programa SoMachine. D'aquesta manera, des de l'ordinador es pot saber l'angle de la pala en cada instant de temps. Veure *Figura 22: Valor LED des del SoMachine amb orientació 30°*.



Figura 22: Valor LED des del SoMachine amb orientació 30°

Per saber l'equivalència del LED encès amb el valor llegit per part del PLC, veure *Taula 14: Equivalència resposta amb el LED*.

6.1.2 Orientació 45°

En aquest cas es força l'orientació de 45°, escrivint el bloc VALOR_TEST un valor u. Tal com es mostra a la *Figura 23: Registre per forçar una orientació de 45°*.

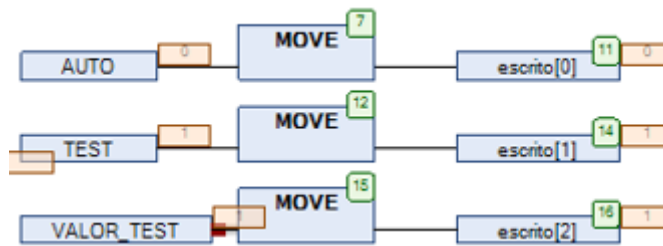


Figura 23: Registre per forçar una orientació de 45°

El microcontrolador executa les dades rebudes en forma de trama. El resultat de l'execució és que s'encén el LED EA4 tal com es mostra a la *Figura 24: Visualització dels LEDs en el microcontrolador per 45°*.

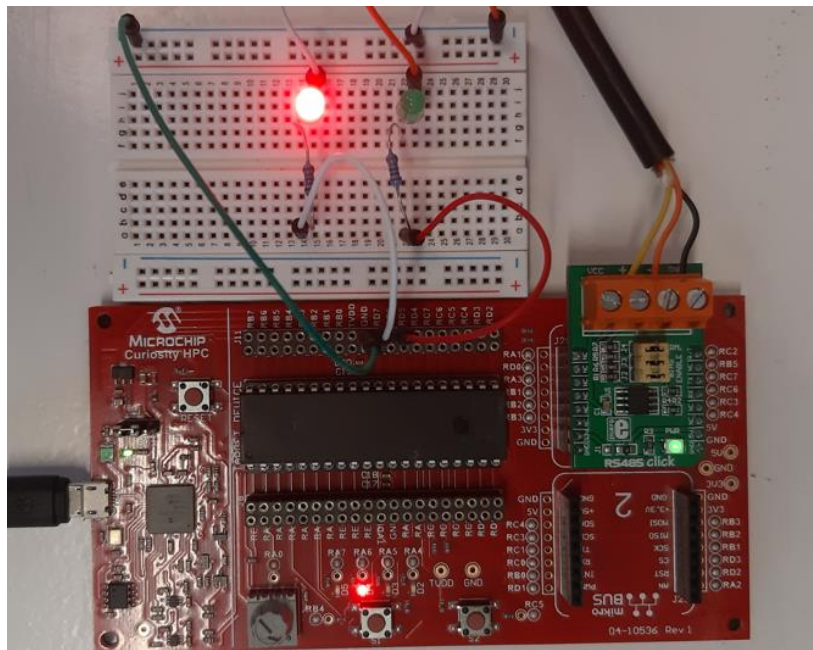


Figura 24: Visualització dels LEDs en el microcontrolador per 45°

A continuació es mostra el valor del LED encès des del programa SoMachine. D'aquesta manera, des de l'ordinador es pot saber l'angle de la pala en cada instant de temps. Veure *Figura 25: Valor LED des del SoMachine amb orientació 45°*.

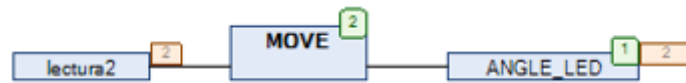


Figura 25: Valor LED des del SoMachine amb orientació 45°

Per saber l'equivalència del LED encès amb el valor llegit per part del PLC en el bloc ANGLE_LED, veure *Taula 14: Equivalència resposta amb el LED*.

6.1.3 Orientació 60°

En aquest cas es força l'orientació de 60°, escrivint el bloc VALOR_TEST un valor dos. Tal com es mostra a la *Figura 26: Registre per forçar una orientació de 60°*.

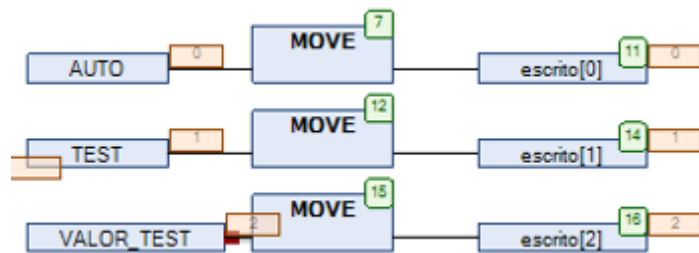


Figura 26: Registre per forçar una orientació de 60°

El microcontrolador executa les dades rebudes en forma de trama. El resultat de l'execució és que s'encén el LED EA3 tal com es mostra a la *Figura 27: Visualització dels LEDs en el microcontrolador per 60°*.

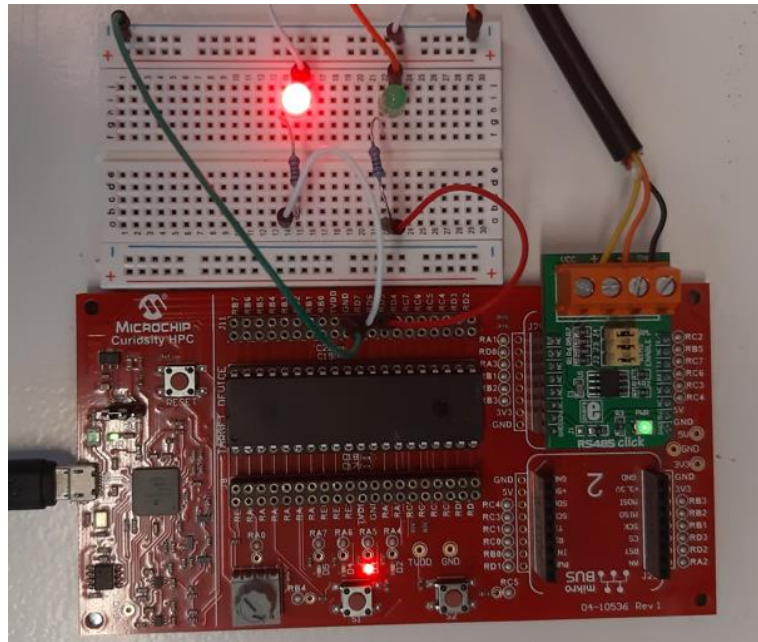


Figura 27: Visualització dels LEDs en el microcontrolador per 60°

A continuació es mostra el valor del LED encès des del programa SoMachine. D'aquesta manera, des de l'ordinador es pot saber l'angle de la pala en cada instant de temps. Veure *Figura 28: Valor LED des del SoMachine amb orientació 60°*.



Figura 28: Valor LED des del SoMachine amb orientació 60°

Per saber l'equivalència del LED encès amb el valor llegit per part del PLC en el bloc ANGLE_LED, veure *Taula 14: Equivalència resposta amb el LED*.

6.1.4 Orientació 90°

En aquest cas es força l'orientació de 90°, escrivint el bloc VALOR_TEST un valor tres. Tal com es mostra a la *Figura 29: Registre per forçar una orientació de 90°*.

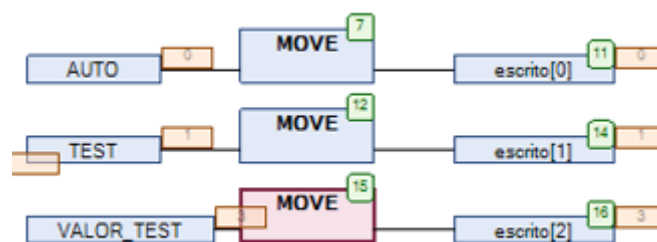


Figura 29: Registre per forçar una orientació de 90°

El microcontrolador executa les dades rebudes en forma de trama. El resultat de l'execució és que s'encén el LED EA2 tal com es mostra a la *Figura 30: Visualització dels LEDs en el microcontrolador per 90°*.

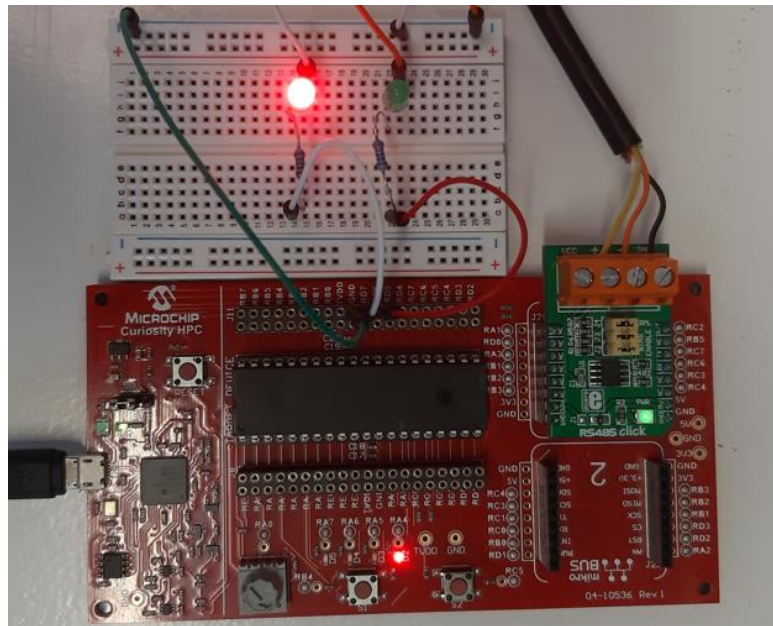


Figura 30: Visualització dels LEDs en el microcontrolador per 90°

A continuació es mostra el valor del LED encès des del programa SoMachine. D'aquesta manera, des de l'ordinador es pot saber l'angle de la pala en cada instant de temps. Veure *Figura 31: Valor LED des del SoMachine amb orientació 90°*.



Figura 31: Valor LED des del SoMachine amb orientació 90°

Per saber l'equivalència del LED encès amb el valor llegit per part del PLC en el bloc ANGLE_LED, veure *Taula 14: Equivalència resposta amb el LED*.

6.2 Mode automàtic

El mode de treball automàtic permet regular l'orientació de les pales en una de les 4 orientacions predefinides segons la intensitat del vent.

En primer lloc, s'ha d'escriure aquest mode de treball escrivint un registre en els blocs de funcions MOVE amb els valors:

- AUTO = 1
- TEST = 0

Veure els registres a la *Figura 32: Registre en mode automàtic*.

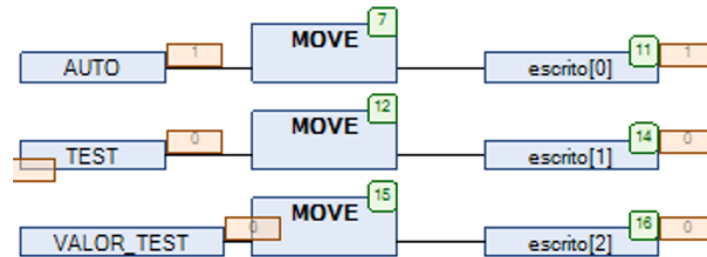


Figura 32: Registre en mode automàtic

En aquest mode de treball no cal afegir quina orientació predefinida es vol forçar, ja que serà causada pel valor del vent (potenciòmetre). Tanmateix, segons la velocitat del vent, les pales s'orienten en una posició predefinida. Veure *Figura 33: Rang de valors del vent i angle de les pales*.

| Valor (m/s) | Visualització PIC | Pin | Descripció |
|-------------|-------------------|-----|----------------------------------|
| [0, 6] | LED EA5 == 1 | RA7 | Representa una orientació de 30° |
| (6, 12] | LED EA4 == 1 | RA6 | Representa una orientació de 45° |
| (12, 19] | LED EA3 == 1 | RA5 | Representa una orientació de 60° |
| (19, 25] | LED EA2 == 1 | RA4 | Representa una orientació de 90° |

Figura 33: Rang de valors del vent i angle de les pales

Quan es fa la transmissió de dades, s'encén el LED de color verd que es troba fora la placa de desenvolupament Curiosity HPC i indica el mode de treball. Veure *Figura 34: Visualització del mode de treball automàtic en el microcontrolador*.

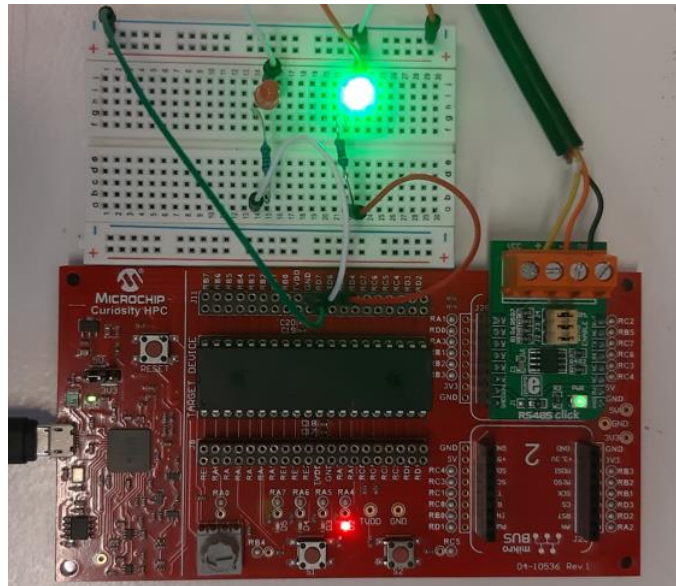


Figura 34: Visualització del mode de treball automàtic en el microcontrolador

A continuació, es fan dues proves de comunicació posant el potenciòmetre en 2 posicions diferents.

6.2.1 Posició 1 del potenciòmetre

En primer lloc, posicionem el potenciòmetre en una posició a l'atzar. Veure *Figura 35: Posició 1 del potenciòmetre*.

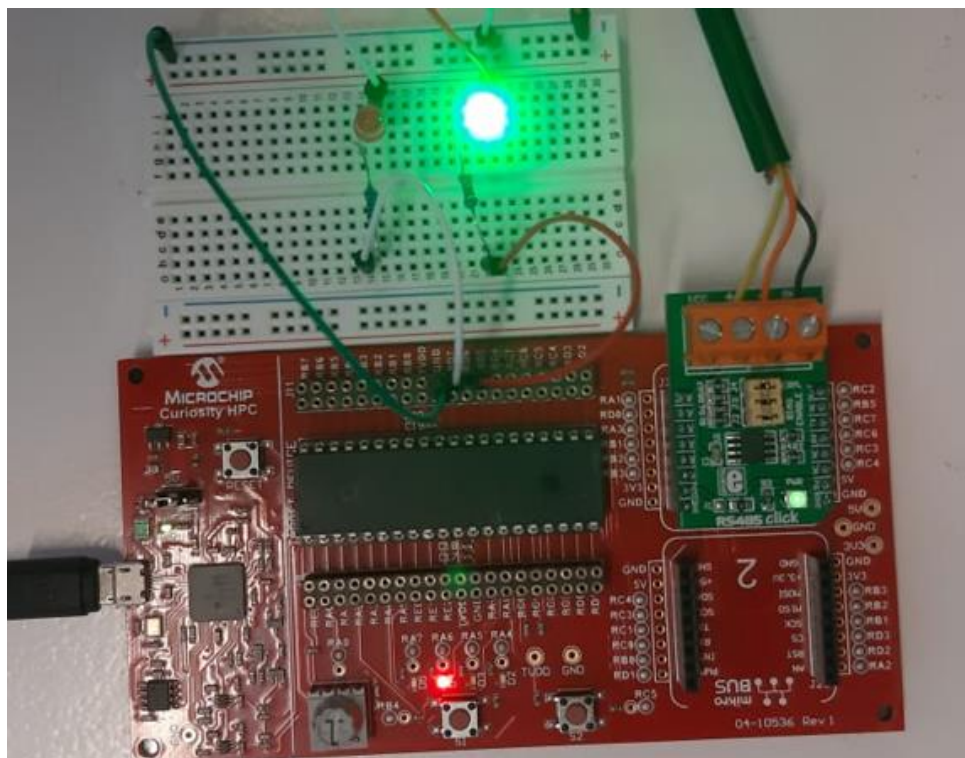


Figura 35: Posició 1 del potenciòmetre

S'observa perfectament el mode de treball a partir del LED extern de color verd. A més, segons els LEDs interns, la posició de les pales correspondria a 45°.

Des del SoMachine, es pot veure la velocitat del vent, que és causada per la posició del potenciòmetre, on pren el valor de 12 m/s. Addicionalment, també es pot veure el LED que es troba encès en tot moment. Veure *Figura 36: Valor del LED i potenciòmetre a SoMachine per la posició 1.*

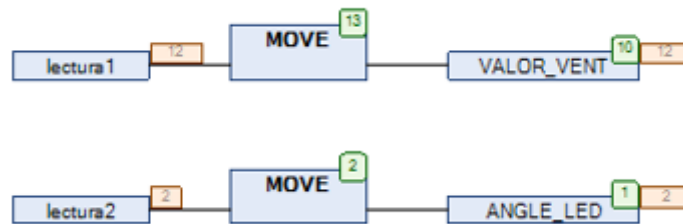


Figura 36: Valor del LED i potenciòmetre a SoMachine per la posició 1

Per saber l'equivalència del LED encès amb el valor llegit per part del PLC en el bloc ANGLE_LED, veure *Taula 14: Equivalència resposta amb el LED.*

6.2.2 Posició 2 del potenciòmetre

Posicionem el potenciòmetre en una segona posició, concretament el posem al màxim. Veure *Figura 37: Posició 2 del potenciòmetre.*

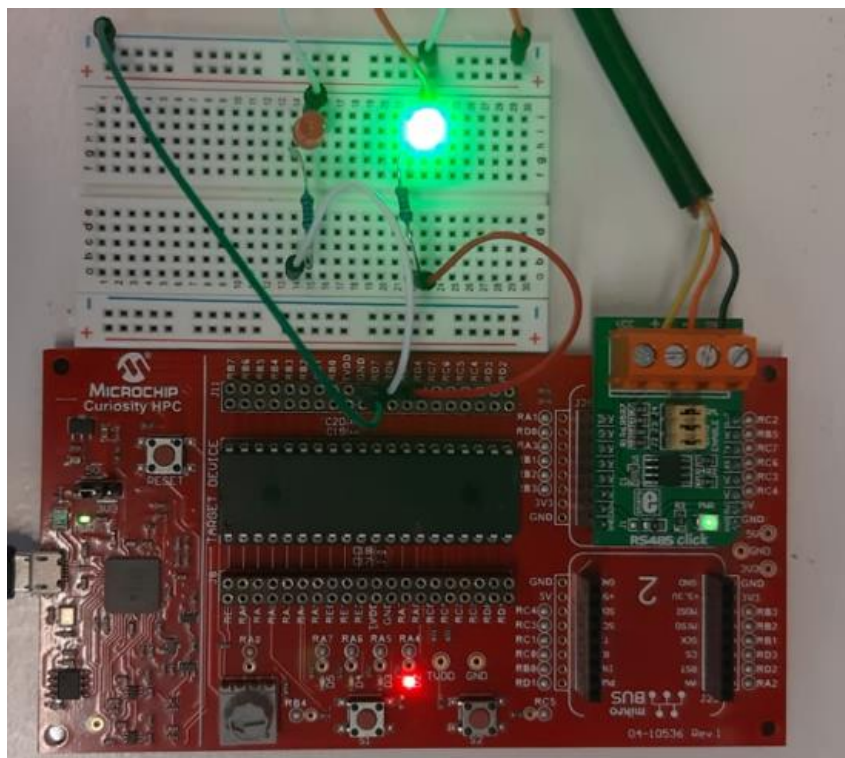


Figura 37: Posició 2 del potenciòmetre

En aquest cas, la velocitat del vent és de 25 m/s i el LED encès és el EA2. Veure *Figura 38: Valor del LED i potenciòmetre a SoMachine per la posició 2*.

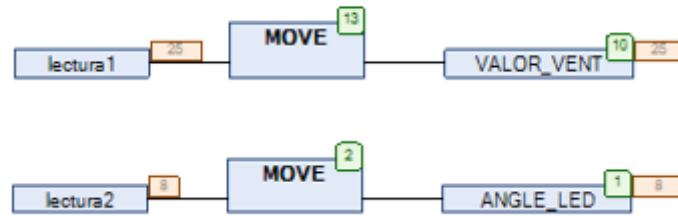


Figura 38: Valor del LED i potenciòmetre a SoMachine per la posició 2

Tant el LED encès en el microcontrolador de la *Figura 37* com el valor del LED encès llegit pels blocs del programa SoMachine de la *Figura 38*, coincideixen.

7 RESUM DEL PRESSUPOST

El conjunt de totes les parts que componen el pressupost tenen un cost total de DOS MIL TRES-CENTS NORANTA-UN EUROS AMB NORANTA-QUATRE CÈNTIMS, amb IVA inclòs (2391,94€).

8 CONCLUSIONS

Després d'haver treballat amb el programa MPLAB per tal de programar el microcontrolador, es pot dir que ha estat de gran ajuda la interfície que aquest ofereix. Addicionalment, esmentar que gràcies als recursos que el programa ofereix, s'ha estalviat programar a mà part del codi com són la declaració dels pins, la configuració del dispositiu i configuracions dels recursos EUSART i ADCC.

Una vegada dissenyada i executada la comunicació descrita en el present document, es constata que funciona correctament en poder-se comprovar mitjançant els dispositius mestre i esclau, els quals combina el hardware i el software que s'ha implementat.

En conseqüència s'ha assolit l'objectiu que es perseguia inicialment, establir la comunicació entre un dispositiu mestre i un dispositiu esclau seguint el protocol de comunicació Modbus RTU.

Finalment cal esmentar que, per al futur projecte final, es pot aprofitar gran part del codi de programació del microcontrolador. Només caldrà canviar els pins d'entrada i sortides pels més convenients, adaptar el codi en altres microcontroladors afegint més codi, acabar d'ajustar paràmetres com és la conversió analògica – digital i adaptar els actuadors i sensors necessaris.

Sergi Riera Toran

Graduat en Enginyeria en Tecnologies Industrials

Girona, 1 de Setembre de 2021

9 RELACIÓ DE DOCUMENTS

El present projecte compren els següents documents:

Document nº1: Memòria i annexos

Document nº2: Plànols

Document nº3: Plec de condicions

Document nº4: Amidaments

Document nº5: Pressupost

Document nº6: Codi (format electrònic)

10 REFERÈNCIES

About Modbus | Simply Modbus Software. (n.d.). <https://www.simplymodbus.ca/FAQ.htm>

Computer, C. (2005). C Compiler Reference Manual. *Computer*, February, 279.

EL MODELO CIM Y JERARQUÍA DE REDES DE COMUNICACIÓN EN LA INDUSTRIA. (n.d.). 25.

Fernández, D. J. L. (n.d.). *Índice de Contenidos u Ventajas Buses de campo.* 14.

Marais, H. (2008). Analog Device Application Note (AN-960), RS-485/RS-422 Circuit Implementation Guide. *An-960*, 1–12. www.analog.com

Mário de Sousa, & Portugal, P. (2016). Modbus. *Industrial Communication Systems*, 50. <https://doi.org/10.1201/9781420041682.ch18>

Mestle, L. (n.d.). DIRRECCIONAMIENTO DE PROTOCOLO MODBUS RTU Y ASCII _ Leche Mestle - Academia.

Microchip Technology Inc. (2020). *PIC16(L)F18855-75 Data Sheet* (p. 673).

MIKROE. (2021). *RS485 click 5V - Breakout board for ADM485 transceiver IC.* <c:%5CUsers%5CUSUARI%5CDocuments%5Creferencies%5CRS485 click 5V - Breakout board for ADM485 transceiver IC.html>

Modicon. (1996). *Modicon Modbus Protocol Reference Guide Modicon Modbus Protocol Reference Guide* (p. 115).

Robotec. (2019). *Modbus Fieldbus Networking* (pp. 1–20).

Telecommunications Industry Association. (2000). *Electrical Characteristics of Generators and Receivers for Use in Balanced Digital Multipoint Systems* (p. 22). <https://www.mikrocontroller.net/attachment/428561/eia485.pdf>

Type, D., & Date, P. (2014). Introduction to Modbus. In *National Instruments* (pp. 1–2). <http://www.ni.com/white-paper/7675/en/>

11 GLOSSARI

AC: Alternating Current

ADCC: Analog to Digital Conversion with Computation

CFC: Countinious Function Chart

CRC: Cyclic Redundancy Check

DC: Direct Current

EUSART: Enhanced Universal Synchronous Asynchronous Receiver Transmitter

LED: Light Emitting Diode

LSB: Least Significant Byte

LUT: Look Up Table

MSB: Most Significant Byte

PLC: Programmable Logic Controller

RTU: Remote Terminal Unit

UART: Universal Asynchronous Receiver Transmitter

ANNEX A CONFIGURACIONS INICIALS DELS RECURSOS A MPLAB

A.1.1 Configuració recurs EUSART

EUSART és una comunicació sèrie de perifèrics d'entrada/sortida que es pot configurar en mode síncron o en mode asíncron. Conté tots els rellotges generadors, registres de desplaçament i memòries intermèdies de dades per realitzar una transferència de dades independentment de l'execució del programa del dispositiu.

En el projecte es fa servir una comunicació asíncrona, ja que el procés de sincronització entre emissor i receptor es realitza en cada paraula de codi transmès. Aquesta relació asíncrona s'estableix sempre que no hi ha cap relació temporal entre l'estació que transmet i la que rep. En altres paraules, aquest tipus de xarxes no saben amb precisió quan rebrà un missatge.

Per generar el codi en el recurs EUSART, abans s'ha de configurar. La configuració general es mostra a la *Figura 39: Configuració general del recurs EUSART*.

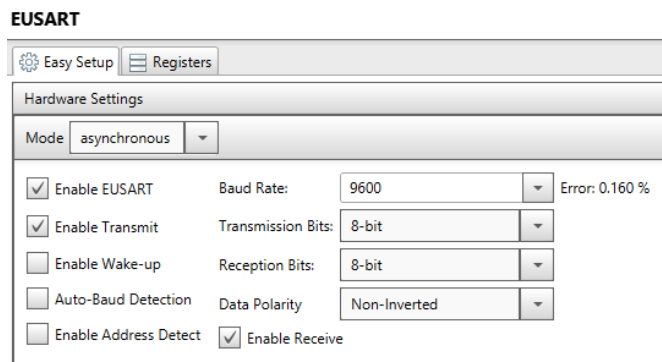


Figura 39: Configuració general del recurs EUSART

A la configuració general s'indica mode asíncron, s'habilita el recurs, la transmissió i la recepció. A més, s'ajusta la velocitat de transmissió dels bits (*Baud Rate*) de 9600 i finalment el nombre de bits de dades és vuit.

També es configuren els bits dels registres com es pot veure a la *Figura 40: Configuració dels registres*.

The figure shows three separate configuration panels for different registers. Each panel has a title bar with the register name and its address, followed by a list of fields with their current values.

- Register: BAUD1CON (0x8)**
 - ABDEN: disabled
 - ABDOVF: no_overflow
 - BRG16: 16bit_generator
 - RCIDL: idle
 - SCKP: Non-Inverted
 - WUE: disabled
- Register: RC1STA (0x90)**
 - ADDEN: disabled
 - CREN: enabled
 - FERR: no_error
 - OERR: no_error
 - RX9: 8-bit
 - RX9D: 0x0
 - SPEN: enabled
 - SREN: disabled
- Register: TX1STA (0x24)**
 - BRGH: hi_speed
 - CSRC: slave
 - SENDB: sync_break_complete
 - SYNC: asynchronous
 - TRMT: TSR_empty
 - TX9: 8-bit
 - TX9D: 0x0
 - TXEN: enabled

Figura 40: Configuració dels registres

BAUD1CON és un registre de control de la velocitat de transmissió i recepció de bits. La configuració inicial d'aquest registre es mostra a la *Taula 22: Configuració registre BAUD1CON*.

| Bit | Nom | Valor | Descripció |
|-----|--------|-------|---|
| 7 | ABDOVF | 0 | Auto-baud timer did not overflow |
| 6 | RCIDL | 0 | Start bit has been received and the receiver is receiving |
| 5 | - | 0 | Not implemented |
| 4 | SCKP | 1 | Idle state for transmit (TX) is a low level |
| 3 | BRG16 | 0 | 8-bit Baud Rate Generator is used |
| 2 | - | 0 | Not implemented |
| 1 | WUE | 0 | RX pin not monitored nor rising edge detected |
| 0 | ABDEN | 0 | Baud rate measurement disabled or completed |

Taula 22: Configuració registre BAUD1CON

RC1STA és un registre de control de recepció. La configuració inicial d'aquest registre es mostra a la *Taula 23: Configuració registre RC1STA*.

| Bit | Nom | Valor | Descripció |
|-----|-------|-------|--|
| 7 | SPEN | 1 | Serial port enabled |
| 6 | RX9 | 0 | Selects 8-bit reception |
| 5 | SREN | 0 | Unused Single Receive Enable bit |
| 4 | CREN | 1 | Enables continuous receive until enable bit CREN is cleared |
| 3 | ADDEN | 0 | Disables address detection, all bytes are received and ninth bit can be used as parity bit |
| 2 | FERR | 0 | No framing error |
| 1 | OERR | 0 | No overrun error |
| 0 | RX9D | 0 | - |

Taula 23: Configuració registre RC1STA

TX1STA és un registre de control de transmissió. La configuració inicial d'aquest registre es mostra a la Taula 24: Configuració registre TX1STA.

| Bit | Nom | Valor | Descripció |
|-----|-------|-------|--|
| 7 | CSRC | 0 | Unused in this mode – value ignored |
| 6 | TX9 | 0 | Selects 8-bit transmission |
| 5 | TXEN | 1 | Transmit enabled |
| 4 | SYNC | 0 | Asynchronous mode |
| 3 | SENDB | 0 | SYNCH BREAK transmission disabled or completed |
| 2 | BRGH | 1 | High speed |
| 1 | TRMT | 0 | Transmit Shift Register Status full |
| 0 | TX9D | 0 | - |

Taula 24: Configuració registre TX1STA

A.1.1.1 EUSART transmissió asíncrona

La transmissió de dades via EUSART és possible sempre que s'habiliten els següents tres bits de control:

- TXEN = 1: Habilita el circuit transmissor
- SYNC = 0: Configura EUSART en mode asíncron

- SPEN = 1: Habilita EUSART

A.1.1.2 EUSART recepció asíncrona

La recepció de dades via EUSART és possible sempre que s'habiliten els següents tres bits de control:

- CREN = 1: Habilita el circuit receptor
- SYNC = 0: Configura EUSART en mode asíncron
- SPEN = 1: Habilita EUSART

A.1.2 Configuració recurs ADCC

El convertidor analògic a digital amb càlcul permet la conversió d'un senyal d'entrada analògica a una representació binària de 10 bits d'aquest senyal mitjançant una aproximació successiva i emmagatzema la conversió del resultat als registres ADC (ADRESH:ADRESL).

Per generar el codi en el recurs ADCC, abans s'ha de configurar. La configuració general es mostra a la *Figura 41: Configuració general del recurs ADCC*.

Figura 41: Configuració general del recurs ADCC

També es configuren els bits dels registres com es pot veure a la *Figura 42: Configuració dels registres ADCC*.

▼ Register: ADCON0 0x84

ADCONT disabled ▼

ADCS FOSC/ADCLK ▼

ADFM right ▼

ADON enabled ▼

ADGO stop ▼

▼ Register: ADCON1 0x0

ADDSEN disabled ▼

ADGPOL digital_low ▼

ADIPEN disabled ▼

ADPPOL VSS ▼

Figura 42: Configuració dels registres ADCC

ADCON0 és un registre de control ADC 0 (*Analog to Digital Conversion*). La configuració inicial d'aquest registre es mostra a la *Taula 25: Configuració registre ADCON0*.

| Bit | Nom | Valor | Descripció |
|-----|--------|-------|---|
| 7 | ADON | 1 | ADC is enabled |
| 6 | ADCONT | 0 | ADGO is cleared upon completion of each conversion trigger |
| 5 | - | 0 | Not implemented |
| 4 | ADCS | 0 | Clock supplied by FOSC, divided according to ADCLK register |
| 3 | - | 0 | Not implemented |
| 2 | ADFRM0 | 1 | ADRES and ADPREV data are right-justified |
| 1 | - | 0 | Not implemented |
| 0 | ADGO | 0 | ADC conversion completed/not in progress |

Taula 25: Configuració registre ADCON0

ADCON1 és un registre de control ADC 1 (*Analog to Digital Conversion*). La configuració inicial d'aquest registre es mostra a la *Taula 26: Configuració registre ADCON1*.

| Bit | Nom | Valor | Descripció |
|-----|--------|-------|--|
| 7 | ADDPOL | 0 | Shorted to VSS |
| 6 | ADIPEN | 0 | The bit is ignored |
| 5 | ADGPOL | 0 | ADC guard ring outputs start as digital low during precharge stage |
| 4 | - | 0 | Not implemented |
| 3 | - | 0 | Not implemented |
| 2 | - | 0 | Not implemented |
| 1 | - | 0 | Not implemented |
| 0 | ADDSEN | 0 | One conversion is performed for each trigger |

Taula 26: Configuració registre ADCON1

ANNEX B PROGRAMARI MICROCONTROLADOR

B.1 Source files: Main

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

uint8_t i = 0;
uint8_t RxBuffer[20];

void main(void)
{
    SYSTEM_Initialize();
    uint8_t rcv;

    while (1)
    {
        LATCbits.LATC2 = 0;    //Deshabilita la direcció de transmissió del PIC
        for(i = 0; i < 2; i++){
            rcv = EUSART_Read();
            RxBuffer[i] = rcv;
        }
        modbus_function();
    }
}
```

B.2 Source files: Modbus

```
/*
 * File:    modbus.c
 * Author:  Sergi Riera Toran *
 * Created on August 5, 2021, 8:22 PM
 */

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

```
#include <stdio.h>
#include <pic16f18875.h>
#include <xc.h>

#define SlaveAddress 1
#define AUTOMATIC_MODE 1
#define TEST_MODE 0

//Global variables
extern uint8_t i;
extern uint8_t RxBuffer[];
uint8_t response[20];
uint8_t error;
bool frame;
bool working_mode;
uint16_t wind_conversion;

//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,
```

Memòria - Programari microcontrolador

```
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, 0x55C1, 0x9481, 0x5440,  
0x9C01, 0x5CC0, 0x5D80, 0x9D41, 0x5F00, 0x5FC1, 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, 0x99C0, 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 };

```

Memòria - Programari microcontrolador

```
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){
            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;
        }
    }
}

//inici leds incorporats dins la placa
void initial_leds_state(void){
    angle_30_SetLow();
    angle_45_SetLow();
    angle_60_SetLow();
    angle_90_SetLow();
}

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++){
            EUSART_Write(response[j]);
        }
    }
}

```


Memòria - Programari microcontrolador

```
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++){
        EUSART_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++){
        EUSART_Write(response[j]);
    }
}

}

void modbus_read_coils (void){
    uint16_t starting_address, coils_num, j, n, crc16 = 0;
    uint8_t    starting_address_Hi,    starting_address_Lo,    coils_num_Hi,
    coils_num_Lo,
        CRC_read_Hi, CRC_read_Lo;
    uint8_t bytes_number, reminder, i, k = 0;
    uint8_t coils[4] = {angle_30_PORT,    angle_45_PORT,    angle_60_PORT,
    angle_90_PORT};
    frame = 1; // inicialitzem la trama com a correcta
```

```

starting_address_Hi = RxBuffer[2];
starting_address_Lo = RxBuffer[3];
coils_num_Hi = RxBuffer[4];
coils_num_Lo = RxBuffer[5];
CRC_read_Hi = RxBuffer[6];
CRC_read_Lo = RxBuffer[7];

if(checkCRC() == true){
    //Passem els 2 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;
    coils_num = coils_num_Hi;
    coils_num = coils_num << 8;
    coils_num = coils_num | coils_num_Lo;

    //Nombre de bytes de resposta
    bytes_number = coils_num / 8;
    remainder = coils_num % 8; //Si 0 = no residu, Si != 0 -> residu

    if(coils_num > 4 || coils_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){
        response[0] = SlaveAddress;
        response[1] = READ_COILS;
    }
}

```

```
    if(reminder != 0){
        response[2] = bytes_number + 1;
    }
    else{
        response[2] = bytes_number;
    }

    k = 3;
    response[k] = 0;
    for(j = 0; j < reminder; j++){
        if(coils[starting_address] == 1){
            n = 1;
        }
        else{
            n = 0;
        }
        response[k] ^= (n << j);
        starting_address++;
    }
    k++;
    crc16 = MODBUS_CRC16_TX(k);
    response[5] = CRC8_Hi(crc16); //crc em surt 0
    response[4] = CRC8_Lo(crc16);

    LATCbits.LATC2 = 1;//habilitem la direcció de transmetre
    for (i = 0; i <= 5; i++){
        EUSART_Write(response[i]);
    }
    __delay_ms(20);
}

}

void modbus_read_input_registers(void){
    uint8_t starting_address_Lo, starting_address_Hi, reg_num_Hi, reg_num_Lo,
        CRC_read_Hi, CRC_read_Lo, wind_conversion_Hi, wind_conversion_Lo,
        j, k = 0;
    uint16_t max_wind = 25; //vent màxim de 25 m/s (minim = 0)
```

```

uint16_t crc16, adc_result, reg_num, starting_address;
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;

    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){
        adc_result = ADCC_GetSingleConversion(pote);
        wind_conversion = (adc_result * max_wind/1023);

        response[0] = SlaveAddress;
    }
}

```

```
        response[1] = READ_INPUT_REGISTERS;
        response[2] = 2 * reg_num_Lo;
        response[3] = 0;
        response[4] = wind_conversion;
        crc16 = MODBUS_CRC16_TX(5);
        response[6] = CRC8_Hi(crc16); //crc em surt 0
        response[5] = CRC8_Lo(crc16);

        LATCbits.LATC2 = 1;
        for (j = 0; j <= 6; j++){
            EUSART_Write(response[j]);
        }
        __delay_ms(10);
    }
}

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
    if(auto_data == 1 && test_data == 0){
        initial_leds_state();
        working_mode = AUTOMATIC_MODE;
        automatic_mode_led_SetHigh(); //pin RD6
        test_mode_led_SetLow();

        if(wind_conversion >= 0 && wind_conversion <= 6){
            angle_30_SetHigh();
            angle_45_SetLow();
        }
    }
}

```

```
        angle_60_SetLow();
        angle_90_SetLow();
    }
    else if(wind_conversion > 6 && wind_conversion <= 12){
        angle_30_SetLow();
        angle_45_SetHigh();
        angle_60_SetLow();
        angle_90_SetLow();
    }
    else if(wind_conversion > 12 && wind_conversion <= 19){
        angle_30_SetLow();
        angle_45_SetLow();
        angle_60_SetHigh();
        angle_90_SetLow();
    }
    else if(wind_conversion > 19 && wind_conversion <= 25){
        angle_30_SetLow();
        angle_45_SetLow();
        angle_60_SetLow();
        angle_90_SetHigh();
    }
}

else if(auto_data == 0 && test_data == 1){
    initial_leds_state();
    working_mode = TEST_MODE;
    test_mode_led_SetHigh(); //pin RD7
    automatic_mode_led_SetLow();
    switch (test_value_data_Lo){
        case 0:
            angle_30_SetHigh();
            angle_45_SetLow();
            angle_60_SetLow();
            angle_90_SetLow();
            break;
        case 1:
            angle_30_SetLow();
            angle_45_SetHigh();
            angle_60_SetLow();
            angle_90_SetLow();
    }
}
```

```

        break;
    case 2:
        angle_30_SetLow();
        angle_45_SetLow();
        angle_60_SetHigh();
        angle_90_SetLow();
        break;
    case 3:
        angle_30_SetLow();
        angle_45_SetLow();
        angle_60_SetLow();
        angle_90_SetHigh();
        break;
    }
}
else if(auto_data == test_data || auto_data > 1 || test_data > 1){
    initial_leds_state();
    test_mode_led_Toggle();
    automatic_mode_led_Toggle();
    frame = 0;
}
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);

LATCbits.LATC2 = 1;
for(uint8_t j = 0; j <= 7; j++){
    EUSART_Write(response[j]);
}
__delay_ms(5);
}
}
}

```



```
void read_all_frame(void){
    if(RxBuffer[i-1] == WRITE_MULTIPLE_REGISTERS){
        uint8_t pending;
        for(pending = i; pending < 15; pending++){ //Sempre tenim 3 registres,
aleshores la longitud de trama és de 15 bytes
            RxBuffer[pending] = EUSART_Read();
        }
    }
    else if(RxBuffer[i-1] == READ_INPUT_REGISTERS){
        uint8_t pending;
        for(pending = i; pending < 8; pending++){
            RxBuffer[pending] = EUSART_Read();
        }
    }
    else if(RxBuffer[i-1] == READ_COILS){
        uint8_t pending;
        for(pending = i; pending < 8; pending++){
            RxBuffer[pending] = EUSART_Read();
        }
    }
}

//Funcions modbus programades
void modbus_function (void){
    if(RxBuffer[0] == SlaveAddress){
        if(RxBuffer[i-1] == WRITE_MULTIPLE_REGISTERS || RxBuffer[i-1] ==
READ_COILS || RxBuffer[i-1] == READ_INPUT_REGISTERS){
            switch (RxBuffer[i-1]){
                case READ_COILS:
                    read_all_frame();
                    modbus_read_coils();
                    break;
                case READ_INPUT_REGISTERS:
                    read_all_frame();
                    modbus_read_input_registers();
                    break;
                case WRITE_MULTIPLE_REGISTERS:
                    read_all_frame();
                    modbus_write_multiple_registers();
            }
        }
    }
}
```

```

        break;
    }
}
else{
    error = ILLEGAL_FUNCTION;
    frame = 0;
}
}
}

```

B.3 Source files: MCC generated files

B.3.1 ADCC

```

#include <xc.h>
#include "adcc.h"
#include "mcc.h"

void ADCC_Initialize(void)
{
    ADLTHL = 0x00;
    ADLTHH = 0x00;
    ADUTHL = 0x00;
    ADUTHH = 0x00;
    ADSTPTL = 0x00;
    ADSTPTH = 0x00;
    ADRPT = 0x00;
    ADPCH = 0x00;
    // ADCAP Additional uC disabled;
    ADCAP = 0x00;
    ADPRE = 0x00;
    // ADDSEN disabled; ADGPOL digital_low; ADIPEN disabled; ADPPOL VSS;
    ADCON1 = 0x00;
    // ADCRS 0; ADMD Basic_mode; ADACLR disabled; ADPSIS ADRES;
    ADCON2 = 0x00;
    // ADCALC First derivative of Single measurement; ADTMD disabled; ADSOI
    ADGO not cleared;
    ADCON3 = 0x00;
    // ADAOV ACC or ADERR not Overflowed;
    ADSTAT = 0x00;
    // ADNREF VSS; ADPREF VDD;

```

Memòria - Programari microcontrolador

```
ADREF = 0x00;
// ADACT disabled;
ADACT = 0x00;
// ADCCS FOSC/2;
ADCLK = 0x00;
// ADGO stop; ADFM right; ADON enabled; ADCONT disabled; ADCS FOSC/ADCLK;
ADCON0 = 0x84;
// ADACQ 5;
ADACQ = 0x05;
}
```

```
adc_result_t ADCC_GetSingleConversion(adcc_channel_t channel)
{
    // select the A/D channel
    ADPCH = channel;
    // Turn on the ADC module
    ADCON0bits.ADON = 1;
    //Disable the continuous mode.
    ADCON0bits.ADCONT = 0
    // Start the conversion
    ADCON0bits.ADGO = 1;
    NOP();
    // Wait for the conversion to finish
    while (ADCON0bits.ADGO)
    {
    }

    // Conversion finished, return the result
    return ((adc_result_t)((ADRESH << 8) + ADRESL));
}
```

B.3.2 Device configuration

```
// CONFIG1
#pragma config FEXTOSC = OFF      // External Oscillator mode selection bits-
>Oscillator not enabled
#pragma config RSTOSC = HFINT1    // Power-up default value for COSC bits-
>HFINTOSC (1MHz)
#pragma config CLKOUTEN = OFF     // Clock Out Enable bit->CLKOUT function is
disabled; i/o or oscillator function on OSC2
#pragma config CSWEN = ON        // Clock Switch Enable bit->Writing to NOSC and
NDIV is allowed
```

```
#pragma config FCMEN = ON      // Fail-Safe Clock Monitor Enable bit->FSCM timer
enabled

// CONFIG2
#pragma config MCLRE = ON      // Master Clear Enable bit->MCLR pin is Master
Clear function
#pragma config PWRT = OFF      // Power-up Timer Enable bit->PWRT disabled
#pragma config LPBOR = OFF      // Low-Power BOR enable bit->ULPBOR disabled
#pragma config BOREN = ON      // Brown-out reset enable bits->Brown-out Reset
Enabled, SBOREN bit is ignored
#pragma config BORV = LO      // Brown-out Reset Voltage Selection->Brown-out
Reset Voltage (VBOR) set to 1.9V on LF, and 2.45V on F Devices
#if (__XC8_VERSION < 1360)
#pragma config ZCDDIS = OFF    // Zero-cross detect disable->Zero-cross detect
circuit is disabled at POR.
#else // __XC8_VERSION
#pragma config ZCD = OFF      // Zero-cross detect disable->Zero-cross detect
circuit is disabled at POR.
#endif // __XC8_VERSION
#pragma config PPS1WAY = ON    // Peripheral Pin Select one-way control->The
PPSLOCK bit can be cleared and set only once in software
#pragma config STVREN = ON     // Stack Overflow/Underflow Reset Enable bit-
>Stack Overflow or Underflow will cause a reset
#pragma config DEBUG = OFF     // Background Debugger->Background Debugger
disabled

// CONFIG3
#pragma config WDTCP = WDTCP_31 // WDT Period Select bits->Divider ratio
1:65536; software control of WDTCP
#pragma config WDTE = OFF      // WDT operating mode->WDT Disabled, SWDTEN is
ignored
#pragma config WDTCS = WDTCS_7 // WDT Window Select bits->window always
open (100%); software control; keyed access not required
#pragma config WDTCCS = SC     // WDT input clock selector->Software Control

// CONFIG4
#pragma config WRT = OFF      // UserNVM self-write protection bits->Write
protection off
#pragma config SCANE = available // Scanner Enable bit->Scanner module is
available for use
#pragma config LVP = ON      // Low Voltage Programming Enable bit->Low Voltage
programming enabled. MCLR/Vpp pin function is MCLR.

// CONFIG5
```

Memòria - Programari microcontrolador

```
#pragma config CP = OFF      // UserNVM Program memory code protection bit-
>Program Memory code protection disabled

#pragma config CPD = OFF     // DataNVM code protection bit->Data EEPROM code
protection disabled
```

B.3.3 Eusart

```
#include "eusart.h"

volatile eusart_status_t eusartRxLastError;

void (*EUSART_FramingErrorHandler)(void);
void (*EUSART_OverrunErrorHandler)(void);
void (*EUSART_ErrorHandler)(void);

void EUSART_DefaultFramingErrorHandler(void);
void EUSART_DefaultOverrunErrorHandler(void);
void EUSART_DefaultErrorHandler(void);

void EUSART_Initialize(void)
{
    // ABDOVF no_overflow; SCKP Non-Inverted; BRG16 16bit_generator; WUE
    disabled; ABDEN disabled;
    BAUD1CON = 0x08;
    // SPEN enabled; RX9 8-bit; CREN enabled; ADDEN disabled; SREN disabled;
    RC1STA = 0x90;
    // TX9 8-bit; TX9D 0; SENDB sync_break_complete; TXEN enabled; SYNC
    asynchronous; BRGH hi_speed; CSRC slave;
    TX1STA = 0x24;
    SP1BRGL = 0x19;
    SP1BRGH = 0x00;

    EUSART_SetFramingErrorHandler(EUSART_DefaultFramingErrorHandler);
    EUSART_SetOverrunErrorHandler(EUSART_DefaultOverrunErrorHandler);
    EUSART_SetErrorHandler(EUSART_DefaultErrorHandler);
    eusartRxLastError.status = 0;
}

uint8_t EUSART_Read(void)
{
    while(!PIR3bits.RCIF)
    {
    }
}
```

```

    eusartRxLastError.status = 0;
    if(1 == RC1STAbits.OERR)
    {
        RC1STAbits.CREN = 0;
        RC1STAbits.CREN = 1;
    }
    return RC1REG;
}

void EUSART_Write(uint8_t txData)
{
    while(0 == PIR3bits.TXIF)
    {
    }
    TX1REG = txData;    // Write the data byte to the USART.
}

void EUSART_DefaultFramingErrorHandler(void){}
void EUSART_DefaultOverrunErrorHandler(void){
    // EUSART error - restart
    RC1STAbits.CREN = 0;
    RC1STAbits.CREN = 1;
}

void EUSART_DefaultErrorHandler(void){
}

void EUSART_SetFramingErrorHandler(void (* interruptHandler)(void)){
    EUSART_FramingErrorHandler = interruptHandler;
}

void EUSART_SetOverrunErrorHandler(void (* interruptHandler)(void)){
    EUSART_OverrunErrorHandler = interruptHandler;
}

void EUSART_SetErrorHandler(void (* interruptHandler)(void)){
    EUSART_ErrorHandler = interruptHandler;
}

```

B.3.4 Mcc

```

#include "mcc.h"

void SYSTEM_Initialize(void)
{

```

82

```
PMD_Initialize();
PIN_MANAGER_Initialize();
OSCILLATOR_Initialize();
ADCC_Initialize();
EUSART_Initialize();
}

void OSCILLATOR_Initialize(void)
{
    OSCCON1 = 0x62;
    OSCCON3 = 0x00;
    OSCEN = 0x00;
    // HFFRQ 4_MHz;
    OSCFRQ = 0x02;
    OSCTUNE = 0x00;
}

void PMD_Initialize(void)
{
    PMD0 = 0x00;
    PMD1 = 0x00;
    PMD2 = 0x00;
    PMD3 = 0x00;
    PMD4 = 0x00;
    PMD5 = 0x00;
}
```

B.3.5 Pin manager

```
#include "pin_manager.h"

void PIN_MANAGER_Initialize(void)
{
    /**
    LATx registers = escriure
    */
    LATE = 0x00;
    LATD = 0x00;
    LATA = 0x00;
    LATB = 0x00;
    LATC = 0x00;
```

```
/**
TRISx registers = 1 (input), 0 (output)
*/
TRISE = 0x07;
TRISA = 0x0F;
TRISB = 0xFF;
TRISC = 0xBB;
TRISD = 0x3F;
```

```
/**
ANSELx registers
*/
ANSELD = 0x3F;
ANSELC = 0x3B;
ANSELB = 0xFF;
ANSELE = 0x07;
ANSELA = 0x0F;
```

```
/**
WPUx registers
*/
WPUD = 0x00;
WPUE = 0x00;
WPUB = 0x00;
WPUA = 0x00;
WPUC = 0x00;
```

```
/**
ODx registers
*/
ODCONE = 0x00;
ODCONA = 0x00;
ODCONB = 0x00;
ODCONC = 0x00;
ODCOND = 0x00;
```

```
/**
SLRCONx registers
```



```
*/
SLRCONA = 0xFF;
SLRCONB = 0xFF;
SLRCONC = 0xFF;
SLRCOND = 0xFF;
SLRCONE = 0x07;

/**
INLVLx registers
*/
INLVLA = 0xFF;
INLVLB = 0xFF;
INLVLC = 0xFF;
INLVLD = 0xFF;
INLVLE = 0x07;
RXPPS = 0x17;    //RC7->EUSART:RX;
RC6PPS = 0x10;    //RC6->EUSART:TX;
}
```

B.4 Header files: Modbus

```
#ifndef XC_HEADER_TEMPLATE_H
#define XC_HEADER_TEMPLATE_H
#include <xc.h>
#ifndef XC_HEADER_TEMPLATE_H
#define XC_HEADER_TEMPLATE_H
#include <xc.h>
#ifdef __cplusplus
extern "C" {
#endif
#ifdef __cplusplus
}
#endif
#endif
#endif

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_coils (void);
void modbus_read_input_registers(void);
void modbus_write_multiple_registers(void);
void read_all_frame(void);
void modbus_function (void);

```

B.5 Header files: MCC generated files

B.5.1 ADCC

```

#ifndef ADCC_H
#define ADCC_H
#include <xc.h>
#include <stdint.h>
#include <stdbool.h>
#ifdef __cplusplus
#endif

typedef uint16_t adc_result_t;
typedef enum
{
    pote = 0x0,
    channel_VSS = 0x3C,
    channel_Temp = 0x3D,
    channel_DAC1 = 0x3E,
    channel_FVR_buf1 = 0x3F
} adcc_channel_t;

void ADCC_Initialize(void);
adc_result_t ADCC_GetSingleConversion(adcc_channel_t channel);

```

B.5.2 Device configuration

```

#ifndef DEVICE_CONFIG_H
#define DEVICE_CONFIG_H
#define _XTAL_FREQ 1000000
#endif

```

B.5.3 Eusart

```

#ifndef EUSART_H

```

Memòria - Programari microcontrolador

```
#define EUSART_H

#include <xc.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>

#ifdef __cplusplus
#endif

#define EUSART_DataReady (EUSART_is_rx_ready())

typedef union {
    struct {
        unsigned perr : 1;
        unsigned ferr : 1;
        unsigned oerr : 1;
        unsigned reserved : 5;
    };
    uint8_t status;
}eusart_status_t;

void EUSART_Initialize(void);
void EUSART_Write(uint8_t txData);
void EUSART_SetFramingErrorHandler(void (* interruptHandler)(void));
void EUSART_SetOverrunErrorHandler(void (* interruptHandler)(void));
void EUSART_SetErrorHandler(void (* interruptHandler)(void));
```

B.5.4 Mcc

```
#ifndef MCC_H
#define MCC_H

#include <xc.h>
#include "device_config.h"
#include "pin_manager.h"
#include <stdint.h>
#include <stdbool.h>
#include <conio.h>
#include "adcc.h"
#include "eusart.h"

void SYSTEM_Initialize(void);
```

```
void OSCILLATOR_Initialize(void);
void PMD_Initialize(void);
#endif
```

B.5.5 Pin manager

```
#ifndef PIN_MANAGER_H
#define PIN_MANAGER_H

#include <xc.h>

#define INPUT    1
#define OUTPUT   0

#define HIGH     1
#define LOW      0

#define ANALOG    1
#define DIGITAL   0

#define PULL_UP_ENABLED      1
#define PULL_UP_DISABLED    0

// get/set pote aliases
#define pote_LAT              LATAbits.LATA0
#define pote_PORT             PORTAbits.RA0
#define pote_SetHigh()       do { LATAbits.LATA0 = 1; } while(0)
#define pote_SetLow()        do { LATAbits.LATA0 = 0; } while(0)
#define pote_Toggle()        do { LATAbits.LATA0 = ~LATAbits.LATA0; } while(0)

// get/set angle_90 aliases
#define angle_90_LAT          LATAbits.LATA4
#define angle_90_PORT         PORTAbits.RA4
#define angle_90_SetHigh()    do { LATAbits.LATA4 = 1; } while(0)
#define angle_90_SetLow()     do { LATAbits.LATA4 = 0; } while(0)
#define angle_90_Toggle()     do { LATAbits.LATA4 = ~LATAbits.LATA4; } while(0)

// get/set angle_60 aliases
#define angle_60_LAT          LATAbits.LATA5
```

Memòria - Programari microcontrolador

```
#define angle_60_PORT          PORTAbits.RA5
#define angle_60_SetHigh()    do { LATAbits.LATA5 = 1; } while(0)
#define angle_60_SetLow()     do { LATAbits.LATA5 = 0; } while(0)
#define angle_60_Toggle()     do { LATAbits.LATA5 = ~LATAbits.LATA5; }
while(0)

// get/set angle_45 aliases
#define angle_45_LAT          LATAbits.LATA6
#define angle_45_PORT          PORTAbits.RA6
#define angle_45_SetHigh()    do { LATAbits.LATA6 = 1; } while(0)
#define angle_45_SetLow()     do { LATAbits.LATA6 = 0; } while(0)
#define angle_45_Toggle()     do { LATAbits.LATA6 = ~LATAbits.LATA6; }
while(0)

// get/set angle_30 aliases
#define angle_30_TRIS          TRISAbits.TRISA7
#define angle_30_LAT          LATAbits.LATA7
#define angle_30_PORT          PORTAbits.RA7
#define angle_30_SetHigh()    do { LATAbits.LATA7 = 1; } while(0)
#define angle_30_SetLow()     do { LATAbits.LATA7 = 0; } while(0)
#define angle_30_Toggle()     do { LATAbits.LATA7 = ~LATAbits.LATA7; }
while(0)

// get/set RS485_TXEN aliases
#define RS485_TXEN_LAT        LATCbits.LATC2
#define RS485_TXEN_PORT        PORTCbits.RC2
#define RS485_TXEN_SetHigh()  do { LATCbits.LATC2 = 1; } while(0)
#define RS485_TXEN_SetLow()   do { LATCbits.LATC2 = 0; } while(0)
#define RS485_TXEN_Toggle()   do { LATCbits.LATC2 = ~LATCbits.LATC2;
} while(0)

// get/set RC6 procedures
#define RC6_SetHigh()          do { LATCbits.LATC6 = 1; } while(0)
#define RC6_SetLow()           do { LATCbits.LATC6 = 0; } while(0)
#define RC6_Toggle()           do { LATCbits.LATC6 = ~LATCbits.LATC6; }
while(0)

// get/set RC7 procedures
#define RC7_SetHigh()          do { LATCbits.LATC7 = 1; } while(0)
#define RC7_SetLow()           do { LATCbits.LATC7 = 0; } while(0)
```

```

#define RC7_Toggle()                do { LATCbits.LATC7 = ~LATCbits.LATC7; }
while(0)

// get/set automatic_mode_led aliases
#define automatic_mode_led_LAT      LATDbits.LATD6
#define automatic_mode_led_PORT     PORTDbits.RD6
#define automatic_mode_led_SetHigh() do { LATDbits.LATD6 = 1; }
while(0)
#define automatic_mode_led_SetLow() do { LATDbits.LATD6 = 0; }
while(0)
#define automatic_mode_led_Toggle() do { LATDbits.LATD6 =
~LATDbits.LATD6; } while(0)

// get/set test_mode_led aliases
#define test_mode_led_LAT           LATDbits.LATD7
#define test_mode_led_PORT          PORTDbits.RD7
#define test_mode_led_SetHigh()     do { LATDbits.LATD7 = 1; } while(0)
#define test_mode_led_SetLow()      do { LATDbits.LATD7 = 0; } while(0)
#define test_mode_led_Toggle()      do { LATDbits.LATD7 = ~LATDbits.LATD7;
} while(0)

void PIN_MANAGER_Initialize (void);
void PIN_MANAGER_IOC(void);
#endif

```

ANNEX C PROGRAMARI PLC

En aquest annex s'adjunta el codi del PLC programat per un altre alumne.

C.1 Variables

```
PROGRAM POU
VAR
    addm_0: ADDM;
    endereco: address;

    BLINK_1: BLINK;
    BLINK_2: BLINK;
    BLINK_3: BLINK;

    ORDRE: WRITE_VAR;
    CONFRIMACIO: READ_VAR;
    CONFRIMACIO2: READ_VAR;

    Inicia: BOOL;
    Llegeix: BOOL;
    Escriu: BOOL;

    escrito: ARRAY[0..2] OF WORD;
    lectural1: WORD;
    lectura2: BYTE;

    AUTO: WORD;
    TEST: WORD;
    VALOR_TEST: WORD;
    VALOR_VENT: WORD;
    ANGLE_LED: WORD;

END_VAR
```

C.2 CFC

El PLC s'ha programat amb un llenguatge de funcions contínues (CFC), que és una extensió de l'estàndard IEC 61131-3 i és un llenguatge de programació gràfic basat en el llenguatge de diagrama de blocs de funcions. Veure *Figura 43: Interfície i diagrames de blocs de SoMachine*.

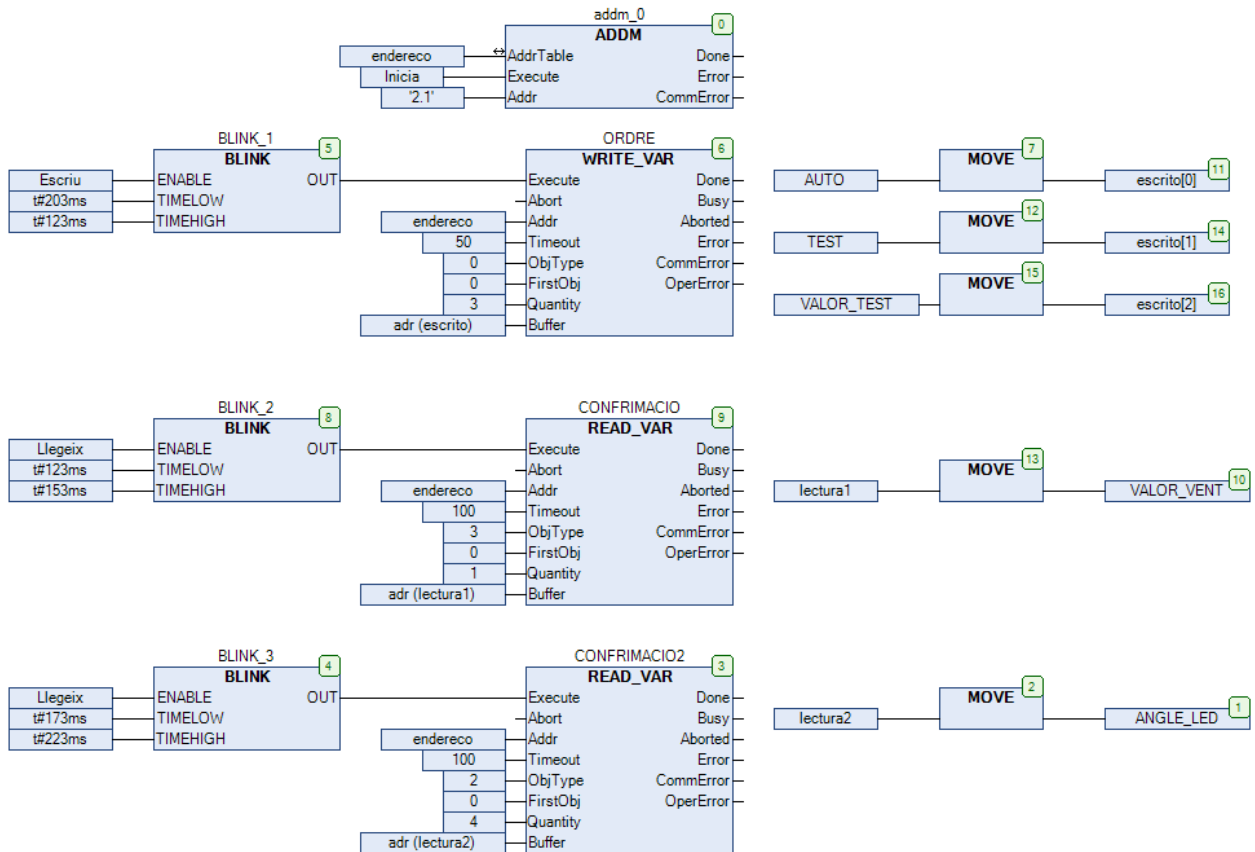


Figura 43: Interfície i diagrames de blocs de SoMachine

C.3 Descripció dels blocs de funcions

C.3.1 Bloc ADDM

El bloc ADDM converteix una adreça de destinació que es representa com una cadena a una estructura ADDRESS com a entrada en un bloc de funcions de comunicacions. Veure *Figura 44: Bloc ADDM*.

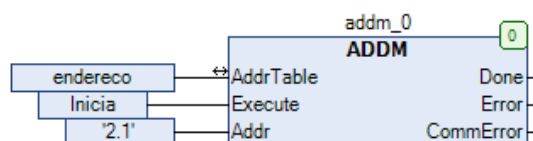


Figura 44: Bloc ADDM

La descripció dels diferents paràmetres que conformen aquest bloc es pot observar a la *Taula 27: Paràmetres bloc ADDM*.

| Paràmetre | Tipo | Direcció | Descripció |
|------------|---------|-----------------|---|
| Addr Table | ADDRESS | Entrada/Sortida | Estructura que ha de completar el bloc |
| Execute | BOOL | Entrada | Executa la funció en flanc ascendent |
| Addr | STRING | Entrada | Direcció en el tipo STRING que converteix al tipo ADDRESS |
| Done | BOOL | Sortida | TRUE: quan la funció s'executa correctament |
| Error | BOOL | Sortida | TRUE: quan la funció es para al detectar un error |
| CommError | BYTE | Sortida | Conté el codi d'error quan n'hi ha |

Taula 27: Paràmetres bloc ADDM

C.3.2 Bloc WRITE_VAR

El bloc de funcions WRITE_VAR escriu les dades en un dispositiu exten amb el protocol Modbus. Veure *Figura 45: Bloc WRITE_VAR*.

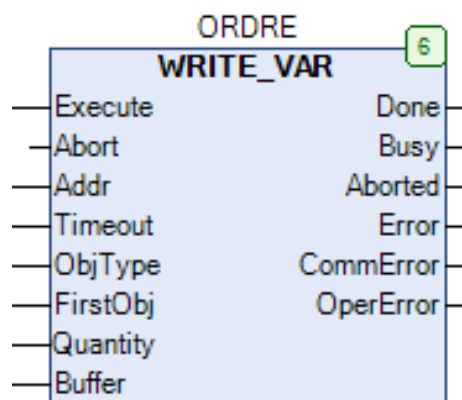


Figura 45: Bloc WRITE_VAR

La descripció dels diferents paràmetres que conformen aquest bloc es pot observar a la *Taula 28: Paràmetres bloc WRITE_VAR*.

| Paràmetre | Tipo | Direcció | Descripció |
|-----------|--------------------|----------|---|
| ObjType | ObjectType | Entrada | Escriu el tipus d'objecte que s'escriurà |
| FirstObj | DINT | Entrada | És l'índex del primer objecte que s'escriurà |
| Quantity | UINT | Entrada | És el número d'objectes a escriure |
| Buffer | POINTER TO BYTE | Entrada | Direcció del punter de la matriu que conté les dades que s'escriuràn en el dispositiu destí |

Taula 28: Paràmetres bloc WRITE_VAR

C.3.3 Bloc READ_VAR

El bloc de funcions READ_VAR llegeix les dades d'un dispositiu extern amb el protocol Modbus.

Verure Figura 46: Bloc READ_VAR.

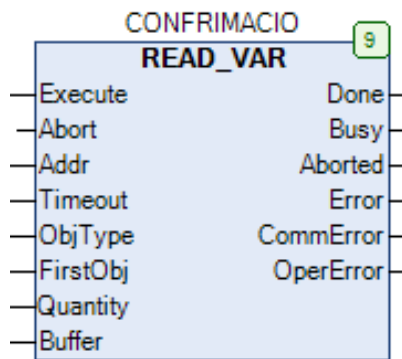


Figura 46: Bloc READ_VAR

La descripció dels diferents paràmetres que conformen aquest bloc es pot observar a la Taula 29: Paràmetres bloc READ_VAR.

| Paràmetre | Tipo | Direcció | Descripció |
|-----------|--------------------|----------|--|
| ObjType | ObjectType | Entrada | Escriu el tipus d'objecte que a llegir |
| FirstObj | DINT | Entrada | És l'índex del primer objecte que a llegir |
| Quantity | UINT | Entrada | És el número d'objectes a llegir |
| Buffer | POINTER TO BYTE | Entrada | Direcció del punter de la matriu que conté les dades que s'han llegit des del dispositiu destí |

Taula 29: Paràmetres bloc READ_VAR

C.3.4 Bloc BLINK

El bloc de funcions BLINK permet generar uns pulsos de senyal i s'utilitzen per executar els blocs de funcions WRITE_VAR i READ_VAR. Veure *Figura 47: Bloc BLINK*.

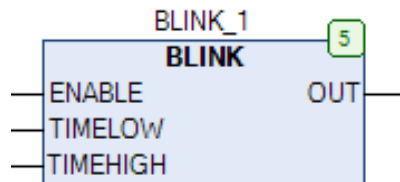


Figura 47: Bloc BLINK

La descripció dels diferents paràmetres que conformen aquest bloc es pot observar a la *Taula 30: Paràmetres bloc BLINK*.

| Paràmetre | Tipo | Direcció | Descripció |
|-----------|-------|----------|--|
| ENABLE | BOOL | Entrada | TRUE: s'habilita el bloc |
| TIMELOW | CONST | Entrada | Temps que el bloc dona senyal |
| TIMEHIGH | CONST | Entrada | Temps que el bloc no dona senyal |
| OUT | BOOL | Sortida | Proporciona senyal o no, depenent de la franja de temps que es trobi |

Taula 30: Paràmetres bloc BLINK

C.3.5 Bloc MOVE

El bloc de funcions MOVE escriu el missatge en una posició de memòria o llegeix el missatge guardat en una posició de memòria. Veure *Figura 48: Bloc MOVE*.

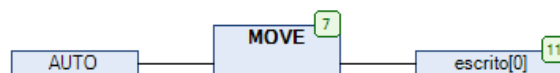


Figura 48: Bloc MOVE

ANNEX D PETIT PROGRAMARI PER VISUALITZAR LES TRAMES AMB ARDUINO

```
#include <SoftwareSerial.h>
#define rxPin 10
#define txPin 11
#define estatRT 7
SoftwareSerial mySerial (rxPin, txPin);

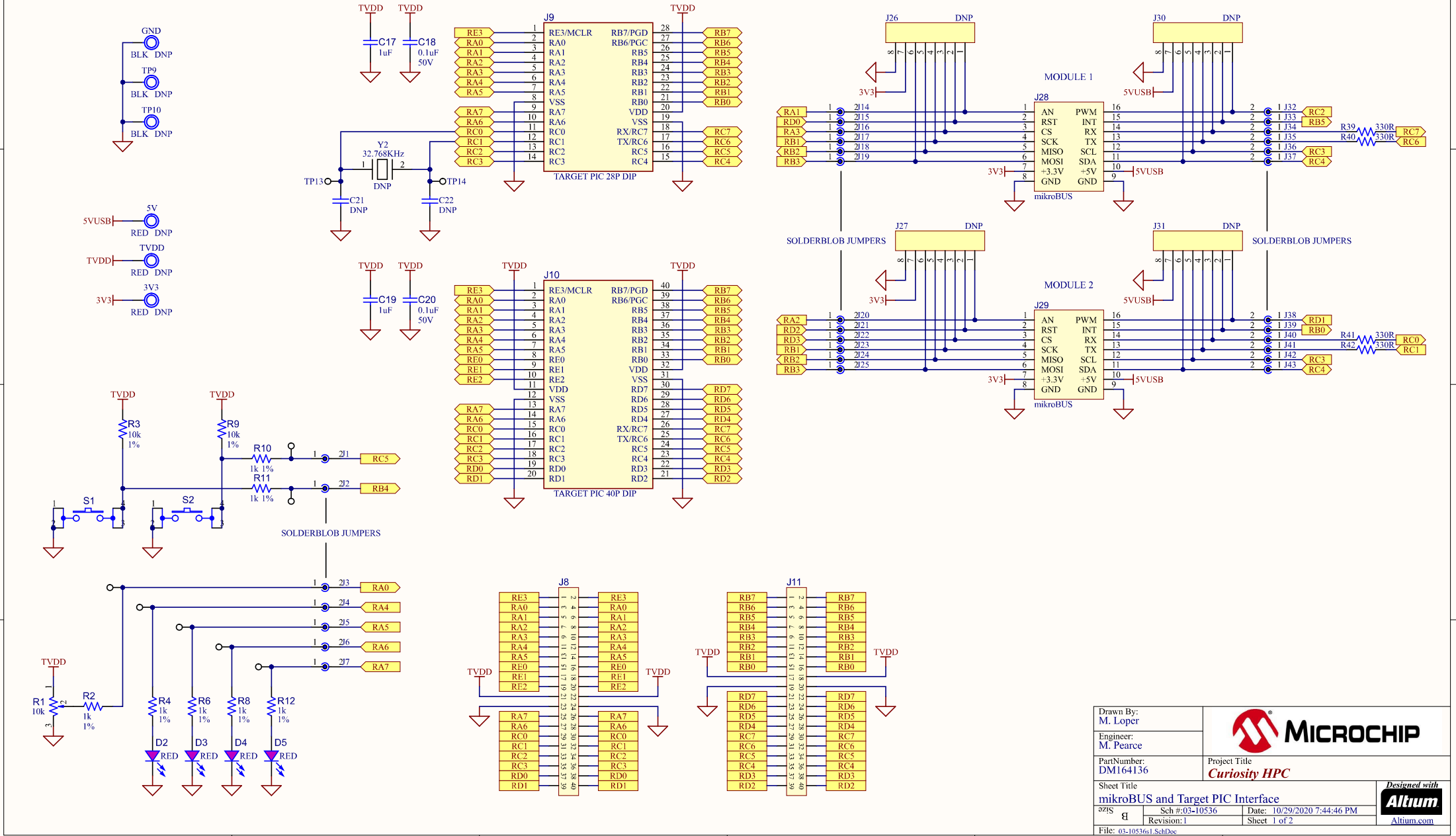
void setup() {
  Serial.begin(9600);
  pinMode(estatRT, INPUT);
  pinMode(rxPin, INPUT);
  pinMode(txPin, OUTPUT);
  mySerial.begin(9600);
}


void loop() {
  if (mySerial.available()) {
    uint16_t angulo = mySerial.read();
    Serial.println(angulo);
  }
}
```



ANNEX E ESQUEMES ADDICIONALS

En el present annex, s'adjunten tots aquells esquemes que contenen informació necessària per desenvolupar el protocol, i que no s'ha esmentat anteriorment. Concretament, s'adjunten els esquemes de la placa de desenvolupament Curiosity HPC i els esquemes del trasnceptor RS485.

| REV | ECO# | DESCRIPTION | DATE |
|-----|------|-------------|------|
| | | | |



| | | |
|---|--|--|
| Drawn By: M. Loper | |  MICROCHIP |
| Engineer: M. Pearce | | |
| PartNumber: DM164136 | | Project Title Curiosity HPC |
| Sheet Title mikroBUS and Target PIC Interface | | |
| Sch #: 03-10536 | | Date: 10/29/2020 7:44:46 PM |
| Revision: 1 | | Sheet 1 of 2 |
| File: 03-10536s1.SchDoc | | |


Designed with
Altium
Altium.com

