UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Politècnica Superior d'Enginyeria
de Manresa

PROTOCOL DE COMUNICACIÓ CAN

4 de juliol de 2018

Autor: Omar Sánchez Arias
Director: Francisco del Águila
Títol: Grau en Enginyeria de Sistemes TIC
Localitat: Manresa

Índex

Índex de figures

Índex de taules

Abstract

This project is divided into 4 parts. In the first part, an introduction to the CAN bus is explained: the history of the protocol, when it was published and for what reason; followed by the objectives of this project.

The second part explains the protocol that implements the CAN bus. It will begin with a brief description of the bus and the most important features. The protocol will be explained in more detail below, starting with the lowest level layer up to the data link layer. The third part explains the application that has been developed and how the CAN bus has been implemented. Finally, i will talk about the conclusions and the possible future projects.

Resum

Aquest projecte està dividit en 3 parts. En la primera part s'explica una introducció del bus CAN: la historia del protocol, quan va ser publicat i per quin motiu; seguit dels objectius d'aquest projecte.

En la segona part s'explica el protocol que implementa el bus CAN. Es començarà amb una breu descripció del bus i de les característiques més importants. A continuació s'explica el protocol més detalladament, començant amb la capa de més baix nivell fins a la capa d'enllaç de dades. En la tercera part s'explica el l'aplicació que s'ha desenvolupat i com s'ha implementat el bus CAN. Finalment es parlarà de les conclusions que s'ha arribat i les possibles línies futures per a darrers projectes.

Capítol 1

Introducció

En els anys 80, la necessitat de millorar la tecnologia dels vehicles va fer incrementar l'electrònica que incorporaven. Originalment, les connexions dels dispositius elecrònics eren punt a punt. L'increment de l'electrònica va comportar un augment excessiu de línies de cable, fent que els vehicles fossin cada cop més pesats i el cablejat més complicat.

Per aquesta raó l'any 1986 el protocol CAN (Controller Area Network) va ser publicat, amb l'objectiu de reduir la quantitat de cablejat utilitzat en la indústria dels automòbils i alhora poder enviar grans quantitats d'informació a llargues distàncies. Amb el bus CAN s'aconsegueix que tots els components electrònics estiguin comunicats entre ells en un únic bus, sense línies dedicades.

1.1 Situació del treball

La implementació del bus CAN ha permès l'augment del nombre de components electrònics que es poden instal·lar en els vehicles sense preocupar-nos d'afegir noves línies de cable. Aquest protocol permet treballar amb busos independents i separar les tasques del vehicle.

Des que el bus CAN va ser implementat, la seva utilització s'ha estandaritzat en tots el vehicles i avui dia el podem veure en autobusos, camions, trens o avions.

1.2 Objectius

L'objectiu d'aquest treball és la investigació del protocol CAN, per tal d'esbrinar quins són els seus punts forts i entendre perquè després de més de 30 anys del desenvolupament del protocol segueix sent utilitzat.

Alhora, s'ha desenvolupat una petita aplicació per tal de comprovar el funcionament del bus i poder entendre'l millor.

Capítol 2

Protocol CAN

2.1 Introducció al protocol CAN

El protocol de comunicació CAN és del tipus CSMA/CD+AMP (Carrier Sense Multiple Acces with Collision Detection and Arbitration on Message Priority).

Cada node de la xarxa escolta el bus, si durant un cert període de temps no detecta activitat, pot enviar un missatge. En cas que 2 o més nodes escriguin alhora, la col·lisió és resolta a partir d'un sistema d'arbitratge on el missatge que tingui la prioritat més alta guanya l'accés al bus.

El CAN és un protocol asíncron, vol dir que no hi ha cap *clock* que se sincronitzi els nodes per l'escriptura i lectura.

El protocol recull només les 2 capes més baixes del model OSI, la capa física i la de dades.

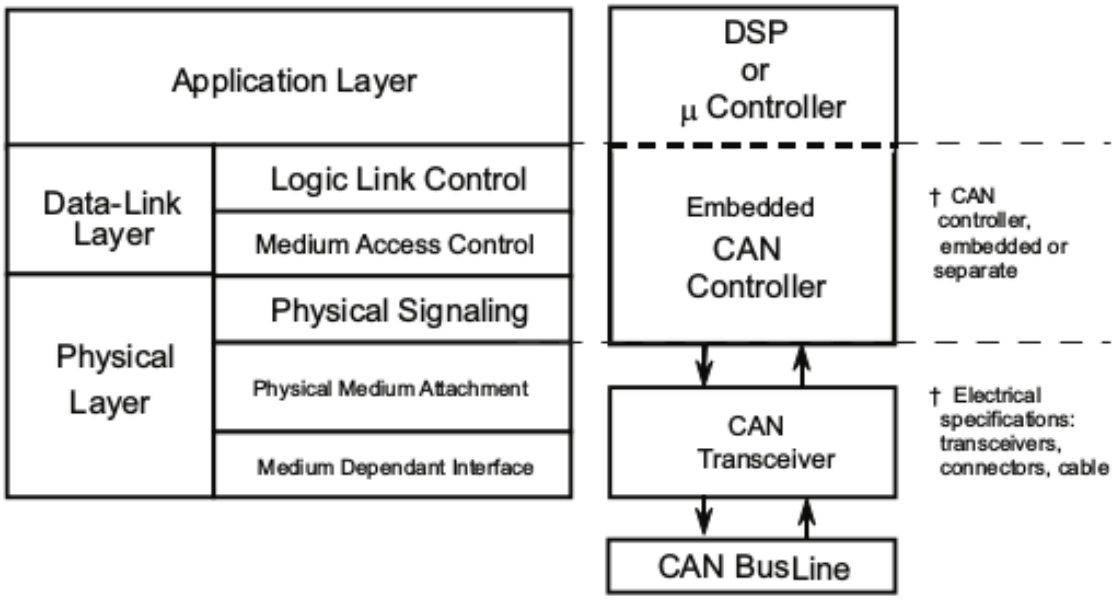

Figura 2.1: Model OSI

Cada node requereix 3 elements:

1. Transductor: Implementa les especificacions elèctriques necessàries pel correcte funcionament del bus.

2. Controlador CAN: Responsable de transferir missatges d'un node a un altre de la xarxa sense errors. Despres d'enviar una trama espera el reconeixement dels receptors.

3. Microprocesador: Capa d'aplicació que proporciona funcions de comunicació a un nivell superior.

2.2 Estandarització

A l'any 1993 es va llançar l'estàndard CAN ISO 11898. Més tard a l'any 2003 es va reestructurar en dues parts, ISO 11898-1 on recull la capa de dades i la ISO 11898-2 la capa física per a una xarxa d'alta velocitat.

El propòsit de la capa física és en general definir de quina manera els bits són codificats en senyals elèctriques per ser transmesos a través del bus.

2.3 Capa fisica

CAN d'alta velocitat

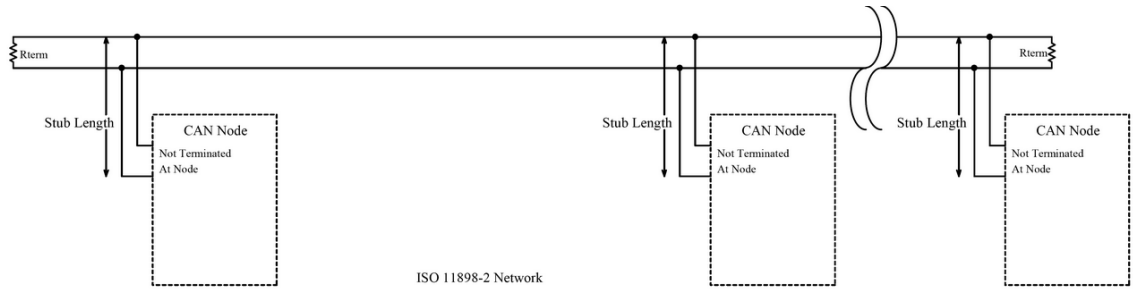

Figura 2.2: CAN d'alta velocitat

El bus d'alta velocitat (ISO 11989-2) consisteix en dos cables trenats amb una impedància característica de 120 ohms. A cada extrem del bus és necessari que es connecti una resistència d'igual valor per tal d'evitar reflexions al final de la línia.

Nivells del bus CAN A cada cable es defineix un nivell de senyal, anomenats 'CAN High' i 'CAN Low'.

CAN especifica dos estats lògics depenent de la tensió diferencial entre els dos cables.

Estat recessiu Quan els dos senyals es troben al mateix nivell de tensió. En l'estat recessiu la diferència de potencial respecte terra és proper 2,5V. L'estat recessiu correspon al valor lògic '1'.

Estat dominant En l'estat dominant la diferència de nivell de tensió entre els senyals és de 2V, estant els senyals 'CAN High' i 'CAN Low' aproximadament a 3,5 i 1,5 V respectivament. L'estat dominant correspon al valor lògic '0', mentre que l'estat dominant correspon al valor lògic '1'.

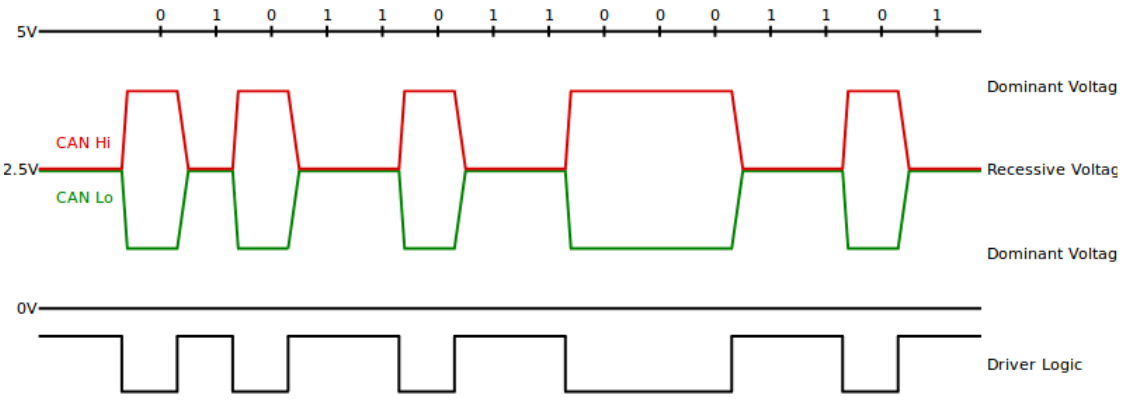

Figura 2.3: Estat dominant i estat recesiu

2.4 Capa d'enllaç de dades

Quan un node vol transmetre un missatge, primer escolta el canal durant un cert temps. Si passat aquest temps no es detecta activitat al bus, el node començarà a escriure el seu missatge.

Els nodes envien trames amb un format fixe i de mida variable. Les trames s'envien bit a bit i sempre comencen amb el bit 'Start of Frame', que serà sempre dominant (valor lògic '0'). El node continuarà enviant fins a transmetre un seguit de bits recessius (valors lògics '1') indicant que la transmissió de la trama ha acabat, tornant a deixar el bus en estat inactiu.

Quan un node ha rebut una trama correctament envia un missatge de reconeixement 'ACK'. Si el node transmissor després d'enviar la trama rep almenys una trama ACK, significarà que el missatge s'ha rebut bé. Si passat un temps no ha rebut cap ACK, tornarà a enviar la trama, ja que entén que el missatge no l'ha rebut per cap node.

2.5 Tipus de trames

Les trames que es poden enviar son les següents:

- Trama de dades
- Trama remota
- Trama d'error
- Trama de sobrecarrega

2.5.1 Trama de dades

Les trames de dades s'utilizen per transmetre informació entre el node transmissor cap a tots els nodes de la xarxa. Les trames CAN no utilitzen una adreça explícita per identificar el receptor del missatge, però cada receptor pot definir quins missatges pot rebre.

Existeixen dos tipus de trames, l'estàndard i l'extensa, la diferència més gran entre elles és el camp d'identificació on tenen 11 i 29 bits respectivament.

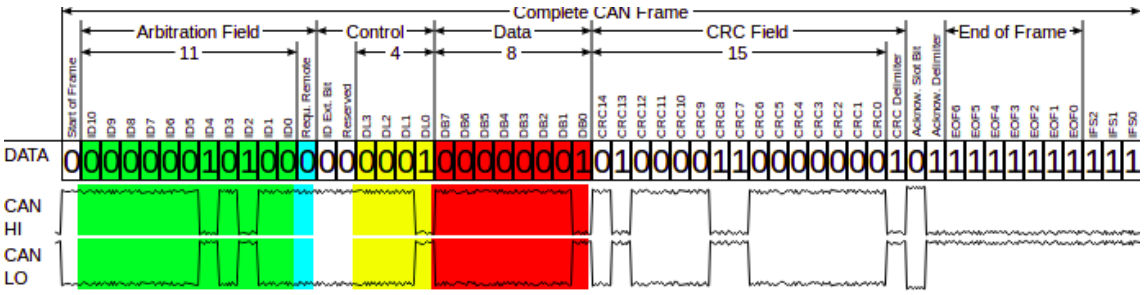

Figura 2.4: Exemple d'una trama de dades amb format estàndard

La trama de dades estàndard es divideix en diferents camps:

SOF (Start of frame)

Longitud d'un bit. Bit que indica inici de trama. És un bit dominant, per tant de valor lògic '0'. Aquest bit interromp l'estat recessiu del bus quan està inactiu.

Identificador

Identificador únic que representa la prioritat de la trama. 11+1 bits pel format estàndard. A la xarxa cada node ha de tenir un identificador únic.

El dotzè bit representa el bit RTR (Remote Transmission Request), per distinguir entre trames remotes o de dades. Per a enviar una trama remota el valor del bit ha de ser domimant, per a la trama de dades recessiu.

Control

Conté 6 bits, el primer bit conté el bit IDE (Identifier Extension Bit) en nivell dominant si el format és estàndard o recessiu si és extensa. El segon bit és reservat, recessiu per defecte. Els 4 bits següents defineixen la quantitat de dades que s'envien, màxim 8 dades per trama.

Camp de dades

Dades de la trama, la longitud ve donada pel camp de control. Màxim 8 dades (64 bits) per trama.

CRC

Codi que verifica que les dades han sigut transmeses correctament.

Delimitador

Ha de ser recessiu.

ACK

El transmissor envia un bit recessiu.

El receptor d'una trama envia la mateixa trama llegida només canviant el bit ACK a dominant. D'aquesta manera si es produeix un conflicte produït per una trama de reconeixement amb un altre missatge del node transmissor, l'estat dominant de la trama reconeixement donarà prioritat.

EOF (End Of Frame)

7 bits recessius indiquen que s'ha acabat la trama.

2.5.2 Trama remota

És possible que un node demani la petició d'una dada a un altre node enviant-li una trama remota. El bit 'RTR' ha de ser recessiu (nivell lògic '1') i no s'ha d'enviar camp de dades. En aquesta trama, el camp de l'identificador correspon al node a qui se li demana la dada.

2.5.3 Trama d'error

Quan un node receptor detecta un missatage amb un format erroni o un CRC incorrecte s'envia una trama d'error.

2.5.4 Trama de sobrecàrrega

És enviada per un node que es troba massa ocupat, el bus proporciona un retard extra entre trames.

2.6 Arbitrarietat per prioritats

L'arbitratge del protocol CAN es basa amb prioritats. El camp d'identificador de cada trama te un valor únic indicant la prioritat del missatge. Quan dos missatges s'envien alhora el missatge menys prioritari perd l'accés al medi i el missatge guanyador continua transmetent sense que el valor s'hagi vist modificat.

Quan un node vol escriure i detecta el canal inactiu, sempre començarà escrivint el bit 'Start of Frame'. En aquest moment, els nodes que transmetin al mateix moment competiran per aconseguir l'accés al medi. Si dos nodes han començat a escriure alhora, el bit dominant (valor lògic '0') sobreescriurà el bit recessiu (valor lògic '1').

Com que després del bit 'Start of Frame' s'escriu el camp d'identificació, el node que enviï un número d'identificació més baix (més prioritari) serà el guanyador.

2.7 Detecció de pèrdua d'accés al medi

Després de la transmissió d'un bit el node escolta el canal. Si el valor que escolta és el mateix valor que l'enviat per ell, el node considera que cap altre node amb més prioritat ha accedit al medi i enviarà el següent bit. En canvi, si el valor és diferent, el node considera que un altre node amb més prioritat que ell ha transmès pel canal i el seu missatge ha estat sobreescrit. En aquest moment deixa d'enviar per intentar retransmetre passat un temps.

Perquè l'arbitratge funcioni correctament, tots els nodes han de poder llegir el mateix bit alhora. Això vol dir que quan es configura la xarxa tots els nodes han d'estar programats per transmetre a la mateixa velocitat. D'igual manera els nodes esperen rebre els missatges també amb aquesta velocitat.

Com que el bus CAN està pensat per enviar missatges a grans distàncies, és necessari calcular quin serà el retard de propagació de la línia, per tal que el node més llunyà respecte el node transmissor pugui rebre el mateix missatge que altres nodes que es troben més propers.

Això a la pràctica significa que la xarxa es configura amb el retard de propagació segons els dos nodes més llunyans de la xarxa. Perquè l'arbitrarietat de la xarxa funcioni correctament, el retard de propagació esperat s'haurà de configurar com el doble del temps de propagació entre els dos nodes més llunyans.

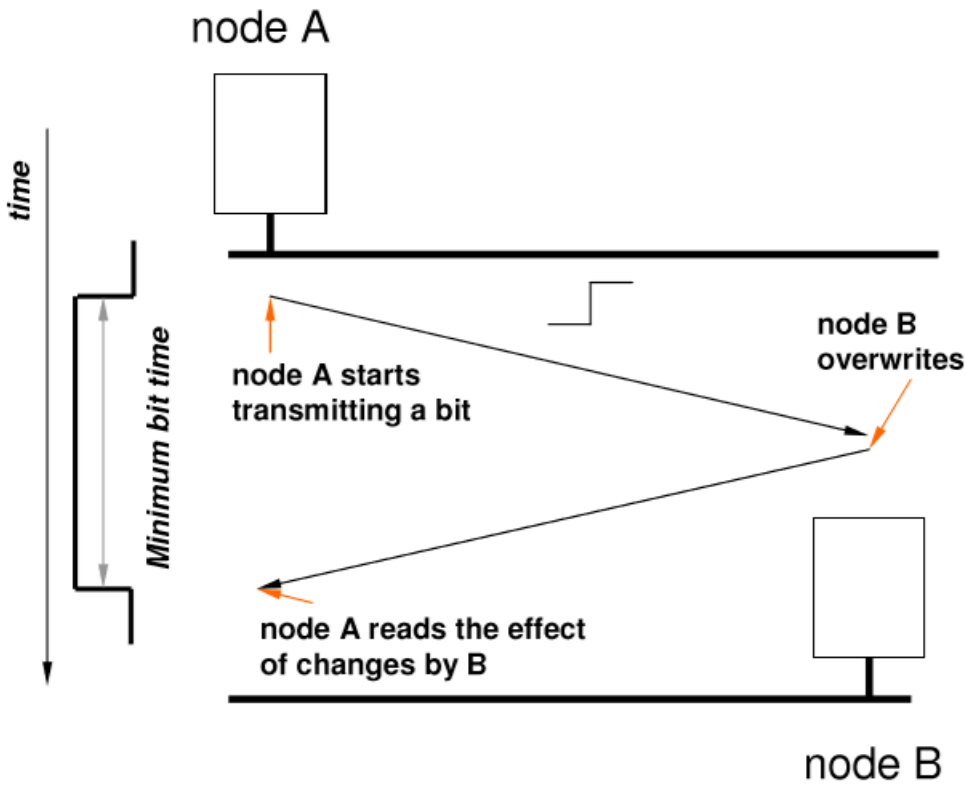

Figura 2.5: Temps nominal de bit mínim

Veient la figura de sobre observem els 2 nodes més llunyans entre ells de tota la xarxa. El node A envia un senyal (imaginem que és recessiu) i recorre tota la línia. Però just abans que arribi al node B, aquest vol escriure, i ho fa, ja que escolta el bus inactiu perquè el senyal encara no ha arribat. El senyal enviada per B és dominant i sobreescriu al senyal del node A. El senyal recorre tota la xarxa fins a arribar node A. En aquest moment si el node A escolta el canal, determinarà que el nivell del bus no és el mateix que el nivell que ell ha enviat i la detecció de la col·lisió es detectarà correctament. Al següent interval de temps el node A deixarà d'escriure i el node B seguirà escrivint.

Quan més d'un node escriuen alhora hem de ser conscient que:

- No es detectarà col·lisió si el nivell dels missatges són els mateixos.
- Només el node que ha perdut l'accés al medi detecta la col·lisió, qui guanya l'accés no s'adona.

2.8 Longitud de la línia VS velocitat de transmissió

Com es pot observar existeix una dependència entre la velocitat de transmissió i el retard de la propagació, i per tant, entre la màxima velocitat de transmissió i la distància del bus.

Aquesta taula recomana la taxa de bits per segon depenent la llargada de la línia.

Longitud del bus (metres)	Taxa de transferència (Mkps)
40	1000
100	500
200	250
500	100
1000	50

Taula 2.1: Longitud VS Velocitat

2.9 Sincronització

Pel fet que el bus CAN és asíncron, no existeix cap clock que sincronitzi el node transmissor i receptor quan es vol fer una transmissió. Això fa que els nodes puguin escriure pel bus quan vulguin i s'espera que els receptors siguin capaços de sincronitzar-se i rebre els missatges correctament.

La sincronització es realitza quan un node detecta un flanc de pujada després d'un temps d'inactivitat. Aquest flanc vol dir que l'estat del bus ha canviat de recessiu a dominant. Després d'aquest primer bit el node espera que els següents arribin a la mateixa velocitat en la qual s'ha configurat el bus.

Si observem la figura 2.4, comprovem que el bit de sincronització es produeix al 'Start of Frame'. Després de la recepció d'aquest primer bit, el node receptor inicialitza un temporitzador amb un temps igual a la velocitat de la xarxa. Durant aquest temps, llegeix la següent dada corresponent al primer bit d'identificació. Quan el temporitzador finalitza tornarà a començar per llegir el següent bit d'identificació. El node anirà llegint les dades del bus segons la velocitat de la xarxa configurada fins que torni a detectar inactivitat.

A aquesta sincronització es diu sincronització *hard*

2.10 Temps de bit

Per la necessitat que tots els nodes han de poder llegir el mateix bit alhora, el temps nominal de bit (temps que triga a enviar-se un bit) es compon de 4 segments:

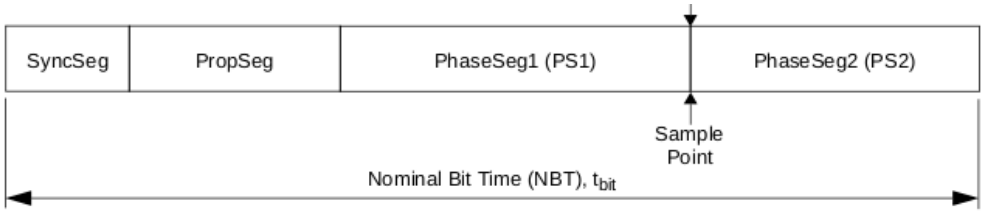

Figura 2.6: Segments del temps de bit

- Segment de sincronització És l'interval de referència utilitzat per la sincronització. S'espera que cada nova dada arribi en aquest segment de temps.
- Segment de propagació Segment de temps per compensar el retard de propagació de la xarxa. Haurà de ser almenys el doble del temps de propagació del bus.
- Segments fase 1 i fase 2 Segment de temps per compensar altres retards de la xarxa.

El lloc indicat com *sample point* és el moment on el nivell del bus és llegit i interpretat com el valor del respectiu bit.

Per aconseguir la sincronització, és obligatori que tots els nodes estiguin configurats amb el mateix temps de bit. En cas que dintre d'una mateixa xarxa s'utilitzin nodes amb controladors que treballin amb diferents freqüències, caldrà ajustar-los amb un 'preescaler' perquè tots treballin a la mateixa velocitat.

Llavors el temps de bit serà determinat pel temps dels quatre segments.

Temps de Quantum

La configuració del temps de bit es realitza afegint unitats de temps a cadascun dels quatre segments. Aquesta unitat de temps es diu *temps de Quantum*

El temps de Quantum ve donada per la següent fòrmula.

$$t_Q = 2 + BRP + T_O = \frac{2 + BRP}{F_O}$$

on tQ és el temps de Quantum, BRP és un preescaler i Fo és la freqüència de clock del controlador CAN.

2.11 Resincronització

En una comunicació real, la tolerància dels cristalls dels 'Clocks' pot produir una deriva en els rellotges de la xarxa. Si durant la transmissió de la trama aquest error es va acumulant, es podria produir una desincronització i la resta de la trama es llegiria erròniament.

Per arreglar aquest problema, el protocol CAN sincronitza el temps nominal de bit dels nodes receptors amb la velocitat de transimissió real del bus mentre s'està rebent la trama. La resincronització es produeix en els flancs de pujada. El protocol espera detectar els flancs de pujada dintre del segment de sincronització, en cas de no ser així el temps de bit s'ajustarà a la velocitat real del bus.

Per assegurar la resincronització, el protocol força l'enviament d'uns bits de polaritat oposada després de 5 bits consecutius de la mateixa polaritat. El node receptor detecta aquests bits afegits i els elimina abans de processar les dades. Aquesta estratègia se li diu *Bit stuffing*.

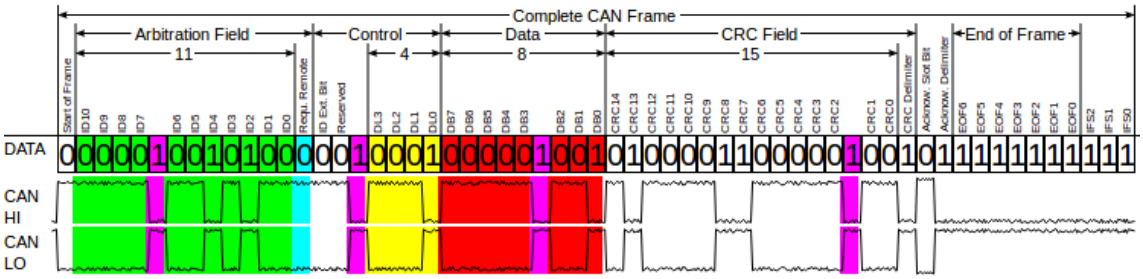

Figura 2.7: Bit stuffing

Cas 1: La sincronització és correcte

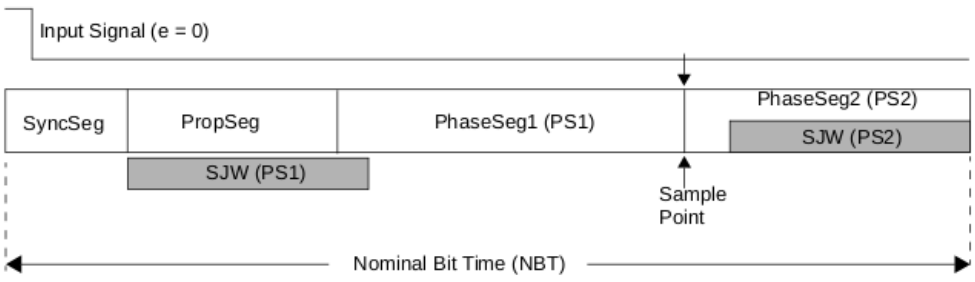

Figura 2.8: No hi ha sincronització

En aquest cas el flanc de pujada és detectat a l'inici de la trama, dintre el segment de sincronització.

Cas 2: La sincronització no és correcte, retard en la línia

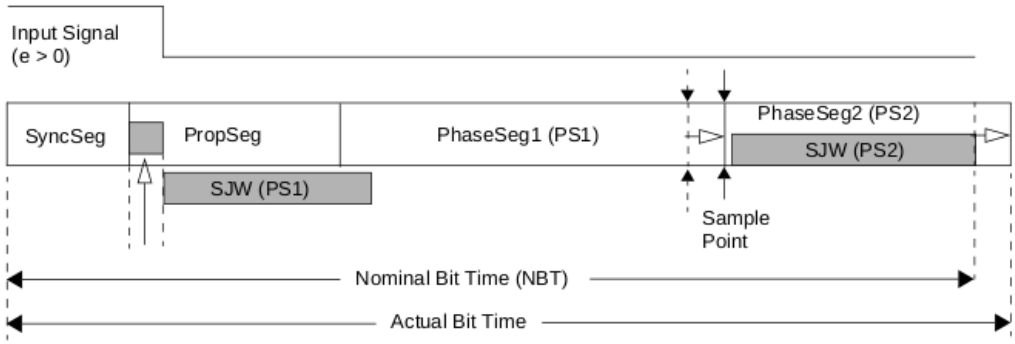

Figura 2.9: Resincronitzar a una transmissió més lenta

Si el flanc de pujada és detectat després del segment de sincronització, la dada ha arribat retardada. El protocol ajusta el temps nominal de bit a la velocitat real del bus afegint-li el temps de retard.

Cas 3: La sincronització no és correcte, anticipació en la línia

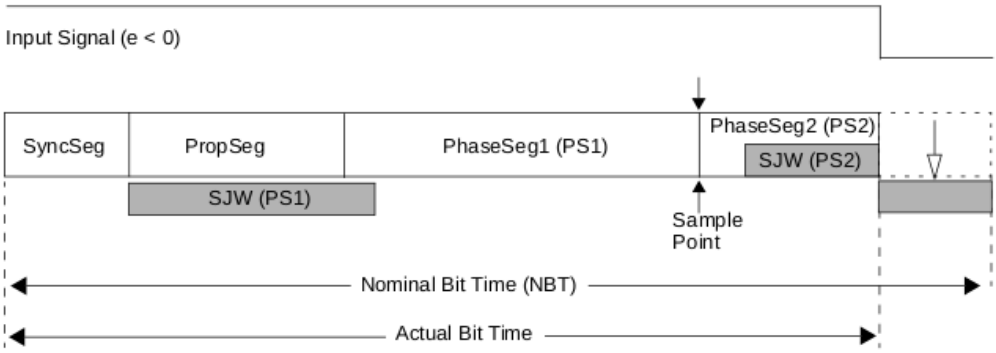

Figura 2.10: Resincronitzar a una transmissió més ràpida

Si el flanc de pujada és detectat abans del segment de sincronització la dada ha arribat abans del que s'esperava. El protocol ajusta el temps nominal de bit traient-li el temps d'anticipació.

Capítol 3

Aplicació

El sistema desenvolupat el formen 3 nodes connectats al bus CAN el qual s'enviaran missatges entre ells. El sistema ha de complir els següents requisits:

- Hi ha 1 node funcionant com a màster i 2 nodes com a esclaus. El màster rep dades dels esclaus.
- El màster és capaç de monitorar les trames enviades pels nodes esclaus.
- Els esclaus només monitorar les trames arribades del màster, les d'un altre esclau les ignora.
- Les trames generades pel màster tindran prioritat respecte als dels esclaus.
- Els nodes esclaus només poden enviar dades si el màster inicia la comunicació. També hauran de dexiar de transmetre si el màster ho demana.
- Els nodes esclaus representen sensors. Un node envia dades que representen velocitat, les dades de l'altre node representen temperatura. Les dades de temperatura s'obtenen d'un sensor de temperatura. Les dades de velocitat són inventades.
- Un cop iniciat l'enviament de missatges, el màster advertirà a l'usuari en cas de pèrdua de comunicació amb un esclau.
- Els esclaus advertiran a través d'un senyal lluminós en cas de fallada de comunicació.
- El màster es comunicarà a partir del bus sèrie amb una petita interfície on enviarà les dades rebudes dels esclaus. Alhora des de la interfície es podrà enviar comandes al màster.
- Amb una petita interfície d'usuari es podrà monitorar les dades rebudes del màster. Alhora, l'usuari podrà interactuar amb el sistema podent realitzar comandes com iniciar o aturar la comunicació. A més es podrà interactuar amb el led d'un node esclau modificant de la intensitat del senyal.

3.1 Estudi de mercat

L'estàndard CAN només especifica la capa de física i de dades, cada empresa desenvolupa la seva aplicació de manera diferent. Tot i així, el sistema utilitzat és sempre molt similar, s'utilitza un transductor, un controlador i un microprocessador. La funcionalitat de cada component està explicat a l'apartat 2.1.

3.1.1 Hardware utilitzat

Per a l'aplicació d'aquest projecte s'ha buscat diferents shields compatibles amb la plataforma Arduino. L'opció escollida ha sigut la shield fabricada per l'empresa Sparkfun. S'ha triat aquesta shield perquè Sparkfun també ha desenvolupat unes llibreries per poder comunicar-se amb el controlador. S'utilitzarà aquesta llibreria per facilitar la recepció i transmissió de dades del bus.

Shield

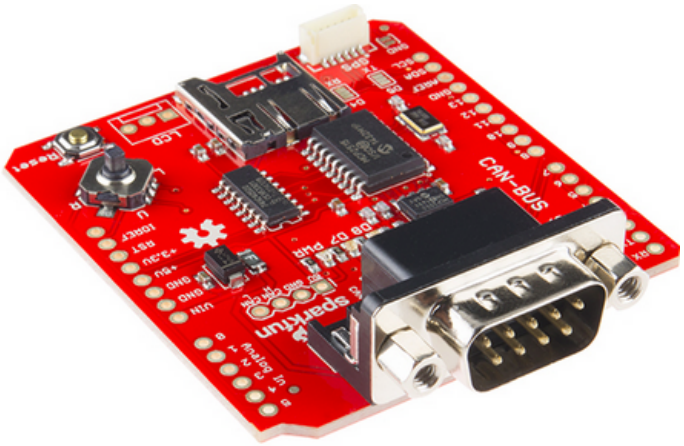

Figura 3.1: Shield Sparfkfun del bus CAN

Els components integrats al circuit imprès són els següents:

- Controlador CAN MCP2515, del fabricant Microchip.
- Transductor CAN MCP2551, del fabricant Microchip.
- Connexió CAN a través de connector estàndard sub-D.
- Socket per al mòdul GPS.
- Targeta micro SD.
- Connector per a LCD.
- Botó de 'Reset'.
- Joystic.
- Dos indicadors LED.

Utilitza la interfície SPI d'alta velocitat (10 MHz) per comunicar el controlador CAN amb l'Arduino.

Arduino UNO

Cada shield està muntada sobre un Arduino UNO on s'utilitzarà per desenvolupar la nostra aplicació. L'arduino UNO és la plataforma amb la que s'ha treballat durant tota la carrera i és ben conegut. Utilitza el microcontrolador ATMEGA328p a una freqüència de 16 MHz.

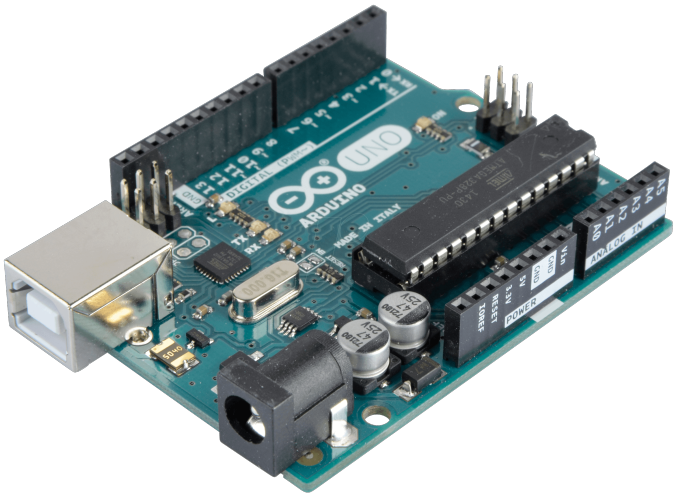

Figura 3.2: Arduino UNO

Connectors

S'utilitzarà els connectors DB9 de 9 pins amb la finalitat de connectar els cables *CAN High* i *CAN Low* a una protoboard.

Figura 3.3: Connectors DB9 de 9 pins

Sensor de temperatura MCP9700

Com a sensor de temperatura s'ha triat l'integrat MCP9700 del fabricant MICROCHIP, ja que que és el que s'utilitza per a l'assignatura *Teoria de Circuits*.

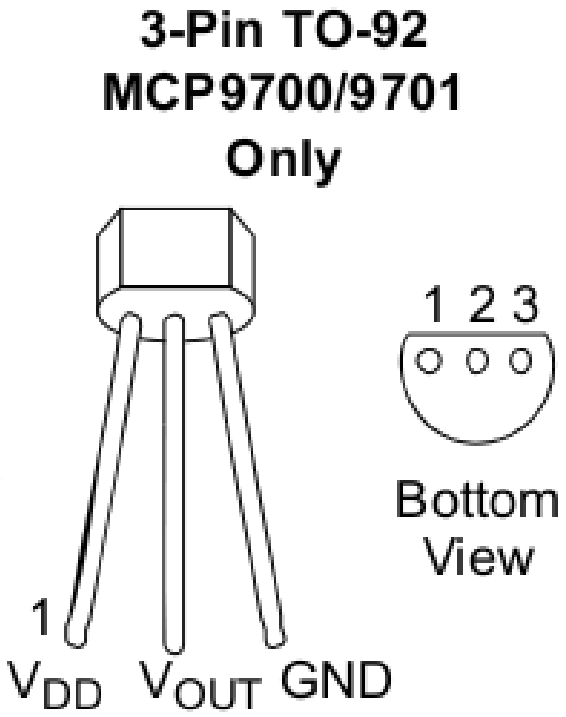

Figura 3.4: Sensor de temperatura MCP9700

Addicionalment s'ha utilitzat un LED amb una resistència de 220 ohms.

3.2 Arquitectura

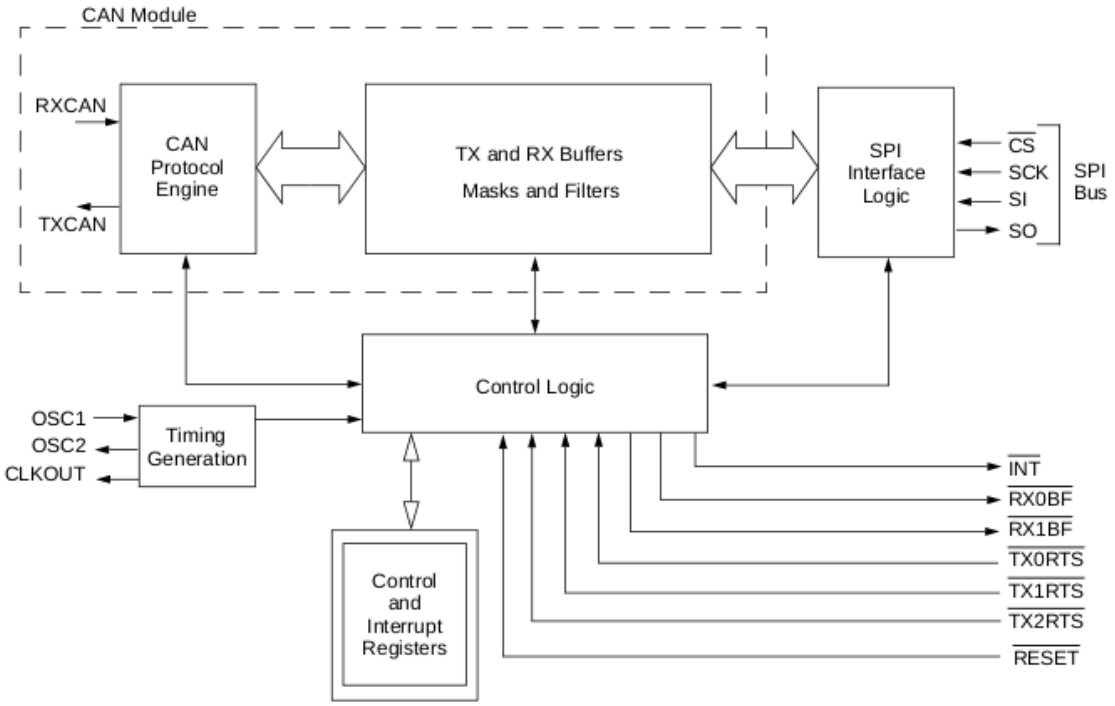

Figura 3.5: Diagrama de blocs del controlador CAN

Aquesta seria una vista simplificada del controlador CAN. Els tres blocks principals són:

1. El mòdul CAN, que inclou el motor del protocol, mascares, filtres i buffers de transmissió i recepció.

2. La lògica de control i registres utilitzats per configurar dispositiu.

3. Bloc del protocol SPI (Serial Peripheral Interface) .

La transmissió dels missatges s'inicialitza utilitzant bits de registre a través de l'interfície SPI o amb els pins d'habilitació. L'estat i els errors es poden comprovar llegint els registres apropiats.

El bloc lògic de control controla la configuració i funcionament del controlador CAN. A més, proporciona pins d'interrupció. En concret el pin d'interrupció PINT s'utilitza tant per la transmissió del missatge com a la recepció.

La lectura i escriptura de tots els registres es realitza a partir de comandes SPI. A més, existeixen comandes especials per facilitar la interacció amb el controlador. Per exemple, demanar la transmissió d'un missatge es pot realitzar amb una comanda concreta.

Buffers

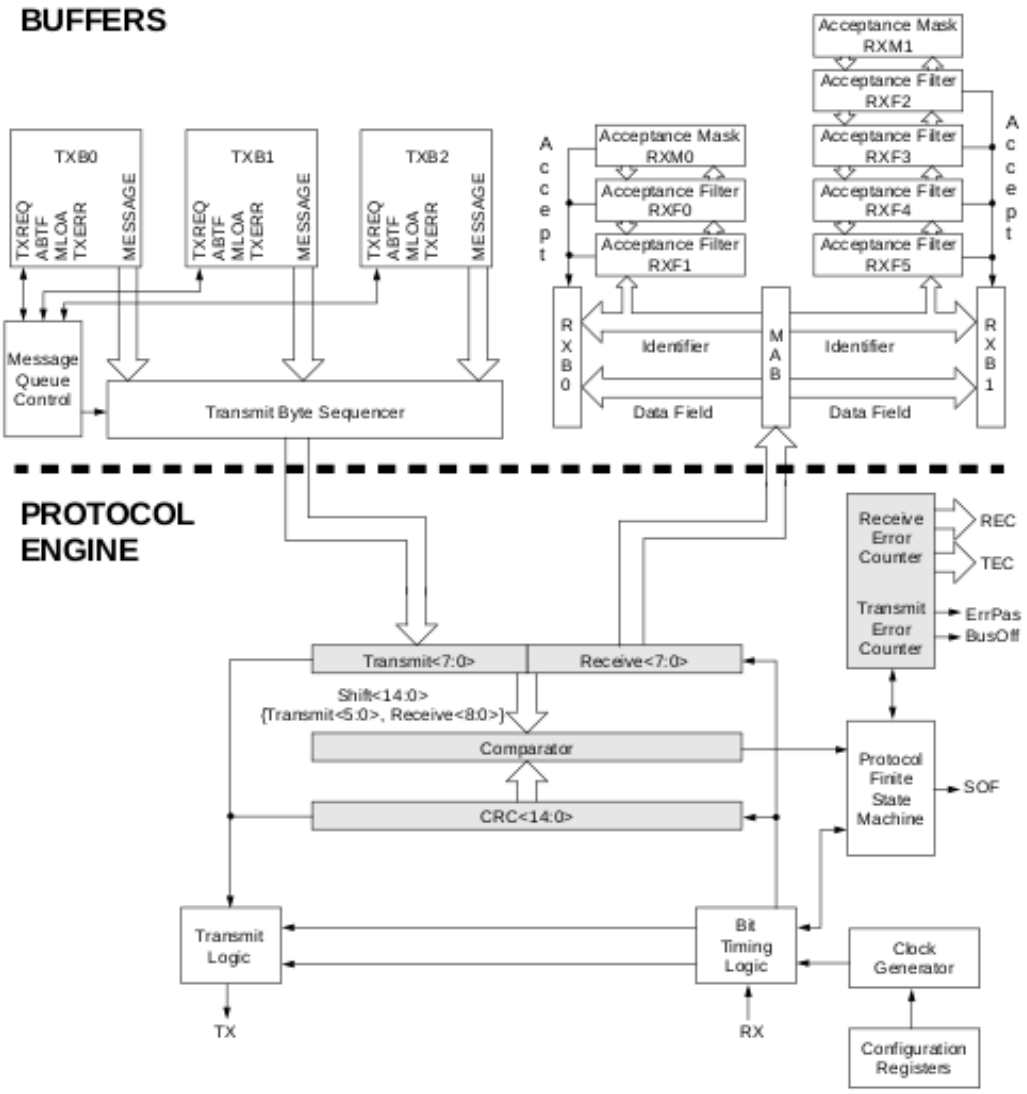

Figura 3.6: Buffers de transmissió i recepció

Buffers de transmissió

Existeixen 3 buffers de transmissió. TXBnCTRL és un registre de control associat a un buffer. La informació d'aquest registre determina les condicions les quals el missatage s'enviarà i indica l'estat de la transmissió del missatge.

Buffers de recepció

Hi ha 2 buffers de recepció. Cadascun incorpora diferents filtres. Les trames que es reben es poden filtrar segons el camp d'identificació.

Ul tercer buffer anomenat MAB està sempre a l'espera de rebre missatges del bus. Aquest buffer reuneix tots els missatges rebuts i seran entregats als buffers RXBn nomes si els criteris de filtratge són els correctes.

Protocol CAN

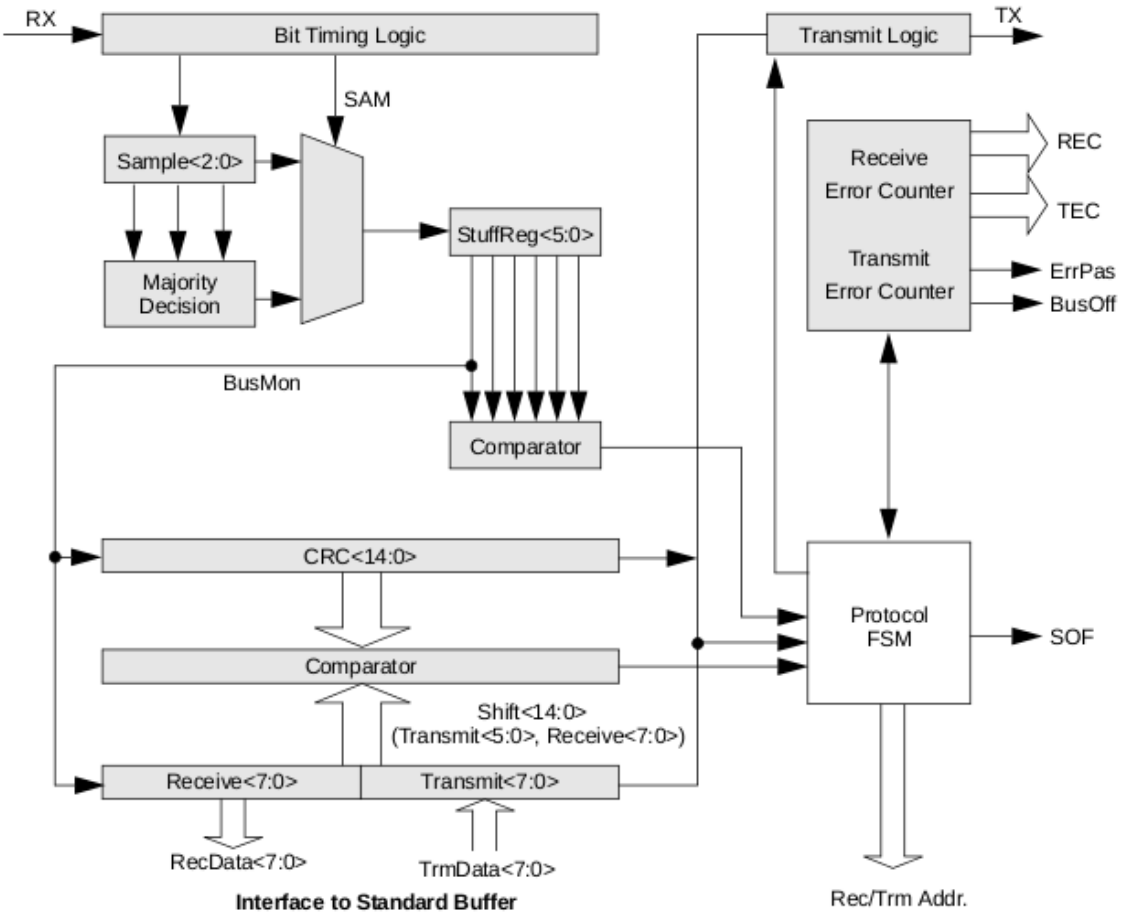

Figura 3.7: Motor del protocol CAN

El motor del controlador CAN és una màquina d'estats que implementa el protocol CAN segons les especificacions. La màquina d'estats garanteix que tots els processos de recepció, arbitratge, transmissió i detecció d'errors es realitzen segons el protocol CAN.

El registre CRC (Cyclic Redundancy Check) genera el codi CRC. És utilitzat per comprovar el camp CRC dels missatges entrants.

El bloc EML (Error Managament Logic) és responsable de la detecció de fallades.

El bloc BTL (Bit Timing Logic) implementa la configuració per la sincronització entre nodes.

Transmissió

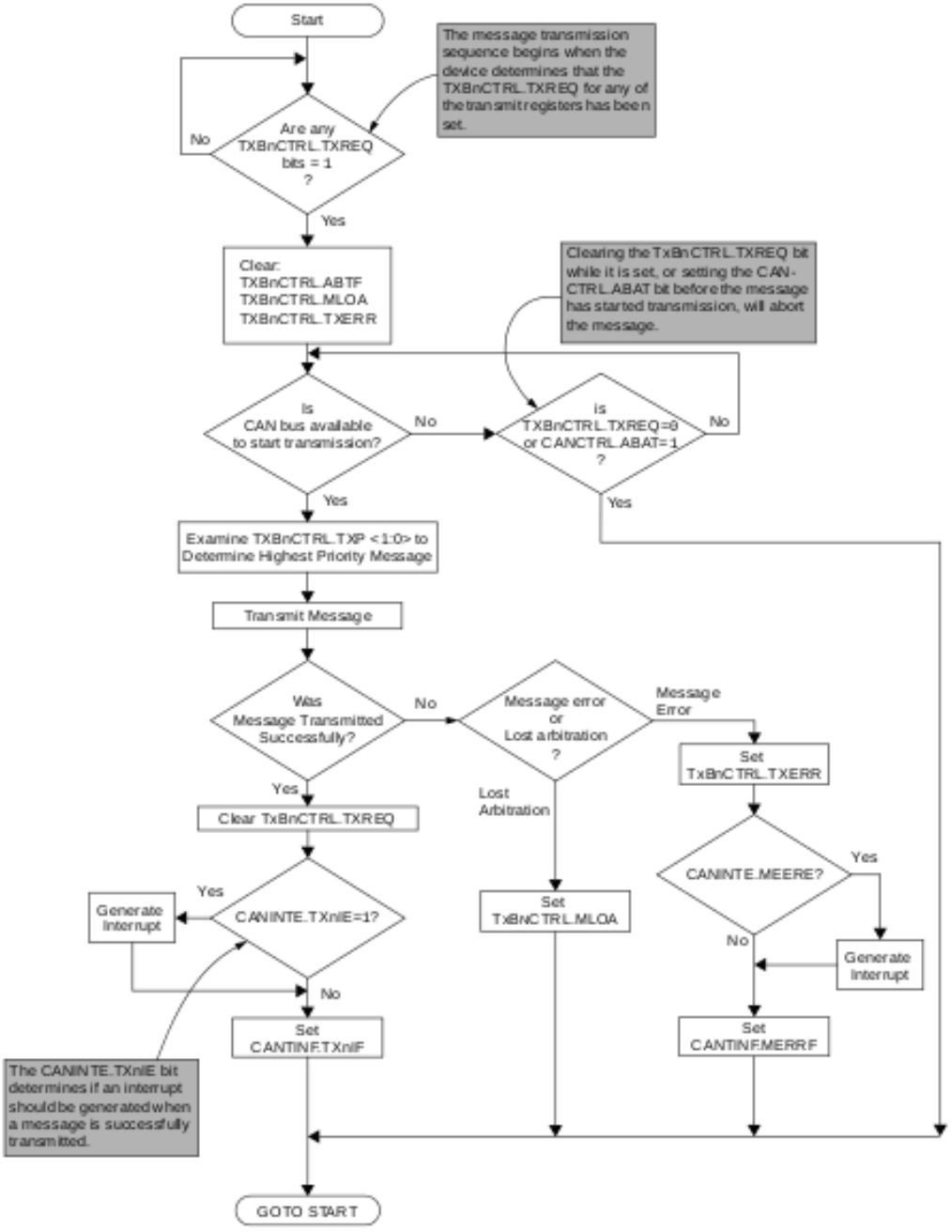

Figura 3.8: Màquina d'estats: Transmissió

La transmissió comença quan el bit TXBnCTRL.TXREQ d'un buffer es posa '1'. Si el bus no està ocupat, envia el missatge. Si el missatge s'ha rebut bé generem una interrupció al pin *INT*.

Si detecta una pèrdua d'arbitrarietat activa el bit TxBnCTRL.MLOA. Si detecta fallada en l'enviament, activa el bit TxBnCTRL.TXERR.

Recepció

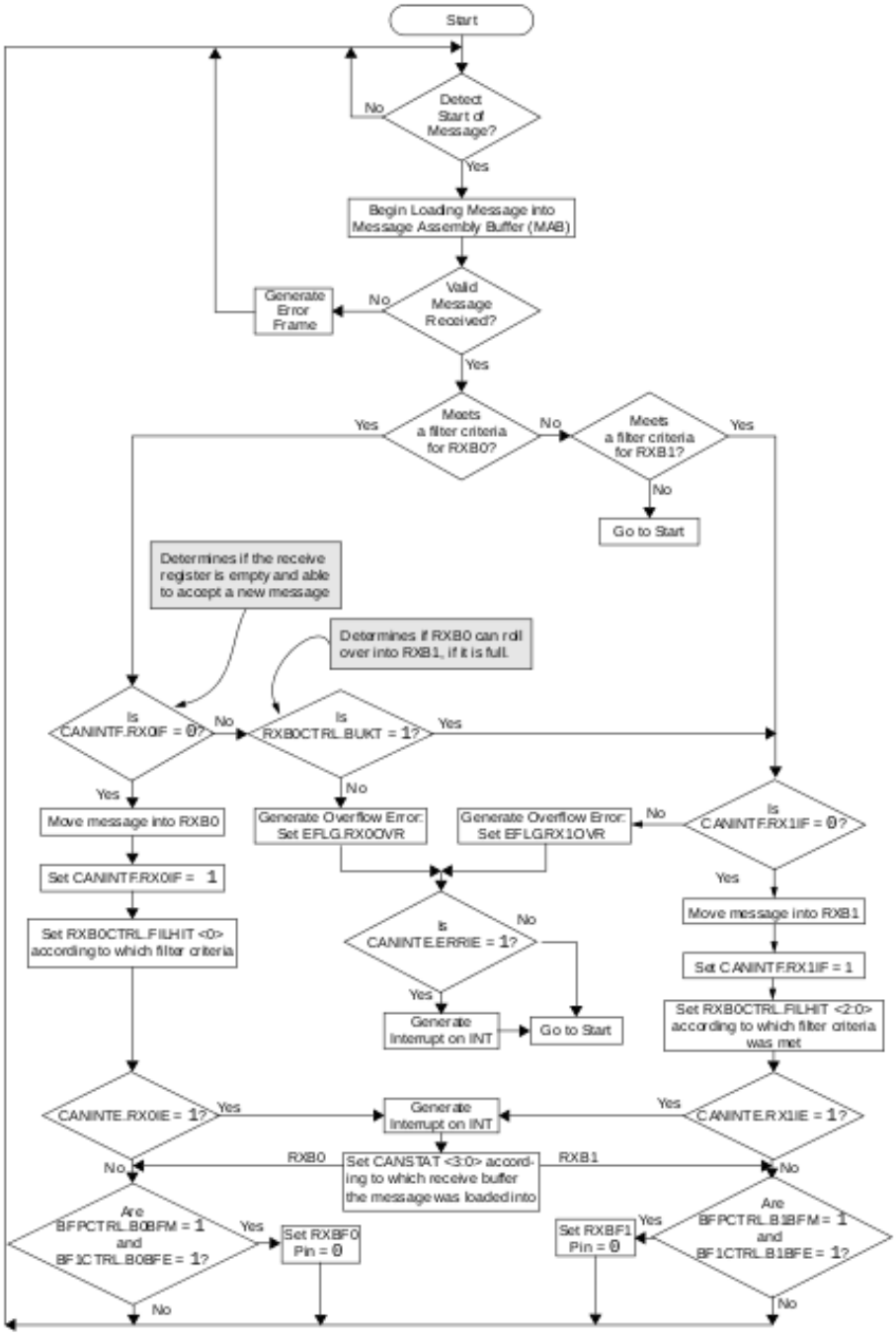

Figura 3.9: Màquina d'estats: Recepció

La recepció comença quan el buffer MAB detecta el 'Start of frame', seguidament anirà agafant els missatges i els anirà ajuntant. Si en aquest procés detecta que el format de la trama és incorrecte generarà una trama d'error. Si la trama rebuda és correcte i el filtratge és correcte l'enviarà al buffer que estigui lliure. Activa el bit CANINTF.RX0F i generarà una interrupció al pin INT.

3.3 Plataforma de software i llibreries

Arduino IDE El codi del programa s'ha desenvolupat amb Arduino IDE per la utilització d'un seguit de llibreries que ens faciliten la comunicació amb el hardware tant del mateix Arduino com del controlador CAN. Tenint la comunicació entre Arduino-Controlador ja donada m'he ocupat solament del desenvolupament de la meva aplicació.

Llibreries

CAN BUS Sparkfun La llibreria utilitzada per la comunicació amb el controlador CAN ve donada pel mateix fabricant de la shield, Sparkfun. En aquesta llibreria les funcions que utilitzo són aquestes.

1. **Canbus.init(velocitat)**: Configura el temps nominal de bit segons la velocitat de transmissió desitjada. Les possibles velocitats del bus configurables segons la llibreria són 500 Mbps, 250 Mbps i 125 Mbps. Per a més detalls com es configura el bus anar a l'apartat 3.5
2. **mcp2515-bit-modify(CANCTRL, (1«REQOP2)|(1«REQOP1)|(1«REQOP0), 0)**: Habilita la transmissió de missatges.
3. **mcp2515-send-message(message)**: Demana una petició de transmissió al bus CAN. Quan el controlador detecti el canal inactiu podrà transmetre. *message* és una estructura que representa una trama CAN.
4. **mcp2515-get-message(message)**: Llegeix la informació de la trama proporcionada pel buffer de recepció corresponent. *message* és una estructura que representa una trama CAN.

Alhora la llibreria implementa una estructura anomenada *tCAN* que representa una trama CAN. L'estructura està definida de la següent manera:

- **tCAN.id** Identificador de la trama.
- **tCAN.data[8]** Camp de dades. Longitud màxima de 8.
- **tCAN.header.rtr** Bit indica si és una trama remota.
- **tCAN.header.length** Longitud del camp de dades

TIMER Llibreria pròpia d'Arduino que implementa esdeveniments temporitzats.

Utilitzo els esdeveniments temporitzats en els nodes esclaus perquè els missatges es transmetin periòdicament. En concret envio 5 trames per segon. També l'utilitzo per comprovar que el node màster rep missatges dels esclaus mentre la comunicació està activa.

Les funcions utilitzades són:

- **Timer-after(funció, temps)** On *funció* és un callback que s'executarà cada *temps* mil·lisegons. Retorna l'identificador del timer.
- **Timer-Cancel(Id)** Cancel·la el timer corresponent a l'identificador *Id*.

3.4 Esquema general

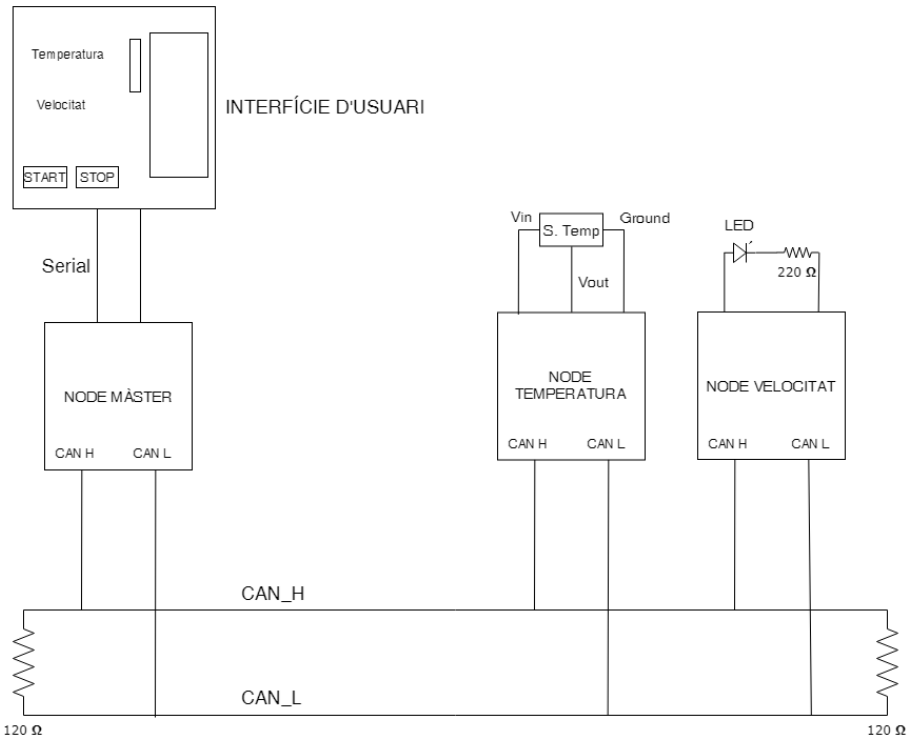

Figura 3.10: Esquema general del sistema

L'aplicació que s'ha volgut realitzar és una xarxa amb un node màster i dos nodes esclaus. Els esclaus representen sensors dintre d'un sistema i el màster la ECU que processa les dades.

Aquest sistema podria ser una simplificació de la xarxa que veuríem en qualsevol indústria. Un node esclau enviarà dades de velocitat en Km/h i un altre enviarà dades de temperatura en graus Celcius.

Els esclaus estan separats del màster a una distància d'uns 10 metres.

Els nodes esclaus aniran enviant missatges periòdicament, alhora el màster també podrà comunicar-se amb els esclaus.

Identificadors

Els identificadors que li dono a cada node depenen de la prioritat que considero que té cadascun.

L'identificador més baix, i per tant el més prioritari el tindrà el node màster, ja que ell tindrà prioritat sobre la resta.

Com és un sistema en temps real, la prioritat dels esclaus depenen de quina dada considero més important pel bon funcionament del sistema.

Per aquesta raó, considero que té una prioritat més alta conèixer el valor de velocitat que el de temperatura i les trames que enviï el sensor de velocitat seran d'una prioritat més alta respecte el sensor de temperatura.

3.5 Configurant el nostre CAN

La configuració del bus es realitza tal com s'explica a l'apartat 2.10 Temps de bit.

Configurant la xarxa a 500 Kbps

Amb una freqüencia de clock de 16MHz i un preescaler de 1, obtenim un temps de *Quantum* de 125 nanosegons.

Determinem quantes unitats de temps volem a cada segment:

- Segment de Sincronització: Una unitat de temps.
- Segment de Propagació: Dos unitats de temps.
- Segment PS1: Dos unitats de temps.
- Segment PS2: Tres unitats de temps.

En total són 8 unitats de temps, a 125 nanosegons cada unitat aconseguim un temps de bit d'1 microsegon. És a dir una velocitat de tranferència de 1 Mbps, el màxim possible.

Si nosaltres volem 500 Mbps simplement modifiquem el preescaler de la fòrmula del temps de *Quantum* a 2. La unitat de temps serà el doble, de 250 nanosegons. De tal manera que sense modificar el segments obtenim una velocitat de tranferència de **500 Mbps**.

3.6 Nivell del bus CAN

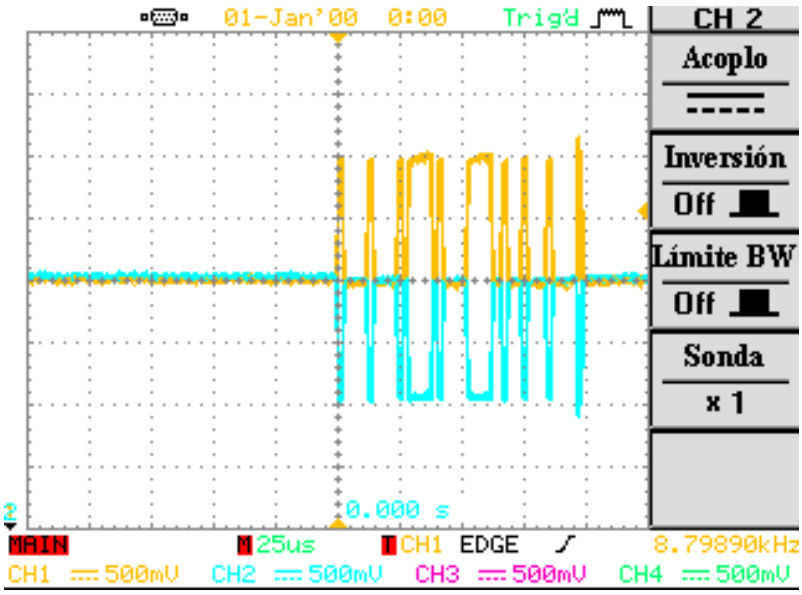

Figura 3.11: Nivells de CAN-H i CAN-L

En la figura de sobre podem observar les senyals CAN-H i CAN-L del nostre bus respecte el terra. Quan el bus està inactiu els dos senyals es troben a 2.5 V. Quan s'escriu un bit dominant CAN-H està a 3.5 V i CAN-L a 1.5 V.

Aquesta imatge correspon a una trama amb un únic byte de dades. Això correspon a l'enviament de 52 bits. Estant el nostre bus configurat per transmetre a 1 bit cada 2 microsegons esperem enviar 52 bits en 104 microsegons.

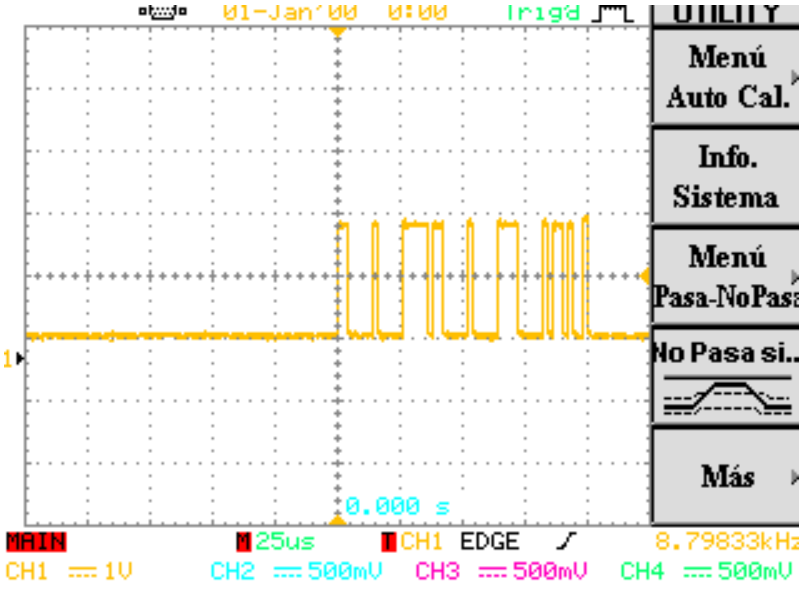

Figura 3.12: Voltatge diferencial del bus

En aquesta imatge observem la diferència de voltatge entre CAN-H i CAN-L. Comprovem com el voltatge diferencial entre ells és de 2V quan s'escriu un bit dominant.

3.7 Codi

3.7.1 Inici de comunicació

La comunicació comença únicament quan el node màster envia un missatge "START"als nodes de la xarxa, enviat per l'usuari. En aquest moment comença el procés d'identificació dels nodes de la xarxa on s'espera que els nodes esclaus connectats a la xarxa s'identifiquin.

Quan un node esclau rep aquest missatge "START"respondrà amb un missatge indicant el seu identificador i amb el camp de dades indicant quin sensor és (velocitat o temperatura).

El màster relacionarà cada identificador amb el tipus de dades que li enviarà. A partir d'aquest moment cada cop que rep una dada amb el corresponent identificador sabrà si correspon a velocitat o a temperatura.

Quan ja tots els nodes esclaus estan identificats comença la comunicació i els nodes enviaran la dada corresponent. El màster anirà llegint cada trama i la interfície li mostrarà a l'usuari.

3.7.2 Pèrdua de comunicació

Si el màster perd la comunicació amb un dels sensors, mostrarà un missatge d'error a la interfície indicant quin sensor s'ha perdut. A més, farà parpellejar un dels seus leds. Quan la comunicació torni ho mostrarà per la interfície i apagarà el seu led.

Aquesta funcionalitat s'aconsegueix gràcies a la utilització d'una funció callback utilitzant la llibreria timer. Periòdicament comprova que s'hagin rebut missatges dels sensors. Si no ha sigut així, es detecta pèrdua de comunicació al sensor corresponent.

Si un node esclau s'ha desconnectat de la xarxa mentre s'envien dades ho advertirà fent parpellejar un led fins que la comunicació torni.

3.7.3 Aturar la comunicació

L'usuari podrà parar l'enviament de dades dels esclaus enviant el missatge "STOP". La funcionalitat de detectar pèrdua de comunicació queda deshabilitada.

Per tornar a iniciar la comunicació s'haurà de tornar a enviar "START".

3.7.4 Manipulació LED esclau

Des de l'interfície, l'usuari podrà manipular la intensitat de lluminositat d'un LED connectat a un dels nodes esclaus.

3.7.5 Transmissió de missatges

La transmissió de missatges es realitza a partir d'esdeveniments temporitzats. Quan un esclau es rep el missatge d'inici de comunicació, just després d'enviar el missatge d'identificació s'inicia un Timer a 200 mil·lisegons. Quan ha passat aquest temps es crida una funció callback. En aquesta funció s'envia la dada amb el seu identificador.

3.7.6 Recepció de missatges

La recepció de missatges es realitza a partir de la interrupció del pin D2. El controlador CAN activarà aquest bit quan tingui una trama en un dels buffers de recepció i llesta per ser llegida.

Quan es detecti un flanc de pujada al pin d'interrupció la rutina d'interrupció s'executarà. Dintre la rutina d'interrupció es cridarà la funció **mcp2515-get-message(message)** on *message* ès una estructura *tCAN*.

Cada cop que el màster rep un missatge comprovarà si l'identificador de la trama és una de les aconseguides al procés d'identificació de nodes. Si és així, enviarà la dada a la interfície indicant de quina dada es tracta. Si no es cap identificador conegut ignora el missatge.

Quan sigui un dels nodes que reben un missatge comprovaràn si l'identificador correspon al node màster (conegut per tots). Si no es tracta del màster ignoren el missatge.

3.7.7 Filtrat de missatges

Una bona solució perquè els nodes esclaus no rebin les trames enviades d'altres esclaus, seria filtar els missatges perque només s'acceptin dades del màster. El problema de fer això és que quan el filtre d'un missatge no és acceptat tampoc s'envien les trames ACK, per tant conèixer si la fallada en l'enviament d'un esclau és per culpa d'ell o del màster no es coneixaria.

Per exemple, si no utilitzo cap filtratge i tinc un error en un esclau (master també detecta fallada) puc estar convençut que la fallada és del node esclau i no del màster. Perquè si el node esclau estiguès dintre la xarxa rebria el missatge ACK de l'altre esclau.

3.8 Intefície d'usuari

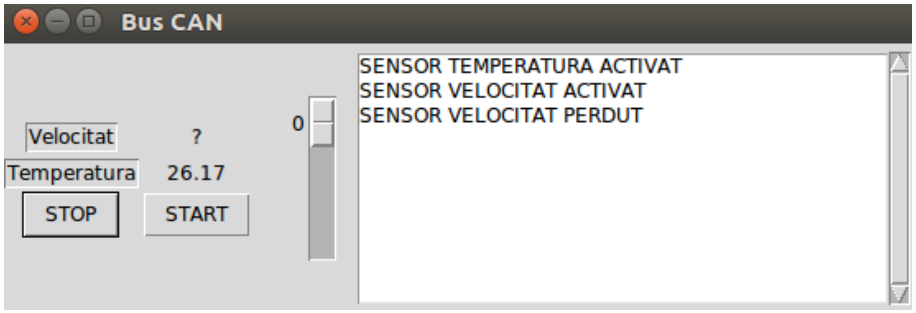

Figura 3.13: Interfície d'usuari de l'aplicació.

Disposem d'una petita interfície, en la qual l'usuari pot visualitzar els missatges que envien els sensors i interactuar amb els nodes sensors.

L'interfície va llegint constantment pel port sèrie les dades dels sensors que li proporciona el node màster. L'intefície actualitza el valor de les dades constantment.

L'inteficie ha estat desenvolupada amb Python amb el paquet TKinter, que és el que té per defecte Python.

He triat TKinter perquè per fer una petita interfície les eines que ofereix són suficients i no he cregut que fes falta utilitzar un altre més potent.

3.8.1 Monitoratge de dades

A la part esquerra tenim uns quadres de text identificats com 'Velocitat' i 'Temperatura' on s'aniran actualitzant les dades corresponents.

A la part dreta de l'interfície tenim un quadre de text on podrem llegir la informació de la xarxa. Els missatges que es poden llegir són:

- SENSOR TEMPERATURA/VELOCITAT ACTIVAT
- SENSOR TEMPERATURA/VELOCITAT PERDUT
- SENSOR TEMPERATURA/VELOCITAT RECUPERAT

3.8.2 Interacció amb la xarxa

A la banda esquerra de l'interfície hi han uns botons identificats com 'START' i 'STOP'. Cada botó envia a la xarxa el missatge corresponent.

També hi ha un *scroll* a la interfície, amb aquest comandament podem interactuar amb el LED d'un dels nodes esclaus modificant la seva intensitat.

Capítol 4

Conclusions

Amb aquest projecte s'ha volgut explicar els aspectes positius del bus CAN per entendre perquè encara avui dia es segueix utilitzant, no només en la indústria de l'automòbil sinó també amb altres sectors.

S'ha vist que el bus CAN permet la transmissió a prova de col·lisions d'una gran quantitat d'informació a nodes que es troben a molta distància.

També un aspecte important és la facilitat de connectar o desconnectar un node de la xarxa. No cal desconnectar la xarxa fer afegir cap node, simplement es connecten els dos cables al bus i node ja està dintre. I en cas de fallada en un dispostiu, la resta de la xarxa no es veurà afectada.

Podem dir que les principals avantatges són la seva robustesa, la velocitat de transferència a distàncies llargues i la simplicitat de connexió. A més, amb el baix cost dels seus component fa que la implementació del bus CAN sigui avui dia una opció molt viable.

4.1 Línies futures

Aquest projecte només engloba el protocol CAN estàndard, especificat a la norma ISO 11989-1 i ISO 11989-2. Segons l'ús que se li vol donar, cada sector ha desenvolupat altres protocols CAN que implementen capes de més alt nivell. Un possible projecte seria investigar de quina manera s'implementa CAN en la indústria de l'automòbil.

Existeixen diferents protocols, una possible opció seria investigar el protocol CANOPEN, desenvolupat per Cia (CAN in Automotion) i implementa la capa de xarxa.

El protocol CANOPEN compta amb suport per:

- La gestió de xarxa
- Monitoratge de dispositius i comunicació entre nodes.
- Petita capa de transport.

El desenvolupament del projecte seria similar al realitzat en aquest. Primer es tractaria d'investigar el protocol i entendre com funciona. La segona part es tractaria de posar en pràctica el protocol. Es buscaria un automòbil que implementés CANOPEN, (o el protocol escollit) connectar-se a la xarxa del vehicle i observar les comunicacions que es produeixen.

Bibliografia

[1] *CAN bus*. URL: https://en.wikipedia.org/wiki/CAN_bus. (accessed: 05.03.2018).

[2] *CAN-BUS Shield Hookup Guides*. URL: https://learn.sparkfun.com/tutorials/can-bus-shield-hookup-guide?_ga=2.266062603.528726824.1530606880-917999329.1530606880. (accessed: 10.03.2018).

[3] *CAN Controller With SPI Interface*. URL: https://www.sparkfun.com/datasheets/DevTools/Arduino/MCP2515.pdf. (accessed: 15.03.2018).

[4] *High Speed CAN Transceiver*. URL: https://www.sparkfun.com/datasheets/DevTools/Arduino/MCP2551.pdf. (accessed: 15.03.2018).

[5] *Introduction to the Controller Area Network*. URL: http://www.ti.com/lit/an/sloa101b/sloa101b.pdf. (accessed: 10.04.2018).

[6] *Controller Area Network Physical Layer Requirements*. URL: http://www.ti.com/lit/an/slla270/slla270.pdf. (accessed: 10.04.2018).

[7] *CANopen*. URL: https://en.wikipedia.org/wiki/CANopen. (accessed: 5.06.2018).

[8] Marco di Natale, Haibo Zeng i Paolo Giusto. *Understanding and using the Controller Area Network*. Springer, 2008.

[9] *MCP9700 Sensor Temperatura*. URL: https://www.mouser.com/datasheet/2/268/20001942F-461622.pdf. (accessed: 8.05.2018).