UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

TREBALL FI DE GRAU

Grau en Enginyeria Electrònica Industrial i Automàtica

DISSENY, IMPLEMENTACIÓ I TEST D'UN MÒDUL DE COMUNICACIONS SÈRIE PER A UN MICROCONTROLADOR

Memòria i Annexos

Autor: Joan Grané Andreu
Director: Jordi Cosp Vilella
Departament: EEL

Convocatòria: Maig 2022

Resum

Actualment existeixen diferents mètodes de comunicació sèrie que poden ser incorporats com un mòdul per microcontrolador. Un d'aquest mètodes és l'anomenat USART (*Universal Synchronous Asynchronous Recieve Transmit*), que permet la comunicació tant en mode síncron com asíncron. El principal problema que presenta és la difícil implementació en llenguatge d'alt nivell i sobretot en FPGA's.

Per tant, aquest projecte pretén dissenyar un mòdul de comunicacions basat en el mètode USART, ja que dins l'àmbit electrònic s'han desenvolupat implementacions teòriques però poques s'han arribat a implementar sobre FPGA's

Aquest s'ha desenvolupat mitjançant el llenguatge de programació d'alt nivell VHDL i s'ha provat sobre una placa FPGA, basant-se en varies idees teòriques no implementades.

Dit això, es presentarà una idea teòrica del mòdul a dissenyar així com la implementació sobre una FPGA i una prova on s'establirà comunicació amb un altre microcontrolador per comprovar la seva funcionalitat.

Resumen

Actualmente existen diferentes métodos de comunicación serie que pueden ser incorporados como módulos para microcontroladores. Uno de estos métodos es el conocido como USART (*Universal Synchronous Asynchronous Recieve Transmit*), que permite la comunicación tanto en modo síncrono como asíncrono. El principal problema que presenta es su difícil implementación con lenguaje de alto nivel y sobre todo en FPGA's

Por tanto, este proyecto pretende diseñar un módulo de comunicaciones basado en el método USART, ya que dentro del ámbito de la electrónica se han desarrollado implementaciones teóricas, pero pocas se han llegado a implementar sobre FPGA's.

Este diseño se ha desarrollado mediante el lenguaje de programación de alto nivel VHDL y se ha testeado sobre una placa FPGA, basándonos en varias ideas teóricas no implementadas.

Dicho esto, se presentará una idea teórica del módulo a diseñar, así como la implementación sobre una placa FPGA y una prueba donde se establecerá una comunicación con otro microcontrolador para poder comprobar su funcionalidad.

Abstract

Nowadays we can find different protocols of serial communication that could be integrated as a microcontroller module.

One of them is the called USART (Universal Synchronous Asynchronous Receive Transmit). The main issue is the difficulty in its implementation in the high-level language, mainly in FPGA's.

Thus, this project aims to design a communication module based on the USART method, as within the electronic field have already been developed several theoretical implementations but not that many have been implemented in FPGA's.

This design has been built up throughout the high-level language VHDL and put into practice on a FPGA board, using different theoretical ideas not implemented.

Therefore, this project presents a theoretical idea of the module to be designed as well as its implementation on a FGPA and a trial where it will be established a communication with another microcontroller to test the functionality.

Índex de contingut

Índex de figures

Índex de taules

Índex de funcions

1. Prefaci

1.1. Motivació

La principal motivació per a la realització d'aquest treball ha sigut el món dels microcontroladors i la programació que aquests comporten. Mitjançant aquest treball es pretén aprendre i millorar conceptes que han sigut adquirits durant la carrera i que resulten molt interesants de cara al futur i món laboral

1.2. Requeriments previs

Per a la realització d'aquest treball és necessari tenir coneixement de electrònica digital, comunicació i programació de VHDL. Per altre, banda per a poder escriure el codi, es requereix del software ModelSim, i per poder sintetitzar-lo, el software ISE de XIlinx així com una FPGA Basys 2.

Cal remarcar que aquest treball es podria realitzar amb altres programes que permetin la simulació de codi VHDL/HDL així com amb qualsevol FPGA i el seu entorn per sintetitzar.

2. Introducció

2.1. Objectius del treball

- Disseny teòric del diagrama de blocs que contindrà el mòdul de comunicacions:
  - Bloc receptor.
  - Bloc transmissor.
  - Bloc de detecció i correcció d'errors.
  - Bloc de control principal.
  - Bloc generador de senyal de rellotge per mode síncron.
- Descripció d'alt nivell mitjançant VHDl de tots els blocs teòrics.
- Simulacions on es comprovi el correcte funcionament dels diferents blocs per separat.
- Disseny d'un bloc que unifiqui els diferents mòduls.
- Simulacions del bloc general.
- Implantació física sobre una FPGA.

2.2. Abast del treball

L'abast del treball queda definit amb els propis objectius comentats en l'apartat anterior, però si que es volen comentar una sèrie de punts sobre aquest projecte.

Primer de tot, tenir en compte que tot i ser un treball sobre el disseny d'un mòdul de comunicacions *USART*, és un projecte de nivell educatiu, pel que s'entén que no tindrà el mateix rendiment i característiques complertes com poden tenir els mòduls que es poden trobar actualment en el mercat. També complementant aquest punt, s'ha de tenir en compte que és un projecte realitzat per un sol estudiant amb la formació rebuda durant els 4 anys d'estudis del grau (dels quals una gran part no estava enfocada en aquest àmbit) i no un equip d'enginyers com es podria tenir en una empresa.

Tot i així es pretén complir els punts esmentats en l'apartat d'objectius del treball.

3. Microcontroladors

3.1. Microcontroladors

Un microcontrolador és un circuit integrat digital que conté principalment una unitat central de procés (CPU o processador), memòria pròpia (RAM o ROM) i sistemes de comunicació d'entrada i sortida. [1]

A continuació es detallen els elements integrats més importants que conformen el chip integrat d'un microcontrolador.

3.1.1. Processador

S'entén per processador (CPU) un circuit integrat que conté la Unitat Central de Processos d'un computador. És l'element més important d'un microcontrolador, doncs és qui determina les principals característiques tant a nivell de *software* com de *hardware* [2]. Es podria dir que el processador és el "cervell " del sistema, ja que és qui decideix les accions que realitzarà el microcontrolador.

Aquest element és utilitzat de forma independent en molts sistemes ja que permet una millor configuració del entorn en el que treballarà. El microcontrolador, per altre banda, ofereix un processador i aquest entorn en un mateix chip integrat, cosa que per determinades aplicacions facilita molt el seu disseny.

Existeixen tres tendències respecte la funcionalitat i arquitectura d'aquests:

- **CISC:** Permet a l'usuari la programació d'instruccions més complexes
- **RISC:** Es caracteritza per tenir instruccions ràpides i senzilles que faciliten l'optimització de hardware i software
- **SISC**: instruccions reduïdes i especifiques per a realitzar aplicacions molt concretes

3.1.2. Memòria

Un microcontrolador ha de contenir una memòria ROM, és a dir, una memòria no volàtil on es guardarà tot el conjunt d'instruccions que conformen el programa; i una RAM, que s'utilitzarà per a guardar variables i elements volàtils i canviants del disseny[2].

Generalment les memòries que contenen els microcontroladors no tenen una gran capacitat, ja que no han de guardar quantitats massives de dades, sinó al revés, perquè generalment han de guardar un únic programa que executa la funció dissenyada per a què realitzi el microcontrolador.

3.1.3. Perifèrics

El microcontrolador disposa de pins que serveixen com ports d'entrada/sortida i permeten que es comuniqui amb altres controladors, processadors o sistemes externs mitjançant ports digitals o analògics.

Depenent del sistema es tindrà un tipo de comunicació o un altre, i un voltatge i amperatge per configurar les entrades i sortides de les dades rebudes (generalment treballen a 3,3V o 5V)

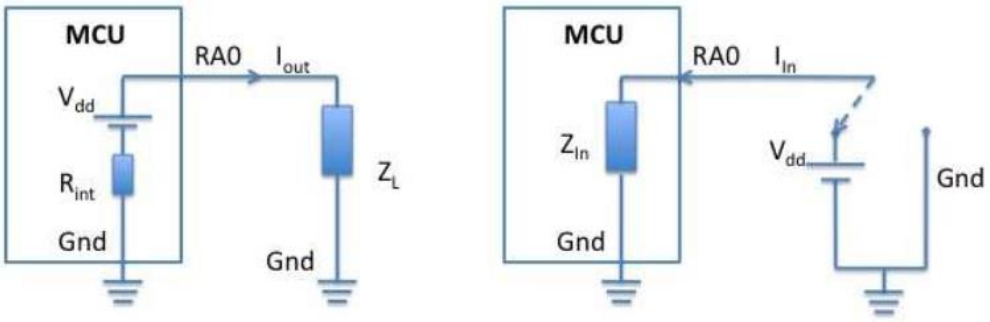


Figura 3-1: Pin A0 configurat com sortida (esquerra) i com entrada (dreta). [2]

La figura 3.1 mostra com seria la connexió d'un pin (A0) de microcontrolador connectada per a enviar i rebre senyals.

3.1.4. Rellotge intern

Els microcontroladors disposen d'un o variïs oscil·ladors que permeten generar senyals quadrats que s'utilitzen després com a senyals de rellotge internes. Durant el disseny de hardware és molt important aquest rellotge intern, perquè permet fer un disseny síncron del sistema. Depenent del circuit integrat del que es disposi, la freqüència del rellotge pot ser fixe o variable, permetent a l'usuari la configuració de la velocitat.

Aquest senyal pot agafar-se del propi rellotge intern del chip integrat, o bé generar-lo de forma externa amb sistemes de ressonància.

El domini i control d'aquest rellotge és molt important, sobretot en l'àmbit de les comunicacions, per a poder ja que si es vol establir un sistema entre dos microcontroladors diferents, aquests no tenen per què treballar amb la mateixa freqüència interna i per tant no executaran les instruccions amb la mateixa velocitat.

3.2. Modes de transmissió

"Per comunicació s'entén el procés mitjançant el qual es transmet una certa informació des d'un punt anomenat font o emissor a un altre anomenat destí o receptor. En el context de les comunicacions, la informació constituirà el conjunt de paràmetres d'interès que transmet el emissor al receptor mitjançant la variació, en la majoria dels casos, d'un senyal elèctric"[4].

Com s'ha esmentat anteriorment, els microcontroladors disposen de diferents ports per a ser utilitzats com entrades i sortides. Aquests ports es faran servir per enviar i rebre dades a diferents perifèrics connectats al sistema. Per a poder establir aquesta transmissió de dades, existeixen diferents protocols i modes de comunicació establerts de manera única per poder definir unes bases que tots dos sistemes coneguin i permeti una comunicació sense errors.

A continuació es presentaran diferents aspectes que s'han de tenir en compte quan es parla de comunicació dins el món de l'electrònica.

3.2.1. Sincronisme

Tota transmissió requereix que el receptor i el transmissor es regeixin per la mateixa base de temps a l'hora de rebre o enviar les dades. Si això no es compleix, resultarà impossible determinar en quin moment s'està enviant cada valor. Per exemple, si es treballa amb un receptor que llegeix el valor rebut cada dos segons però el transmissor està enviant la informació cada segon, per cada bit que el receptor llegeixi, es perdrà un altre.

Per evitar aquest problema, existeix el que anomenem sincronisme en les comunicacions, que permet establir una base de temps que marca el moment en que un nou bit ha de ser enviat i llegit. Trobem dos possibles modes de sincronitzar una transmissió.

3.2.1.1. Transmissió síncrona

Durant la comunicació síncrona, el receptor i el transmissor treballen en funció d'una base de temps externa. Aquesta línia addicional marca una pauta pels dos mòduls sobre quan s'ha de rebre i enviar cada bit. A la figura 3.2 es pot observar el funcionament d'una transmissió síncrona, on es pot veure com la base temps marca amb un impuls el moment de canvi de cada bit. En moltes ocasions, i per evitar possibles errors, aquesta línia de temps és generada pel transmissor i enviada a través d'una línia extra a la de comunicació.

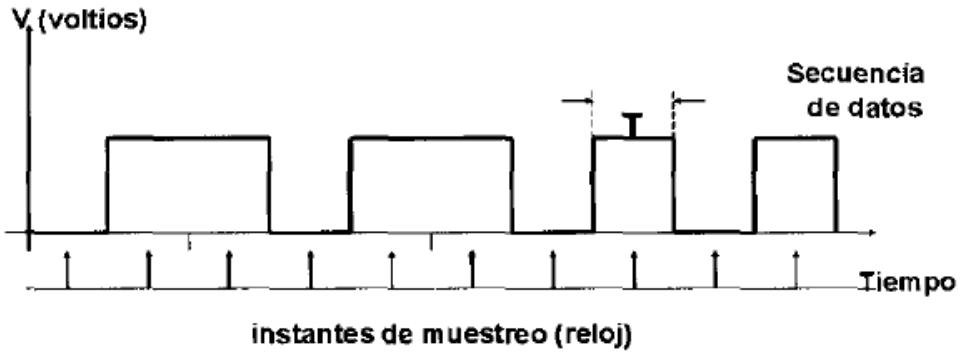


Figura 3-2: Senyal de dades síncrona. [4]

3.2.1.2. Transmissió asíncrona

La comunicació asíncrona no disposa de cap línia que marqui les bases temporals com és el cas vist anteriorment, sinó que cada uns dels mòduls treballa amb el seu senyal de rellotge propi. Per tal de poder establir un ordre durant la comunicació, es busca una velocitat de transmissió (*bps* o Bauds) a la que tots dos sistemes puguin treballar i comunicar-se de manera ordenada.

Com es pot observar a la figura 3.3, aquestes transmissions generalment estan formades per un bit d'inici, els bits que conformen la paraula i uns bits de parada.

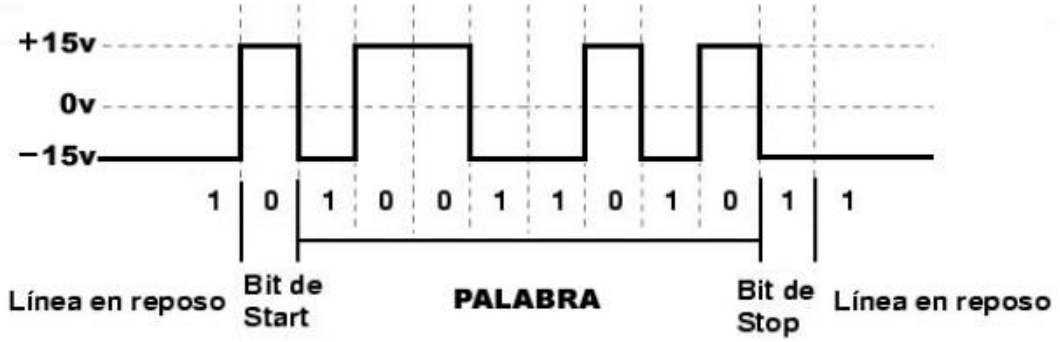


Figura 3-3: Senyal de dades asíncrona. [4]

3.2.2. Comunicació sèrie/paral·lel

La comunicació en paral·lel es caracteritza per la transmissió simultània de tots els bits que formen la paraula. Això implica que s'ha de disposar d'una línia per cada bit simultani que es vulgui transmetrepel que els costos i les possibilitats d'error augmenten per cada línia extra. Un gran desavantatge que presenten els sistemes amb comunicació paral·lel és quan es treballa a grans distàncies, ja que es

necessita una gran quantitat de cables i això produeix interferències entre les línies. Pel contrari, si es treballa a distàncies curtes i ambients amb poca interferència electromagnètica, aquesta opció presenta una gran velocitat de transmissió.

A diferència de les comunicacions en paral·lel, la comunicació sèrie transmet les dades bit a bit a través d'una única línia. Per tant, és la recomanada si es vol treballar a gran distàncies o ambients que presentin més interferència electromagnètica.

Per exemple, si es té una transmissió d'un byte, en una comunicació en paral·lel s'hauria de disposar de vuit línies de transmissió, a diferència de la comunicació sèrie, que només en necessitaria una. Pel contrari, la comunicació en paral·lel presentaria una velocitat vuit vegades més ràpida que la sèrie.

3.2.3. Mètodes de detecció d'error

Per tal de poder detectar si durant la transmissió de les dades s'ha produït algun error, existeixen mètodes de detecció d'errors, sent els més comuns els que es comentaran seguidament.

3.2.3.1. Paritat

Quan es comunica amb bits de paritat, s'envia un bit juntament amb la cadena de dades que indica si el número de 1's que conté el missatge és parell o senar. El bit afegit tindrà un valor de '1' o '0' per tal de completar la cadena de bits i complir la paritat escollida per l'usuari.

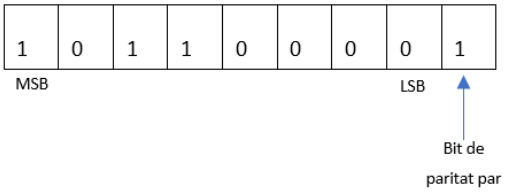


Figura 3-4: Cadena de bits amb un bit de paritat par

A la figura 3-4 es pot observar com, en cas de tenir una paritat par, aquest bit prèn el valor de '1'. Això és perquè el número de '1' que hi ha al missatge és un nombre senar i amb el de paritat passa a ser parell.

En la detecció d'errors, un cop s'ha enviat el missatge, el receptor efectuarà la mateixa operació i es comprovaran si els dos bits de paritats (el rebut i el calculat pel receptor) coincideixen.

Aquest mètode presenta el principal error de que en cas que es rebin de forma errònia dos bits del mateix valor, no afectarà a la paritat del global del missatge i per tant no es detectarà que hi ha hagut un error.

3.2.3.2. *Checksum*

Consisteix en enviar juntament amb els paquets de dades enviats, el valor total de la suma d'aquets. En quant el missatge arriba al receptor, aquest realitzarà la mateixa operació i comprovarà que el valor calculat coincideixi amb el rebut.

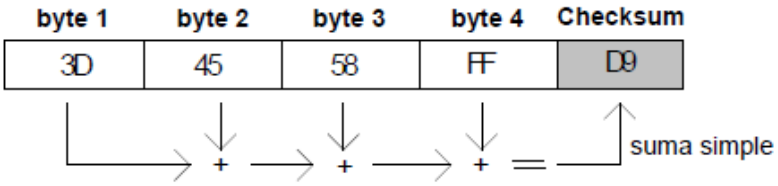


Figura 3-5: Càlcul del valor de checksum. [4]

Aquest mètode de detecció d'errors es bastant robust, ja que per a què el *Checksum* coincideixi en els dos sistemes i que realment hi hagi hagut un error durant la comunicació, hauria de baixar el valor d'un dels paquets i augmentar aquesta mateixa diferència en un altre, situació que resulta difícil que suceeixi.

3.2.3.3. Detectar bits de parada

Per aquest mètode de detecció d'error s'afegeixen al final de la cadena de bits, uns bits de parada. El receptor al rebre el missatge podrà detectar si en el moment en que s'han rebut tots el bits, es rep també la cadena de bits de parada. Si aquesta es rep de forma correcta, voldrà dir que no hi ha hagut cap bit que s'hagi perdut durant la transmissió.

El principal problema que presenta aquest mètode és que els bits de parada coincideixin amb els propis de la cadena del missatge. En aquest cas podria coincidir que el receptor donés per finalitzada la comunicació i al comprovar els bits de parada realment estigués comprovant uns del mateix valor del propi missatge.

3.2.3.4. CRC (Cyclic Redundancy Code)

A diferència dels mètodes anteriors, aquest és molt més complex i per tant, més precís.

Al utilitzar mètodes més complexos:

- Permeten una detecció més segura dels errors.
- Aconsegueixen una millor eficiència.

Aquest mètode és equivalent a enviar darrera el missatge el resto que es treuria de fer la divisió del valor dels bits de dades amb un número acordat. Per exemple, si volem enviar el valor de 30 i acordem que el número divisor sigui 4 el resto serà de 2 (Eq. 3.1).

$$Resto = 30 \; mod \; 4 = 2 \qquad \textbf{(Eq. 3.1)}$$

Així doncs, el missatge final enviat seria de 302. El receptor faria la mateixa operació (amb el mateix divisor que el transmissor) i comprovaria que el valor coincidís.

3.2.4. Protocols de comunicació

Els protocols de comunicació són un conjunt de normes que regeixen la comunicació entre dos sistemes. Aquestes normes recullen una sèrie de configuracions, algunes de les comentades anteriorment, i permeten que les dues entitats segueixin la mateixa estructura. A continuació es comenten alguns dels protocols més coneguts per a la transmissió sèrie.

3.2.4.1. BUS I2C

Per a poder establir una comunicació I2C, es requereixen dues línies i una massa o comú. Aquest protocol estableix una comunicació sèrie síncrona, on una línia transmet les dades i l'altre marca el temps de transmissió. Aquest permet una transmissió que comunica un dispositiu mestre, el qual enviarà les dades i la senyal de rellotge, amb variïs dispositius esclaus, els quals reben les dades. El dispositiu que actua com a mestre, pot enviar les dades a tots els esclaus o escollir a quin d'ells enviar la informació concretament.

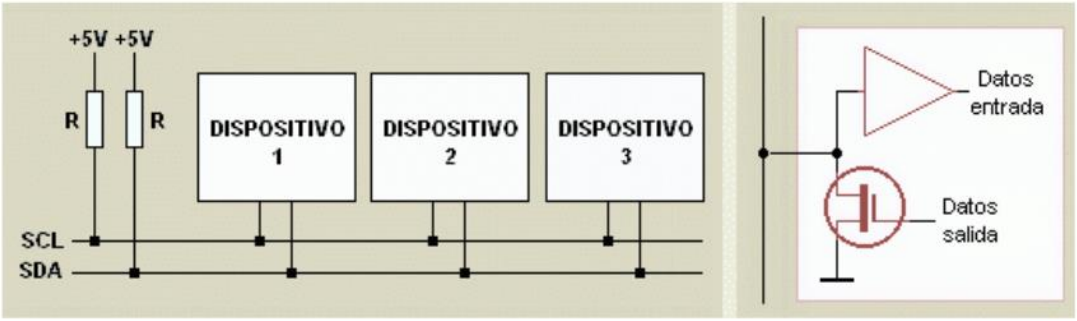


Figura 3-6: Protocol de comunicació I2C. SCL (System Clock) i SDA (Signal Data) son les senyal de rellotge i dades respectivament. [6]

Com es pot observar a la figura 3.6, quan línies SCL (senyal de rellotge) i SDA (senyal de dades) es troben en un estat inactiu, tenen un valor lògic alt. Per a poder iniciar la comunicació, el mestre enviarà un bit d'inici a nivell baix a la línia SDA i mantindrà la línia SCL a nivell alt. Seguidament, d'aquest bit s'enviarà un byte per indicar l'adreça a la que es dirigeix el missatge, és a dir, escull a quin dels esclaus va

destinada la transmissió. Aquest byte contindrà 7 bits per l'adreça, i el vuitè bit indicarà quina operació es vol realitzar, si de lectura o d'escriptura. En cas que la direcció correspongui a un dispositiu que es trobi en el bus, aquest respon amb un bit a nivell baix per a confirmar la transmissió i que pugui començar. Cada un dels esclaus tenen una direcció única i en cas que no es trobi, s'induirà a un error del sistema.

La línia de bits transmesa està conformada per 8 bits de dades als que se li suma un bit d'inici i un de parada. És possible afegir un mode de funcionament en el que el esclau enviï un bit de confirmació després de rebre cada Byte, i així indicar al mestre que està preparat per acceptar el següent missatge.

3.2.4.2. SPI

Per establir una comunicació SPI es necessiten les següents línies :

- **SLCK (*Signal Clock*):** Senyal de rellotge enviada pel màster als esclaus.
- **MOSI (*Master Out-Slave*):** Senyal de dades del màster al esclau.
- **MISO(*Master In-Slave*):** Senyal de dades del esclau al màster.
- **SSn (*Slave Select*):** Permet seleccionar amb quin màster es realitzarà la comunicació.

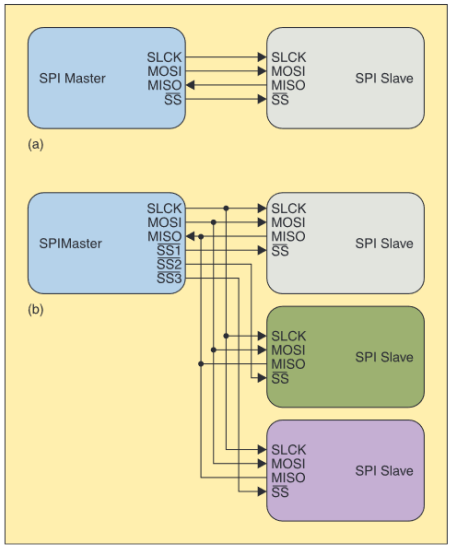


Figura 3-7: Protocol de comunicació SPI amb un o variïs esclaus. [7]

Com es pot intuir amb la definició dels senyals i la figura 3.7, es tracta d'un protocol semblant al I2C que permet una comunicació d'un esclau a diferents màsters.

Quan el màster desitja comunicar-se amb un esclau, posa la seva línia corresponent (SS) a nivell baix per a indicar-li a aquest que serà qui interactuarà amb ell. Juntament amb això activarà la línia de rellotge i usarà les línies MISO i MOSE per rebre i transmetre bits.

Tant si el màster desitja rebre o transmetre bits, serà ell qui enviarà el senyal de rellotge cap els sistemes esclaus. En cas de actuar com a transmissor, enviarà els bits juntament amb la línia de rellotge; si per altre banda actua com a receptor, enviarà el senyal de rellotge quan vulgui que el sistema esclau enviï les dades.

3.2.4.3. USART

USART (Universal Synchronous Asynchronous Receive Transmit), és un protocol que permet una comunicació adaptable sèrie entre dos sistemes segons les preferències de l'usuari. Les principals característiques que presenta son les següents:

- **Comunicació *Full Duplex.***
- **Comunicació síncrona i asíncrona.**
- **Comunicació Master-Slave amb generació de senyal de rellotge.**
- **Velocitat de comunicació variable.**
- **Suporta cadenes de *frames* entre 5-9 bits de dades, bit de paritat i 1 o 2 bits de parada.**

La principal diferència respecte altres protocols de comunicació sèrie és que permet una comunicació tant síncrona com asíncrona, és a dir, quan estigui en mode asíncron no farà falta tenir un senyal de rellotge compartit entre els dos sistemes.

- **Comunicació síncrona:** La comunicació es produeix en dues línies, una pel senyal de dades i un altre pel senyal de rellotge (generat pel transmissor).
- **Comunicació asíncrona:** Només es fa servir una línia de dades.

En cas de tenir el mode de transmissió asíncrona, es té una línia I/O que permet la comunicació entre els dos sistemes sense la necessitat d' enviar un senyal de rellotge. En comptes d'aquest senyal, tots dos sistemes acorden prèviament un *Baud Rate* que serà utilitzat durant la transmissió, i cada un dels dos mòduls generarà el seu senyal de rellotge propi.

Si es treballa en mode síncron, realitza un funcionament semblant al del protocol SPI. Per tant s'enviarà un bit d'inici seguit d'un bit de adreça i posteriorment s'enviaran les dades seguint la velocitat marcada pel senyal de rellotge enviat.

Aquest protocol permet configurar diferents paràmetres a nivell d'usuari, els quals són:

- **Mode de comunicació:** Permet seleccionar si la comunicació serà síncrona o asíncrona.
- **Mida de la paraula:** es permet enviar entre 5 a 8 bits de missatge. Això permet una comunicació més optimitzada en la que es guanya velocitat si es configura correctament.

- **Selecció de la velocitat de transmissió:** permet seleccionar la velocitat a la que es comunicaran els dos sistemes.
- **Error de paritat par o senar:** Permet triar si es genera un bit de paritat o no. En cas de generar aquest bit, també es permet la selecció de si serà paritat par o senar.
- **Bits de parada:** Decidir el nombre de bits de parada al final de cada transmissió (1 o 2).
- **Bit d'adreça:** En cas que es vulgui treballar amb un USART actuant com a màster i variïs com esclaus, s'activa el bit d'adreça.
- **Mode de doble velocitat:** Permet doblar la velocitat escollida per la transmissió de dades.

El protocol USART és el mes conegut i usat en quant a comunicació asíncrona, doncs permet una gran configuració dels paràmetres que conformen la comunicació.

En l'apartat 4 d'aquest projecte es realitza una explicació més detallada sobre aquest protocol de comunicació.

4. USART

Dins de l'apartat genèric sobre els microcontroladors, s'ha comentat breument el funcionament d'aquest protocol, però ja que aquest projecte tracta sobre el disseny i implementació d'un mòdul de comunicacions que es comuniqui mitjançant aquest estàndard, a continuació es farà una explicació més complexa sobre aquest.

4.1. Protocol de comunicació

La cadena de bits transmesa pel protocol USART consisteix en una cadena de 5 a 8 bits de dades, iniciada per un bit d'inici, seguida per un possible bit de paritat (en cas de tenir el mode de paritat actiu) i un o dos bits de parada. A la figura 4-1 es pot veure de manera més clara el format de la comunicació.

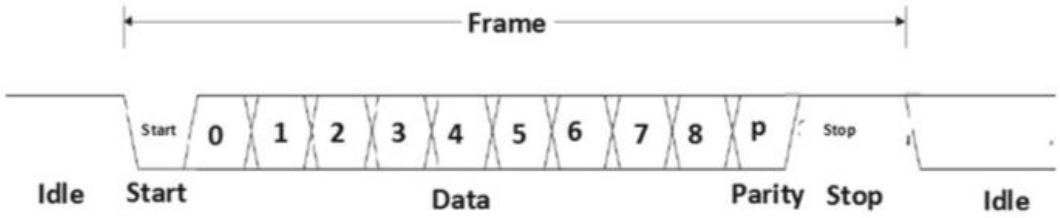


Figura 4-1: Cadena de bits d'una comunicació USART[10]

Un sistema USART es comunica mitjançant dues línies de dades:

- **Senyal de dades:** el senyal es manté en un estat *idle* de valor alt fins que el transmissor dóna l'ordre d'iniciar una comunicació
- **Senyal de rellotge:** el senyal només s'utilitza si es treballa en mode síncron, però és recomanable tenir-lo sempre actiu per a no perdre el sincronisme entre tots dos sistemes.

L'ordre dels bits que es transmeten a través d'una comunicació USART (com es pot visualitzar a la foto anterior) és la següent:

- **Bit d'inici:** S'envia un bit a nivell baix.
- **Bits de dades:** Entre 5 a 9 bits enviats des del LSB al MSB.
- **Bit de paritat:** Bit de paritat (opcional) parell o senar.
- **Bits de parada:** 1 o 2 bits de parada a nivell alt.

4.1.1. Comunicació síncrona

Quan es treballa en mode síncron, el protocol USART treballa de manera semblant al protocol SPI. Per tant, el sistema transmissor funciona en mode Màster i el receptor com a esclau. Com que en un mòdul USART es permet la selecció de la velocitat, serà el transmissor qui faci les respectives operacions per tal de poder ajustar el senyal del rellotge del sistema a la velocitat especificada (el mode de regular la velocitat està explicat en el següent apartat).

Un cop el transmissor tingui generada la velocitat de rellotge s'enviaran les dades per la corresponent línia i els impulsos que marquen els períodes de temps en els que s'ha de treballar per tal de rebre correctament tots els bits.

4.1.2. Comunicació asíncrona

Quan es treballa amb una comunicació asíncrona (UART), no es fa servir la línia que envia el senyal de rellotge, sinó que només es treballa amb la senyal de dades. Igual que en la comunicació sèrie, la velocitat és regulable, pel que s'han de ajustar tant el rellotge de transmissor com el del receptor de manera independent, ja que cadascun pot treballar a una freqüència diferent.

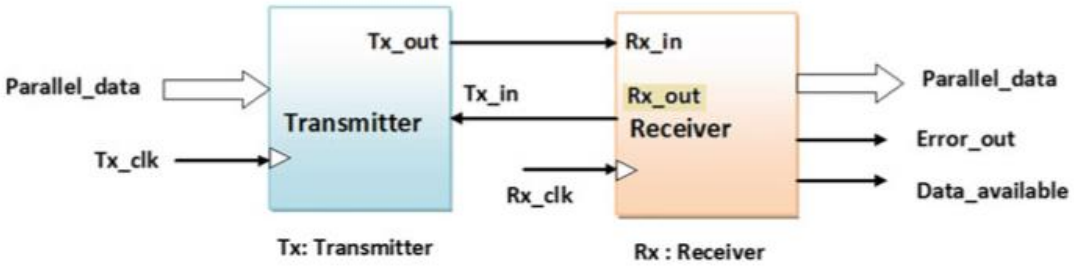


Figura 4-2: Comunicació asíncrona d'un mòdul USART[10].

4.2. Registres

El comportament de la comunicació USART ve determinat per registres d'estat, control i dades que permeten llegir tota la informació corresponent a la comunicació.

A continuació es mostraran els registres teòrics que ha de contenir un mòdul USART. La informació que es mostra sobre els registres s'ha extret de la referència [14].

4.2.1. Registre Baud Rate (UDR)

Tant si es treballa de manera síncrona com asíncrona, la velocitat de transmissió és regulable. Per això, els mòduls USART disposen d'un registre d'entrada que conté el valor de Baud Rate. Aquest valor permet ajustar el senyal de rellotge del sistema per a què treballi a la velocitat desitjada. Sempre es permet reduir velocitats, però en cap cas es podrà treballar amb velocitats superiors a la del propi sistema.

Així, tenint en compte la velocitat pròpia del rellotge de cada mòdul, es pot calcular quin valor ha de tenir el registre per tal de poder treballar a la velocitat desitjada.

$$Registre\ Baud = \frac{f_{oscil·lació}}{16 * VELOCITAT} - 1 \qquad \textbf{(Eq. 4-1)}$$

On:

- **Registre Baud:** Valor que ha de tenir el registre per regular la velocitat.
- **Velocitat:** La velocitat desitjada a la que es comunicarà.
- **Freqüència d'oscil·lació:** Freqüència a la que funciona el mòdul on s'implementa el disseny.

Mitjançant l'equació 4.1 es pot calcular quin valor ha de tenir el registre de Baud per tal de poder transmetre a la velocitat desitjada. Per tant, en cas de tenir dos senyals de rellotge diferents permetrà una comunicació sincronitzada utilitzant un registre de valor 'x' per a cada sistema

Per exemple, si es treballa amb un mòdul receptor que treballa a 100MHz i un mòdul transmissor que treballa a 50MHz i es desitja una velocitat de 9600 Bauds, el valor de Baud Rate per cada sistema serà el següent:

$$Registre_{Transmissor} = \frac{50MHz}{19 * 9600} - 1 = 324{,}52 \approx 325 \qquad \textbf{(Eq. 4-2)}$$

$$Registre_{Receptor} = \frac{100MHz}{19 * 9600} - 1 = 650{,}04 \approx 650 \qquad \textbf{(Eq. 4-3)}$$

Com es pot veure en l'exemple anterior, el valor ha de ser un número enter i per tant s'ha de considerar la possibilitat de que es produeixi cert error. L'equació 4-4 mostra com es pot calcular aquest error.

$$Error[\%] = 100\% * \left(\frac{BaudRate\ Real}{BaudRate}\right) \quad \textbf{(Eq. 4-4)}$$

Aquest valor s'entra al sistema mitjançant un registre de 16 bits

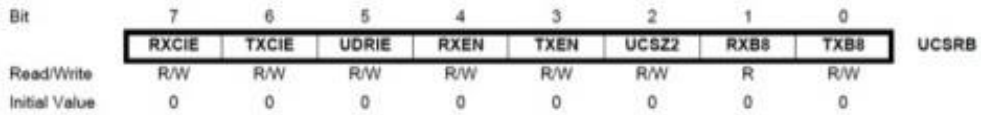
4.2.2. Registre d'estat i control A (UCSRA)

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

Figura 4-3: Registre USART d'estat i control A (UCSRA) [14].

- **Bit 7 [Rxc]** ***USART Recieve Complet flag*****:** Es posa a nivell alt per a indicar que una comunicació ha finalitzat i el buffer receptor conté dades emmagatzemades. També es pot utilitzar aquest bit com interrupció en cas d'activar-se prèviament.
- **Bit 6 [TxC]** ***USART Transmit Complet flag*****:** S'activa quan totes les dades del buffer transmissor han estat enviades. També es pot utilitzar aquest bit com interrupció en cas d'activar-se prèviament.
- **Bit 5 [UDRE]** ***USART Data Register Empty*****:** Indica que el buffer transmissor està buit i llest per rebre noves dades.
- **Bit 4 [FE]** ***Frame Error*****:** S'activa si es detecta un error en el *frame.*
- **Bit 3 [DOR]** ***Data Overrun Error*****:** S'activa si es detecta un error de overrun. És a dir, que el buffer rebi més paraules d'espais del que disposa.
- **Bit 2 [PE]** ***Parity Error*****:** S'activa si es detecta un error de paritat.
- **Bit 1 [U2X]** ***Double Transmition Speed*****:** Només afecta a la comunicació asíncrona i activa la transmissió de doble velocitat quan es troba a nivell alt.
- **Bit 0 [MPCM]** ***Multi-Processor Communication Mode*****:** Activa la comunicació d'adreça, si el receptor rep un missatge que no contingui una adreça disponible a l'inici les dades seran ignorades.

4.2.3. Registre d'estat i control B (UCSRB)

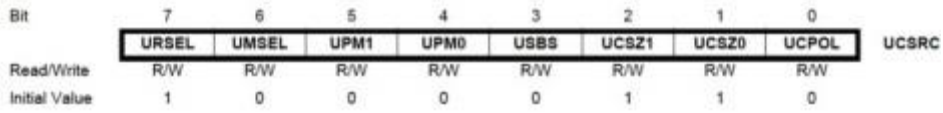


Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 4-4: Registre USART d'estat i control B (UCSRB) [14].

- **Bit 7 [RXCIE]** ***RX Complete Interrupt Enable*****:** Activa el mode d'interrupció RxC del registre UCSRA. S'activa posant el bit a nivell alt.
- **Bit 6 [TXCIE]** ***TX Complete Interrupt Enable*****:** Activa el mode d'interrupció TxC del registre UCSRA. S'activa posant el bit a nivell alt.
- **Bit 5 [UDRIE]** ***USART Data Register Empty Interrupt Enable*****:** s'ha de posar a nivell alt per activa les interrupcions que genera el bit UDRE del registre UCSRA.
- **Bit 4 [RXEN]** ***Receiver Enable*****:** S'escriu a nivell alt per activar el mòdul receptor.
- **Bit 3 [TXEN]** ***Transmitter Enable*****:** S'escriu a nivell alt per activar el mòdul transmissor.
- **Bit 2 [UCSZ2]** ***Character Size*****:** Juntament amb els bits de selecció de mida serveix per seleccionar el novè bit de dades. En cas que tingui valor de 1 es pot activar aquest bit si la combinació de l'altre registre ho permet.
- **Bit 1 [RXB8]** ***Receive Data Bit 8*****:** Serveix per a guardar el novè bit de dades rebut en cas que s'hagi activat prèviament.
- **Bit 0 [RXB8]** ***Transmit Data Bit 8*****:** Serveix per a guardar el novè bit de dades transmès en cas que s'hagi activat prèviament.

4.2.4. Registre d'estat i control C (UCSRB)



Bit	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	0	0	0	0	1	1	0	

Figura 4-5: Registre USART d'estat i control C (UCSRC) [14].

- **Bit 7 [URSEL]** ***USART Register Select*****:** Serveix per accedir al registre URSEL. Quan es llegeix aquest registre es posa el bit a '1'. Per poder escriure sobre aquest registre s'ha de tenir a nivell alt.
- **Bit 6 [UMSEL]** ***USART Mode Select*****:** Serviex per seleccionar el mode de comunicació.
  - **Síncrona:** Es posa el bit a '1'.
  - **Asíncrona:** Es posa el bit a '0'.

- **Bit 5:4 [UPM1:0] *Pari*tyBit:** Permet activar el bit de paritat, així com seleccionar el tipo.
  - **Paritat par:** "10".
  - **Paritat senar:** "11".
  - **Resservat:** "01".
  - **Desactivar paritat:** "11".
- **Bit 3 [USBS] *Stop Bit Select*:** Permet seleccionar el número de bits de parada.
  - **1 bit de parada:** "0".
  - **2 bits de parada:** "1".
- **Bit 2:1 [UCSZ1:0] *Character Size*:** Permet seleccionar la mida de les dades transmeses.
  - **5 bits de dades:** "00".
  - **6 bits de dades:** "01".
  - **7 bits de dades:**"10".
  - **8 bits de dades:** "11".
- **Bit 0 [UCPOL] *Clock Polarity*:** En la comunicació síncrona permet seleccionar si el bit que marca la recepció de dades serà a nivell alt o baix.

5. Disseny de l'estructura del mòdul USART

A continuació, es plantejarà i dissenyarà un mòdul USART. Durant el disseny es diferenciaran dos gran blocs: el transmissor i el receptor. Cada bloc tindrà els seu propi registre on emmagatzemarà estats de variables i senyals interns, així com un registre de control comú des d'on es podrà configurar els paràmetres de la comunicació desitjada.

5.1. Registres

5.1.1. Registre de control

Per a la realització del disseny s'han agafat les característiques més destacables dels registres esmentats a l'apartat 4.2 d'aquest treball i s'han implementat dins del disseny creat.

El registre més important és el que anomenem registre de control . Aquest és un registre d'entrada que conté part de la informació sobre les característiques de la comunicació. La següent taula conté la informació i l'estat dels bits d'aquest registre.

Taula 5-1: Descripció registre control.

Registre	Estat	Descripció
Bit reservat	0	-
	1	-
Funcionament [6]	0	Comunicació asíncrona
	1	Comunicació síncrona
Paritat [5-4]	00	Sense bit de paritat
	01	Sense bit de paritat
	10	Bit de paritat senar
	11	Bit de paritat parella
Bits parada [3]	0	1 bit de parada
	1	2 bits de parada

Mida de dades [2-1]	00	5 bits
	01	6 bits
	10	7 bits
	11	8 bits
clk_pol_aux [0]	0	Pujada
	1	Baixada

5.1.2. Registre d'estat

Aquest registre guarda bits sobre l'estat del sistema.

Taula 5-2: Descripció registre control.

Registre	Estat	Descripció
RX_EN [7]	0	El receptor pot rebre dades
	1	El receptor no pot rebre dades
U2X [6]	0	Velocitat normal
	1	Comunicació a doble velocitat
UCSZ2 [5]	0	No es treballa amb 9 bits de dades
	1	Es treballa amb 9 bits de dades
RXB8 [4]	0	Guarda l'estat del 9 bit del receptor
	1	
TXB8 [3]	0	Guarda l'estat del 9 bit del receptor
	1	

PE [2]	0	No es detecta error de paritat
	1	Es detecta error de paritat
FE [1]	0	No es detecta error de *frame*
	1	Es detecta error de *frame*
Reservat	0	-
	1	-

5.1.3. Registre de Baud Rate

Per simplificar el treball i el disseny, ja que no es disposen de més eines per poder seleccionar un registre molt complex, s'ha optat per un registre de 3 bits (8 possibles combinacions) amb el que seleccionar 8 de les velocitats més comunes en l'àmbit actualment. Per tant, simplement seleccionant l'estat del registre, es podrà seleccionar una de les velocitats disponibles.

A la següent taula es mostren els estats del registre i les velocitats equivalents:

Taula 5-3: Valors fixats pel registre Baud Rate

Estat	Velocitat
000	2400 Bauds
001	4800 Bauds
010	9600 Bauds
011	14400 Bauds
100	19200 Bauds
101	28800 Bauds
110	38400 Bauds
111	57600 Bauds

5.2. Estructura dels blocs

Per tal de tenir una millor organització i una programació més senzilla i estructurada que permetrà facilitar la posterior programació dels sistemes, s'ha dividit cada mòdul en diferents blocs que realitzen instruccions de manera independent.

Aquests blocs es comuniquen entre ells mitjançant senyals d'activació, permetent un disseny estructurat i guiat. Així, si un bloc requereix d'un senyal o registre que es genera en un altre, aquest no actuarà fins a rebre el senyal d'activació del seu predecessor.

A continuació es mostrarà l'estructura dels mòduls transmissor i receptor de manera global, separant cada bloc segons la funció específica que realitza. Un cop es tingui organitzat, programat i testejat cada uns dels blocs de manera separada del sistema global, es realitzarà un *Port Map* per tal de tenir un complet e integrat dels mòduls.

Dins aquest capítol també es comentarà la funció que tindrà cada un dels blocs i com interactuarà amb la resta de blocs dins d'un sistema. Per tant, s'han creat unes taules que expressen els senyal d'entrada i sortida que conté cada un dels blocs i si aquestes interaccionen amb algun altre bloc.

Per tal de seleccionar la velocitat, es permet modificar un registre d'entrada de 3 bits per ajustar fins a 8 velocitats diferents. Mitjançant aquest registre es seleccionarà un registre intern que contindrà els valors necessaris per fer el comptador del *prescaler*. Per tant, els càlculs necessaris per poder ajustar el senyal de rellotge del nostre sistema s'han realitzat prèviament i han estat introduïts al disseny.

5.2.1. Registre de Baud Rate

El funcionament per ajustar la velocitat dins d'un microprocessador és l'ús d'un senyal de *prescaler*. Aquest s'encarrega de marcar una base temporal, sempre més lenta que la original, a la que actuarà el sistema. Mitjançant un comptador que augmenta el seu valor en 1 per cada senyal de rellotge que efectua el processadorm es comprova si aquest valor és igual al carregat amb el registre de Baud. Si aquests dos coincideixen, s'activa el senyal de *prescaler* a nivell alt durant un cicle de rellotge. Com es pot observar, el registre ha estat calculat i dividit per 16. És a dir, aquest senyal de *prescaler* s'haurà d'activar 16 vegades per obtenir la veritable marca temporal a la que es comunicarà el sistema.

A continuació es mostren les velocitats disponibles així com l'estat del registre per poder seleccionar-les. També es mostra el valor del registre intern que servirà per comprar el comptador.

Taula 5-4: Valor del Registre Baud Rate per una freqüència de rellotge de 50 MHz

Estat	Valor	Valor registre (decimal)	Valor registre (binari)
000	2400 bps	1302	10100010110
001	4800 bps	651	01010001011
010	9600 bps	325	00101000101
011	14400 bps	217	00011011001
100	19200 bps	162	00010100010
101	28800 bps	108	00001101100
110	38400 bps	81	00001010001
111	57600 bps	54	00000110110

Durant la comunicació USART, es pot activar un bit que permet augmentar la velocitat de comunicació el doble. Per implementar aquets disseny, s'ha agafat el mateix comptador de 16 que es té per la comunicació normal i es redueix el seu límit fins a 8. D'aquesta manera, es genera un senyal de *prescaler* el doble de ràpid permetent una transmissió a doble velocitat.

El bloc de Baud Rate s'implementa dins dels blocs de *TSR* i *Sampling* per els mòduls de transmissor I receptor respectivament.

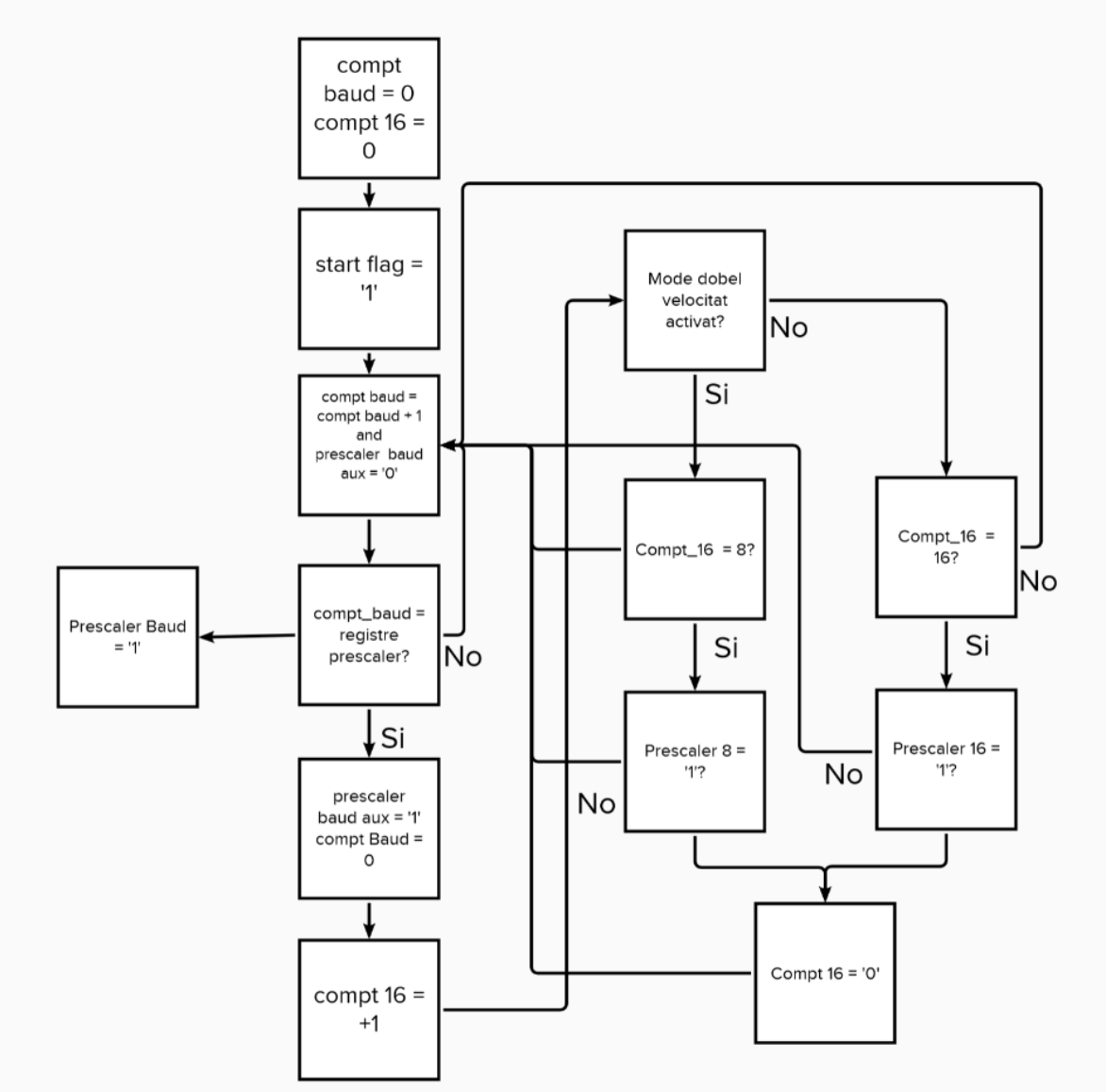


Figura 5-1: Disseny del bloc de Baud Rate.

5.3. Transmissor

El bloc transmissor consta de 4 entitats que actuen en sèrie, és a dir, per a què un bloc executi la seva funció, ha de rebre un senyal del bloc anterior informant de que aquest ha executat la seva funció de forma correcte i per tant el següent bloc pot començar. D'aquesta manera, es té un sistema complex en que cada un dels elements treballa amb sincronia amb la resta.

Aquesta forma de funcionar s'ha pensat ja que en VHDL, com s'ha comentat anteriorment, el codi s'executa de forma constant i per tant, servirà perquè el sistema pugui diferenciar cadascun dels canvis. Aquest fet es pot veure més clar amb el següent cas:

El bloc de paritat, per exemple, calcula el nombre de 1's que té la paraula i crea el bit de paritat. Per a què aquest bloc sàpiga quan arriba un nou Byte, juntament amb aquest s'enviarà un senyal d'activació durant un cicle de rellotge que informa al bloc de Paritat que pot començar a executar la seva funció.

Aquets mòdul enviarà senyals externs mitjançant els registres comentats anteriorment.

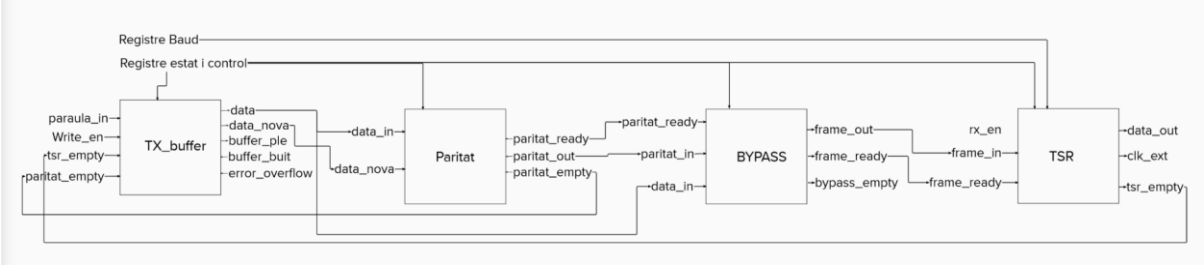


Figura 5-2:Diagrama de blocs general del mòdul transmissor.

Si es transmet un impuls a nivell alt a través del port de *Write Enable,* volent dir que es vol escriure dins del mòdul, es guardarà el byte (8 bits) dins el buffer intern que es trobi al port d'entrada en el moment de rebre aquest impuls. En cas que el missatge anterior hagi estat enviat sense cap interferència o error, s'enviarà un senyal intern informant al buffer de que els sistema esta buit i pot enviar la següent paraula. Un cop el buffer enviï el Byte al següent bloc, es calcularà la paritat en funció de les opcions configurades i es crearà el *frame* sencer que serà enviat al receptor. Finalment, s'utilitzarà el registre Baud Rate per a generar els impulsos que marcaran els períodes temporals i s'enviaran els bits en sèrie a la velocitat configurada. En cas de tenir una transmissió síncrona, aquest impuls generat amb el registre de Baud Rate serà també transmès a través de la línia *clk_ext.*

5.3.1. Buffer transmissor

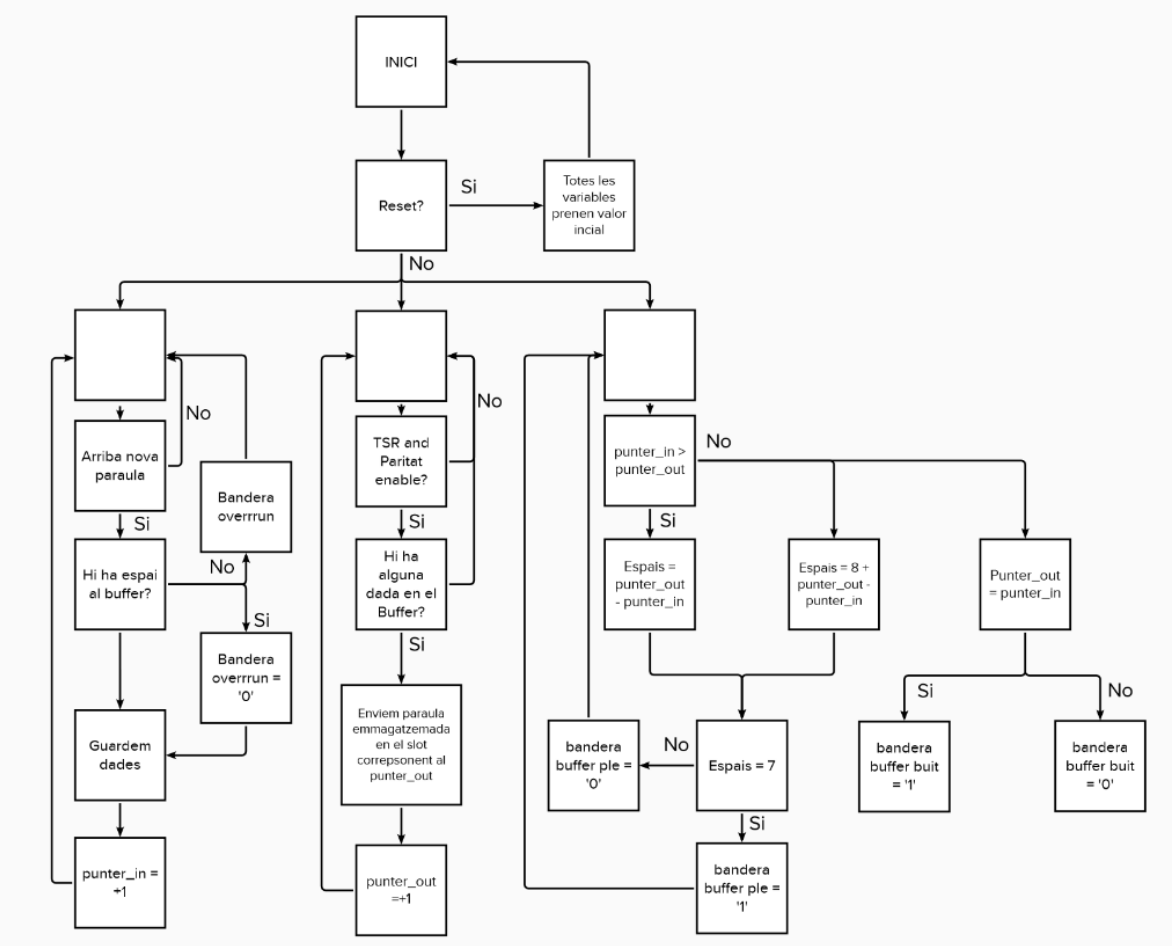


Figura 5-3: Disseny del buffer del mòdul transmissor.

Les dades a enviar són emmagatzemades al buffer transmissor fins que el receptor dóna el senyal de que esta llest per rebre-les. El disseny d'aquest buffer és un estil FIFO *(First In First Out*), és a dir, les paraules seran llegides amb el mateix ordre en el que siguin escrites. Aquestes paraules de sortida aniran al bloc de paritat juntament amb un impuls de nivell alt que indicarà que una nova paraula s'ha d'enviar. S'ha escollit aquest estil de buffer perquè en una transmissió és important mantenir l'orde en que s'envien les dades.

Per a què el buffer llegeixi i guardi les dades, un impuls a nivell alt ha de ser rebut a través port de *write enable* juntament amb les dades que vulguin ser guardades. Per altra banda, pera què el buffer enviï aquesta paraula a través de tot el sistema, aquest ha de rebre els senyals conforme el TSR i el bloc de paritat estan buits i disponibles per actuar.

Pel disseny que es preté, que és un prototip, es dissenyarà un buffer amb 8 espais per guardar 1 byte a cada un.

La següent taula recull les entrades i sortides d'aquest bloc:

Taula 5-5: Entrades i sortides del bloc Buffer Transmissor

I/O	Senyal	Origen/Destinació	Descripció
IN	TSR buit	TSR/Buffer	Indica que el bloc TSR està buit
IN	Paritat buit	Paritat/Buffer	Indica que el bloc de generar la paritat està buit
IN	Dades entrada	Sistema/Buffer	Byte rebut del sistema extern
IN	*Write enable*	Sistema/Buffer	Impuls per indicar al buffer que ha de llegir un nou byte
OUT	Dades sortida	Buffer /Paritat	Byte enviat pel buffer
OUT	Nova paraula	Buffer/Paritat	Impuls per indicar que entra un nou byte al següent bloc
OUT	Buffer ple	Buffer/Sistema	Indica que el buffer no té mes caselles de memòria disponibles
OUT	Buffer buit	Buffer/Sistema	EL buffer no te dades per transmetre
OUT	*Overflow*	Buffer/Sistema	Bandera que indica que s'ha intentat carregar una nova paraula al buffer quan aquest estava ple

El disseny pensat del *buffer* consta de 3 processos, un per llegir noves dades, un per enviar bytes i un altre que mitjançant operacions aritmètiques calcula el nombre de espais disponibles a la memòria.

El sistema pel que s'ha optat es la utilització de dos punters, un per registrar l'entrada de dades, i un altre que en registra la sortida. Aquests punters serviran per marcar la casella sobre la que ha d'actuar cada un dels dos processos.

Rebre dades

Per tal de poder escriure Bytes dins del buffer, per posteriorment ser transmesos al receptor, s'ha de rebre un bit a nivell alt que en permeti l'escriptura. Quan es rep aquest bit, el buffer llegirà els bits disponibles als ports d'entrada i les emmagatzemarà a la casella corresponent. En cas que el buffer estigui ple, per molt que aquest bit d'entrada estigui a '1', s'ha dissenyat per a què no admeti més paraules i sobreescrigui les que ja estan guardades. Si això passés, s'activaria la bandera de *overrun*, que indica que s'està introduint una paraula al transmissor que no podrà ser guardada.

Enviar dades

Per a què el buffer pugui enviar les dades, primer s'ha de comprovar que el buffer conté alguna dada, i per tant que el transmissor té dades que transmetre. Si es dóna aquest cas, el transmissor esperarà rebre el bits que indiquen que el bloc que calcula la paritat i el TSR estan buits i llestos per rebre noves dades. En aquest moment, i comprovant que el receptor està disponible per acceptar dades, el buffer enviarà el Byte guardat a la casella que marca el punter de sortida corresponent. Juntament amb les dades, s'envia també un senyal d'activació al següent bloc.

Càlcul dels espais disponibles

Per tant, per saber la el número de espais restants al buffer només s'ha de restar el punter de sortida amb el d'entrada. El principal problema és que es tracta d'un buffer circular, i per tant, es pot donar la situació en que el punter de sortida sigui menor que el de entrada. Si això passa, el resultat de l'operació no seria vàlid. La solució es realitzar abans de l'operació una comprovació respecte els dos punters i en funció d'això utilitzar una fórmula o una altre.

- **Punter entrada > Punter sortida:**

$$Espais\ disponibles = Punter\ entrada - \ Punter\ sortida \quad \textbf{(Eq. 5-1)}$$

- **Punter entrada < Punter sortida:**

$$Espais\ disponibles = (Punter\ entrada - \ Punter\ sortida) + N_t\ \ espais \quad \textbf{(Eq. 5-2)}$$

- **Punter entrada = Punter sortida:** Buffer buit.

En cas que el punter de sortida sigui inferior, es suma el nombre total de caselles de les que disposa el buffer per tal de compensar el valor final i rebre un resultat correcte.

A través d'aquest punters podem saber dues coses primordials i bàsiques dins del sistema:

- **Buffer buit:** El buffer estarà buit en quan els dos punters es trobin a la mateixa casella. Això implicarà que s'activarà una bandera perquè el receptor no demani dades.
- **Buffer ple:** El buffer estarà ple en quan la resta dels dos punters sigui el número màxim de "slots" - 1 . Si això passa, s'enviarà una bandera que indicarà que el mòdul transmissor no admet més dades 'dentrada.

Aquests dos senyals seran enviats als registres d'estats, juntament amb el senyal de *overrun*.

5.3.2. Calcular paritat

En quant aquest bloc rep el senyal a nivell alt de que noves dades estan sent enviades, agafa les dades que hi ha al port d'entrada i les guarda en un registre. Això es fa per si en algun moment del càlcul de la paritat les dades canvien, poder tenir un registre guardat amb els bits correctes.

El disseny implementat agafa aquest registre i comprova el nombre de 1's que conté guardant el resultat en un registre intern. Per no ocupar molt espai aquets comptador és de 2 bits, i en cas que s'arribi al seu màxim automàticament torna al seu valor inicial i pot seguir sumant. Aquest mètode s'ha escollit ja que en el càlcul de la paritat, només interessa saber si el nombre d'1' és parell o senar, per tant, un comptador de dos bits és suficient.

Per tal de saber quan ha de parar d'analitzar (un cop es recorren tots els bits de dades), es fa un comptador que augmenta després de cada bit analitzat, i en quant aquest té el mateix valor que la mida de les dades, el procés s'atura. S'ha de fer així ja que la mida de les dades és variable.

El següent procés del disseny és analitzar el resultat obtingut del comptador de paritat. Per tant, s'agafa el tipus de paritat configurada del registre de control i, en funció del seu estat, es completa el bit amb un '1' o un '0'. Per exemple, si es treballa amb una paritat senar i el nombre d'uns del comptador és senar, el bit de paritat es posa a 0. Per altre banda, si en aquest mateix cas es tingui una paritat parella, el bit de paritat prendria un valor de '1'.

Un cop calculat aquest bit, s'envia juntament amb un bit d'activació al següent bloc.

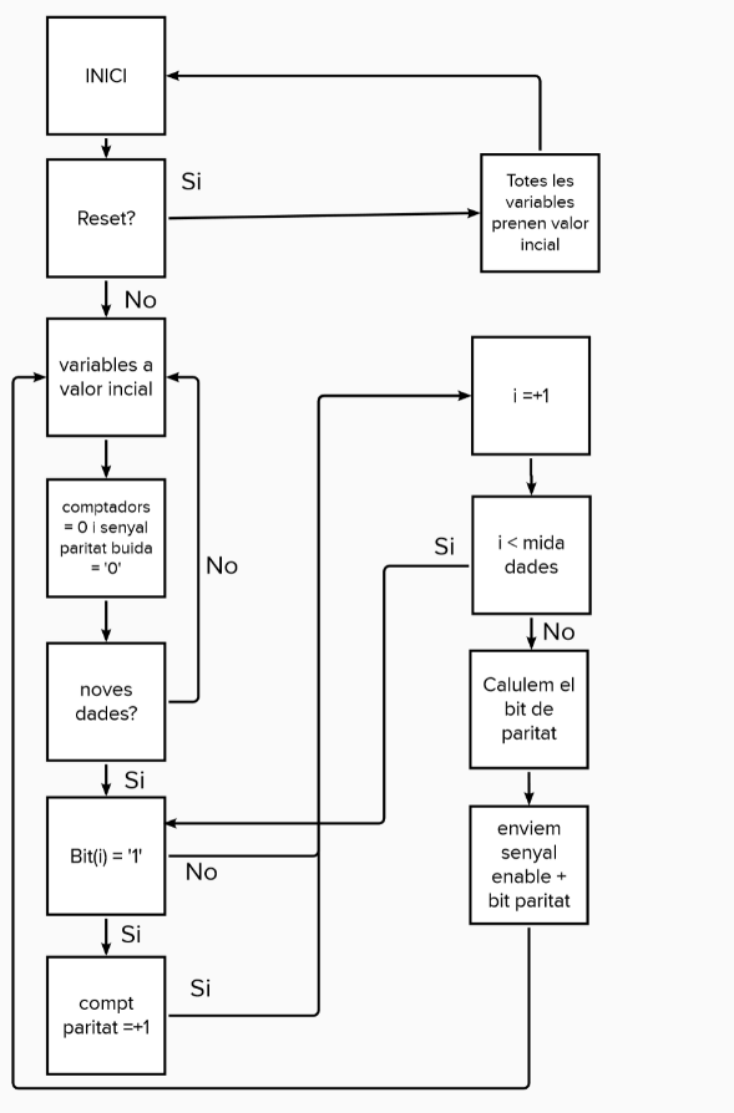


Figura 5-4: Disseny del bloc de càlcul de paritat.

La següent taula recull les entrades i sortides del bloc de paritat:

Taula 5-6: Entrades i sortides del bloc de càlcul de paritat.

I/O	Senyal	Origen/Destinació	Descripció
IN	Paritat	Sistema/Paritat	Indica el tipus de paritat utilitzada
IN	Mida de les dades	Sistema/Paritat	Indica la mida de les dades

IN	Dades entrada	Buffer transmissor /Paritat	Byte rebut
OUT	Bit de paritat	Paritat/Generar *frame* complert	S'envia el bit de paritat generat
OUT	Paritat calculada	Paritat/Generar *frame* complert	Impuls que indica que el bit de paritat s'ha generat correctament
OUT	Bloc buit	Paritat/Buffer transmissor	El bloc està buit i llest per rebre el següent byte

5.3.3. Generar *frame* complert

Un cop el bit de paritat ha estat calculat i generat, s'envia al bloc de generació del *frame* complert, juntament amb el senyal que indica que la paritat ja està llesta.

Un cop el bloc rep aquest senyal d'activació del bloc de paritat, s'agafa la informació sobre la mida de les dades, si la paritat està activada i el nombre de bits de parada, i es crea el registre que contindrà tots els bits que seran enviats. Un cop s'ha creat el registre, aquest serà enviat juntament amb una senyal d'activació al bloc TSR.

En la comunicació USART els bits de dades s'envien de LSB a MSB, per tant aquest bloc també realitza la funció d'inversió del registre de dades. Aquesta inversió es realitza amb dos comptadors, un que comença a valor 7 i l'altre a 0. Aquests comptadors s'utlitzen per invertir els bits del registre de dades.

Aquesta funció es realitza una única vegada abans de incloure el registre de dades dins la cadena del *frame* complert.

La següent taula recull les entrades i sortides d'aquest bloc:

Taula 5-7: Entrades i sortides del bloc de generació del frame complert.

I/O	Senyal	Origen/Destinació	Descripció
IN	Paritat	Sistema/Generar *frame* complert	Indica el tipus de paritat utilitzada

IN	Mida de les dades	Sistema/Generar *frame* complert	Indica la mida de les dades
IN	Bits de parada	Sistema/Generar *frame* complert	Indica el número de bits de parada que conté la cadena de dsdes
IN	Dades entrada	Paritat/Generar *frame* complert	Byte rebut
IN	Bit de paritat	Paritat/Generar *frame* complert	Es rep el bit de paritat generat
IN	Paritat calculada	Paritat/Generar *frame* complert	Impuls que indica que es pot generar el nou *frame*
IN	UCSZ2	Sistema/Generar *frame* complert	Ativa la comunicació amb 9 bits de dades
IN	TXB8	Sistema/Generar *frame* complert	Valor del 9 bit de dades
OUT	Bloc buit	Generar *frame* complert/Paritat	El bloc està buit i llest per rebre el següent byte
OUT	*Frame*	Generar *frame* complert/TSR	El *frame* complert que serà enviat al receptor
OUT	*Frame* generat	Generar *frame* complert/TSR	Impuls per indicar que el *frame* s'ha generat correctament

El *frame* generat pot tenir una mida d'entre 7 a 12 bits en funció de les possibles configuracions disponibles. En el cas més reduit, es tindrà una comunicació de 5 bits de dades, 1 bit d'inici i 1 bit de parada sense bit de paritat. Pel contrari, el cas en que tindrem 12 bits serà si es transmeten 8 bits de dades, 1 bit d'inici, 1 bit de paritat i 2 bits de parades.

Ja que la mida és variable, s'ha generat un registre de 12 bits (el màxim possible) i es van omplint els bits de dreta a esquerra en funció de la informació disponible al registre de control. En cas que el nombre de bits útils per la comunicació sigui inferior a 12, es deixaran els bits sobrant a en un estat de '1'.

Per exemple, si tenim el *frame* de 12 bits "0 + 10101010 + 0 + 11" però es vol transmetre amb 1 bit de parada i sense bit de paritat, es mouran els bits a la següent posició "00 + 0 + 10101010 + 1". D'aquesta manera s'enviaran només 10 bits en compte de 12 i es guanyarà velocitat de transmissió.

En cas de tenir el novè bit de dades activat, aquest s'agafarà del registre d'estat corresponent i s'implementarà de la mateixa forma que es faria amb el bit de paritat.

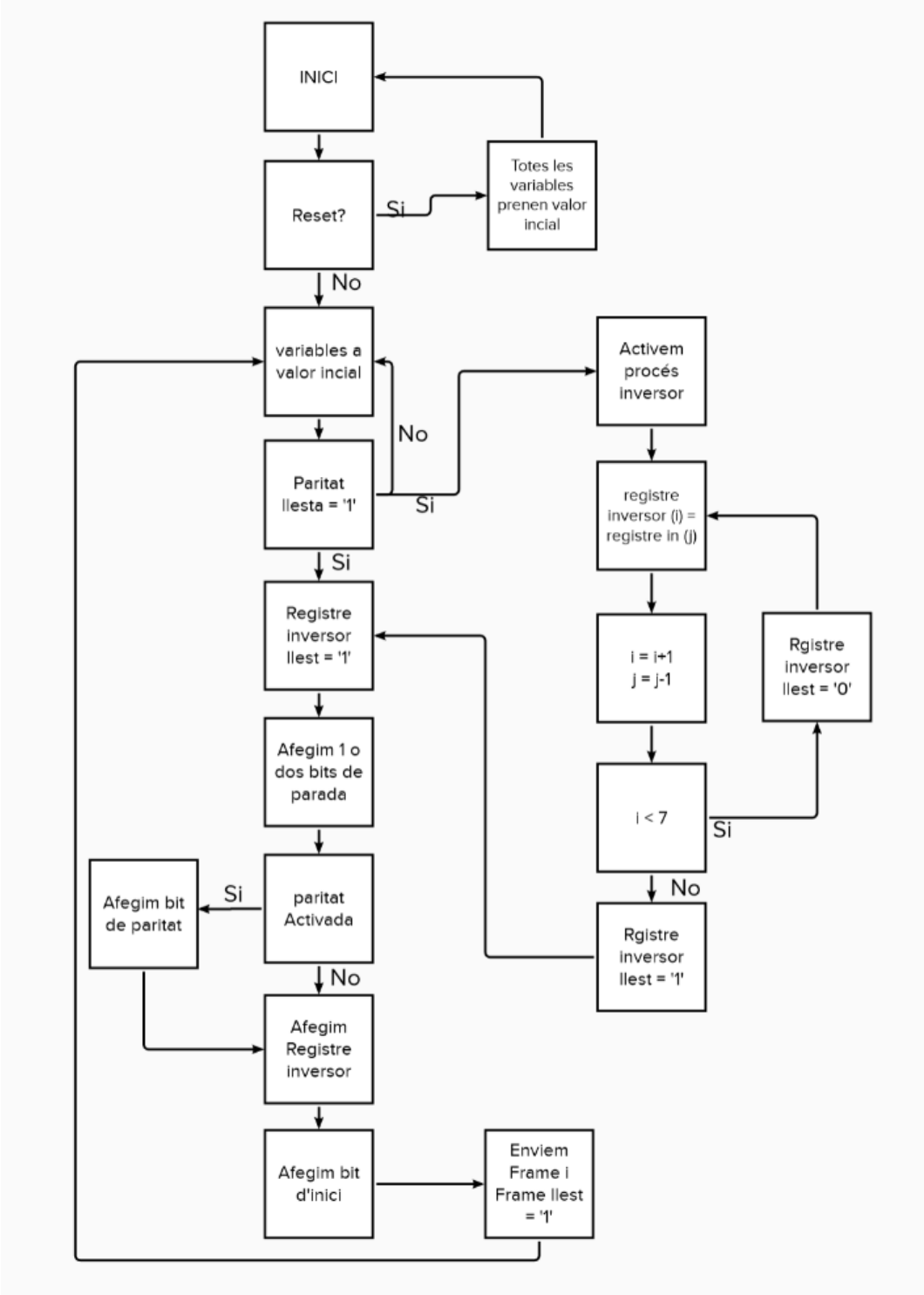


Figura 5-5: Disseny del bloc de generació del frame complert.

5.3.4. TSR (Transmitter Shit Register)

L'últim bloc del mòdul transmissor és el TSR (*Transmitter Shift Register*), que s'encarrega de transmetre la cadena de bits al mòdul receptor. Com ja s'ha comentat, el mode de comunicació USART és un mètode de comunicació sèrie, per tant, aquest bloc fa la funció de convertidor paral·lel/sèrie, doncs fins ara les dades es transmetien d'un bloc a un altre en forma de registres paral·lels.

Aquesta transmissió es pot donar de forma síncrona o asíncrona, modes que fins ara, pels blocs anteriors, eren igual ja que les funcions realitzades per aquests eren independents per un o altre tipus de comunicació. En cas de tenir una comunicació síncrona, juntament amb les dades s'enviarà el senyal d'impulsos generat pel bloc de Baud Rate; si es té una comunicació asíncrona, només s'enviaran els bits de dades i es mantindrà el senyal de rellotge a nivell baix.

Així doncs, quan es rebi el senyal d' activació del sistema receptor, i el senyal de que el *frame* ha estat generat correctament, es calcularà el nombre de bits útils que conté el *frame*. Els bits estan ubicats dins el registre de manera que el bit de parada es troba a la posició 0 i el bit d'inici és variable en funció dels paràmetres de configuració. Per tant, abans de transmetre les dades, s'analitza de MSB a LSB fins a trobar el primer '0' del *frame*, posició que marcarà l'inici dels bits de sortida.

Fent aquesta última funció (detectar el primer '0'), es redueix el temps de comunicació en cas que la cadena de bits transmesa sigui inferior a 12.

En quant el receptor enviï el senyal de que està preparat per rebre, s'activa el comptador de Baud Rate i s'envia bit per bit la cadena del *frame* cada cop que el senyal de *prescaler* s'activa. Aquest procés es repetirà fins assolir la mida màxima del *frame* generat.

La següent taula recull les entrades i sortides d'aquest bloc:

Taula 5-8: Entrades i sortides del bloc TSR.

I/O	Senyal	Origen/Destinació	Descripció
IN	Paritat	Sistema/TSR	Indica el tipus de paritat utilitzada
IN	Mida de les dades	Sistema/TSR	Indica la mida de les dades
IN	Bits de parada	Sistema/TSR	Indica el número de bits de parada que conté la cadena de dsdes

IN	Funcionament	Sistema/TSR	Indica si es treballa amb comunicació síncrona o asíncrona
IN	Registre Baud	Sistema/TSR	Registre per ajustar la velocitat de comunicació
IN	*Frame* complert	Generar *frame* complert/TSR	*Frame* complert per enviar
IN	*Frame* generat	Generar *frame* complert/TSR	Impuls que indica que s'ha generat el *frame* correctament
IN	RX *Enable*	Receptor/TSR	Bit que indica que el receptor està llest per a rebre les dades
IN	UCSZ2	Sistema/TSR	Activa la comunicació amb 9 bits de dades
IN	U2X	Sistema/Generar *frame* complert	Activa transmissió a doble velocitat de Baud Rate
OUT	Senyal rellotge	TSR/Receptor	Senyal de rellotge generat pel transmissor per les comunicacions síncrones
OUT	TSR buit	TSR/Buffer	Bit enviat al buffer que indica que el TSR pot iniciar una nova transmissió
OUT	Data out	TSR/Receptor	Bit sèrie enviat al receptor

Un cop ha finalitzat la comunicació i el registre de dades del TSR s'ha buidat, s'activa el senyal corresponent que indica que ja està llest per la següent transmissió. Durant la comunicació, aquest senyal es mantindrà a nivell baix, indicant que no està buit i està realitzant una comunicació o bé té una cadena de bits carregada esperant a ser tramesa.

El disseny que s'ha implementat, permet enviar un senyal conforme el TSR està buit i per tant que el buffer enviï un byte i aquest quedi carregat al registre del TSR, però no sigui enviat fins rebre el senyal

de que el receptor està llest per a rebre dades. Amb aquest disseny es permet una execució de 1 byte extra, ja que disposa dels 8 propis del buffer, més aquest que pot esperar carregat al TSR.

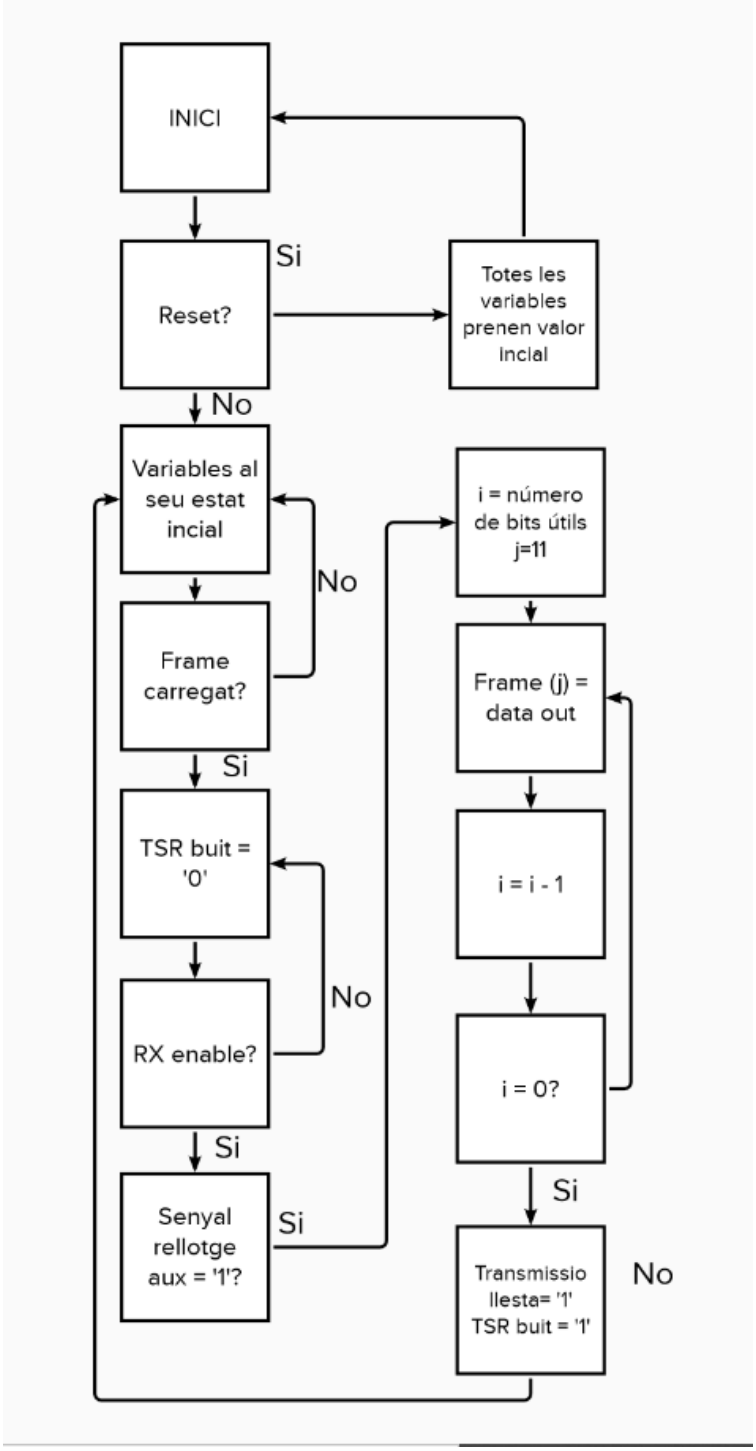


Figura 5-6: Disseny del bloc TSR.

5.3.5. Transmissor complert

Un cop s'ha realitzat el disseny dels 4 blocs que conformen el mòdul transmissor, es realitza un disseny que inclogui tots 4 i les seves interaccions. Aquest disseny es realitza mitjançant un Port Map, que permet connectar entrades i sortides de variïs dissenys.

Així doncs, s'hauran de connectar les sortides d'un bloc amb les entrades del següent, en cas que una sigui la mateixa que l'altre, i les sortides externes que es treuen d'alguns blocs col·locar-les als registres corresponents.

Aquests senyals que van connectats entre si es poden veure a les taules anteriors de cada un dels blocs.

5.4. Receptor

EL bloc receptor està format per 4 mòduls que actuen en sèrie i paral·lel. En el cas d'actuar en sèrie, per a què un bloc executi la seva funció, ha de rebre un senyal del bloc anterior per a que pugui començar a executar les seves accions. En el cas de tenir blocs paral·lel, permet l'execució de tots dos dissenys de manera simultània.

Igual que passa amb el mòdul transmissor, s'ha dissenyat el sistema de manera que per a què un bloc executi la seva funció, abans ha de rebre un bit de confirmació del bloc anterior. En el cas dels dos blocs que treballen en paral·lel, tots dos reben el mateix bloc del sistema de mostreig.

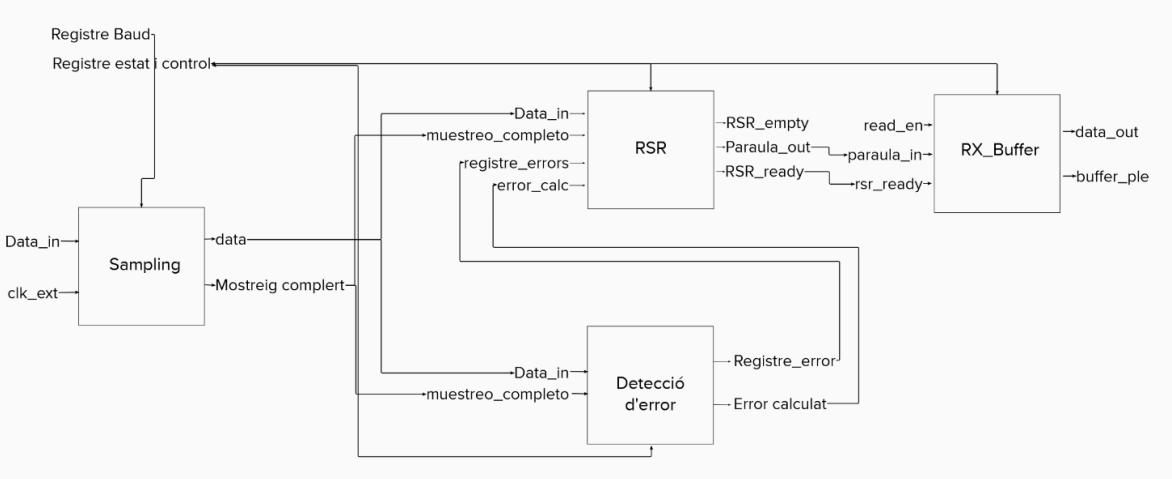


Figura 5-7: Diagrama de blocs general del mòdul receptor.

El bloc de *Sampling* rep les dades en sèrie del transmissor i les guarda en un registre de 12 bits (mida màxima que pot tenir un missatge). Un cop s'han afegit tots els bits al registre, s'envia un senyal als

blocs *RSR* i detecció d'error per avisar que el registre està complert. El missatge està format per la següent informació.

- **Missatge:** De 5 a 8 bits que contenen la informació que es vol transmetre.
- **Errors:** Mètodes de detecció d'errors i els bits que requereixen.
- **Bit d'inici i parada:** per detectar quan es comença i es para la comunicació.

Per tant, els blocs *RSR* i *detecció d'error* s'encarreguen d'analitzar i tractar el missatge rebut. Per una banda, com bé diu el seu nom, el bloc de detecció d'error s'encarrega de comprovar el bit de paritat rebut i els bits de parada i en cas de detectar algun error, avisar al sistema. El bloc de *RSR* s'encarrega de separar els bits de dades del *frame* de 12 bits generat pel bloc de *sampling* i invertir el bits, ja que es reben de LSB a MSB. Un cop feta la inversió es comprova si la comunicació disposa de un novè bit d'adreça i en cas afirmatiu, es guarda aquest en el registre d'estat.

En el cas que algun error sigui detectat, es pararà la comunicació i el byte rebut serà descartat. Així, s'activaran les banderes d'error que indicaran el tipus d'error i serà necessari activar un *reset* del sistema per tal de tornar a posar en marxa una comunicació. Cal tenir en compte que els bits de dades s'envien de LSB a MSB.

A la figura anterior, es pot veure els diferents components que formen el bloc complert i com interactuen uns amb altres. El bloc de RSR buit indica al transmissor que el següent missatge pot ser enviat, ja que el sistema receptor no està treballant amb cap tipus de dada.

El funcionament d'interconnexió dels diferents components del mòdul receptor s'ha dissenyat de manera que només s'hagi de treballar amb el registre de Baud Rate, en cas de tenir comunicació asíncrona, en el primer component. Per tant, aquets agafarà les dades en funció del senyal de rellotge generat pel registre i les pesarà juntament amb un senyal d'avís. Aquest funcionament és el mateix que el que es feia servir pel mòdul transmissor.

El registre amb el que es treballa en tots els blocs està pensat perquè en cas de tenir una comunicació amb menys de 12 bits d'importància, es deixin els LSB a valor alt. Un cop dins de cada bloc, es recorre aquest registre en funció dels paràmetres establerts al registre de control per tal de saber quins són important i quins no.

5.4.1. Mostreig de dades

Igual que passava amb el bloc TSR del transmissor, aquest és l'únic bloc del mòdul que es veurà afectat pel mode de comunicació, ja que és el primer bloc i el que treballava en conjunt amb el mòdul transmissor. Per tant en funció del mode de comunicació escollit, es tindran les següents opcions:

- **Comunicació síncrona:** S'utilitzarà el senyal generat pel transmissor per tal de sincronitzar tots dos mòduls. Aquest senyal es farà servir per saber en quin moment es rep cada un dels bits enviats.
- **Comunicació asíncrona:** S'utilitzarà el registre de Baud Rate, que en funció de la velocitat de rellotge del sistema receptor i la velocitat de transmissió desitjada , tindrà un valor o un altre. Així, quan aquest arribi al seu valor màxim es generarà el senyal de "enable" que permetrà llegit el bit que contingui el port d'entrada de dades.

Quan es treballa de manera asíncrona, es poden donar errors al llegir el bit. Això es deu a que es fa servir un registre i un comptador que regula la velocitat del sistema, però aquest registre no es precís al 100%. Entre això i possibles errors degut a la diferència de la velocitats de tots dos sistemes, es pot donar el cas que es llegeixi un bit dues vegades o es salti un bit per llegir si el senyal d'*enable* es dóna just en el moment en que es canvia una dada.

Per a evitar que es produeixen aquest tipus d'errors, aquest bloc s'encarrega de fer un recompte en el que analitza 16 vegades el bit rebut i analitza quin ha sigut el resultat més popular i assigna aquest com el correcte.

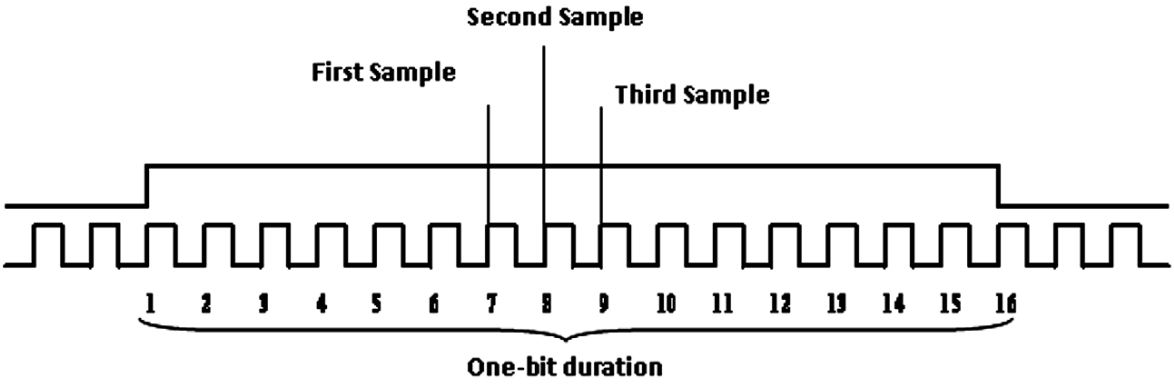


Figura 5-8: Funcionament del mostreig de dades [8].

Per a poder fer aquest mostreig, es divideix el registre de Baud Rate entre 16 i s'analitza el estat del bit rebut en 16 ocasions. Així el valor que s'obtingui més vegades serà el correcte. A la figura 5-8 es pot

observar el funcionament d'aquest sistema. El bit rebut té el valor de '1' i es fa un recompte de 16 vegades on s'analitza aquest bit. Es pot veure que en totes les mostres s'agafa el valor correcte del bit excepte en l'ultima (16) que el valor ja ha canviat a nivell baix.

En cas de tenir una comunicació síncrona, no és necessari fer el mostreig ja que el mateix mòdul transmissor diu quan s'ha de llegir el bit d'entrada.

Un cop realitzat el mostreig de cada bit, aquest es guarda en un registre format de 12 bits que guardarà la informació del missatge transmès. En cas que la transmissió disposi de menys de 12 bits, els LSB, es deixaran a nivell alt.

La següent taula recull les entrades i sortides d'aquest bloc:

Taula 5-9: Entrades i sortides del bloc de mostreig.

I/O	Senyal	Origen/Destinació	Descripció
IN	Funcionament	Sistema/Mostreig	Indica si es treballa amb comunicació síncrona o asíncrona
IN	Senyal de rellotge	TSR/Mostreig	Senyal de rellotge utilitzat en el mode síncron
IN	Paritat	Sistema/Mostreig	Indica el tipus de paritat utilitzada
IN	Mida de les dades	Sistema/Mostreig	Indica la mida de les dades
IN	Bits de parada	Sistema/Mostreig	Indica el número de bits de parada que conté la cadena de dsdes
IN	Registre Baud	Sistema/Mostreig	Registre per ajustar la velocitat de comunicació
IN	UCSZ2	Sistema/TSR	Activa la comunicació amb 9 bits de dades
IN	U2X	Sistema/TSR	Activa transmissió a doble velocitat de Baud Rate
IN	Data in	TSR/TSR	Dades sèrie rebudes

IN	Data out	Mostreig/RSR i Detecció d'error	Registre amb els 12 bits transmesos
IN	Mostreig complert	Mostreig/RSR i Detecció d'error	Bit que indica que el registre de 12 bits està generat correctament

El disseny final pel que s'ha optat és realitzar una màquina d'estats en la que és diferencien els dos modes de comunicació (síncron o asíncron). Un cop es seleccioni una de les dues modalitats, si es vol canviar a l'altre, s'haurà de fer un *reset* del sistema.

Dins de cada una de les dues opcions, es reben les dades i un cop mostrejades es guarda el bit resultant en un registre de *frame*. En cas de tenir una comunicació que contingui menys de 12 bits, es deixaran els LSB a valor alt.

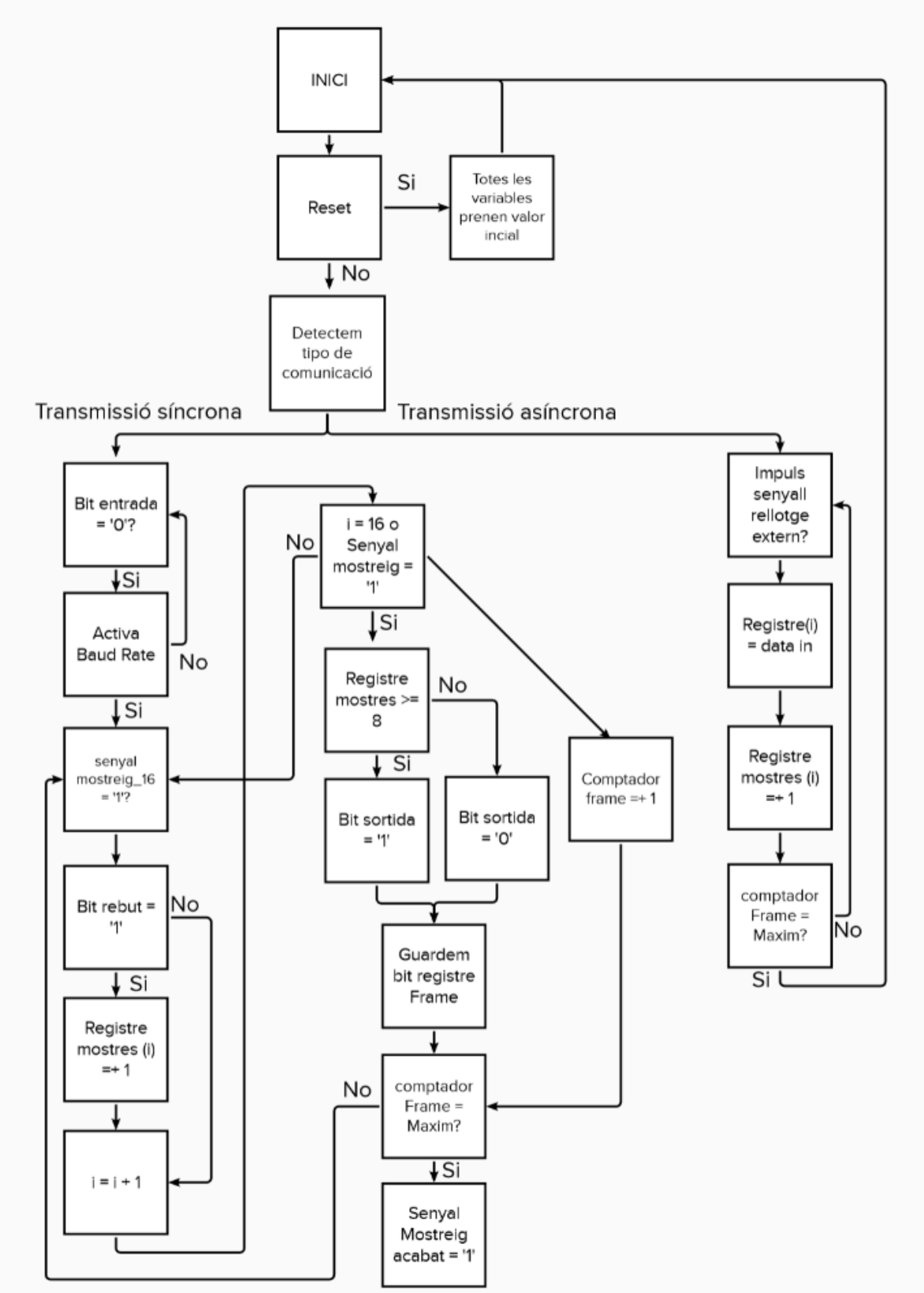


Figura 5-9: Disseny del bloc de mostreig de dades.

5.4.2. Detecció d'errors

En quant arriba dl senyal avisant de que el registre generat pel bloc S*ampling* s'ha completat, s'activa aquest bloc. Com que en una comunicació *USART* la mida del missatge és variable, es treballa amb un comptador que suma els bits del registre i els analitza en funció de la configuració establerta. Es pot donar el cas que es tinguin més o menys bits de dades, es tingui o no un bit de paritat o es tingui 1 o 2 bits de parada. Per això s'ha dissenyat de manera que el comptador analitzi bit a bit el registre donant les següent opcions:

Mida de les dades (de 5 a 8 bits)

En el que els primers bits que formes la comunicació són el d'inici i els bits de dades que porten la informació. Per tant, aquest comptador té un límit variable que es modifica en funció de les dades establertes al registre de control.

En cas que la comunicació disposi de bit de paritat, s'analitzaran els bits de dades i es generarà un comptador que augmentarà per cada valor de '1' que trobi. D'aquesta manera, un cop s'hagi analitzat el registre sencer, aquest comptador servirà per generar el bit de paritat pel receptor. Si no es disposa de bit de paritat, s'augmentarà el comptador fins passar aquests bits.

Bit de paritat o novè bit

Un cop s'han rebut les dades, es comprova si la comunicació disposa o no d'un bit de paritat. En cas que sigui afirmatiu, aquest va després de les dades i per tant es guarda en una senyal per poder ser tractat posteriorment.

En cas que la transmissió disposi d'un novè bit de dades, aquest ocupa la posició de la paritat, i s'ubica dins la mateixa posició dins del registre. Si això passa, s'actua de la mateixa forma i s'augmenta el valor del comptador. Si aquets bit no es tingués en compte, es detectaria que no hi ha paritat i el comptador es quedaria un bit atraçat. Aquest novè bit no serà tractat en aquest bloc, sinó que es fa en el de *RSR.* Cal dir que el sistema ha estat dissenyat per a què executi la mateixa funció de guardar el bit rebut en un senyal intern, independentment de si es tracta d'un bit de paritat o un novè bit de dades. Això s'ha realitzat per estalviar línies de codi i possibles complicacions, doncs més endavant ja es té en compte.

En cas que no es disposi de bit de paritat o novè bit de dades, aquest bloc es saltaria sense augmentar el valor del comptador.

Bits de parada

Seguint el comptador, els següents bits que analitzaria el sistema seran els de parada. Per tant, en funció si es té 1 o 2, es comprovarà si tenen un valor alt i si es detecta que això no és així, s'activarà la bandera d'Error de *Frame*.

Com s'ha comentat abans, en cas de tenir una comunicació que requereixi menys de 12 bits, no resultarà en problemes, ja que el comptador està dissenyat perquè tingui aquest valor en compte i es pari al moment en que arribi al màxim de bits útils.

Un cop s'han analitzat tots els bits del registre, es procedeix a analitzar i tractar el bit de paritat rebut per transmissor i el comptador que el bloc ha generat. Primer es mira que aquest bit guardat durant la anàlisis del registre no sigui un novè bit de dades, ja que abans s'ha guardat sense tenir-ho en compte, i en cas negatiu es procedeix a comprovar la paritat del missatge. En cas de tenir la paritat activada i el novè bit de dades desactivat, mitjançant el comptador de '1' de les dades i la informació dels registres, es calcula el bit de paritat de les dades rebudes i es comprova si aquest coincideix amb el enviat. En cas negatiu s'activa la bandera d'error de paritat.

Un cop acabades de tractar totes les dades, es comprova si s'ha detectat algun error i s'envia un senyal d'avís al bloc *RSR* conforme les dades ja han sigut comprovades

La següent taula recull les entrades i sortides d'aquest bloc:

Taula 5-10: Entrades i sortides del bloc de detecció d'errors.

I/O	Senyal	Origen/Destinació	Descripció
IN	Paritat	Sistema/Detecció d'errors	Indica el tipus de paritat utilitzada
IN	Mida de les dades	Sistema/Detecció d'errors	Indica la mida de les dades
IN	Bits de parada	Sistema/Detecció d'errors	Indica el número de bits de parada que conté la cadena de dsdes
IN	Data in	Mostreig/Detecció d'error	El registre de bits rebuts

IN	Mostreig complert	Mostreig/Detecció d'error	Impuls a nivell alt que indica quan s'ha completat el registre
IN	UCSZ2	Sistema/Mostreig	Activa la comunicació a doble velocitat
OUT	Registre error	Detecció d'error /Sistema	Registre on es guarden les dues banderes d'error
OUT	Error comprovat	Detecció d'error /RSR	Bit que es posa a nivell alt un cop s'hagin comprovat tots dos errors

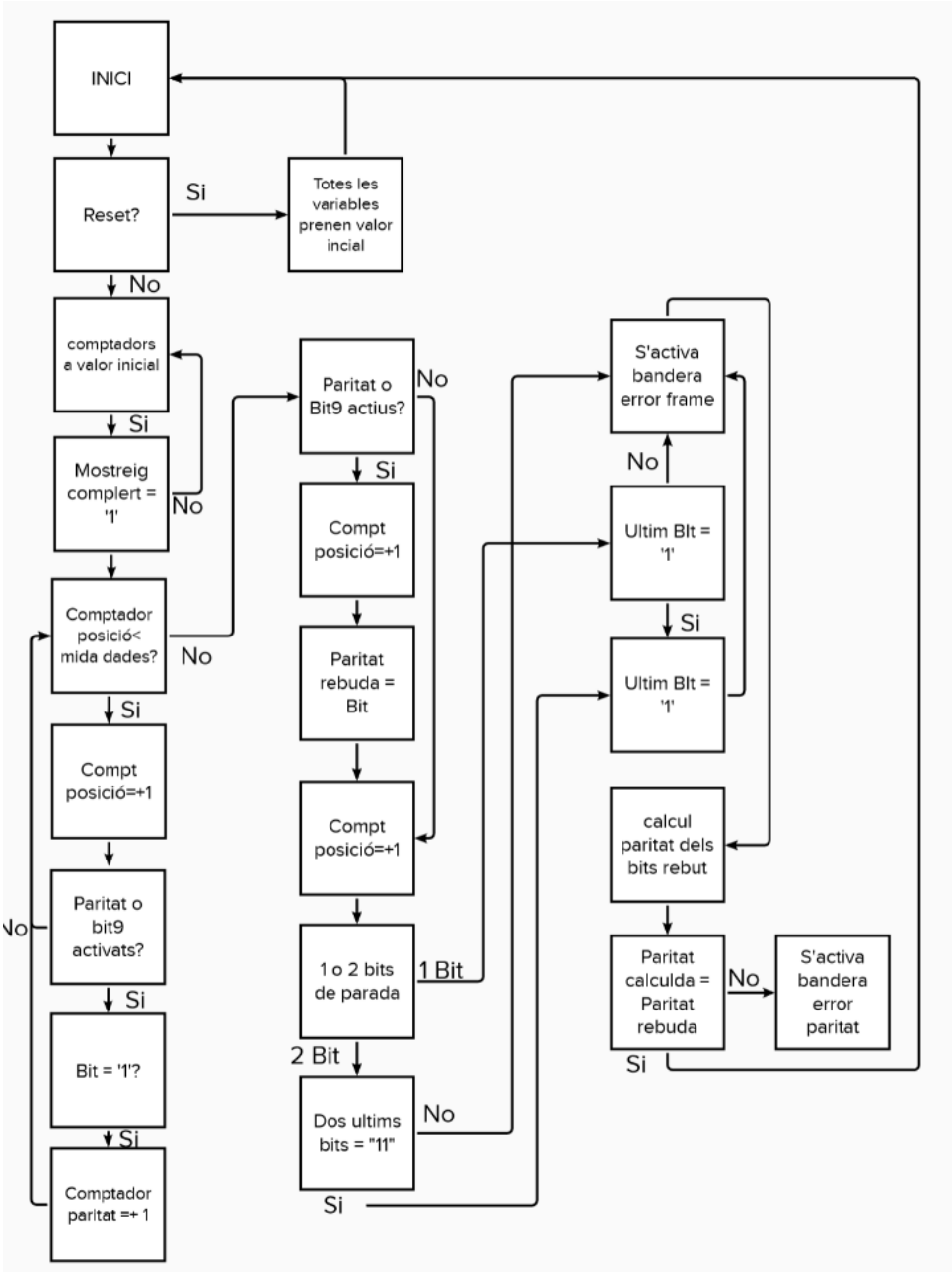


Figura 5-10: Disseny del bloc de detecció d'erros.

El comptador de posició del que es parla en el diagrama, es refereix al comptador que analitza bit per bit (d'esquerra a dreta, és a dir, del bit d'inici al bit de parada final) i el comptador de *frame* al valor variable en funció del nombre de dades, si es té bit de paritat o novè bit i el nombre de bits de parades configurats.

5.4.3. RSR

La funció d'aquest bloc és comprovar la mida de les dades rebudes i separar-les del *frame* generat pel bloc de *sampling*. Es farà servir un comptador de posició igual que el utilitzat en el bloc de detecció d'errors, però en aquets cas només arribarà fins la posició on es trobi l'últim bit de dades. En cas de tenir activada la funció d'un novè bit de dades, el comptador arribarà fins a una posició més i guardarà la dada rebuda en la seva corresponent posició en el registre d'estat.

Cal recordar que els bits s'envien de LSB a MSB i per tant, un cop s'ha generat el registre de dades, s'han d'invertir els bits d'aquets.

Un cop es té el registre complert i invertit, el bloc romandrà en espera de rebre el missatge de que els errors han estat comprovats, i en cas de detectar-se algun error durant la comunicació, el registre no serà enviat al buffer.

La següent taula recull les entrades i sortides d'aquest bloc:

Taula 5-11: : Entrades i sortides del bloc RSR

I/O	Senyal	Origen/Destinació	Descripció
IN	Paritat	Sistema/Detecció d'errors	Indica el tipus de paritat utilitzada
IN	Mida de les dades	Sistema/Detecció d'errors	Indica la mida de les dades
IN	Bits de parada	Sistema/Detecció d'errors	Indica el número de bits de parada que conté la cadena de dsdes
IN	Data in	Mostreig/RSR	El registre de bits rebuts
IN	Mostreig complert	Mostreig/RSR	Impuls a nivell alt que indica quan s'ha completat el registre
IN	Registre Errors	Detecció d'error/RSR	Registre que conté les banderes que indiquen possibles errors durant la comunicació

IN	Comprovació errors	Detecció d'error/RSR	Bit que indica que s'ha comprovat si hi ha errors durant la comunicació
IN	UCSZ2	Sistema/Mostreig	Activa la comunicació a doble velocitat
OUT	RSR buit	RSR/Sistema	Indica que el RSR por rebre una nova cadena de bits
OUT	RSR llest	RSR/Buffer Receptor	Impuls a nivell alt que indica que el RSR ha generat el Byte
OUT	Data out	RSR/Buffer Receptor	Byte enviat al Buffer
OUT	9 bit de dades	RSR/Sistema	Novè bit de dades

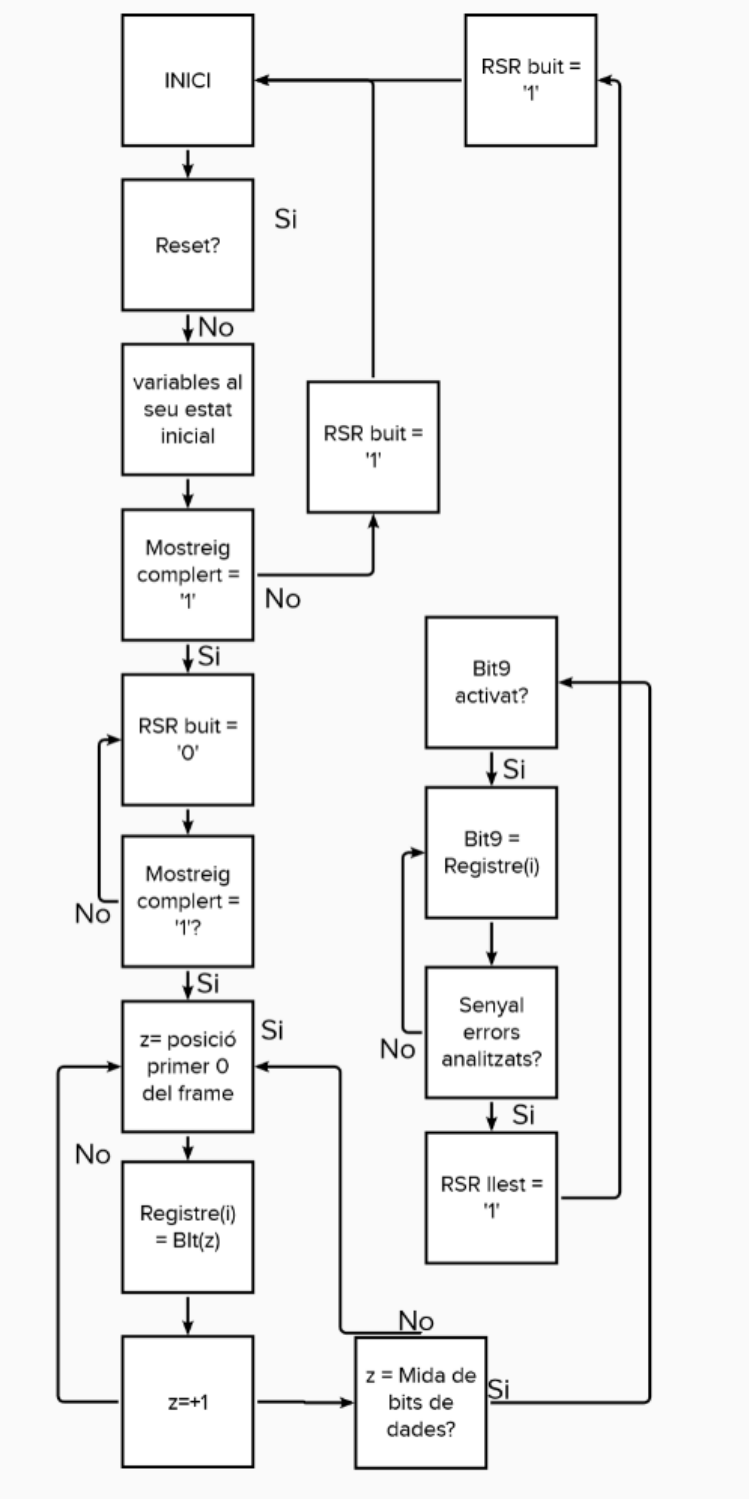


Figura 5-11: Disseny del bloc RSR.

5.4.4. Buffer receptor

Els registres de 8 bits rebuts seran emmagatzemats dins del buffer. El funcionament i estructura d'aquest és el mateix que l'explicat en el transmissor però variant els seus senyals d'entrada i sortida. En aquets cas, per a poder escriure en el buffer, es requereix l'impuls a nivell alt de confirmació del RSR que indica que el registre de 8 bits ha estat creat. Per llegir del buffer s'ha d'enviar un impuls a nivell alt al port de *Read enable*.

La següent taula recull les entrades i sortides d'aquest bloc:

Taula 5-12: Entrades i sortides del Buffer Receptor.

I/O	Senyal	Origen/Destinació	Descripció
IN	Paraula in	RSR/Buffer	Byte d'entrada
IN	RSR llest	RSR/Buffer	Impuls a nivell alt que indica que el RSR ha generat el Byte
IN	*Read enable*	RSR/Buffer	Bit que s'ha d'activar per tal de poder llegir del Buffer
OUT	Data out	Buffer/Sistema	Byte de sortida del Buffer
OUT	Buffer buit	Buffer/Sistema	Bit que indica que el buffer està buit i pot emmagatzemar noves paraules

En quant el disseny optat, s'ha escollit el mateix que el utilitzat pel transmissor però canviant les variables d'entrada i sortida per adequar-lo al mòdul receptor.

6. Simulacions

Per tal de provar que el disseny comentat en l'anterior apartat funciona de forma correcte, s'han simulat els arxius VHDL mitjançant el programa ModelSim. Als annexos (apartat A) es poden veure aquests arxius amb codi VHDL creats.

6.1. Baud Rate

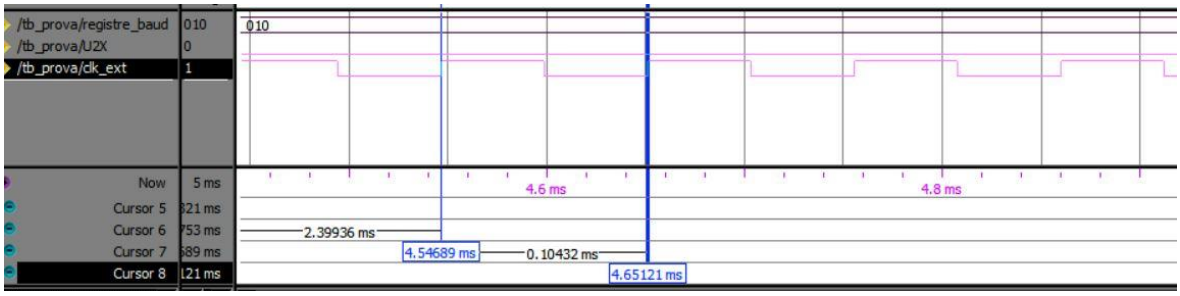


Figura 6-1: Baud Rate configurat a 9600 bauds.

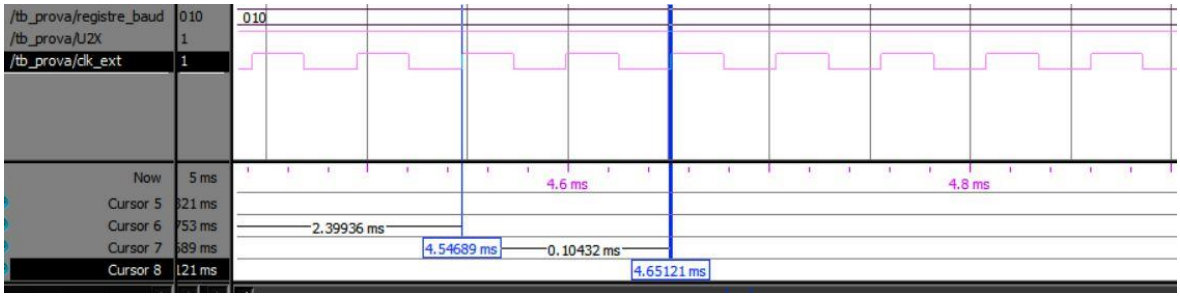


Figura 6-2: Baud Rate configurat a 9600 Bauds i doble velocitat.

A les anteriors figures es pot comprovar el funcionament de la doble velocitat. Totes dues s'han configurat per transmetre a 9600 Bauds, (Registre Baud = "010") però la segona executa dos cicles de rellotge amb el mateix temps en que la primera només en fa 1.

Es pot observar que el temps d'un cicle de rellotge i per tant el d'enviar 1 bit es de 0.10432 (en velocitat normal) que es l'equivalent a dividir 1/9600.

Si es canvia l'entrada del registre Baud, la velocitat augmenta o disminueix en funció del valor introduït. Per exemple, a la següent figura es pot observar que l'estat del registre = "101", que equival a una velocitat de 28800 bauds.

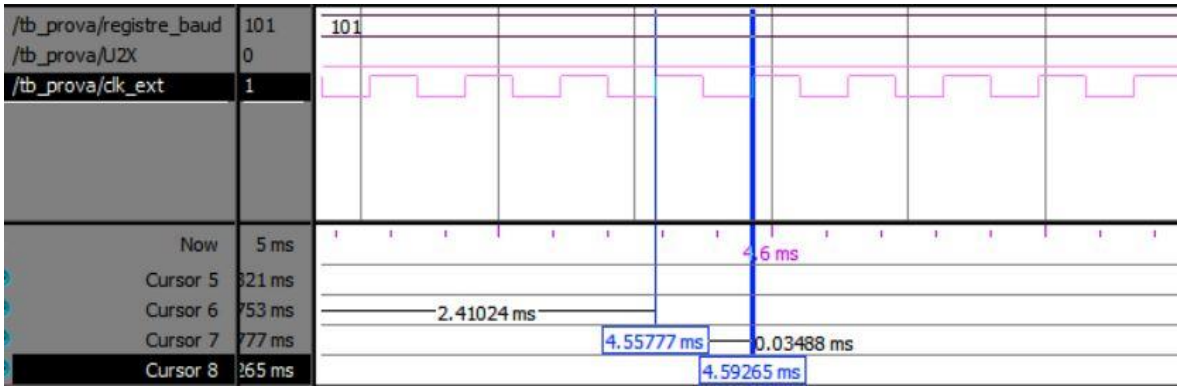


Figura 6-3: Baud Rate configurat a 28800 Bauds i velocitat normal.

6.2. Buffer

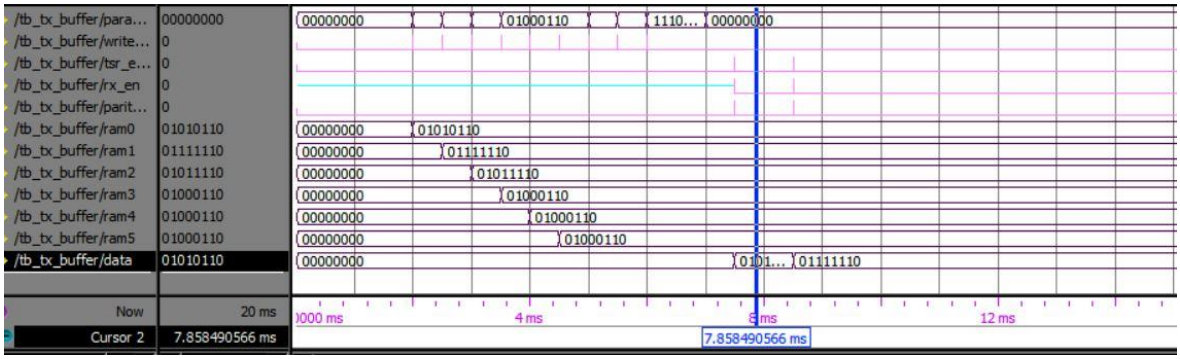


Figura 6-4: Simulacions del TX Buffer.

Com es pot comprovar a la figura, el buffer emmagatzema els bytes a les seves caselles en funció de l'ordre en que entren. Per que un byte sigui guardat al buffer s'ha de rebre a '1' al senyal de *Write Enable.*

També es pot observar el funcionament d'un *buffer* estil FIFO, on per moltes dades que s'hi guardin, sense passar el límit, la primera dada en sortir serà la primer que hagi entrat. La senyal de *Data* mostra les dades que surten cap al bloc del càlcul de paritat quan es rep la senyal de que el TSR i el bloc de paritat estan buits.

Si totes les caselles del *buffer* estan ocupades, s'activa la bandera de que el *buffer* està ple i no es desactiva fins que alguna casella ha quedat buida.

Per saber sobre quina casella s'ha de guardar o extreure informació i el nombre que hi ha disponibles es fan servir punter d'entrada i sortida.

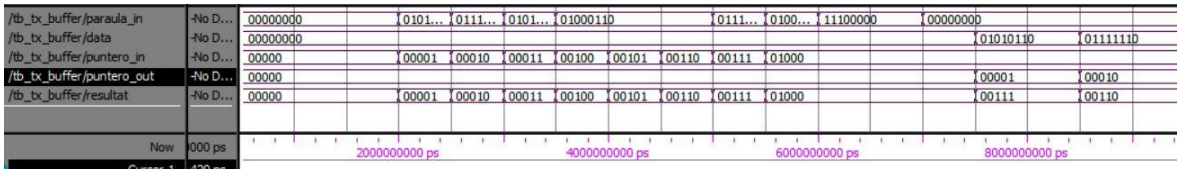


Figura 6-5: Gestió dels punters dins del Buffer.

6.3. Càlcul del bit de paritat

A continuació es mostren els resultat obtinguts de simular l'entrada d'un *Byte* al bloc de paritat. El tipus de paritat es pot observar amb la senyal de *paritat_tipo*, acord amb l'establert al registre descrit a l'apartat de disseny d'aquest projecte.

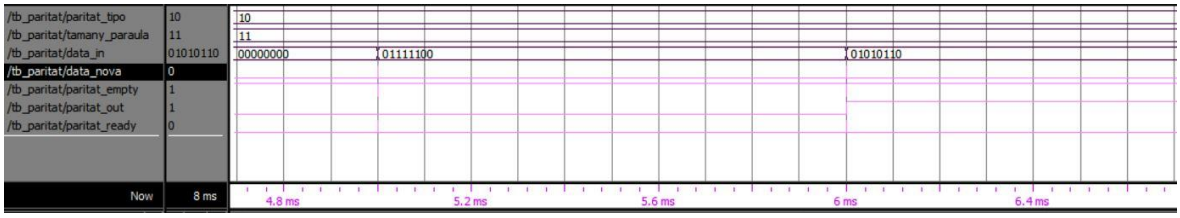


Figura 6-6: Simulacions càlcul del bit de paritat amb paritat senar.

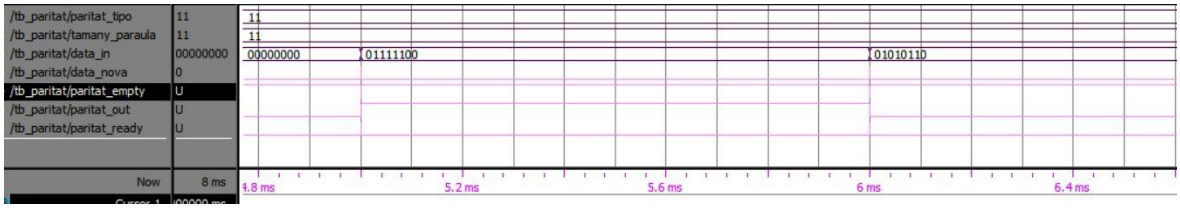


Figura 6-7: Simulacions càlcul del bit de paritat amb paritat parella.

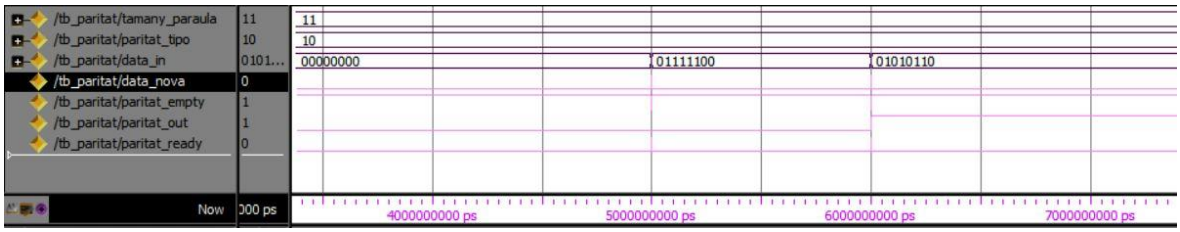


Figura 6-8: Simulacions càlcul del bit de paritat amb paritat parella.

A la figura 6-6 es pot observar el temps que requereix la generació del bit de paritat. Aquest es mira des de que es rep el bit de data nova (entra un nou byte al sistema) fins que es genera el bit de paritat *ready*.

A les figures 6-7 i 6-8 es pot observar com es genera el bit pels mateixos *bytes* però canviant el tipus de paritat.

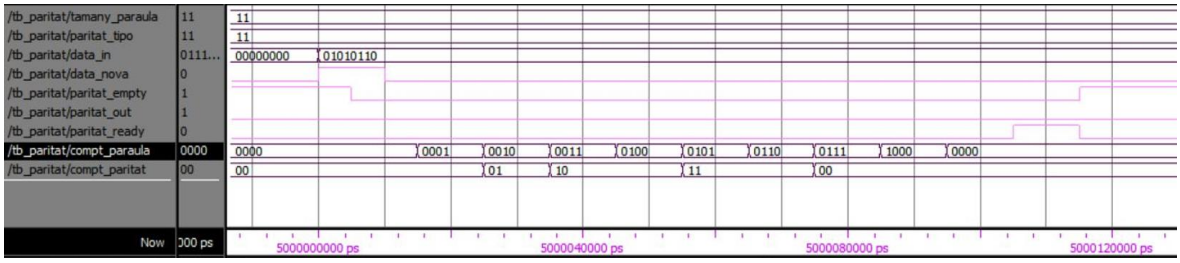


Figura 6-9: Comptadors utilitzats per generar el bit de paritat.

En la figura 6-9 es mostren el comptadors utilitzats tant per saber el nombre d'1s que conté el byte com per saber el nombre de bits recorreguts dins el registre.

6.4. Generar *frame* complert

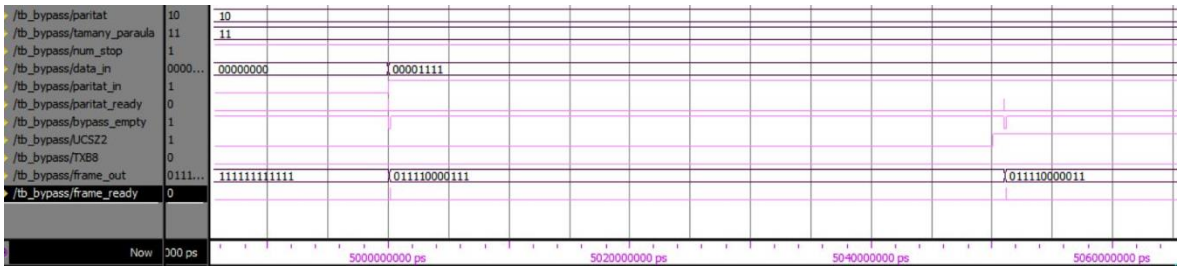


Figura 6-10: Simulacions del bloc de generació del frame complert treballant amb mode normal (1r frame) i 9 bits (segon frame).

S'han generat dos *frames* per a poder observar el mode normal i el mode en que s'activa el 9bit per a ser utilitzat com a generació d'adreces.

Es pot observar com el bloc s'activa al rebre el senyal de *paritat_ready* i en aquest moment el senyal de *bypass_empty* es posa a nivell baix fins que s'activa el senyal de *frame_ready*, moment en que torna a '1'.

En la figura 6-10 es pot observar el funcionament del 9bit (segon *frame* generat). En el primer *frame*, després dels bits de dades, es posa el de paritat; mentre que en el segon, quan es té activat UCSZ2, aquest bit queda anul·lat pel bit guardat al registre UCSRA, el TXB8.

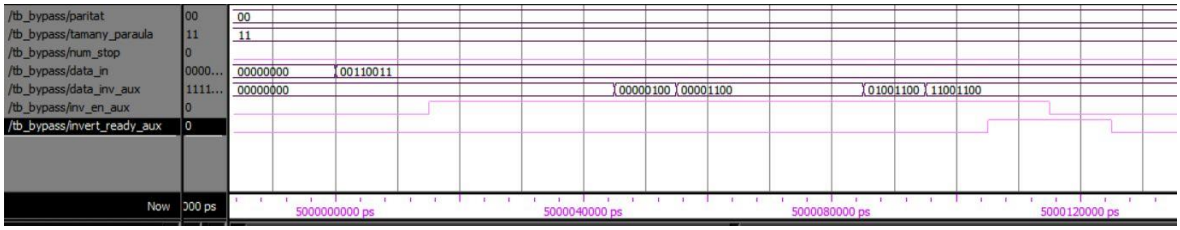


Figura 6-11: Funcionament del inversor del bits de dades.

Una altre acció que realitza aquest bloc és la de invertir els bits de dades per enviar de LSB a MSB, com es pot observar a la figura 6-11.

Per acabar cal comentar que si es tenen menys bits de 12 (mida del registre frame), es deixaran els primers bits del registre (tants com sobrin) a nivell alt, d'aquesta manera quna es comuniqui i es rebi el primer bit inicial només es tindran els bits importants fins el final de la transmissió. (Figura 7-9).

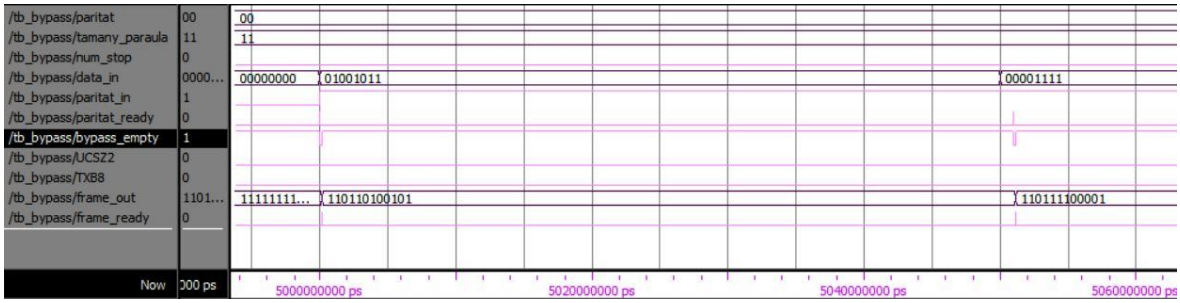


Figura 6-12: Frames generats amb 8 bits de dades, 1 sol bit de parada, i sense bit de paritat.

6.5. TSR

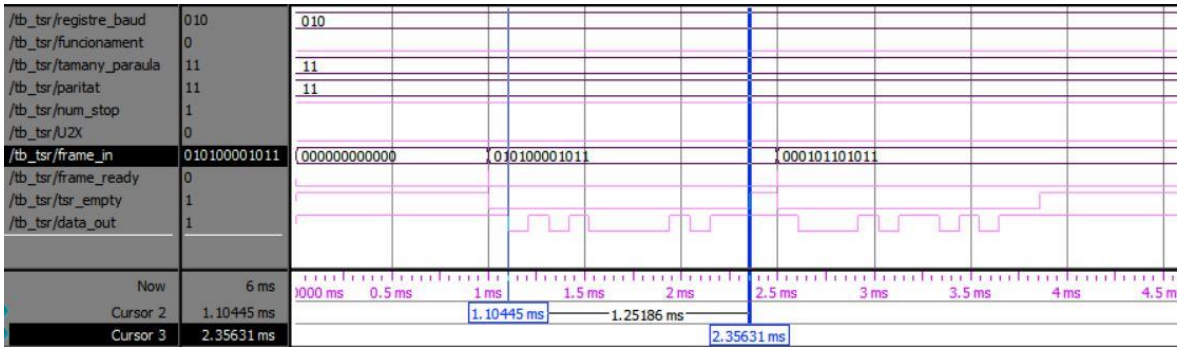


Figura 6-13: Simulació bloc TSR en mode asíncron i velocitat 9600 Baud.

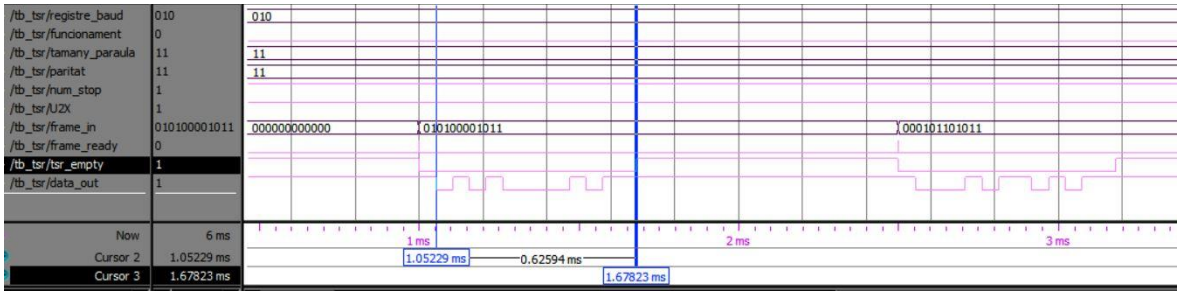


Figura 6-14: Simulació bloc TSR en mode asíncron i velocitat 9600 Baud amb selecció de doble velocitat

En quant rep el *frame* del bloc de generació, el senyal de tsr_empty passa a nivell baix fins que tots els bits han estat transmesos, moment en que torna a nivell alt.

A les figures 7-13 i 7-14 es pot observar el procés d'enviar el mateix *frame* però tenint activada o no l'opció de doble velocitat.

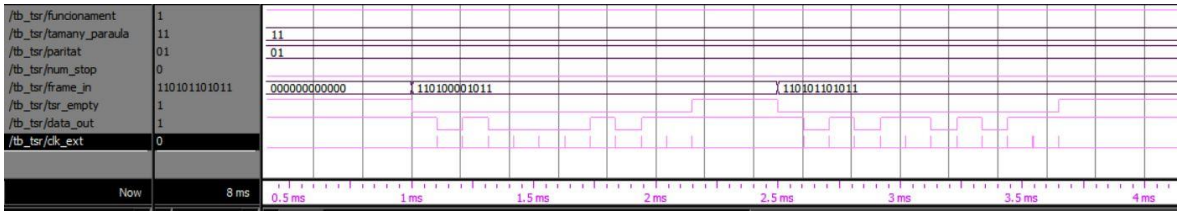


Figura 6-15: Simulació bloc TSR en mode síncron i velocitat 9600 Baud.

A la figura 7-15 es pot observar el mode síncron, que presenta la diferència d'enviar també el senyal amb un impuls sobre quan llegir el següent bit

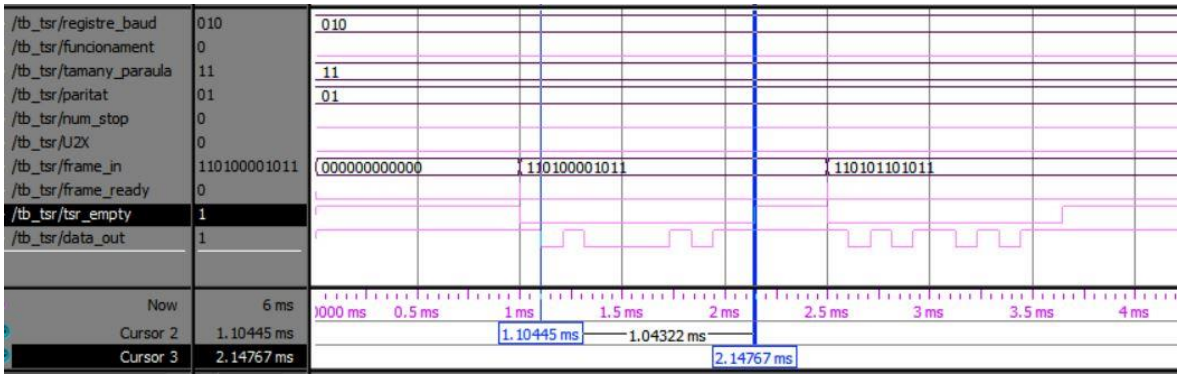


Figura 6-16: Asíncron amb frme de 10 bits

A la figura 7-16 es pot observar el funcionament comentat prèviament en el que si la mida del *frame* és inferior a 12, s'envien els bits de manera que la transmissió duri menys.

6.6. TX complert

A continuació es mostra el funcionament general del sistema transmissor i les seves variant:

- ***Frame* complert (figura 6-17):** 8 bits de dades, paritat activada i 2 bits de parada a 9600 Bauds.
- ***Frame* complert (figura 6-18):** 8 bits de dades, paritat activada i 2 bits de parada a 9600 Bauds amb mode doble velocitat activada.
- ***Frame* de tamany reduït (figura 6-19):** 8 bits de dades pero desactivant el bit de paritat i tenint 1 bit de parada.

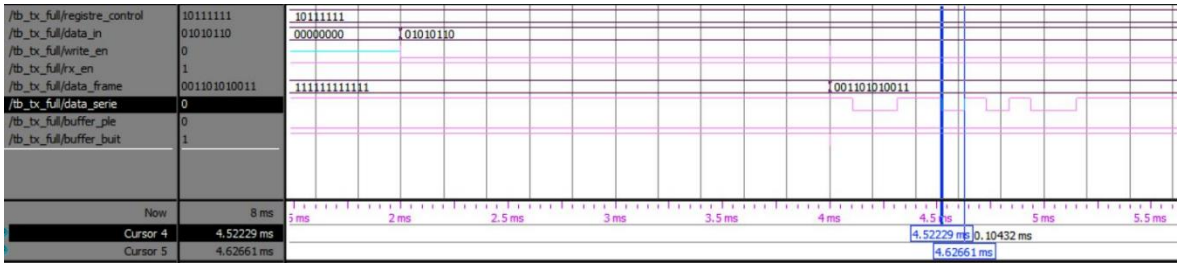


Figura 6-17: Simulacions del mòdul receptor amb 8 bits de dades, paritat activada i 2 bits de parada treballant a 9600 Bauds.

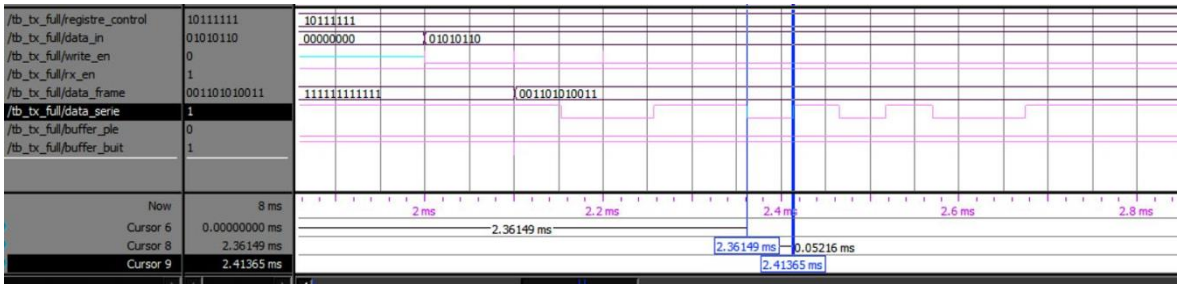


Figura 6-18: Simulacions del mòdul receptor amb 8 bits de dades, paritat activada i 2 bits de parada treballant a 9600 Bauds.

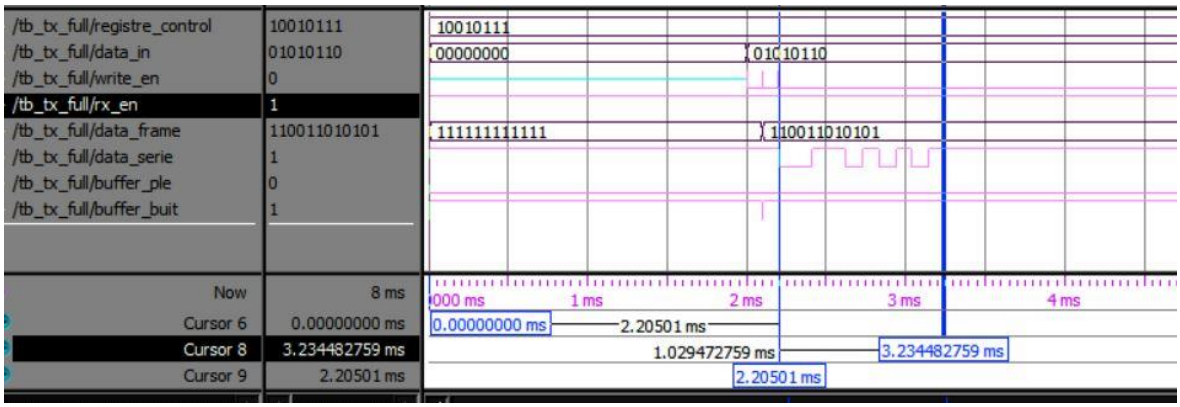


Figura 6-19: Simulacions del mòdul receptor amb 8 bits de dades, paritat desactivada i 1 bits de parada treballant a 9600 Bauds.

6.7. Mostreig

6.7.1. Recepció asíncrona

Es mostren 3 simulacions, una en la que es treballa en mode normal tenint una recepció completa de 12 bits, una altre amb la mateixa configuració per activar el bit de doble velocitat i per acabar una simulació en que no es reben 12 bits ja que es desactiva el bit de paritat.

Figura 6-20: Recepció asíncrona amb 8 bits de dades, bit de paritat i 2 bits de parada a 9600 Bauds.

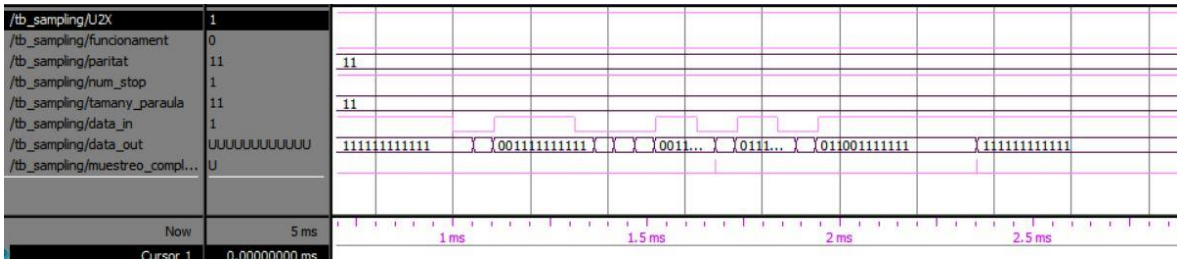


Figura 6-21: Recepció asíncrona amb 8 bits de dades, bit de paritat i 2 bits de parada a 9600 Bauds pero activant la doble comunicació.

A la figura 6-21 es pot observar la recepció funcionant a doble comunicació. Per a simular aquesta funció, s'ha utilitzat el mateix entorn de simulació que per a la velocitat normal, però es pot comprovar que quan treballa a doble velocitat, per cada bit rebut està agafant dues mostres. Per tant, tan bon punt acaba la recepció, ha generat dos *frames* complerts.

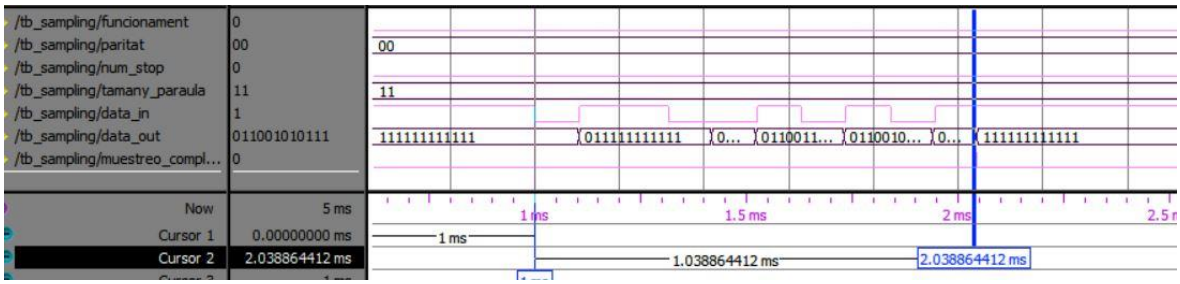


Figura 6-22: Recepció asíncrona amb 8 bits de dades, sense bit de paritat i 1 bits de parada a 9600 Bauds.

En aquesta última simulació es pot observar el funcionament en cas que es rebin menys bits del màxim del registre (12 bits). Els bits sobrant LSB es deixaran en un estat de nivell alt. En cas de tenir menys bits, es pot observar que el temps general de recepció també es redueix.

6.7.2. Recepció síncrona

Es mostraran dos simulacions, primer rebent 12 bits i després reduint els bits rebuts a 10.

A la següent figura s'observa el funcionament de la recepció síncrona. Es fa servir el mateix comptador que per la asíncrona però rebent les dades cada cop que es rep un senyal a '1' de clk_ext.

Figura 6-23: Recepció síncrona amb 8 bits de dades, bit de paritat i 2 bits de parada.

A continuació, es mostra una figura en la que es simula una recepció en la que no hi h a bit de paritat i només un bit de parada, reduint així el nombre de bits rebuts. El marcador de la figura indica el moment en que la senyal de mostreig complert es posa a nivell alt, enviant el registre generat als següents blocs del sistema.

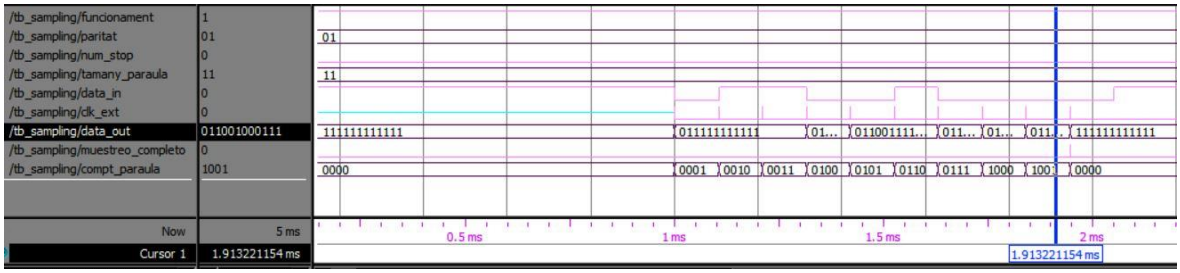


Figura 6-24: Recepció síncrona amb 8 bits de dades, bit de paritat i 2 bits de parada.

6.8. Detecció d'errors

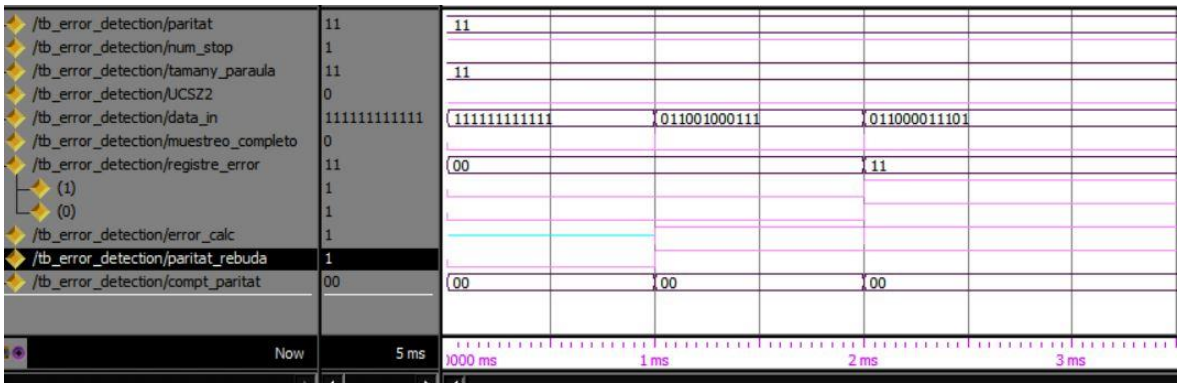


Figura 6-25: Simulació bloc detecció error. Paritat parella, 2 bits de parada i 8 bits de dades.

En la figura 6-25 s'han forçat dues entrades, una que no presenta cap error i una altra que que conté tant un error de paritat com un de *frame*.

Es pot observar que el sistema calcula correctament els bits d'error i activa la seva bandera correctament. El senyal de *error_calc*, que avisa que els errors han estat calculats, es posa a nivell alt quan s'ha analitzat el *frame* sencer. En aquest cas, no torna a nivell baix perquè es reben els dos *frames* seguits i un cop finalitzat el segon no se'n rep cap de nou i per tant queda en estat de '1'.

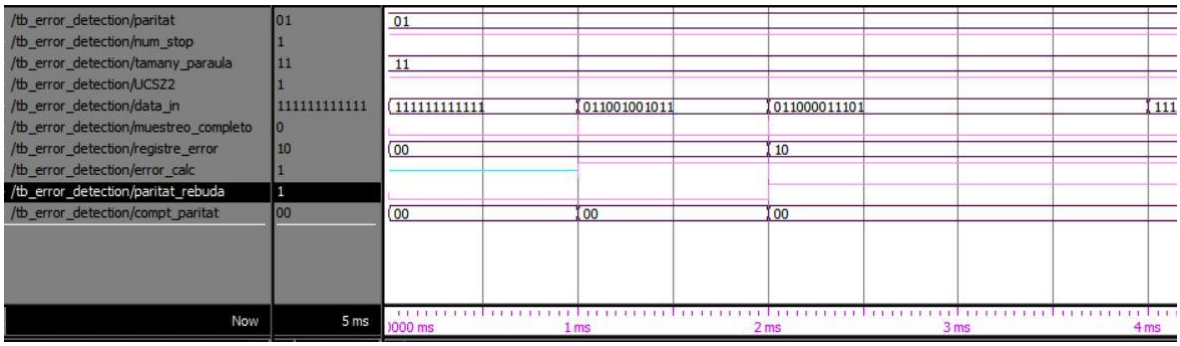


Figura 6-26: Simulació bloc detecció error. Novè bit activat, 2 bits de parada i 8 bits de dades.

Per comprovar que es té en compte correctament l'activació de 9 bits de dades, s'ha carregat el mateix *frame* amb la mateixa configuració que la primera simulació però canviant la paritat pel novè bit de dades (figura 6-26). Es pot observar com en el segon *frame*, que si que presentaria un error de paritat, aquesta queda desactivada per tenir el novè bit de dades activat.

6.9. RSR

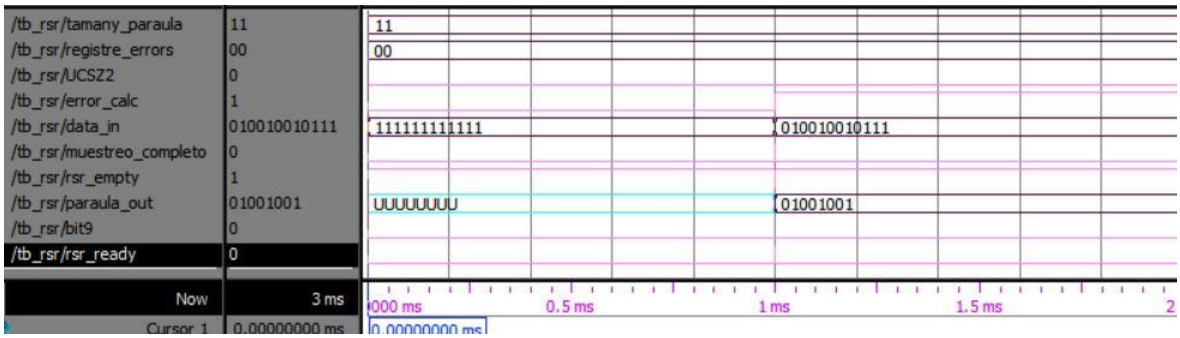


Figura 6-27: Simulació bloc RSR amb 6 bits de dades i el bit UCSZ2 activat.

Les dades rebudes s'analitzen i s'extreuen correctament els bits de dades. A la figura 7-27, es pot observar una comunicació de 8 bits de dades. Aquestes són "10010010" que, invertides, queda el valor de "01001001" que es mostra en la figura. En aquestes simulacions no es força cap valor de paritat ni de bits de parada, ja que amb el disseny plantejat no es requereix dels seus valors.

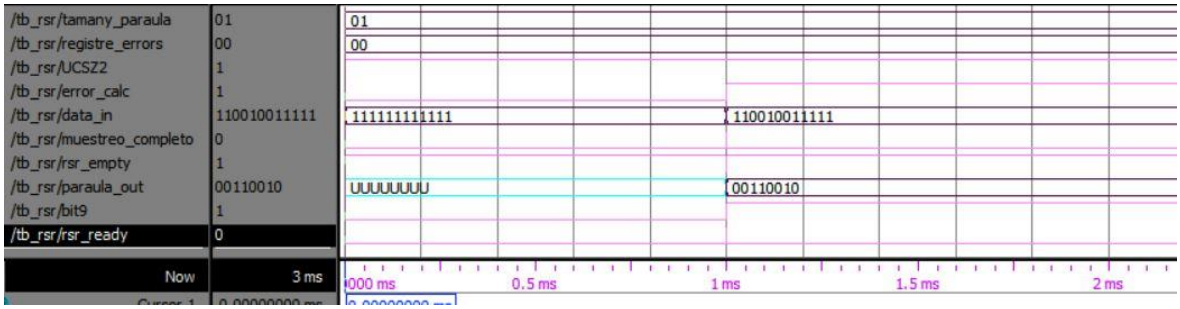


Figura 6-28: Simulació bloc RSR amb 6 bits de dades i el bit UCSZ2 activat.

A la figura 7-28 es pot observar el funcionament en cas de tenir menys de 8 bits de dades (6 en aquest cas) i el bit de UCSZ2 activat. El registre de 8 bits de sortida mantindrà a valor 0 els MSB en cas de tenir menys de 8 bits de dades.

Es pot observar també la sortida del 9 bit de manera correcte.

6.10. RX complert

S'ha simulat el sistema receptor complert, incloent tots els blocs esmentats durant l'apartat del disseny d'aquest projecte.

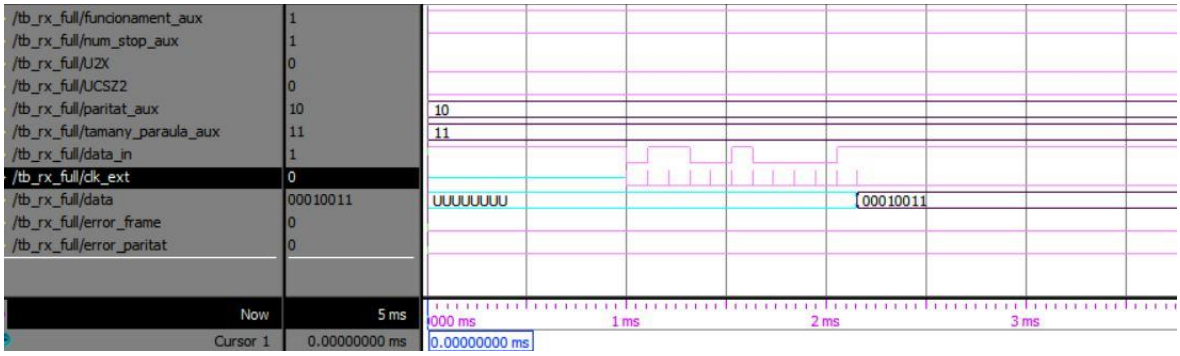


Figura 6-29: Recepció síncrona amb 8 bits de dades, paritat senar i 2 bits de parada a 9600 Bauds.

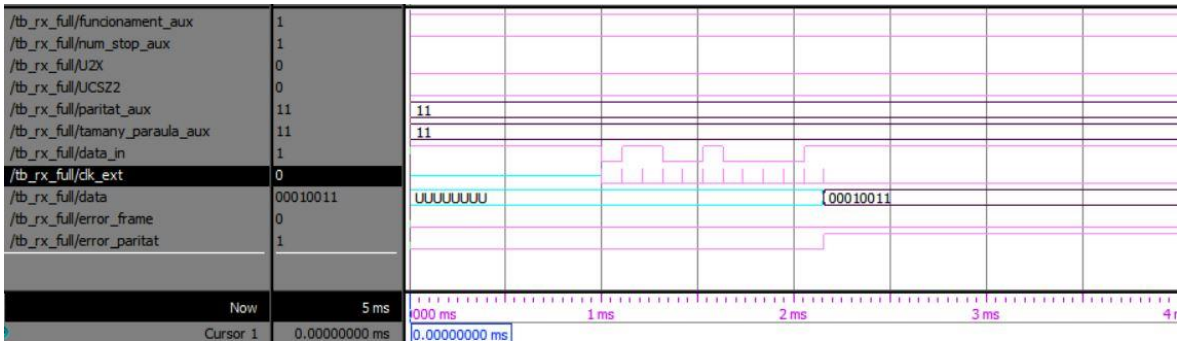


Figura 6-30: Recepció síncrona amb 8 bits de dades, paritat parella i 2 bits de parada a 9600 Bauds.

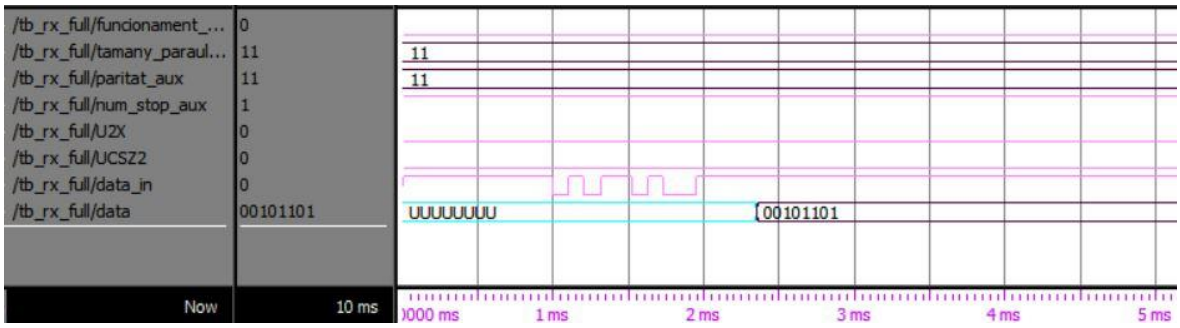


Figura 6-31: Recepció asíncrona amb 8 bits de dades, paritat parella i 2 bits de parada a 9600 Bauds.

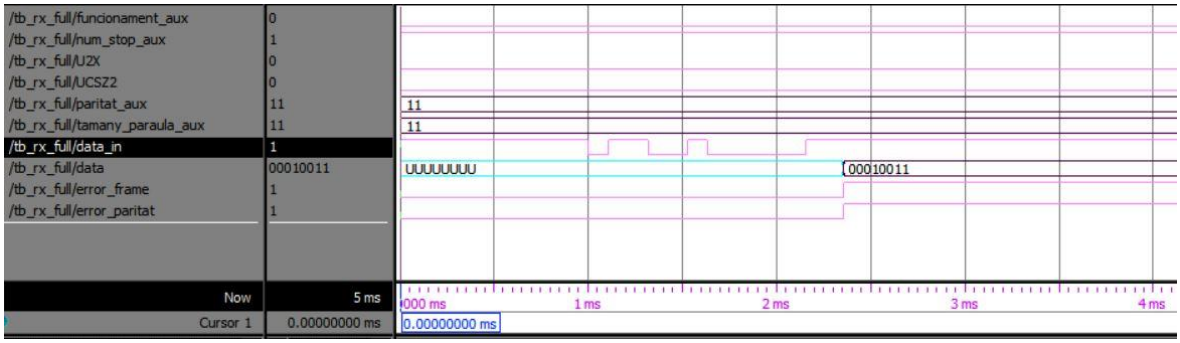


Figura 6-32: Recepció asíncrona amb 8 bits de dades, paritat parella i 2 bits de parada a 9600 Bauds.

7. Implementació del disseny

Un cop s'ha dissenyat el projecte i verificat el funcionament mitjançant les simulacions realitzades amb el ModelSim, es realitza el test sobre una placa FPGA per poder verificar el correcte funcionament del disseny. La placa escollida per la implementació es una BASYS2, de l'empresa *Diligent*.

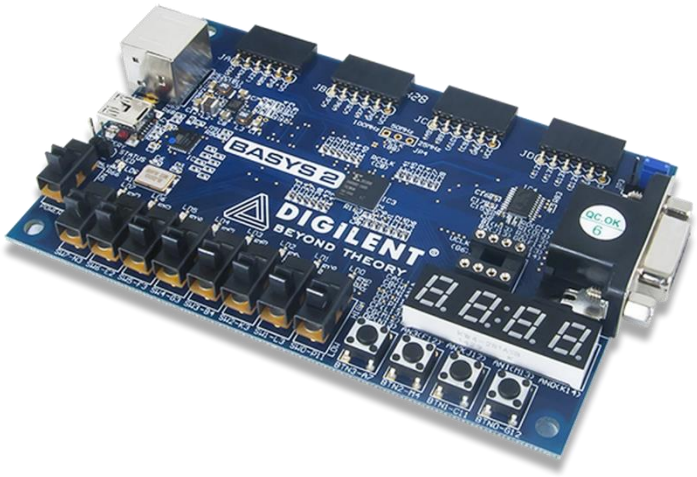


Figura 7-1: FPGA BASYS2 de l'empresa diligent [18].

Les prestacions més destacables que ofereix i que interessen per la implementació del dissney són les següents: [18]

- 72 Kbit de memòria *ram.*
- Rellotge amb freqüència variable de 25, 50 i 100 MHz.
- Reguladors de voltatge de 1,2V, 2,5V i 3,3V.
- Mini USB port.
- 8 interruptors, 8 LED'S I 4 botons.
- 4 grups de pins d'entrada i sortida amb 6 connectors cada un.

La *FPGA* Basys 2 requereix del software *Adept2* de *Digilent* per tal de poder carregar els diferents programes un cop dissenyats.

Per poder sintetitzar el codi VHDL creat, s'ha utilitzat el software ISE, que ens permet sintetitzar el codi juntament amb l'arxiu de restriccions sobre la placa.

7.1. Síntesis

Els arxius necessaris per a poder sintetitzar el disseny són el codi VHDL, que s'ha creat i simulat amb ModelSim, i un arxiu de restriccions que es crea amb el mateix software ISE.

Un arxiu de restriccions (*.ucf*) conté la informació que permet relacionar les entrades i sortides dels ports i components dissenyats amb components físics. Per exemple, es permet relacionar una sortida d'un sol bit amb dels ports I/O per poder encendre un *LED.*

A continuació, es mostraran les variables i els components físics utilitzats per tal de fer el test sobre la placa.

Degut a la falta de components, per tal d'implementar una prova que permeti verificar el funcionament, s'han dissenyat blocs que permeten de la mateixa forma que amb el Baud Rate, seleccionar una possible configuració.

7.1.1. Transmissor

Taula 7-1: Senyals del transmissor del arxiu de restriccions per la síntesis sobre la Basys2

Senyals	I/0	I/O Basys2	Component
Clk	IN	B8	Senyal de rellotge de 50 MHz
rst	IN	A7	Polsador
Data_in [1:0]	IN	N3-E2	Dos interruptors
Registre Control [2:0]	IN	F3-G3-B4	Dos interruptors
UCSZ2	IN	C12	Connector
TXB8	IN	A13	Connector
U2X	IN	C13	Connector
Registre Baud [2:0]	IN	K3-L3-P11	Tres interruptors

Read Enable	IN	M4	Polsador
RX *Enable*	IN	C11	Polsador
Buffer Buit	OUT	G1	LED
Buffer ple	OUT	P4	LED
Data out	OUT	J3	Port I/O per a ser connectat a un altre element
Clk generat	OUT	B2	Port I/O per a ser connectat a un altre element

Els senyals de dades d'entrada s'han implementat mitjançant polsadors [1:0] que permeten 4 possibles combinacions de dades. No fan falta més ja que són simples bytes, i amb 4 per testejar és més que suficient.

El registre de control, per altre banda, si que té mes possibles combinacions i s'ha implementat amb interruptors. Mitjançant 3 interruptors, es configuren 8 entrades de registre amb diferents modes de paritat funcionament síncron o asíncron i diferents bits de parada i de dades.

La sortida sèrie del transmissor, s'ha connectat a un oscil·loscopi per tal de poder observar la cadena de bits transmesa.

7.1.2. Receptor

Taula 7-2: Senyals del transmissor del arxiu de restriccions per la síntesis sobre la Basys2

Senyals	I/0	I/O Basys2	Component
Clk	IN	B8	Senyal de rellotge de 50 MHz
rst	IN	A7	Polsador

Data_in	IN	C6	Connector
CLK extern	IN	B6	Connector
Registre Control [1:0]	IN	F3-G3-B4	Dos interruptors
UCSZ2	IN	C12	Connector
TXB8	IN	A13	Connector
U2X	IN	C13	Connector
Registre Baud [2:0]	IN	K3-L3-P11	Tres interruptors
Read Enable	IN	M4	Polsador
Data_out [7:0]	IN	C5-B7-A9-B9-A10-C9-B5-B2	Connector
RXB8	IN	D12	Connector
RX *Enable*	OUT	P7	LED
Buffer Buit	OUT	G1	LED
Buffer ple	OUT	P4	LED

La sortida del mòdul ve donada del buffer receptor, la qual es connecta a una sèrie de 8 LED's que permeten la visualització de les dades rebudes. La resta d'informació corresponent als registres s'observa amb els propis LED's de la placa.

8. Conclusions

Per poder realitzar aquest disseny s'ha estudiar prèviament el funcionament del mòdul *USART*, mòdul no tractat en específic i de forma amplia durant els 4 anys d'estudis de grau. Tot i no disposar de molta informació sobre aquests mòduls, s'ha aconseguit fer una disseny que permet treballar amb totes les possibles configuracions que ofereix aquest tipus de comunicacions.

En l'etapa de les simulacions, també s'ha assolit l'objectiu del disseny, doncs s'han obtingut uns bons resultats sobre el funcionament del sistema i s'ha deixat registrat no només el funcionament general del sistema, sinó que també de cada un dels blocs amb les possibles funcionalitats que ofereixen cada un.

Cal esmentar que la part de síntesis i prova sobre placa no ha presentat tants bons resultats com les etapes anteriors, doncs tot i que per una banda el mòdul transmissor si que ha funcionat de manera correcta i s'ha comprovat aquest resultat mitjançant un oscil·loscopi, el mòdul receptor no ha acabat de funcionar de la manera que s'esperava. Si que és veritat que permetia el treball a la velocitat adequada i configurada, però després de provar-lo juntament amb una placa A*rduino* que li enviava missatges, aquests no mostraven el missatge esperat.

Les possibles causes d'aquests errors poden ser d'interconnexió dels senyals dins del propi disseny, que al treballar a una velocitat molt elevada pugui causar algun error, tot i no mostrar-ho a les simulacions, o bé que no s'hagi configurat de manera correcta la placa externa *Arduino* amb la que s'han realitzat les proves.

Per tant, a part d'aconseguir la bona implementació sobre la placa del mòdul receptor, aquest treball dóna opció a possibles projectes futurs com podria ser la implementació d'una xarxa de comunicació mitjançant aquets mòduls on es treballi amb múltiples microcontroladors en forma de *master-slave*. El disseny seguit per aquest projecte està disponible i detallat pas per pas, per tant, juntament amb el codi *VHDL* disponible als annexos, es podria plantejar un treball on es tracti aquest tema.

9. Cost econòmic del projecte

A continuació es realitzarà un estudi econòmic sobre els costos que implica la realització d'aquest projecte. Es considera que el preu per contractar un enginyer es de 12 euros l'hora.

9.1. Cost de la mà d'obra

Taula 9-1:Temps dedicat per cada apartat del projecte i el seu preu associat

TITULACIÓ	TASQUES	HORES	PREU
Enginyer	Plantejament del projecte	50 h	400,00 €
	Disseny teòric dels sistemes	100 h	800,00 €
	Implementació del disseny mitjançant VHDL	180 h	1440,00 €
	Realització de les corresponents simulacions	140 h	1120,00 €
	Síntesis del disseny sobre una placa Basys 2	50 h	400,00 €
	Test sobre placa *Arduino*	60 h	480,00 €
	Observació resultats en oscil·loscopi	20 h	160,00 €
	TOTAL	**600 h**	**4.800,00 €**

*El preu indicat es l'estipulat per la universitat UPC per la realització de pràctiques remunerades d'un estudiant d'enginyeria.

$$Preu = 600\ h * 8 * \frac{€}{h} = 4.800{,}00\ € \quad \text{(Eq. 9-1)}$$

9.2. Material

Taula 9-2: Cost del material utilitzat en aquest projecte

NOM	QUANTITAT	PREU UNITARI	VIDA ÚTIL	COST ATRIBUIT AL PROJECTE
FPGA Basys 2	1	150 €	4 anys	18,75 €
Portàtil HP Pavilion**	1	800 €	4 anys	100,00 €
Placa elegoo UNO R3	1	25 €	4 anys	3,13 €
TOTAL				**121,88 €**

Es té en compte que la vida útil del material utilitzat és de 4 anys, per tant s'agafa 1/8 part del seu cost ja que la realització d'aquest projecte presenta una durada de 6 mesos.

$$Cost\ atribuit = Quantitat * Preu\ unitari * \frac{duració\ del\ projecte[mesos]}{Vida\ útil\ [mesos]} \quad \textbf{(Eq. 9-2)}$$

9.3. Cost de les llicències i programes

Taula 9-3: Cost de les llicències i programes

NOM	TOTAL ATRIBUIT AL PROJECTE
ModelSim	0,00 €
ISE Xilinx	0,00 €
Llicència ISE webpack	0,00 €

ADEPT2	0,00 €
IDE arduino	0,00 €
Microsoft Office	50 €*
TOTAL	**50 €**

*Per estipular el preu de la llicència de *Microsoft office* s'ha seguit el mateix raonament que pel material. En aquest cas, les llicències són anuals i per tant s'ha calculat el preu per una període de mig any.

$$Total\ atribuit\ al\ projecte = Cost\ programa * \frac{duració\ del\ projecte[mesos]}{durada\ llicència\ [mesos]} \quad \textbf{(Eq. 9-3)}$$

9.4. Balanç total

El cost total del projecte complert sumant tots els gestos dels anteriors apartats és:

Taula 9-4: Cost total del projecte

NOM	TOTAL
Ma d'obra	4.800,00 €
Materials	121,88 €
Llicències + programes	50,00 €
TOTAL	**4.971,88 €**

10. Anàlisi mediambiental

El projecte consisteix en el disseny hardware de microcontroladors, pel que l'impacte mediambiental que produeix és pràcticament nul, perquè es treballa únicament amb un ordinador i les pròpies plaques (arduino i FPGA) les quals s'alimenten del propi ordinador.

Així doncs, s'ha decidit que no s'ha de realitzar un estudi ambiental ja que l'única despesa energètica és el de la càrrega da la pròpia bateria del portàtil i no es generen residus.

Bibliografia

[1] Aguayo, J. *Introducción a los microcontroladores*, 2004, En: [en línia]. [consulta: desembre 2021]. Disponible a : https://itm201512.webnode.es/_files/200000038-6e59e6f573/Microcontrolador.pdf

[2] Úbeda, B. *Apuntes SISTEMAS EMBEBIDOS*, tema 2, pàgines 1-15 2009. En: [en línia]. [consulta: desembre 2021] Disponible a: https://www.um.es/documents/4874468/19345367/ssee-t02.pdf/bdf91c12-df9a-4b80-9e15-2241ee6eabad

[3] Artés Rodríguez, A., Pérez González, F., Cid Sueiro, J., López Valcarce, R., Mosquera Nartallo, C. I Pérez Cruz, F. *Comunicaciones digitales*, 2012. En: [en línia]. [consulta: desembre 2021]. Disponible a: https://www.tsc.uc3m.es/~antonio/libro_comunicaciones/El_libro_files/comdig_artes_perez.pdf

[4] Gámiz, J., Gámiz, J. i Ponsa P. *Comunicaciones Industriales*, 2018. En: [Documentació teòrica de l'assignatura Sistemas de *Comunicación Información y comunicación Industrial* (SICI) de l'*EEBE] [consulta: desembre 2021]*.

[5] Guerrero, V., Yuste, R. i Martínez, L. *Comunicaciones industriales*, 2009. En: [en línia]. [consulta: desembre 2021] Disponible a: https://books.google.es/books?hl=es&lr=&id=fPCVCoDCa8IC&oi=fnd&pg=PT41&dq=deteccion+de+error+en+comunicacion+industrial+&ots=q_hMxb1Gdt&sig=v_jbDQlaQMj6MNroh2pSXtyiSUo#v=onepage&q=deteccion%20de%20error%20en%20comunicacion%20industrial&f=false

[6] Carletti, J. *Comunicación – Bus I2C*. En: [en línia][consulta: gener 2022]. Disponible a: https://www.bolanosdj.com.ar/MOVIL/ARDUINO2/ComunicacionBusI2C.pdf

[7] Leens, F "An introduction to I2C and SPI protocols," in IEEE Instrumentation & measurement Magazine, vol. 12, no. 1, pp. 8-13, February 2009, doi: 10.1109/MIM.2009.4762946. En [en línia]. [consulta: gener 2022]. Disponible a: https://ieeexplore-ieee-org.recursos.biblioteca.upc.edu/document/4762946

[8] El-Mousa, A., Suyyagh, A., *A high speed reconfigurable USART IP core with support for multi-drop networks*, 2010

https://www.researchgate.net/publication/229036511_A_high_speed_reconfigurable_USART_IP_core_with_support_for_multi-drop_networks

[9] Frenzel, L. *Handbook of Serial Communications Interfaces. A Comprehensive Compendium of Serial Digital Input/Output (I/O) Standards*, 2016. En [en línia][consulta: gener 2022]. Disponible a : https://www-sciencedirect-com.recursos.biblioteca.upc.edu/book/9780128006290/handbook-of-serial-communications-interfaces#book-description

[10] Taraate, V. *Advanced HDL Synthesis ans SOC Prototyping*, 2019. ISBN: 978-981-10-8775-2. DOI: 10.1007/978-981-10-8776-9.

[11] Brock J. LaMeres. *Quick Start Guide to VHDL*, 2019. ISBN: 978-3-030-04515-9. DOI: 10.1007/978-3-030-04516-6

[12] Brock J. LaMeres. *Introduction to Logic Circuits & Logic Design with VHDL*, 2019. ISBN: 978-3-030-12488-5/ 978-3-030-12491-5. DOI: 10.1007/978-3-030-12489-2.

[13] Money Harris, D i L. Harris S. Digital Design and Computer Architecture, 2007. ISBN: 978-0-12-370-497-9. En [en línia]. [consulta: febrer 2022] Disponible a:

[14] Article: *The USART of the AVR.* Pàgina web sobre *embedded electronics* https://maxembedded.com/2013/09/the-usart-of-the-avr/

[15] Xilinx [Pàgina web oficial del proveïdor]. Enllaç per descarregar el software *ISE*: https://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.html

[16] Intel [Pàgina web oficial del proveïdor]. Enllaç per descarregar el software *ModelSim*: https://www.intel.es/content/www/es/es/software/programmable/quartus-prime/model-sim.html

[17] Digilent, C. *Basys 2™ FPGA Board Reference Manual*, 2016. En[en línia][consulta: març 2022]. Disponible a: https://www.intel.es/content/www/es/es/software/programmable/quartus-prime/model-sim.html

Annex A. Disseny VHDL

A.1 *PortMap* Transmissor

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4
5  entity TX_FULL is
6  port(
7          clk, rst: in std_logic;
8          registre_estat: inout std_logic_vector (7 downto 0);
9          U2X: in std_logic;
10         UCSZ2: in std_logic;
11         TXB8: in std_logic;
12         registre_control: in std_logic_vector (7 downto 0);
13         registre_baud: in std_logic_vector(2 downto 0);
14         data_in: in std_logic_vector (7 downto 0);
15         write_en: in std_logic;
16         rx_en: in std_logic;
17
18         clk_ext: out std_logic;
19         buffer_ple, buffer_buit, error_overflow: out std_logic;
20         data_serie: out std_logic);
21
22 end TX_FULL;
23
24 architecture Behavioral of TX_FULL  is
25
26
27 alias RXB8: std_logic is registre_estat(4);
28 alias PE : std_logic is registre_estat(2);
29 alias FE : std_logic is registre_estat(1);
30
31 --REgistre de control
32 alias funcionament_aux: std_logic is registre_control (6);
33 alias paritat_aux: std_logic_vector (1 downto 0) is
34 registre_control(5 downto 4);
35 alias num_stop_aux: std_logic is registre_control(3);
36 alias tamany_paraula_aux: std_logic_vector(1 downto 0) is
37 registre_control (2 downto 1);
38 alias clk_pol_aux: std_logic is registre_control(0);
39
40 component TX_buffer is
41
42         port (  rst, clk: in std_logic;
```

```
43                paraula_in: in std_logic_vector (7 downto 0);
44                write_en: in std_logic;
45                tsr_empty: in std_logic;
46                paritat_empty: in std_logic;
47                rx_en: in std_logic;
48
49                data: out std_logic_vector(7 downto 0);
50                data_nova: out std_logic;
51
52                buffer_ple: out std_logic;
53                buffer_buit: out std_logic;
54                error_overflow: out std_logic);
55        end component;
56
57 component PARITAT is
58
59        port (  clk, rst: in std_logic;
60                paritat_tipo: in std_logic_vector (1 downto 0);
61                tamany_paraula: in std_logic_vector (1 downto 0);
62                data_in: in std_logic_vector (7 downto 0);
63                data_nova: in std_logic;
64                paritat_ready: out std_logic;
65                paritat_empty: out std_logic;
66                paritat_out: out std_logic);
67        end component;
68
69 component BYPASS is
70
71        port (  clk, rst: in std_logic;
72                paritat: in std_logic_vector (1 downto 0);
73                tamany_paraula: in std_logic_vector (1 downto 0);
74                num_stop: in std_logic;
75                UCSZ2: in std_logic;
76                TXB8: in std_logic;
77                data_in: in std_logic_vector (7 downto 0); --
78 arriva des de tx_buffer
79                paritat_in: in std_logic; --variable creada per
80 PARITY.
81                paritat_ready: in std_logic;
82                bypass_empty: out std_logic;
83                frame_ready: out std_logic;
84                frame_out: out std_logic_vector (11 downto 0));
85        end component;
86
87 component TSR is
88
89        port (  clk, rst: in std_logic;
```

```
90                paritat: in std_logic_vector (1 downto 0);
91                tamany_paraula: in std_logic_vector (1 downto 0);
92                num_stop: in std_logic;
93                funcionament: std_logic;
94                registre_baud: in std_logic_vector(2 downto 0);
95                U2X: in std_logic;
96                UCSZ2: in std_logic;
97                frame_in: in std_logic_vector(11 downto 0);
98                frame_ready: in std_logic;
99                rx_en: in std_logic;
100               tsr_empty: out std_logic;
101               data_out: out std_logic;
102               clk_ext: out std_logic);
103       end component;
104
105 --TX_buffer
106 signal data_out_buffer: std_logic_vector (7 downto 0);
107 signal data_nova_buffer: std_logic;
108
109 --Paritat
110 signal paritat_ready_paritat: std_logic;
111 signal paritat_empty_paritat: std_logic;
112 signal paritat_out_paritat: std_logic;
113
114 --Bypass
115 signal frame_out_bypass: std_logic_vector (11 downto 0);
116 signal bypass_empty_bypass: std_logic;
117 signal frame_ready_bypass: std_logic;
118
119 --TSR
120 signal tsr_empty_tsr: std_logic;
121 signal data_out_tsr: std_logic;
122
123 --REGISTRE
124 signal buffer_buit_aux:std_logic;
125 signal buffer_ple_aux: std_logic;
126
127 begin
128 u1: TX_buffer port map (clk => clk , rst =>rst, paraula_in =>
129 data_in, write_en => write_en, tsr_empty => tsr_empty_tsr,
130 paritat_empty => paritat_empty_paritat,
131                       rx_en=>rx_en, buffer_ple => buffer_ple_aux,
132 buffer_buit => buffer_buit_aux, error_overflow => error_overflow,
133 data => data_out_buffer,
134                       data_nova => data_nova_buffer);
135
136
```

```
u2: PARITAT   port map (clk => clk , rst =>rst, paritat_tipo =>
paritat_aux, tamany_paraula=>tamany_paraula_aux, data_in =>
data_out_buffer, data_nova => data_nova_buffer,
                        paritat_ready => paritat_ready_paritat ,
paritat_empty => paritat_empty_paritat, paritat_out =>
paritat_out_paritat);

u3: BYPASS   port map (clk => clk , rst =>rst, paritat =>
paritat_aux, tamany_paraula=>tamany_paraula_aux,
num_stop=>num_stop_aux,  UCSZ2=>UCSZ2, TXB8=>TXB8,  data_in =>
data_out_buffer, paritat_in => paritat_out_paritat,  paritat_ready
=> paritat_ready_paritat,
                        bypass_empty => bypass_empty_bypass ,
frame_ready => frame_ready_bypass, frame_out => frame_out_bypass);

u4: TSR      port map (clk => clk , rst =>rst, paritat =>
paritat_aux, tamany_paraula=>tamany_paraula_aux,
num_stop=>num_stop_aux, funcionament=>funcionament_aux, U2X=>U2X,
UCSZ2=> UCSZ2,
                registre_baud => registre_baud, frame_in =>
frame_out_bypass,  frame_ready => frame_ready_bypass, rx_en =>
rx_en, data_out => data_out_tsr , tsr_empty => tsr_empty_tsr,
clk_ext=>clk_ext);

data_serie<=data_out_tsr;
buffer_buit<=buffer_buit_aux;
buffer_ple<=buffer_ple_aux;
error_overflow<=frame_ready_bypass;
```

A.2 TB *PortMap* Transmissor

```
1
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.std_logic_unsigned.all;
5
6  entity TB_TX_FULL is
7  end TB_TX_FULL;
8
9
10 architecture Behavior of TB_TX_FULL is
11 component TX_FULL                                    -- Declarem
12 entrades i sortides del component descrit
13        port(   clk, rst: in std_logic;
```

```
14                registre_estat: inout std_logic_vector (7 downto
15 0);
16                U2X: in std_logic;
17                UCSZ2: in std_logic;
18                TXB8: in std_logic;
19                registre_control: in std_logic_vector (7 downto 0);
20                registre_baud: in std_logic_vector(2 downto 0);
21                data_in: in std_logic_vector (7 downto 0);
22                write_en: in std_logic;
23                rx_en: in std_logic;
24
25                buffer_ple, buffer_buit, error_overflow: out
26 std_logic;
27                data_serie: out std_logic);
28
29 end component;
30
31        signal rst:std_logic;
32        signal clk:std_logic;
33        signal registre_estat: std_logic_vector (7 downto 0):=
34 "00000000";
35        signal registre_control: std_logic_vector (7 downto 0):=
36 "10010111";
37        signal U2X: std_logic:='0';
38        signal UCSZ2: std_logic:='0';
39        signal TXB8: std_logic:='0';
40        signal registre_baud: std_logic_vector (2 downto 0) :=
41 "010";
42        signal data_in: std_logic_vector (7 downto 0);
43        signal write_en: std_logic;
44        signal rx_en: std_logic:='1';
45        signal data: std_logic_vector (7 downto 0);
46        signal buffer_ple: std_logic;
47        signal buffer_buit: std_logic;
48        signal error_overflow: std_logic;
49        signal data_serie: std_logic;
50
51
52        signal data_frame: std_logic_vector (11 downto 0);
53        constant CLK_PERIOD: time := 20 ns;
54
55
56 --TX_buffer
57 signal data_out_buffer: std_logic_vector (7 downto 0);
58 signal data_nova_buffer: std_logic;
59
60 --Paritat
```

```
signal paritat_ready_paritat: std_logic;
signal paritat_empty_paritat: std_logic;
signal paritat_out_paritat: std_logic;

--Bypass
signal frame_out_bypass: std_logic_vector (11 downto 0);
signal bypass_empty_bypass: std_logic;
signal frame_ready_bypass: std_logic;

--TSR
signal tsr_empty_tsr: std_logic;



        begin                               -- Connectem (relacionem)
entrades i sortides amb les senyals.
        UUT: TX_FULL port map
        (clk => clk, rst => rst, registre_estat=>registre_estat,
registre_control=>registre_control, registre_baud =>
registre_baud, data_in => data_in, write_en => write_en,
          buffer_ple=>buffer_ple, buffer_buit=>buffer_buit,
error_overflow=>error_overflow, data_serie => data_serie,
        rx_en=>rx_en, U2X=>U2X, UCSZ2=>UCSZ2, TXB8=>TXB8 );

-- Donem impuls a les entrades

        CLK_PROCESS : process
        begin
        clk <= '0';
        wait for CLK_PERIOD/2;
        clk <= '1';
        wait for CLK_PERIOD/2;
        end process;

        RST_PROC : process
        begin
        rst <='1';
        wait for 1 ms;
        rst <='0';
        wait;
        end process;

        DATA_IN_PROC: process
        begin
        data_in<="00000000";
        wait for 2 ms;
        data_in<="01010110";
```

```
108         write_en<='1';
109         wait for 10 ns;
110         write_en<='0';
111         wait for 100 us;
112         data_in<="01010110";
113         write_en<='1';
            wait for 10 ns;
            write_en<='0';
            wait for 100 us;
            data_in<="01010110";
            write_en<='1';
            wait for 10 ns;
            write_en<='0';
            wait;
            end process;

    end Behavior;
```

A.3 TX Buffer

```
 1
 2 library IEEE;
 3 use IEEE.std_logic_1164.all;
 4 use IEEE.numeric_std.all;
 5 use IEEE.std_logic_unsigned.all;
 6 use IEEE.STD_LOGIC_ARITH.ALL;
 7
 8
 9 entity TX_BUFFER is
10         port(
11                 rst, clk: in std_logic;
12
13                 paraula_in: in std_logic_vector (7 downto 0); --
14 Equivale al paraula_out del rsr
15                 write_en: in std_logic;
16
17                 tsr_empty: in std_logic;
18                 paritat_empty: in std_logic;
19                 rx_en: in std_logic;
20
21                 data: out std_logic_vector(7 downto 0);
22                 data_nova: out std_logic;
23
24
```

```
25                buffer_ple: out std_logic;  --indica al modul
26 extern don vinguin les dades que el buffer no pot rebre res mes i
27 es perdran dades
28                buffer_buit: out std_logic; --serveix per indicar
29 al receptor que no tenim cap paraula per enviar
30                error_overflow: out std_logic);
31 end TX_BUFFER;
32
33 architecture CODE of TX_BUFFER  is
34
35 --ram
36 type ram_type is array (0 to 8) of std_logic_vector (7 downto 0);
37         signal ram_aux : ram_type;
38
39 signal data_aux: std_logic_vector (7 downto 0);
40 signal data_nova_aux: std_logic;
41
42 --punteros
43 signal puntero_in_aux: std_logic_vector (4 downto 0);
44 signal puntero_out_aux: std_logic_vector (4 downto 0);
45 signal resultat_aux: std_logic_vector (4 downto 0);
46
47 type tipus is (inici, enable, fi);
48 signal estat:tipus;
49
50 type tipus_2 is (inici_2, enable_2, fi_2);
51 signal estat_2:tipus_2;
52
53 signal buffer_ple_aux: std_logic;
54 signal buffer_buit_aux: std_logic;
55 signal error_overflow_aux: std_logic;
56
57 begin
58
59 -------------- REBEM PARAULA ----------------
60 process (clk)
61 variable i:integer := 0;
62 begin
63 if (clk'event and clk='1') then
64         if (rst='1') then
65                 puntero_in_aux<="00000";
66                 i :=0;
67                 error_overflow_aux<='0';
68                 ram_aux (0)<= "00000000";
69                 ram_aux (1)<= "00000000";
70                 ram_aux (2)<= "00000000";
71                 ram_aux (3)<= "00000000";
```

```
72                ram_aux (4)<= "00000000";
73                ram_aux (5)<= "00000000";
74                ram_aux (6)<= "00000000";
75                ram_aux (7)<= "00000000";
76                estat<= inici;
77        else
78                case estat is
79                        when inici=>
80                                if (buffer_ple_aux='0') then
81                                        estat<= enable;
82                                end if;
83                        when enable =>
84                                if (write_en = '1') then
85                                        ram_aux(i)<= paraula_in;
86                                        puntero_in_aux <=
87 puntero_in_aux + '1';
88                                        i := i+1;
89                                        if (puntero_in_aux >=
90 "01000") then
91                                                puntero_in_aux<=
92 "00000";
93                                                i:=0;
94                                        end if;
95                                                estat<=fi;
96                                end if;
97                        when fi=>
98                                if (write_en = '0') then --Realitzem
99 aixo per que no guardi molts cops les mateixes dades. Es demanara
100 un impuls de READ_EN igual a dues vegades la senyal de rellotje
101                                        estat<= inici;
102                               end if;
103               end case;
104       end if;
105 end if;
106 end process;
107
108 -------------- ENVIEM PARAULA ----------------
109 process (clk)
110 variable j:integer := 0;
111 begin
112 if (clk'event and clk='1') then
113       if (rst='1') then
114                puntero_out_aux<="00000";
115                data_aux<= "00000000";
116                data_nova_aux<='0';
117                j:=0;
118                estat_2<=inici_2;
```

```
119         else
120                 case estat_2 is
121                         when inici_2=>
122                                 data_nova_aux<='0';
123                                 if (buffer_buit_aux = '0') then
124                                         estat_2<= enable_2;
125                                 end if;
126                         when enable_2 =>
127                                 if (paritat_empty = '1' and
128 tsr_empty='1' and rx_en = '1') then
129                                         data_aux<=ram_aux(j);
130                                         data_nova_aux<='1';
131
132         puntero_out_aux<=puntero_out_aux + '1';
133                                         j:=j+1;
134                                         if (puntero_out_aux >=
135 "01000") then
136
137         puntero_out_aux<="00000";
138                                                 j:=0;
139                                         end if;
140                                         estat_2<=fi_2;
141                                 end if;
142                         when fi_2=>
143                                 if (paritat_empty = '0' and
144 tsr_empty='0') then --Realitzem aixo per que no guardi molts cops
145 les mateixes dades. Es demanara un impuls de READ_EN igual a dues
146 vegades la senyal de rellotje
147                                         estat_2<= inici_2;
148                                 end if;
149                 end case;
150         end if;
151 end if;
152 end process;
153
154 process(clk)
155 begin
156 if (clk'event and clk='1') then
157         if (rst='1') then
158                 resultat_aux <="00000";
159         else
160                 if (puntero_in_aux > puntero_out_aux) then
161                         resultat_aux<= puntero_in_aux -
162 puntero_out_aux;
163                 elsif (puntero_in_aux = puntero_out_aux) then
164                         resultat_aux<="00000";
165                 else
```

```
                        resultat_aux<= (("01000"+ puntero_in_aux) -
    puntero_out_aux);
                end if;
            end if;
    end if;
    end process;

    with resultat_aux select
                buffer_ple_aux <= '1' when "01000",
                                  '0' when others;
    with resultat_aux select
                buffer_buit_aux <= '1' when "00000",
                                   '0' when others;

    data_nova<=data_nova_aux;
    buffer_buit<=buffer_buit_aux;
    buffer_ple<=buffer_ple_aux;
    data<=data_aux;

    end CODE;
```

A.4 TB TX Buffer

```
-- Declarem les llibreries que utilizarem.
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity TB_TX_BUFFER is
end TB_TX_BUFFER;

architecture CODI of TB_TX_BUFFER is
        component TX_BUFFER                                    --
Declarem entrades i sortides del component descrit
        port(
                rst, clk: in std_logic;

                paraula_in: in std_logic_vector (7 downto 0); --
Equivale al paraula_out del rsr
                write_en: in std_logic;

                tsr_empty: in std_logic;
                paritat_empty: in std_logic;
                rx_en: in std_logic;
```

```
                data: out std_logic_vector(7 downto 0);
                data_nova: out std_logic;

                buffer_ple: out std_logic;  --indica al modul extern
don vinguin les dades que el buffer no pot rebre res mes i es perdran
dades
                buffer_buit: out std_logic; --serveix per indicar al
receptor que no tenim cap paraula per enviar
                error_overflow: out std_logic);

        end component;

        signal rst:std_logic;
        signal clk:std_logic;
        signal paraula_in: std_logic_vector (7 downto 0);
        signal write_en: std_logic;
        signal tsr_empty: std_logic;
        signal rx_en: std_logic;
        signal paritat_empty: std_logic;
        signal data: std_logic_vector (7 downto 0);
        signal data_nova: std_logic;
        signal buffer_ple: std_logic;
        signal buffer_buit: std_logic;
        signal error_overflow: std_logic;
        constant CLK_PERIOD: time := 10 ns;

        begin                           -- Connectem (relacionem)
entrades i sortides amb les senyals.
        UUT: TX_BUFFER port map
        (clk => clk, rst => rst,  paraula_in=>paraula_in, write_en =>
write_en, tsr_empty => tsr_empty, paritat_empty => paritat_empty,
rx_en=>rx_en,
          data=>data, data_nova=>data_nova, buffer_ple=>buffer_ple,
buffer_buit => buffer_buit, error_overflow=> error_overflow);

-- Donem impuls a les entrades

        CLK_PROCESS : process
        begin
        clk <= '0';
        wait for CLK_PERIOD/2;
        clk <= '1';
        wait for CLK_PERIOD/2;
        end process;

        RST_PROC : process
        begin
```

```
rst <='1';
wait for 1 ms;
rst <='0';
wait;
end process;

DATA_IN_PROC: process
begin
paraula_in<="00000000";
write_en<= '0';
tsr_empty<='0';
paritat_empty<='0';
wait for 2 ms;
paraula_in<="01010110";
write_en<='1';
wait for 10 ns;
write_en<='0';
wait for  500000 ns;
paraula_in<="01111110";
write_en<='1';
wait for 10 ns;
write_en<='0';
wait for 500000 ns;
paraula_in<="01011110";
write_en<='1';
wait for 10 ns;
write_en<='0';
wait for  500000 ns;
paraula_in<="01000110";
write_en<='1';
wait for 10 ns;
write_en<='0';
wait for  500000 ns;
paraula_in<="01000110";
write_en<='1';
wait for 10 ns;
write_en<='0';
wait for  500000 ns;
paraula_in<="01000110";
write_en<='1';
wait for 10 ns;
write_en<='0';
wait for  500000 ns;
paraula_in<="01111110";
write_en<='1';
wait for 10 ns;
write_en<='0';
```

```
        wait for  500000 ns;
        paraula_in<="01000000";
        write_en<='1';
        wait for 10 ns;
        write_en<='0';
        wait for  500000 ns;
        paraula_in<="11100000";
        write_en<='1';
        wait for 10 ns;
        write_en<='0';
        wait for 1 ms;

        paraula_in<="00000000";
        wait for 500 us;
        paritat_empty<='1';
        tsr_empty<='1';
        rx_en<='1';
        wait for 10 ns;
        paritat_empty<='0';
        tsr_empty<='0';
        rx_en<='0';
        wait for 1 ms;
        paritat_empty<='1';
        tsr_empty<='1';
        rx_en<='1';
        wait for 10 ns;
        paritat_empty<='0';
        tsr_empty<='0';
        rx_en<='0';
        wait;
        end process;

end CODI;
```

A.5 Càlcul de paritat

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.std_logic_unsigned.all;
4
5 entity PARITAT is
6 port(          clk, rst: in std_logic;
7                paritat_tipo: in std_logic_vector (1 downto 0);
8                tamany_paraula: in std_logic_vector (1 downto 0);
```

```
 9                data_in: in std_logic_vector (7 downto 0); --
10 arriva des de tx_buffer
11                data_nova: in std_logic; --variable creada per
12 TX_buffer.
13                paritat_ready: out std_logic;
14                paritat_empty: out std_logic;
15                paritat_out: out std_logic);
16 end PARITAT;
17
18 architecture  CODI of PARITAT is
19
20 signal compt_paraula_aux: std_logic_vector(3 downto 0);
21 signal compt_paritat_aux: std_logic_vector (1 downto 0);
22 signal tamany: std_logic_vector(3 downto 0);
23 signal tipo_paritat: std_logic;
24
25 signal paritat_aux:std_logic;
26 signal paritat_ready_aux:std_logic;
27 signal paritat_empty_aux:std_logic;
28
29 type tipus is (inici, calcular,fi);
30 signal estat : tipus;
31
32 begin
33
34 with tamany_paraula select
35                tamany <= "0101" when "00",
36                          "0110" when "01",
37                          "0111" when "10",
38                          "1000" when others;
39
40 tipo_paritat<=paritat_tipo(0);
41
42
43 process (clk)
44 variable i:integer := 0;
45 variable j:integer := 0;
46 begin
47 if (clk'event and clk='1') then
48        if (rst='1') then
49                paritat_aux<='0';
50                compt_paritat_aux<="00";
51                compt_paraula_aux<="0000";
52                paritat_empty_aux<='1';
53                paritat_ready_aux<='0';
54                i:=0;
55                estat<= inici;
```

```
56          else
57                  case estat is
58                          when inici=>
59                                  compt_paritat_aux<="00";
60                                  compt_paraula_aux<="0000";
61                                  paritat_empty_aux<='1';
62                                  paritat_ready_aux<='0';
63                                  i:=0;
64                                  if (data_nova='1') then
65                                          paritat_aux<='0';
66                                          estat<=calcular;
67                                          paritat_empty_aux<='0';
68                                  end if;
69
70                          when calcular=>
71
72          compt_paraula_aux<=compt_paraula_aux+'1';
73                                  if (compt_paraula_aux<tamany) then
74                                          if (data_in(i)='1') then
75
76          compt_paritat_aux<=compt_paritat_aux+'1';
77                                          end if;
78                                          i:=i+1;
79                                  else
80                                          compt_paraula_aux<="0000";
81                                          i:=0;
82                                          estat<=fi;
83                                  end if;
84                          when fi=>
85                                  if (tipo_paritat='1') then --PARTIAT
86 PAR
87                                          if ((compt_paritat_aux="01")
88 or (compt_paritat_aux="11")) then --Calculem paritat par
89                                                  paritat_aux<='1';
90                                          else
91                                                  paritat_aux<='0';
92                                          end if;
93                                  else ---PARITAT IMPAR
94                                          if ((compt_paritat_aux="00")
95 or (compt_paritat_aux="10")) then --Calculem paritat par
96                                                  paritat_aux<='1';
97                                          else
98                                                  paritat_aux<='0';
99                                          end if;
100                                 end if;
101                                 paritat_ready_aux<='1';
102                                 estat<=inici;
```

```
103                 end case;
104         end if;
    end if;
    end process;

    paritat_out<=paritat_aux;
    paritat_ready<=paritat_ready_aux;
    paritat_empty<=paritat_empty_aux;
    end CODI;
```

A.6 TB Càlcul de paritat

```
1  -- Declarem les llibreries que utilizarem.
 2 library ieee;
 3 use ieee.std_logic_1164.all;
 4 use ieee.std_logic_unsigned.all;
 5
 6 entity TB_PARITAT is
 7 end TB_PARITAT;
 8
 9 -- Describim el funcionament del TB.
10
11 architecture CODI of TB_PARITAT is
12         component PARITAT                                          --
13 Declarem entrades i sortides del component descrit
14         port(
15         clk, rst: in std_logic;
16         paritat_tipo: in std_logic_vector (1 downto 0);
17         tamany_paraula: in std_logic_vector (1 downto 0);
18         data_in: in std_logic_vector (7 downto 0); -- arriva des de
19 tx_buffer
20         data_nova: in std_logic; --variable creada per TX_buffer.
21
22         paritat_ready: out std_logic;
23         paritat_empty: out std_logic;
24         paritat_out: out std_logic
25 );
26         end component;
27
28         signal rst:std_logic;
29         signal clk:std_logic;
30         signal paritat_tipo: std_logic_vector (1 downto 0):="10";
31         signal tamany_paraula: std_logic_vector (1 downto 0):=
32 "11";
```

```
33          signal data_in: std_logic_vector (7 downto 0);
34          signal data_nova: std_logic;
35          signal paritat_ready: std_logic;
36          signal paritat_empty: std_logic;
37          signal paritat_out: std_logic;
38          constant CLK_PERIOD: time := 10 ns;
39
40          begin                           -- Connectem (relacionem)
41 entrades i sortides amb les senyals.
42          UUT: PARITAT port map
43          (clk => clk, rst => rst, paritat_tipo => paritat_tipo,
44 tamany_paraula => tamany_paraula, data_in=>data_in, data_nova =>
45 data_nova, paritat_ready => paritat_ready,
46          paritat_empty=>paritat_empty, paritat_out=>paritat_out);
47
48          -- Donem impuls a les entrades
49
50          CLK_PROCESS : process
51          begin
52          clk <= '0';
53          wait for CLK_PERIOD/2;
54          clk <= '1';
55          wait for CLK_PERIOD/2;
56          end process;
57
58          RST_PROC : process
59          begin
60          rst <='1';
61          wait for 1 ms;
62          rst <='0';
63          wait;
64          end process;
65
66
67          DATA_IN_PROC: process
68          begin
69          data_in<="00000000";
70          data_nova<='0';
71          wait for 5 ms;
72          data_in<="01111100";
73          data_nova<='1';
74          wait for 10 ns;
75          data_nova<='0';
76          wait for 1 ms;
77          data_in<="01010110";
78          data_nova<='1';
            wait for 10 ns;
```

```
        data_nova<='0';
        wait;
        end process;

end CODI;
```

A.7 Generador de *frame* complert

```
 1 library ieee;
 2 use ieee.std_logic_1164.all;
 3 use ieee.std_logic_unsigned.all;
 4
 5 entity BYPASS is
 6 port(
 7                clk, rst: in std_logic;
 8                paritat: in std_logic_vector (1 downto 0);
 9                tamany_paraula: in std_logic_vector (1 downto 0);
10                num_stop: in std_logic;
11                UCSZ2: in std_logic;
12                TXB8: in std_logic;
13                data_in: in std_logic_vector (7 downto 0); --
14 arriva des de tx_buffer
15                paritat_in: in std_logic; --variable creada per
16 PARITY.
17                paritat_ready: in std_logic;
18
19                bypass_empty: out std_logic;
20                frame_ready: out std_logic;
21                frame_out: out std_logic_vector (11 downto 0));
22 end BYPASS;
23
24 architecture  CODI of BYPASS is
25
26 signal compt_paraula_aux: std_logic_vector(3 downto 0);
27 signal tamany: std_logic_vector(3 downto 0);
28 signal bits_stop: std_logic_vector (1 downto 0);
29 signal paritat_act: std_logic;
30
31 type tipus is (inici, invertir, ajuntar, fi);
32 signal estat:tipus;
33
34 signal data_inv: std_logic_vector (7 downto 0);
35 signal inv_en: std_logic;
36 signal invert_ready:std_logic;
```

```
37
38 signal frame_out_aux: std_logic_vector (11 downto 0);
39 signal frame_ready_aux: std_logic;
40 signal bypass_empty_aux: std_logic;
41
42
43 begin
44
45 with tamany_paraula select
46                 tamany <= "0101" when "00",
47                           "0110" when "01",
48                           "0111" when "10",
49                           "1000" when others;
50 with num_stop select
51                 bits_stop <= "01" when '0',
52                                 "11" when others;
53 with paritat select
54                 paritat_act <= '0' when "01",
55                                 '0' when "00",
56                                 '1' when others;
57
58 process (clk)
59 variable i: integer :=0;
60 begin
61 if (clk'event and clk='1') then
62         if (rst = '1') then
63                 frame_out_aux<="111111111111";
64                 frame_ready_aux<='0';
65                 bypass_empty_aux<='1';
66                 inv_en<='0';
67                 i:=0;
68                 frame_ready_aux<='0';
69         else
70                 case estat is
71                         when inici=>
72                                 frame_ready_aux<='0';
73                                 bypass_empty_aux<='1';
74                                 i:=0;
75                                 if (paritat_ready = '1') then
76                                         estat<= invertir;
77                                         bypass_empty_aux<='0';
78                                 end if;
79                         when invertir =>
80                                 inv_en<='1';
81                                 if (invert_ready = '1') then
82                                         estat<=ajuntar;
83                                         inv_en<='0';
```

```
84                                  end if;
85                          when ajuntar=>
86                                          if (num_stop = '1') then
87                                                  frame_out_aux(1
88 downto 0)<="11";
89                                                  i:=2;
90                                          else
91
92          frame_out_aux(0)<='1';
93                                                  i:=1;
94                                          end if;
95
96                                          if (UCSZ2 = '1') then
97
98          frame_out_aux(i)<=TXB8;
99                                                  i:= i+1;
100                                         else
101                                                 if (paritat_act =
102 '1') then
103
104         frame_out_aux(i)<=paritat_in;
105                                                         i:=i+1;
106                                                 end if;
107                                         end if;
108
109                                         frame_out_aux (i+7 downto i)
110 <= data_inv;
111
112                                         if (tamany = "0101") then
113                                                 frame_out_aux(i+5)<=
114 '0';
115                                         elsif (tamany = "0110") then
116                                                 frame_out_aux(i+6)<=
117 '0';
118                                         elsif (tamany = "0111") then
119                                                 frame_out_aux(i+7)<=
120 '0';
121                                         else
122                                                 frame_out_aux(i+8)<=
123 '0';
124                                         end if;
125                                         estat<=fi;
126                         when fi=>
127                                 bypass_empty_aux<='1';
128                                 frame_ready_aux<='1';
129                                 estat<=inici;
130
```

```
131                 end case;
132         end if;
133 end if;
134 end process;
135
136 process(clk)
137 variable z: integer :=0;
138 variable j: integer :=7;
139 begin
140 if (clk'event and clk='1') then
141         if (rst='1') then
142                 data_inv <="00000000";
143                 z:=0;
144                 j:=7;
145                 invert_ready<='1';
146         else
147                 if (inv_en = '1') then
148                         if (z<8) then
149                                 data_inv(z)<=data_in(j);
150                                 z:= z+1;
151                                 j:= j-1;
152                         else
153                                 z:=0;
154                                 j:=7;
155                                 invert_ready<='1';
                        end if;
                else
                        invert_ready<='0';
                end if;
        end if;
    end if;
    end process;

    frame_out<=frame_out_aux;
    frame_ready<=frame_ready_aux;
    bypass_empty<=bypass_empty_aux;

    end codi;
```

A.8 TB Generador de *frame* complert

```
1  -- Declarem les llibreries que utilizarem.
 2 library ieee;
 3 use ieee.std_logic_1164.all;
```

```
 4 use ieee.std_logic_unsigned.all;
 5
 6
 7 -- Definim l'enitat.
 8
 9 entity TB_BYPASS is
10 end TB_BYPASS;
11
12 -- Describim el funcionament del TB.
13
14 architecture CODI of TB_BYPASS is
15 component BYPASS                                          -- Declarem
16 entrades i sortides del component descrit
17         port(
18                 clk, rst: in std_logic;
19                 paritat: in std_logic_vector (1 downto 0);
20                 tamany_paraula: in std_logic_vector (1 downto 0);
21                 num_stop: in std_logic;
22                 UCSZ2: in std_logic;
23                 TXB8: in std_logic;
24                 data_in: in std_logic_vector (7 downto 0); -- arriva
25 des de tx_buffer
26                 paritat_in: in std_logic; --variable creada per
27 PARITY.
28                 paritat_ready: in std_logic;
29                 bypass_empty: out std_logic;
30                 frame_ready: out std_logic;
31
32                 frame_out: out std_logic_vector (11 downto 0));
33
34         end component;
35
36         signal rst:std_logic;
37         signal clk:std_logic;
38         signal paritat: std_logic_vector (1 downto 0) := "00";
39         signal tamany_paraula: std_logic_vector (1 downto 0) :=
40 "11";
41         signal num_stop: std_logic := '0';
42         signal  UCSZ2: std_logic;
43         signal  TXB8: std_logic;
44         signal data_in: std_logic_vector (7 downto 0);
45         signal paritat_in: std_logic;
46         signal paritat_ready: std_logic;
47         signal bypass_empty: std_logic;
48         signal frame_ready: std_logic;
49         signal frame_out: std_logic_vector(11 downto 0);
50         constant CLK_PERIOD: time := 10 ns;
```

```
51
52
53          begin                                -- Connectem (relacionem)
54 entrades i sortides amb les senyals.
55          UUT: BYPASS port map
56          (clk => clk, rst => rst, paritat => paritat, tamany_paraula
57 => tamany_paraula, num_stop=>num_stop,  UCSZ2=>UCSZ2, TXB8=>TXB8,
58 data_in=>data_in, paritat_in => paritat_in, bypass_empty =>
59 bypass_empty,
60          paritat_ready => paritat_ready,
61          frame_ready=>frame_ready, frame_out => frame_out);
62
63          -- Donem impuls a les entrades
64
65          CLK_PROCESS : process
66          begin
67          clk <= '0';
68          wait for CLK_PERIOD/2;
69          clk <= '1';
70          wait for CLK_PERIOD/2;
71          end process;
72
73          RST_PROC : process
74          begin
75          rst <='1';
76          wait for 1 ms;
77          rst <='0';
78          wait;
79          end process;
80
81          DATA_IN_PROC: process
82          begin
83          data_in<="00000000";
84          paritat_in<='0';
85          paritat_ready<='0';
86          UCSZ2<='0';
87          TXB8<='0';
88          wait for 5 ms;
89          data_in<="01001011";
90          paritat_in<='1';
91          paritat_ready<='1';
92          wait for 10 ns;
93          paritat_ready<='0';
94          wait for 50000 ns;
95          data_in<="00001111";
96          paritat_in<='1';
97          UCSZ2<='0';
```

```
        TXB8<='0';
        wait for 1000 ns;
        paritat_ready<='1';
        wait for 10 ns;
        paritat_ready<='0';
        wait;
        end process;

end CODI;
```

A.9 TSR

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4
5  entity TSR is
6  port(
7          clk, rst: in std_logic;
8          paritat: in std_logic_vector (1 downto 0);
9          tamany_paraula: in std_logic_vector (1 downto 0);
10         num_stop: in std_logic;
11         funcionament: in std_logic;
12         UCSZ2: in std_logic;
13         U2X: in std_logic;
14         registre_baud: in std_logic_vector(2 downto 0);
15         frame_in: in std_logic_vector(11 downto 0);
16         frame_ready: in std_logic;
17         rx_en: in std_logic;
18
19         tsr_empty: out std_logic;
20         data_out: out std_logic;
21         clk_ext: out std_logic);
22
23 end TSR;
24
25 architecture  CODI of TSR is
26
27 signal compt_paraula_aux: std_logic_vector(3 downto 0);
28 signal tamany: std_logic_vector(3 downto 0);
29 signal bits_stop: std_logic_vector (1 downto 0);
30 signal paritat_act: std_logic;
31
32 signal registre_prescaler: std_logic_vector (10 downto 0); --
33 almacenaremos el nÃºmero en funciÃ³n d ela velocidad escogida
34 signal prescaler_16_aux: std_logic;
```

```
35 signal prescaler_8_aux: std_logic;
36 signal prescaler_aux: std_logic;
37 signal prescaler_baud_aux: std_logic;
38 signal compt_baud: std_logic_vector (10 downto 0);
39 signal compt_16_aux: std_logic_vector (4 downto 0);
40 signal compt_8_aux: std_logic_vector (4 downto 0);
41 signal inici_baud_aux: std_logic;
42 signal compt_paraula: std_logic_vector(3 downto 0);
43
44 signal desfas:std_logic;
45
46 signal tsr_empty_aux: std_logic;
47 signal clk_ext_aux: std_logic;
48
49 type tipus is (inici, ple, calcular_mida, transmet, fi);
50 signal estat:tipus;
51
52 signal data_out_aux: std_logic;
53
54 ----------------------------SINCRON-------------------------------
55 --
56
57
58
59 begin
60
61 with tamany_paraula select
62                 tamany <= "0101" when "00",
63                           "0110" when "01",
64                           "0111" when "10",
65                           "1000" when others;
66
67 with registre_baud select
68                 registre_prescaler <= "10100010110" when "000",
69                                       "01010001011" when "001",
70                                       "00101000101" when "010",
71                                       "00011011001" when "011",
72                                       "00010100010" when "100",
73                                       "00001101100" when "101",
74                                       "00001010001" when "110",
75                                       "00000110110" when others;
76
77 paritat_act<=paritat(1);
78
79
80 with num_stop select
81                 bits_stop <= "01" when '0',
```

```
                                "11" when others;
---------------SINCRONITZAR EN MODE ASINCRON ELS DOS DISPOSITUS
AMB EL BAUD RATE----------------------------------------
process(clk)
begin
if (clk'event and clk='1') then
        if (rst='1') then
                prescaler_baud_aux<='0';
                compt_baud<="00000000000";
        else
                if (inici_baud_aux ='1') then
                        compt_baud<=compt_baud+'1';
                        if (compt_baud=registre_prescaler) then
                                prescaler_baud_aux<='1';
                                compt_baud<="00000000000";
                        else
                                prescaler_baud_aux<='0';
                        end if;
                else

                end if;
        end if;
end if;
end process;

process(clk)
begin
if (clk'event and clk='1') then
        if (rst='1') then
                prescaler_16_aux<='0';
                compt_16_aux<="00000";
        else
                prescaler_16_aux<='0';
                if (prescaler_baud_aux='1') then
                        if (compt_16_aux < "01111") then
                                compt_16_aux<=compt_16_aux+'1';
                                prescaler_16_aux<='0';
                        else
                                compt_16_aux<="00000";
                                prescaler_16_aux<='1';
                        end if;
                end if;
        end if;
```

```
129 end if;
130 end process;
131
132 process(clk)
133 begin
134 if (clk'event and clk='1') then
135         if (rst='1') then
136                 prescaler_8_aux<='0';
137                 compt_8_aux<="00000";
138         else
139                 prescaler_8_aux<='0';
140                 if (prescaler_baud_aux='1') then
141                         if (compt_8_aux < "00111") then
142                                 compt_8_aux<=compt_8_aux+'1';
143                                 prescaler_8_aux<='0';
144                         else
145                                 compt_8_aux<="00000";
146                                 prescaler_8_aux<='1';
147                         end if;
148                 end if;
149         end if;
150 end if;
151 end process;
152
153
154 with U2X select
155                 prescaler_aux <= prescaler_16_aux when '0',
156                                 prescaler_8_aux when others;
157
158 with funcionament select
159                 clk_ext_aux <= '0' when '0',
160                         prescaler_aux when others;
161 -----------------------------------
162
163
164 process (clk)
165 variable i:integer := 0;
166 begin
167 if (clk'event and clk='1') then
168         if (rst='1') then
169                 tsr_empty_aux<='1';
170                 data_out_aux<='1';
171                 inici_baud_aux<='0';
172                 i:=0;
173                 desfas<='0';
174                 estat<=inici;
175         else
```

```
176                 case estat is
177                                 when inici=>
178                                         data_out_aux<='1';
179                                         tsr_empty_aux<='1';
180                                         inici_baud_aux<='0';
181                                         tsr_empty_aux<='1';
182                                         desfas<='0';
183                                         i:= 0;
184                                         if (frame_ready = '1') then
185                                                 estat<=ple;
186                                                 tsr_empty_aux<='0';
187                                         end if;
188                                 when ple=>
189                                         if (rx_en='1') then
190
191         estat<=calculari_mida;
192                                         end if;
193                                 when calculari_mida=>
194                                         i:= 1;
195                                         if (num_stop = '1') then
196                                                 i:= i+2;
197                                         else
198                                                 i:=i+1;
199                                         end if;
200
201                                         if (paritat_act = '1' or
202 UCSZ2='1') then
203                                                 i:=i+1;
204                                         end if;
205
206                                         if (tamany_paraula = "00")
207 then
208                                                 i:= i + 5;
209                                         elsif (tamany_paraula =
210 "01") then
211                                                 i:= i + 6;
212                                         elsif (tamany_paraula =
213 "10") then
214                                                 i:= i + 7;
215                                         else
216                                                 i:= i + 8;
217                                         end if;
218                                         estat<=transmet;
219                                         i:=i-1;
220                                         desfas<='1';
221                                 when transmet=>
222                                         inici_baud_aux<='1';
```

```
223                                         if (prescaler_aux = '1' and
224 i>0) then
225                                                 if (desfas='1') then
226
227                                                         desfas<='0';
228
229         data_out_aux<=frame_in (i);
230                                                 else
231                                                         i:=i-1;
232
233         data_out_aux<=frame_in (i);
234                                                 end if;
235                                         elsif (prescaler_aux = '1'
236 and i=0) then
237
238         data_out_aux<=frame_in (0);
239                                                 estat<=fi;
240                                         end if;
241                                 when fi=>
242                                         i := 0;
243                                         desfas<='0';
                                            data_out_aux<='1';
                                            tsr_empty_aux<='1';
                                            inici_baud_aux<='0';
                                            estat<=inici;
                    end case;
            end if;
    end if;
    end process;

    tsr_empty<=tsr_empty_aux;
    data_out<=data_out_aux;
    clk_ext<=clk_ext_aux;

    end CODI;
```

A.10 TB TSR

```
1  -- Declarem les llibreries que utilizarem.
 2 library ieee;
 3 use ieee.std_logic_1164.all;
 4 use ieee.std_logic_unsigned.all;
 5
 6
```

```
 7 -- Definim l'enitat.
 8
 9 entity TB_TSR is
10 end TB_TSR;
11
12 -- Describim el funcionament del TB.
13
14 architecture CODI of TB_TSR is
15 component TSR                                    -- Declarem entrades
16 i sortides del component descrit
17         port(
18                 clk, rst: in std_logic;
19                 paritat: in std_logic_vector (1 downto 0);
20                 tamany_paraula: in std_logic_vector (1 downto 0);
21                 num_stop: in std_logic;
22                 funcionament: in std_logic;
23                 U2X: in std_logic;
24                 UCSZ2: in std_logic;
25                 registre_baud: in std_logic_vector(2 downto 0);
26                 frame_in: in std_logic_vector(11 downto 0);
27                 frame_ready: in std_logic;
28                 rx_en: in std_logic;
29                 tsr_empty: out std_logic;
30                 data_out: out std_logic;
31                 clk_ext: out std_logic);
32
33         end component;
34
35         signal rst:std_logic;
36         signal clk:std_logic;
37         signal clk_ext: std_logic;
38         signal paritat: std_logic_vector (1 downto 0):="01";
39         signal tamany_paraula: std_logic_vector (1 downto 0):="11";
40         signal num_stop: std_logic:= '0';
41         signal funcionament: std_logic:= '1';
42         signal U2X: std_logic := '0';
43         signal UCSZ2: std_logic :='0';
44         signal registre_baud: std_logic_vector(2 downto 0) :=
45 "010";
46         signal frame_in: std_logic_vector (11 downto 0);
47         signal frame_ready: std_logic;
48         signal rx_en: std_logic;
49         signal tsr_empty: std_logic;
50         signal data_out: std_logic;
51         signal inici_baud: std_logic;
52         constant CLK_PERIOD: time := 20 ns;
53
```

```
54
55          begin                               -- Connectem (relacionem)
56 entrades i sortides amb les senyals.
57          UUT: TSR port map
58          (clk => clk, rst => rst, paritat => paritat, tamany_paraula
59 => tamany_paraula, num_stop => num_stop, funcionament =>
60 funcionament, registre_baud=>registre_baud, frame_in => frame_in,
61 frame_ready => frame_ready,
62          rx_en=>rx_en, tsr_empty => tsr_empty, data_out=>data_out,
63 U2X=>U2X, clk_ext=>clk_ext, UCSZ2=>UCSZ2);
64
65          -- Donem impuls a les entrades
66
67          CLK_PROCESS : process
68          begin
69          clk <= '0';
70          wait for CLK_PERIOD/2;
71          clk <= '1';
72          wait for CLK_PERIOD/2;
73          end process;
74
75          RST_PROC : process
76          begin
77          rst <='1';
78          wait for 1 ms;
79          rst <='0';
80          wait;
81          end process;
82
83
84          DATA_IN_PROC: process
85          begin
86          frame_in<="000000000000";
87          frame_ready<='0';
88          rx_en<='0';
89          wait for 1 ms;
90          frame_in<="110100001011";
91          frame_ready<='1';
92          wait for 20 ns;
93          frame_ready<='0';
94          wait for 30 ns;
95          rx_en<='1';
96          wait for 1.5 ms;
97          frame_in<="110101101011";
98          frame_ready<='1';
99          wait for 20 ns;
            frame_ready<='0';
```

```
        wait for 2500 us;
        rx_en<='0';
        wait;
        end process;

end CODI;
```

A.11 *PortMap* Receptor

```
 1 library ieee;
 2 use ieee.std_logic_1164.all;
 3 use ieee.std_logic_unsigned.all;
 4
 5 entity RX_FULL is
 6 port(
 7         clk, rst: in std_logic;
 8         funcionament_aux, num_stop_aux, U2X, UCSZ2 : in std_logic;
 9         paritat_aux, tamany_paraula_aux: in std_logic_vector (1
10 downto 0);
11         registre_baud: in std_logic_vector(2 downto 0);
12         data_in: in std_logic;
13         clk_ext: in std_logic;
14         read_en: in std_logic;
15
16         data_sampli: out std_logic_vector(11 downto 0);
17         enable_vot, enable_error: out std_logic;
18
19         RXB8: out std_logic;
20         error_frame, error_paritat: out std_logic;
21         data: out std_logic_vector(7 downto 0);
22         buffer_ple, RxC: out std_logic);
23 end RX_FULL;
24
25 architecture Behavioral of RX_FULL  is
26
27 component SAMPLING is
28
29         port(   rst, clk: in std_logic;
30                 registre_baud: in std_logic_vector(2 downto 0);
31                 funcionament: in std_logic;
32                 paritat: in std_logic_vector (1 downto 0);
33                 num_stop: in std_logic;
34                 tamany_paraula: in std_logic_vector (1 downto 0);
35                 data_in: in std_logic;
```

```
36                clk_ext: in std_logic;
37
38                UCSZ2: in std_logic;
39                U2X: in std_logic;
40                muestreo_completo: out std_logic;
41                data_out: out std_logic_vector (11 downto 0));
42        end component;
43
44 component ERROR_DETECTION is
45
46        port (  rst, clk: in std_logic;
47                data_in: in std_logic_vector (11 downto 0);  ---
48 bits en serie que llegan del sampling
49                muestreo_completo: in std_logic;
50                paritat: in std_logic_vector (1 downto 0);
51                num_stop: in std_logic;
52                tamany_paraula: in std_logic_vector (1 downto 0);
53                UCSZ2: in std_logic;
54
55                registre_error: out std_logic_vector (1 downto 0);
56                error_calc: out std_logic);
57        end component;
58
59 component RSR is
60
61        port(   rst, clk: in std_logic;
62                tamany_paraula: in std_logic_vector (1 downto 0);
63                UCSZ2: in std_logic;
64                data_in: in std_logic_vector(11 downto 0);
65                muestreo_completo: in std_logic;
66                registre_errors: in std_logic_vector (1 downto 0);
67                error_calc: in std_logic;
68
69                bit9: out std_logic;
70                rsr_empty, rsr_ready: out std_logic;
71                paraula_out: out std_logic_vector (7 downto 0));
72        end component;
73
74 component RX_BUFFER is
75
76        port (  rst, clk: in std_logic;
77                paraula_in: in std_logic_vector (7 downto 0); --
78 Equivale al paraula_out del rsr
79                rsr_ready: in std_logic;
80                read_en: in std_logic;
81
82                data: out std_logic_vector(7 downto 0);
```

```
 83                buffer_ple: out std_logic;
 84                buffer_buit: out std_logic);
 85        end component;
 86
 87 signal data_in_aux: std_logic;
 88 --Sampling
 89 signal data_out_sampling: std_logic_vector(11 downto 0);
 90 signal muestreo_completo_sampling: std_logic;
 91
 92 --ERROR_DETECTION
 93 signal registre_error_error: std_logic_vector (1 downto 0);
 94 signal error_calc_error: std_logic;
 95
 96 --RSR
 97 signal rsr_empty_rsr: std_logic;
 98 signal paraula_out_rsr: std_logic_vector (7 downto 0);
 99 signal rsr_ready_rsr: std_logic;
100 signal bit9_rsr: std_logic;
101
102 --RX_BUFFER
103 signal data_aux: std_logic_vector (7 downto 0);
104
105
106 begin
107
108 data_in_aux<=data_in;
109
110
111
112 u1: SAMPLING port map                    (clk => clk , rst =>rst,
113 registre_baud=>registre_baud, funcionament => funcionament_aux,
114 paritat => paritat_aux, num_stop => num_stop_aux, tamany_paraula
115 => tamany_paraula_aux,
116                                 data_in => data_in_aux,  clk_ext =>
117 clk_ext, UCSZ2=>UCSZ2,U2X=>U2X, data_out => data_out_sampling,
118 muestreo_completo => muestreo_completo_sampling);
119
120 u2: ERROR_DETECTION port map   (clk => clk , rst =>rst, data_in =>
121 data_out_sampling, paritat => paritat_aux, num_stop =>
122 num_stop_aux, tamany_paraula => tamany_paraula_aux,
123 muestreo_completo => muestreo_completo_sampling,
124                                   UCSZ2=>UCSZ2, registre_error =>
125 registre_error_error, error_calc=>error_calc_error);
126
127 u3: RSR port map               (clk => clk , rst =>rst,
128 tamany_paraula=>tamany_paraula_aux, UCSZ2=>UCSZ2,
    registre_errors=>registre_error_error,
```

```
error_calc=>error_calc_error, data_in => data_out_sampling,
muestreo_completo => muestreo_completo_sampling,
                                rsr_empty => rsr_empty_rsr,
rsr_ready => rsr_ready_rsr, paraula_out => paraula_out_rsr,
bit9=>bit9_rsr);

u4: RX_BUFFER port map                (clk => clk , rst =>rst,
paraula_in => paraula_out_rsr, rsr_ready => rsr_ready_rsr,
read_en => read_en,
                                data => data_aux , buffer_ple =>
buffer_ple, buffer_buit=>RxC);

data<=paraula_out_rsr;
enable_vot<=muestreo_completo_sampling;
data_sampli<=data_out_sampling;
error_frame<=registre_error_error(1);
error_paritat<= registre_error_error(0);
enable_error<=error_calc_error;
RX8<=bit9_rsr;

end Behavioral;
```

A.12 TB *PortMap* Receptor

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.std_logic_unsigned.all;
4
5 entity TB_RX_FULL is
6 end TB_RX_FULL;
7
8 -- Describim el funcionament del TB.
9
10 architecture Behavioral of TB_RX_FULL is
11         component RX_FULL                                    --
12 Declarem entrades i sortides del component descrit
13         port(
14                 clk, rst: in std_logic;
15                 funcionament_aux, num_stop_aux, U2X, UCSZ2 : in
16 std_logic;
17                 paritat_aux, tamany_paraula_aux: in
18 std_logic_vector (1 downto 0);
19                 registre_baud: in std_logic_vector(2 downto 0);
20                 data_in: in std_logic;
```

```
21                clk_ext: in std_logic;
22                read_en: in std_logic;
23
24                data_sampli: out std_logic_vector(11 downto 0);
25                enable_vot, enable_error: out std_logic;
26
27                error_frame, error_paritat: out std_logic;
28                RXB8: out std_logic;
29                data: out std_logic_vector(7 downto 0);
30                buffer_ple, RxC: out std_logic);
31
32        end component;
33        signal rst: std_logic;
34        signal clk:  std_logic;
35        signal funcionament_aux: std_logic :='0';
36        signal num_stop_aux: std_logic :='1';
37        signal U2X: std_logic :='0';
38        signal UCSZ2: std_logic :='0';
39        signal paritat_aux: std_logic_vector (1 downto 0):= "11";
40        signal tamany_paraula_aux: std_logic_vector (1 downto 0):=
41 "11";
42        signal registre_baud: std_logic_vector(2 downto 0) :=
43 "010";
44        signal data_in: std_logic;
45        signal clk_ext: std_logic;
46        signal read_en: std_logic;
47        signal error_frame: std_logic;
48        signal error_paritat: std_logic;
49        signal RXB8: std_logic;
50        signal data: std_logic_vector (7 downto 0);
51        signal buffer_ple: std_logic;
52        signal RxC: std_logic;
53
54        signal data_sampli: std_logic_vector(11 downto 0);
55        signal enable_vot: std_logic;
56        signal enable_error: std_logic;
57
58        constant CLK_PERIOD: time := 20 ns;
59        begin                          -- Connectem (relacionem)
60 entrades i sortides amb les senyals.
61        UUT: RX_FULL port map
62        (clk => clk, rst => rst, funcionament_aux =>
63 funcionament_aux, num_stop_aux => num_stop_aux, U2X => U2X,
64 UCSZ2=>UCSZ2, paritat_aux => paritat_aux,
65         tamany_paraula_aux=>tamany_paraula_aux, registre_baud =>
66 registre_baud, data_in=>data_in, clk_ext => clk_ext, read_en =>
67 read_en, error_frame=>error_frame,
```

```
68          error_paritat=>error_paritat, RXB8=>RXB8, data=>data,
69 buffer_ple=>buffer_ple, RxC=>RxC,
70          data_sampli=>data_sampli, enable_vot=>enable_vot,
71 enable_error=>enable_error);
72
73
74          -- Donem impuls a les entrades
75
76          CLK_PROCESS : process
77          begin
78          clk <= '0';
79          wait for CLK_PERIOD/2;
80          clk <= '1';
81          wait for CLK_PERIOD/2;
82          end process;
83
84          RST_PROC : process
85          begin
86          rst <='1';
87          wait for 1 ms;
88          rst <='0';
89          wait;
90          end process;
91
92
93          DATA_IN_PROC: process
94          begin
95          data_in<='1';
96          wait for 1 ms;
97          data_in<='0';------start bit
98          wait for 1000 ns;
99          clk_ext<='1';
100         wait for 20 ns;
101         clk_ext<='0';
102         wait for 105 us;
103         data_in<='1';  ------inci trama
104         clk_ext<='1';
105         wait for 20 ns;
106         clk_ext<='0';
107         wait for 105 us;
108         data_in<='1';
109         clk_ext<='1';
110         wait for 20 ns;
111         clk_ext<='0';
112         wait for 105 us;
113         data_in<='0';
114         clk_ext<='1';
```

```
115         wait for 20 ns;
116         clk_ext<='0';
117         wait for 105 us;
118         data_in<='0';
119         clk_ext<='1';
120         wait for 20 ns;
121         clk_ext<='0';
122         wait for 105 us;
123         data_in<='1';
124         clk_ext<='1';
125         wait for 20 ns;
126         clk_ext<='0';
127         wait for 105 us;
128         data_in<='0';
129         clk_ext<='1';
130         wait for 20 ns;
131         clk_ext<='0';
132         wait for 105 us;
133         data_in<='0';
134         clk_ext<='1';
135         wait for 20 ns;
136         clk_ext<='0';
137         wait for 105 us;
138         data_in<='0';  ------fi de la trama
139         clk_ext<='1';
140         wait for 20 ns;
141         clk_ext<='0';
142         wait for 105 us;
143         data_in<='0';  ------bit de paritat
144         clk_ext<='1';
145         wait for 20 ns;
146         clk_ext<='0';
147         wait for 105 us;
148         data_in<='0';  ------stop bit
149         clk_ext<='1';
150         wait for 20 ns;
            clk_ext<='0';
            wait for 105 us;
            data_in<='1';
            clk_ext<='1';
            wait for 20 ns;
            clk_ext<='0';
            wait for 105 us;
            data_in<='1';
            wait;
            end process;
```

```
end Behavioral;
```

A.13 Sampling

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

--------------ENTITAT DEL BUFFER---------------------------------
entity SAMPLING is
        port(
                rst, clk: in std_logic;
                registre_baud: in std_logic_vector(2 downto 0);
                funcionament: in std_logic;
                paritat: in std_logic_vector (1 downto 0);
                num_stop: in std_logic;
                tamany_paraula: in std_logic_vector (1 downto 0);
                data_in: in std_logic;
                clk_ext: in std_logic;
                UCSZ2: in std_logic;
                U2X: in std_logic;
                data_out: out std_logic_vector (11 downto 0);
                muestreo_completo: out std_logic);

end SAMPLING;


architecture CODI of SAMPLING is


signal compt_sinc: std_logic_vector (3 downto 0);
signal tamany: std_logic_vector(3 downto 0);
signal bits_stop: std_logic_vector (1 downto 0);
signal paritat_act: std_logic;

signal registre_prescaler: std_logic_vector (10 downto 0); --
almacenaremos el numero en funcion d ela velocidad escogida
signal start_flag_aux:std_logic;
signal prescaler_16_aux: std_logic;
signal prescaler_8_aux: std_logic;
signal prescaler_baud_aux: std_logic;
signal compt_baud: std_logic_vector (10 downto 0);
signal compt_16_aux: std_logic_vector (4 downto 0);
signal compt_8_aux: std_logic_vector (4 downto 0);
```

```
41 signal prescaler_aux: std_logic; --sortida
42
43
44 type tipus is (inici,
45                 asincron, vot, fi_asincron,
46                 sincron, espera, enable, disable, fi_sincron);
47 signal estat:tipus;
48
49 signal data_out_aux: std_logic_vector(11 downto 0);
50 signal muestreo_completo_aux: std_logic;--sortida
51 signal compt_paraula_aux: std_logic_vector(3 downto 0);
52
53 signal bit_muestreado_asin: std_logic;
54 signal bit_muestreado_sinc: std_logic;
55 signal bit_complet: std_logic;
56
57 signal registro_muestreo_aux: std_logic_vector (15 downto 0); --
58 iremos almacenando los bits que se cojan para depsues realizar una
59 comparacion por mayorÃa absoluta
60 signal compt_votos_aux: std_logic_vector (4 downto 0);
61
62
63 ----------------------------SINCRON--------------------------------
64 --
65
66
67
68 begin
69
70 with tamany_paraula select
71                 tamany <= "0101" when "00",
72                           "0110" when "01",
73                           "0111" when "10",
74                           "1000" when others;
75
76 with registre_baud select
77                 registre_prescaler <= "10100010110" when "000",
78                                       "01010001011" when "001",
79                                       "00101000101" when "010",
80                                       "00011011001" when "011",
81                                       "00010100010" when "100",
82                                       "00001101100" when "101",
83                                       "00001010001" when "110",
84                                       "00000110110" when others;
85
86 with num_stop select
87                 bits_stop <= "01" when '0',
```

```
88                                  "11" when others;
89
90 paritat_act<=paritat(1);
91
92
93
94
95 ---------------SINCRONITZAR EN MODE ASINCRON ELS DOS DISPOSITUS
96 AMB EL BAUD RATE-----------------------------------------
97 process(clk)
98 begin
99 if (clk'event and clk='1') then
100         if (rst='1') then
101                 prescaler_baud_aux<='0';
102                 compt_baud<="00000000000";
103         else
104                 if (start_flag_aux ='1') then
105                         compt_baud<=compt_baud+'1';
106                         if (compt_baud=registre_prescaler) then
107                                 prescaler_baud_aux<='1';
108                                 compt_baud<="00000000000";
109                         else
110                                 prescaler_baud_aux<='0';
111                         end if;
112                 else
113
114                 end if;
115         end if;
116 end if;
117 end process;
118
119 process(clk)
120 begin
121 if (clk'event and clk='1') then
122         if (rst='1') then
123                 prescaler_16_aux<='0';
124                 compt_16_aux<="00000";
125         else
126                 prescaler_16_aux<='0';
127                 if (prescaler_baud_aux='1') then
128                         if (compt_16_aux < "01111") then
129                                 compt_16_aux<=compt_16_aux+'1';
130                                 prescaler_16_aux<='0';
131                         else
132                                 compt_16_aux<="00000";
133                                 prescaler_16_aux<='1';
134                         end if;
```

```
135                 end if;
136         end if;
137 end if;
138 end process;
139
140 process(clk)
141 begin
142 if (clk'event and clk='1') then
143         if (rst='1') then
144                 prescaler_8_aux<='0';
145                 compt_8_aux<="00000";
146         else
147                 prescaler_8_aux<='0';
148                 if (prescaler_baud_aux='1') then
149                         if (compt_8_aux < "00111") then
150                                 compt_8_aux<=compt_8_aux+'1';
151                                 prescaler_8_aux<='0';
152                         else
153                                 compt_8_aux<="00000";
154                                 prescaler_8_aux<='1';
155                         end if;
156                 end if;
157         end if;
158 end if;
159 end process;
160
161
162 with U2X select
163                 prescaler_aux <= prescaler_16_aux when '0',
164                                 prescaler_8_aux when others;
165 -------------------------MUESTREO DE LES DADES REBUDES ASINCRON--
166 ------------------------------------------------
167 process(clk)
168 begin
169 if (clk'event and clk='1') then
170         if (rst='1') then
171                 compt_votos_aux<="00000";
172                 bit_muestreado_asin<='1';
173                 bit_complet<='0';
174         else
175                 bit_complet<='0';
176                         if (start_flag_aux='1') then
177                                 if (prescaler_baud_aux ='1') then
178                                         if (data_in = '1') then
179
180
181         compt_votos_aux<=compt_votos_aux+'1';
```

```
182                                     end if;
183                                 end if;
184                                 if (prescaler_aux='1') then
185                                     if (U2X='0') then
186                                         if
187 (compt_votos_aux>"01000") then
188
189         bit_muestreado_asin<='1';
190
191         bit_complet<='1';
192                                         else
193
194         bit_muestreado_asin<='0';
195
196         bit_complet<='1';
197                                         end if;
198
199         compt_votos_aux<="00000";
200                                     else
201                                         if
202 (compt_votos_aux>"00100") then
203
204         bit_muestreado_asin<='1';
205
206         bit_complet<='1';
207                                         else
208
209         bit_muestreado_asin<='0';
210
211         bit_complet<='1';
212                                         end if;
213
214         compt_votos_aux<="00000";
215                                     end if;
216                                 else
217                                     bit_complet<='0';
218                                 end if;
219                             end if;
220         end if;
221 end if;
222 end process;
223
224 process(clk)
225 variable i: integer:=11;
226 begin
227 if (clk'event and clk='1') then
228         if (rst='1') then
```

```
229                 estat<=inici;
230                 start_flag_aux<='0';
231                 compt_paraula_aux<="0000";
232                 data_out_aux<="111111111111";
233                 muestreo_completo_aux<='0';
234                 i:=11;
235         else
236                         case estat is
237                                 when inici=>
238                                         if (funcionament = '0') then
239                                                 estat<=asincron;
240                                         else
241                                                 estat<= sincron;
242                                         end if;
243                                 when asincron=>
244                                         start_flag_aux<='0';
245                                         compt_paraula_aux<="0000";
246                                         muestreo_completo_aux<='0';
247                                         i:=11;
248                                         if (data_in='0') then
249
250         data_out_aux<="111111111111";
251                                                 estat<=vot;
252                                                 start_flag_aux<='1';
253                                         end if;
254                                 when vot=>
255                                         if (bit_complet='1') then
256                                                 if
257 (compt_paraula_aux<(tamany + bits_stop + (paritat(1) or UCSZ2)))
258 then
259
260         compt_paraula_aux<=compt_paraula_aux+'1';
261
262         data_out_aux(i)<=bit_muestreado_asin;
263                                                                 i:=i-1;
264                                                 else
265
266         estat<=fi_asincron;
267                                                 end if;
268                                         end if;
269                                 when fi_asincron=>
270                                         muestreo_completo_aux<='1';
271                                         estat<=asincron;
272
273                                 when sincron=>
274                                         compt_paraula_aux<="0000";
275                                         muestreo_completo_aux<='0';
```

```
276                                     i:=11;
277                                     estat<=espera;
278                             when espera=>
279                                     if (clk_ext = '1') then
280                                             estat<= enable;
281                                     end if;
                                when enable=>
                                        if (compt_paraula_aux<
    (tamany + bits_stop + (paritat(1) or UCSZ2))-'1')  then

            compt_paraula_aux<=compt_paraula_aux+'1';

            data_out_aux(i)<=data_in;
                                                i:=i-1;

            muestreo_completo_aux<='0';
                                                estat<=disable;
                                        else
                                                estat<=fi_sincron;
                                        end if;
                                when disable=>
                                        if (clk_ext='0') then
                                                estat<=espera;
                                        end if;
                                when fi_sincron=>
                                        muestreo_completo_aux<='1';
                                        estat<=sincron;
                        end case;
            end if;
    end if;
    end process;
    muestreo_completo<=muestreo_completo_aux;
    data_out<=data_out_aux;ux;
    compt_votos<=compt_votos_aux;

    end CODI
```

A.14 TB *Sampling*

```
1 -- Declarem les llibreries que utilizarem.
2 library ieee;
3 use ieee.std_logic_1164.all;
4 use ieee.std_logic_unsigned.all;
5
6
7 -- Definim l'enitat.
```

```
 8
 9 entity TB_SAMPLING is
10 end TB_SAMPLING;
11
12 -- Describim el funcionament del TB.
13
14 architecture CODI of TB_SAMPLING is
15         component SAMPLING                                          --
16 Declarem entrades i sortides del component descrit
17         port(
18                 rst, clk: in std_logic;
19                 clk_ext: in std_logic;
20                 registre_baud: in std_logic_vector(2 downto 0);
21                 funcionament: in std_logic;
22                 paritat: in std_logic_vector (1 downto 0);
23                 num_stop: in std_logic;
24                 tamany_paraula:in std_logic_vector(1 downto 0);
25                 data_in: in std_logic;
26                 UCSZ2: in std_logic;
27                 U2X: in std_logic;
28                 data_out: out std_logic_vector (11 downto 0);
29                 muestreo_completo: out std_logic);
30         end component;
31
32         signal rst: std_logic;
33         signal clk:  std_logic;
34         signal clk_ext: std_logic;
35         signal registre_baud: std_logic_vector(2 downto 0) :=
36 "010";
37         signal funcionament: std_logic:='1';
38         signal paritat: std_logic_vector (1 downto 0):="11";
39         signal num_stop: std_logic:='1';
40         signal tamany_paraula: std_logic_vector (1 downto
41 0):="11";
42         signal data_in: std_logic;
43         signal data_out: std_logic_vector(11 downto 0);
44         signal UCSZ2: std_logic:='0';
45         signal U2X:std_logic:='1';
46         signal bit_muestreado: std_logic;
47         signal muestreo_completo: std_logic;
48         constant CLK_PERIOD: time := 20 ns;
49
50         begin                              -- Connectem (relacionem)
51 entrades i sortides amb les senyals.
52         UUT: SAMPLING port map
53         (clk => clk, rst => rst, registre_baud => registre_baud,
54 funcionament => funcionament, paritat => paritat, num_stop =>
```

```
55 num_stop, tamany_paraula => tamany_paraula, data_in => data_in,
56 data_out => data_out, muestreo_completo=>muestreo_completo,
57         U2X=>U2X, clk_ext=>clk_ext, UCSZ2=>UCSZ2);
58
59         -- Donem impuls a les entrades
60
61         CLK_PROCESS : process
62         begin
63         clk <= '0';
64         wait for CLK_PERIOD/2;
65         clk <= '1';
66         wait for CLK_PERIOD/2;
67         end process;
68
69         RST_PROC : process
70         begin
71         rst <='1';
72         wait for 1 ms;
73         rst <='0';
74         wait;
75         end process;
76
77
78         DATA_IN_PROC: process
79         begin
80         data_in<='1';
81         wait for 1 ms;
82         data_in<='0';------start bit
83         wait for 1000 ns;
84         clk_ext<='1';
85         wait for 20 ns;
86         clk_ext<='0';
87         wait for 105 us;
88         data_in<='1';  ------inci trama
89         clk_ext<='1';
90         wait for 20 ns;
91         clk_ext<='0';
92         wait for 105 us;
93         data_in<='1';
94         clk_ext<='1';
95         wait for 20 ns;
96         clk_ext<='0';
97         wait for 105 us;
98         data_in<='0';
99         clk_ext<='1';
100        wait for 20 ns;
101        clk_ext<='0';
```

```
102         wait for 105 us;
103         data_in<='0';
104         clk_ext<='1';
105         wait for 20 ns;
106         clk_ext<='0';
107         wait for 105 us;
108         data_in<='1';
109         clk_ext<='1';
110         wait for 20 ns;
111         clk_ext<='0';
112         wait for 105 us;
113         data_in<='0';
114         clk_ext<='1';
115         wait for 20 ns;
116         clk_ext<='0';
117         wait for 105 us;
118         data_in<='0';
119         clk_ext<='1';
120         wait for 20 ns;
121         clk_ext<='0';
122         wait for 105 us;
123         data_in<='0';  ------fi de la trama
124         clk_ext<='1';
125         wait for 20 ns;
126         clk_ext<='0';
127         wait for 105 us;
128         data_in<='0';  ------bit de paritat
129         clk_ext<='1';
130         wait for 20 ns;
131         clk_ext<='0';
132         --wait for 105 us;
133         --data_in<='0';------stop bit
134         --clk_ext<='1';
135         --wait for 20 ns;
136         --clk_ext<='0';
137         --wait for 105 us;
138         --data_in<='1';
139         --clk_ext<='1';
140         --wait for 20 ns;
            --clk_ext<='0';
            wait for 105 us;
            data_in<='1';
            wait;
            end process;

    end CODI;
```

A.15 Detecció d'errors

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.std_logic_unsigned.all;
4
5 --------------ENTITAT DEL BUFFER--------------------------------
6 entity ERROR_DETECTION is
7         port(
8                 rst, clk: in std_logic;
9                 data_in: in std_logic_vector (11 downto 0);  ---
10 bits en serie que llegan del sampling
11                muestreo_completo: in std_logic;
12                paritat: in std_logic_vector (1 downto 0);
13                num_stop: in std_logic;
14                tamany_paraula: in std_logic_vector (1 downto 0);
15                UCSZ2: in std_logic;
16
17                registre_error: out std_logic_vector (1 downto 0);
18                error_calc: out std_logic);
19
20 end ERROR_DETECTION;
21
22
23 architecture CODE of ERROR_DETECTION is
24
25 signal compt_paraula_aux: std_logic_vector(3 downto 0);
26 signal tamany: std_logic_vector(3 downto 0);
27 signal stop_bit: std_logic_vector (1 downto 0);
28 signal paritat_act: std_logic;
29
30 signal compt_paritat_aux: std_logic_vector (1 downto 0);
31 signal paritat_rebuda_aux: std_logic;
32 signal compt_stop_aux: std_logic_vector(1 downto 0);
33 signal paritat_aux: std_logic;
34
35 type tipus is (inici, paraula, bit_paritat, frame, comparacio,
36 fi);
37 signal estat:tipus;
38
39 signal error_frame:std_logic;
40 signal error_paritat:std_logic;
41 signal error_calc_aux: std_logic;
42
```

```
43 signal data_serie_aux: std_logic;
44
45 begin
46
47
48 with tamany_paraula select
49                 tamany <= "0101" when "00",
50                           "0110" when "01",
51                           "0111" when "10",
52                           "1000" when others;
53
54 with num_stop select
55                 stop_bit <= "01" when '0',
56                         "11" when others;
57
58 paritat_act<=paritat(1);
59
60 --------------------------------CALCULEM EL NUMERO DE '1' QUE REB
61 EL MISSATGE I COMPROVEM QUE ELS BITS DE STOP SIGUIN '1'-----------
62 -------------------
63 process (clk)
64 variable i: integer :=11;
65 variable j: integer :=0;
66 begin
67         if (clk='1' and clk'event) then
68                 if (rst = '1') then
69                         compt_stop_aux<="00";
70                         compt_paritat_aux<="00";
71                         paritat_rebuda_aux<='0';
72                         compt_paraula_aux<="0000";
73                         error_frame<='0';
74                         error_paritat<='0';
75                         data_serie_aux<='1';
76                         i:= 11;
77                 else
78                         case estat is
79                                 when inici=>
80                                         compt_stop_aux<="00";
81                                         compt_paritat_aux<="00";
82                                         compt_paraula_aux<="0000";
83                                         compt_paritat_aux<="00";
84                                         i:= 11;
85                                         j:=0;
86                                         if (muestreo_completo = '1')
87 then
88                                                 error_calc_aux<='0';
89                                                 estat<= paraula;
```

```
 90                                      end if;
 91                                  when paraula=>
 92                                      if
 93 (compt_paraula_aux<=tamany) then
 94
 95          compt_paraula_aux<=compt_paraula_aux+'1';
 96                                                  if (data_in(i) =
 97 '1') then
 98
 99          compt_paritat_aux<=compt_paritat_aux + '1';
100                                                  end if;
101                                                  i:=i-1;
102                                      else
103                                                  estat<=bit_paritat;
104                                      end if;
105                                  when bit_paritat =>
106                                      if (paritat_act='1' or
107 UCSZ2='1') then
108
109          paritat_rebuda_aux<=data_in(i);
110                                                  i:=i-1;
111
112          compt_paraula_aux<=compt_paraula_aux+'1';
113                                                  estat<=frame;
114                                      else
115                                                  estat<=frame;
116                                      end if;
117                                  when frame=>
118                                      if (stop_bit = "01") then
119                                                  if (data_in(i) ='0')
120 then
121
122          error_frame<='1';
123                                                  else
124
125          error_frame<='0';
126
127                                                  end if;
128                                                  estat<=comparacio;
129                                      else
130                                                  if (data_in(i) =
131 '0') then
132
133          error_frame<='1';
134
135          estat<=comparacio;
136                                                  else
```

```
137
138         error_frame<='0';
139
140                                                 end if;
141                                                 if (j< 1) then
142                                                         i:=i-1;
143                                                         j:=1;
144                                                 else
145
146         estat<=comparacio;
147                                                 end if;
148                                         end if;
149                                 when comparacio=>
150                                         if (UCSZ2 = '1' or
151 paritat(1) ='0' ) then
152                                                 error_paritat<='0';
153                                         else
154                                                 if (paritat (0) =
155 '1') then --paritat par
156                                                         if
157 (compt_paritat_aux="11" or compt_paritat_aux="01") then
158                                                                 if
159 (paritat_rebuda_aux = '1') then
160
161         error_paritat <='0';
162                                                                 else
163
164
165         error_paritat<='1';
166                                                                 end
167 if;
168                                                         else
169                                                                 if
170 (paritat_rebuda_aux = '1') then
171
172         error_paritat <='1';
173                                                                 else


            error_paritat<='0';
                                                                end
    if;
                                                        end if;
                                                else            ---
    paritat impar
                                                        if
    (compt_paritat_aux="11" or compt_paritat_aux="01") then
```

```
                                                                    if
    (paritat_rebuda_aux = '0') then

            error_paritat <='0';
                                                                    else


            error_paritat<='1';
                                                                    end
    if;
                                                            else
                                                                    if
    (paritat_rebuda_aux = '0') then

            error_paritat <='1';
                                                                    else


            error_paritat<='0';
                                                                    end
    if;
                                                                end if;
                                                        end if;
                                                    end if;
                                                    estat<=fi;
                                            when fi=>
                                                    error_calc_aux<='1';
                                                    estat<=inici;
                                    end case;
                            end if;
                    end if;
    end process;

    registre_error(0)<=error_paritat;
    registre_error(1)<=error_frame;
    error_calc<=error_calc_aux;
    end CODE;
```

A.16 TB Detecció d'errors

```
1  -- Declarem les llibreries que utilizarem.
 2 library ieee;
 3 use ieee.std_logic_1164.all;
 4 use ieee.std_logic_unsigned.all;
```

```
 5
 6
 7 -- Definim l'enitat.
 8
 9 entity TB_ERROR_DETECTION is
10 end TB_ERROR_DETECTION;
11
12 -- Describim el funcionament del TB.
13
14 architecture CODI of TB_ERROR_DETECTION is
15         component ERROR_DETECTION
16         -- Declarem entrades i sortides del component descrit
17         port(
18                 rst, clk: in std_logic;
19                 data_in: in std_logic_vector (11 downto 0);  ---bits
20 en serie que llegan del sampling
21                 muestreo_completo: in std_logic;
22                 paritat: in std_logic_vector (1 downto 0);
23                 num_stop: in std_logic;
24                 tamany_paraula: in std_logic_vector (1 downto 0);
25                 UCSZ2: in std_logic;
26                 registre_error: out std_logic_vector (1 downto 0);
27                 error_calc: out std_logic);
28         end component;
29
30         signal rst:std_logic;
31         signal clk:std_logic;
32         signal paritat: std_logic_vector (1 downto 0):="01";
33         signal num_stop: std_logic:='1';
34         signal tamany_paraula: std_logic_vector (1 downto 0):=
35 "11";
36         signal UCSZ2: std_logic:='1';
37         signal status_registre: std_logic_vector (7 downto 0);
38         signal data_in: std_logic_vector (11 downto 0);
39         signal muestreo_completo: std_logic;
40         signal registre_error: std_logic_vector (1 downto 0);
41         signal error_calc: std_logic;
42
43         --signal data_serie: std_logic;
44         constant CLK_PERIOD: time := 20 ns;
45
46
47         begin                              -- Connectem (relacionem)
48 entrades i sortides amb les senyals.
49         UUT: ERROR_DETECTION port map
50
51
```

```
52         (clk => clk, rst => rst, paritat => paritat, num_stop =>
53 num_stop, tamany_paraula => tamany_paraula, data_in=>data_in,
54 muestreo_completo => muestreo_completo,
55          registre_error=>registre_error,
56 error_calc=>error_calc,UCSZ2=>UCSZ2);
57
58         -- Donem impuls a les entrades
59
60         CLK_PROCESS : process
61         begin
62         clk <= '0';
63         wait for CLK_PERIOD/2;
64         clk <= '1';
65         wait for CLK_PERIOD/2;
66         end process;
67
68         RST_PROC : process
69         begin
70         rst <='1';
71         wait for 1 ms;
72         rst <='0';
73         wait;
74         end process;
75
76
77         DATA_IN_PROC: process
78         begin
79         data_in<="111111111111";
80         muestreo_completo<='0';
81         wait for 1 ms;
82         data_in<="011001001011";
83         muestreo_completo<='1';
84         wait for 20 ns;
85         muestreo_completo<='0';
86         wait for 1 ms;
87         data_in<="011000011110";
           muestreo_completo<='1';
           wait for 20 ns;
           muestreo_completo<='0';
           wait for 2 ms;
           data_in<="111111111111";
           wait; ------stop bit
           end process;

   end CODI;
```

A.17 RSR

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.std_logic_unsigned.all;
4
5 entity RSR is
6         port(
7                 rst, clk: in std_logic;
8                 tamany_paraula: in std_logic_vector (1 downto 0);
9                 UCSZ2: in std_logic;
10                data_in: in std_logic_vector(11 downto 0);
11                muestreo_completo: in std_logic;
12                registre_errors: in std_logic_vector (1 downto 0);
13                error_calc: in std_logic;
14                bit9: out std_logic;
15                rsr_empty, rsr_ready: out std_logic;
16                paraula_out: out std_logic_vector (7 downto 0));
17 end RSR;
18
19 architecture CODE of RSR is
20
21
22 signal rsr_empty_aux: std_logic;
23 signal rsr_ready_aux: std_logic;
24
25
26 signal paraula: std_logic_vector(7 downto 0);  ------8 bits de
27 salida hacia el buffer
28 signal paraula_aux: std_logic_vector (7 downto 0);
29 signal start_bit: std_logic;
30 signal bit9_aux: std_logic;
31
32 signal compt_datos_aux: std_logic_vector(3 downto 0);
33 signal bits_stop: std_logic_vector (1 downto 0);
34 signal tamany: std_logic_vector(3 downto 0);
35 signal tamany_datos: std_logic_vector(3 downto 0);
36 signal nove_bit_aux: std_logic;
37
38 type tipus is (espera,rebre,invertir,fi);
39 signal estat:tipus;
40
41 begin
42
43 with tamany_paraula select
44                 tamany <= "0101" when "00",
45                           "0110" when "01",
```

```
46                          "0111" when "10",
47                          "1000" when others;
48
49 process(clk)
50 variable i:integer :=0;
51 variable j:integer :=7;
52 variable z: integer :=11;
53 begin
54         if (clk'event and clk='1') then
55                 if (rst ='1') then
56                         paraula<="00000000";
57                         i:=0;
58                         j:=7;
59                         z:=11;
60                         start_bit<='0';
61                         compt_datos_aux<= "0000";
62                         rsr_ready_aux<='0';
63                         rsr_empty_aux<='1';
64                         bit9_aux<='0';
65                         estat<=espera;
66                 else
67                         case estat is
68                                 when espera=>
69                                         rsr_empty_aux<='1';
70                                         rsr_ready_aux<='0';
71                                         if (muestreo_completo='1')
72 then
73                                                 paraula<="00000000";
74
75         paraula_aux<="00000000";
76                                                 rsr_empty_aux<='0';
77                                                 estat<=rebre;
78                                         end if;
79                                 when rebre=>
80                                         if (UCSZ2='1') then
81                                                 --if
82 (data_in(z)='1') then
83                                                 --      z:=z-1;
84                                                 --else
85                                                         z:=z-1;
86                                                         if (tamany =
87 "0101") then
88
89         paraula(7 downto 3)<=data_in(z downto z-4);
90
91         bit9_aux<=data_in(z-5);
92
```

```
 93                                                      elsif (tamany
 94 = "0110") then
 95
 96         paraula(7 downto 2)<=data_in(z downto z-5);
 97
 98         bit9_aux<=data_in(z-6);
 99                                                      elsif (tamany
100 = "0111") then
101
102         paraula(7 downto 1)<=data_in(z downto z-6);
103
104         bit9_aux<=data_in(z-7);
105                                                      else
106
107         paraula(7 downto 0)<=data_in(z downto z-7);
108
109         bit9_aux<=data_in(z-8);
110                                                      end if;
111                                                      estat<=
112 invertir;
113                                                 --end if;
114                                           else
115                                                 --if
116 (data_in(z)='1') then
117                                                 --      z:=z-1;
118                                                 --else
119                                                      z:=z-1;
120                                                      if (tamany =
121 "0101") then
122
123         paraula(7 downto 3)<=data_in(z downto z-4);
124                                                      elsif (tamany
125 = "0110") then
126
127         paraula(7 downto 2)<=data_in(z downto z-5);
128                                                      elsif (tamany
129 = "0111") then
130
131         paraula(7 downto 1)<=data_in(z downto z-6);
132                                                      else
133
134         paraula(7 downto 0)<=data_in(z downto z-7);
135                                                      end if;
136                                                      estat<=
137 invertir;
138                                                 --end if;
139                                           end if;
```

```
140                                 when invertir=>
141                                         if (i<8) then
142
143         paraula_aux(i)<=paraula(j);
                                                    j:=j-1;
                                                    i:=i+1;
                                            else
                                                    j:=7;
                                                    i:=0;
                                                    estat<=fi;
                                            end if;
                                    when fi=>
                                            --if (error_calc = '1' and
    registre_errors="00") then
                                                    rsr_ready_aux <='1';

            compt_datos_aux<="0000";
                                                    estat<=fi;
                                                    estat<=espera ;
                                            --end if;
                            end case;
                    end if;
            end if;
    end process;

    paraula_out<= paraula_aux;
    rsr_empty <= rsr_empty_aux;
    rsr_ready <= rsr_ready_aux;
    bit9<=bit9_aux;

    --compt_datos<=compt_datos_aux;

    end CODE;
```

A.18 TB RSR

```
1  -- Declarem les llibreries que utilizarem.
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.std_logic_unsigned.all;
5
6
7  -- Definim l'enitat.
8
9  entity TB_RSR is
10 end TB_RSR;
```

```
11
12 -- Describim el funcionament del TB.
13
14 architecture CODI of TB_RSR is
15         component RSR                                    -- Declarem
16 entrades i sortides del component descrit
17         port(
18                 rst, clk: in std_logic;
19                 tamany_paraula: in std_logic_vector (1 downto 0);
20                 UCSZ2: in std_logic;
21                 data_in: in std_logic_vector(11 downto 0);
22                 muestreo_completo: in std_logic;
23                 registre_errors: in std_logic_vector (1 downto 0);
24                 error_calc: in std_logic;
25
26                 rsr_empty, rsr_ready: out std_logic;  ---Esto se
27 podrÃa mirar de meter en el registro de estado o en un registro
28 solo del receptor
29                 bit9: out std_logic;
30                 paraula_out: out std_logic_vector (7 downto 0));
31         end component;
32
33         signal rst:std_logic;
34         signal clk:std_logic;
35         signal tamany_paraula: std_logic_vector (1 downto 0):=
36 "11";
37         signal registre_errors: std_logic_vector (1 downto 0):=
38 "00";
39         signal error_calc: std_logic;
40         signal data_in: std_logic_vector(11 downto 0);
41         signal muestreo_completo: std_logic;
42         signal rsr_empty: std_logic;
43         signal rsr_ready: std_logic;
44         signal paraula_out: std_logic_vector(7 downto 0);
45         signal UCSZ2: std_logic:='0';
46         signal bit9: std_logic;
47         constant CLK_PERIOD: time := 20 ns;
48
49         begin                              -- Connectem (relacionem)
50 entrades i sortides amb les senyals.
51         UUT: RSR port map
52         (clk => clk, rst => rst, tamany_paraula => tamany_paraula,
53 registre_errors => registre_errors, data_in=>data_in,
54 muestreo_completo => muestreo_completo, rsr_empty => rsr_empty,
55 rsr_ready=>rsr_ready,
56         paraula_out => paraula_out, error_calc=>error_calc,
57 UCSZ2=>UCSZ2, bit9=>bit9);
```

```
58
59          -- Donem impuls a les entrades
60
61          CLK_PROCESS : process
62          begin
63          clk <= '0';
64          wait for CLK_PERIOD/2;
65          clk <= '1';
66          wait for CLK_PERIOD/2;
67          end process;
68
69          RST_PROC : process
70          begin
71          rst <='1';
72          wait for 1 ms;
73          rst <='0';
74          wait;
75          end process;
76
77
78          DATA_IN_PROC: process
79          begin
80          data_in<="111111111111";
81          muestreo_completo<='0';
82          error_calc<='0';
            wait for 1 ms;
            data_in<="010010010111";
            muestreo_completo<='1';
            error_calc<='1';
            wait for 20 ns;
            muestreo_completo<='0';
            wait; ------stop bit
            end process;

    end CODI;
```

B.2 TX Buffer

```
1 library IEEE;
2 use IEEE.std_logic_1164.all;
3 use IEEE.numeric_std.all;
4 use IEEE.std_logic_unsigned.all;
5
6
7 entity RX_BUFFER is
8           port(
```

```
 9                rst, clk: in std_logic;
10                paraula_in: in std_logic_vector (7 downto 0); --
11 Equivale al paraula_out del rsr
12                rsr_ready: in std_logic;
13                read_en: in std_logic;
14
15                data: out std_logic_vector(7 downto 0);
16                buffer_ple: out std_logic;
17                buffer_buit: out std_logic);
18 end RX_BUFFER;
19
20 architecture CODE of RX_BUFFER  is
21
22 type ram_type is array (0 to 8) of std_logic_vector (7 downto 0);
23        signal ram_aux : ram_type;
24
25 signal data_aux: std_logic_vector (7 downto 0);
26 signal data_nova_aux: std_logic;
27
28 --punteros
29 signal puntero_in_aux: std_logic_vector (4 downto 0);
30 signal puntero_out_aux: std_logic_vector (4 downto 0);
31 signal resultat_aux: std_logic_vector (4 downto 0);
32
33 type tipus is (inici, enable, fi);
34 signal estat:tipus;
35
36 type tipus_2 is (inici_2, enable_2, fi_2);
37 signal estat_2:tipus_2;
38
39 signal buffer_ple_aux: std_logic;
40 signal buffer_buit_aux: std_logic;
41 signal error_overflow_aux: std_logic;
42
43 begin
44
45 -------------- REBEM PARAULA ----------------
46 process (clk)
47 variable i:integer := 0;
48 begin
49 if (clk'event and clk='1') then
50        if (rst='1') then
51                puntero_in_aux<="00000";
52                i :=0;
53                error_overflow_aux<='0';
54                ram_aux (0)<= "00000000";
55                ram_aux (1)<= "00000000";
```

```
56                ram_aux (2)<= "00000000";
57                ram_aux (3)<= "00000000";
58                ram_aux (4)<= "00000000";
59                ram_aux (5)<= "00000000";
60                ram_aux (6)<= "00000000";
61                ram_aux (7)<= "00000000";
62                estat<= inici;
63        else
64                case estat is
65                        when inici=>
66                                if (buffer_ple_aux='0') then
67                                        estat<= enable;
68                                end if;
69                        when enable =>
70                                if (rsr_ready = '1') then
71                                        ram_aux(i)<= paraula_in;
72                                        puntero_in_aux <=
73 puntero_in_aux + '1';
74                                        i := i+1;
75                                        if (puntero_in_aux >
76 "01000") then
77                                                puntero_in_aux<=
78 "00000";
79                                                i:=0;
80                                        end if;
81                                                estat<=fi;
82                                end if;
83                        when fi=>
84                                if (rsr_ready = '0') then --
85 Realitzem aixo per que no guardi molts cops les mateixes dades. Es
86 demanara un impuls de READ_EN igual a dues vegades la senyal de
87 rellotje
88                                        estat<= inici;
89                                end if;
90                end case;
91        end if;
92 end if;
93 end process;
94
95
96
97 -------------- ENVIEM PARAULA ----------------
98 process (clk)
99 variable j:integer :=0;
100 begin
101 if (clk'event and clk='1') then
102         if (rst='1') then
```

```
103                 puntero_out_aux<="00000";
104                 data_aux<= "00000000";
105                 data_nova_aux<='0';
106                 j:=0;
107                 estat_2<=inici_2;
108         else
109                 case estat_2 is
110                         when inici_2=>
111                                 if (buffer_buit_aux = '0') then
112                                         estat_2<= enable_2;
113                                 end if;
114                         when enable_2 =>
115                                 if (read_en='1') then
116                                         data_aux<=ram_aux(j);
117                                         data_nova_aux<='1';
118
119         puntero_out_aux<=puntero_out_aux + '1';
120                                         j:=j+1;
121                                         if (puntero_out_aux >
122 "01000") then
123
124         puntero_out_aux<="00000";
125                                                 j:=0;
126                                         end if;
127                                         estat_2<=fi_2;
128                                 end if;
129                         when fi_2=>
130                                 if (read_en='0') then --Realitzem
131 aixo per que no guardi molts cops les mateixes dades. Es demanara
132 un impuls de READ_EN igual a dues vegades la senyal de rellotje
133                                         estat_2<= inici_2;
134                                 end if;
135                 end case;
136         end if;
137 end if;
138 end process;
139
140
141
142
143 process(clk)
144 begin
145 if (clk'event and clk='1') then
146         if (rst='1') then
147                 resultat_aux <="00000";
148         else
149                 if (puntero_in_aux > puntero_out_aux) then
```

```
150                         resultat_aux<= puntero_in_aux -
151 puntero_out_aux;
152                 elsif (puntero_in_aux = puntero_out_aux) then
153                         resultat_aux<="00000";
154                 else
155                         resultat_aux<= (("01000"+ puntero_in_aux) -
156 puntero_out_aux);
157                 end if;
158         end if;
    end if;
    end process;

    with resultat_aux select
                buffer_ple_aux <= '1' when "01000",
                                  '0' when others;
    with resultat_aux select
                buffer_buit_aux <= '1' when "00000",
                                   '0' when others;

    data<=data_aux;
    buffer_ple<=buffer_ple_aux;
    buffer_buit<=buffer_buit_aux;
    end CODE
```

B.3 Càlcul de paritat

```
1  -- Declarem les llibreries que utilizarem.
 2 library ieee;
 3 use ieee.std_logic_1164.all;
 4 use ieee.std_logic_unsigned.all;
 5
 6
 7 -- Definim l'enitat.
 8
 9 entity TB_RX_BUFFER is
10 end TB_RX_BUFFER;
11
12 -- Describim el funcionament del TB.
13
14 architecture CODI of TB_RX_BUFFER is
15         component RX_BUFFER                                    --
16 Declarem entrades i sortides del component descrit
17         port(
18                 rst, clk: in std_logic;
```

```
19
20                paraula_in: in std_logic_vector (7 downto 0); --
21 Equivale al paraula_out del rsr
22                rsr_ready: in std_logic;
23                read_en: in std_logic;
24
25                data: out std_logic_vector(7 downto 0);
26                buffer_ple: out std_logic;
27                buffer_buit: out std_logic);
28
29        end component;
30
31        signal rst:std_logic;
32        signal clk:std_logic;
33        signal control_registre: std_logic_vector (7 downto 0):=
34 "11110111";
35        signal paraula_in: std_logic_vector (7 downto 0);
36        signal rsr_ready: std_logic;
37        signal read_en: std_logic;
38        signal buffer_ple: std_logic;
39        signal buffer_buit: std_logic;
40        signal data: std_logic_vector (7 downto 0);
41        constant CLK_PERIOD: time := 20 ns;
42
43        begin                              -- Connectem (relacionem)
44 entrades i sortides amb les senyals.
45        UUT: RX_BUFFER port map
46        (clk => clk, rst => rst, paraula_in=>paraula_in, rsr_ready
47 => rsr_ready, buffer_ple=>buffer_ple, read_en=>read_en, data=>data,
48 buffer_buit=>buffer_buit);
49
50 -- Donem impuls a les entrades
51
52        CLK_PROCESS : process
53        begin
54        clk <= '0';
55        wait for CLK_PERIOD/2;
56        clk <= '1';
57        wait for CLK_PERIOD/2;
58        end process;
59
60        RST_PROC : process
61        begin
62        rst <='1';
63        wait for 1 ms;
64        rst <='0';
65        wait;
```

```
66          end process;
67
68
69          DATA_IN_PROC: process
70          begin
71          paraula_in<="00000000";
72          rsr_ready<= '0';
73          read_en<='0';
74          wait for 1 ms;
75          paraula_in<="01010110";
76          rsr_ready<='1';
77          wait for 20 ns;
78          rsr_ready<='0';
79          wait for  1 ms;
80          paraula_in<="01111110";
81          rsr_ready<='1';
82          wait for 20 ns;
83          rsr_ready<='0';
84          wait for 1 ms;
85          paraula_in<="01010110";
86          rsr_ready<='1';
87          wait for 20 ns;
88          rsr_ready<='0';
89          wait for 1 ms;
90          read_en<='1';
91          wait for 20 ns;
92          read_en<='0';
93          wait for 1 ms;
94          read_en <='1';
            wait for 20 ns;
            read_en<='0';
            wait;
            end process;

    end CODI;
```