Joaquim Cavallé Vidiella

Avaluació del mòdul SDR Adalm Pluto per comunicacions digitals

TREBALL DE FI DE GRAU

dirigit per David Girbau Sala

Grau d'Enginyeria de Sistemes i Serveis de Telecomunicacions

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2021

ÍNDEX

Índex de Figures

1 Introducció

Les comunicacions digitals estan més presents en la nostra societat que mai, la tecnologia avança ràpidament i cada cop depenem més d'ella. L'aparició de les telecomunicacions va suposar un abans i un després en la manera de propagar la informació i han tingut un impacte enorme en la vida quotidiana de les persones i en l'evolució de diversos conflictes al llarg dels anys. Com s'acostuma a dir, la informació és poder i això s'ha demostrat durant tota la nostra història amb la creació de noves tècniques criptogràfiques per tal de protegir la informació que es transmetia durant les guerres o avui en dia per a mantenir la seguretat de les nostres dades.

En pocs anys el número de transmissions i la seva complexitat ha augmentat exponencialment i s'han hagut de trobar noves maneres per a regular, controlar i integrar les noves comunicacions en l'espectre radioelèctric. Aquesta situació s'ha fet més present des de ja fa més d'un any amb l'aparició de la pandèmia de COVID-19 que ha provocat que un gran nombre de treballadors i estudiants es confinin. Les empreses, escoles i universitats han hagut d'adaptar-se per a poder treballar des de casa mantenint el mateix nivell de qualitat i d'eficiència mitjançant les telecomunicacions.

1.1 Motivació

Aquest treball parteix de l'interès d'aprofundir més en la teoria apresa en varies assignatures al llarg del Grau d'Enginyeria de Sistemes i Serveis de Telecomunicacions (GESST) i plasmar aquests coneixements en un projecte pràctic.

D'ençà que em vaig interessar pel món de les telecomunicacions em va captivar tot allò que hi ha darrere de com transmetem la informació. És fascinant com s'aconsegueix transformar qualsevol informació en un senyal que podem enviar a llargues distàncies en un temps gairebé instantani i amb mètodes a l'abast de tothom. Els estudiants del grau hem après la teoria darrere d'aquestes transmissions gràcies a assignatures com Comunicacions Digitals i Fonaments de Comunicacions, i ens han ajudat a entendre com es generen, es detecten i es processen els senyals.

A més a més, en els últims anys han aparegut una sèrie de dispositius Software Design Radio (SDR, veure apartat 2) els quals són molt flexibles i versàtils ja que permeten implementar canvis en el seu funcionament únicament modificant el seu software. Com era d'esperar, aquests dispositius han tingut molt d'èxit entre els radioaficionats d'arreu del món i cada cop les empreses han invertit més en SDR fins al punt que avui en dia es troben a l'abast de tothom i a molt baix cost.

Així doncs, ara que disposem d'aquesta tecnologia que ens permet generar senyals digitals modulats de forma senzilla i de manera assequible, amb aquest projecte se n'ha fet una avaluació inicial per tal de veure la viabilitat d'aplicar-ho en les pràctiques de les assignatures de la carrera i passar de casos teòrics o simulacions a l'ús de dispositius SDR per tal que els alumnes puguin crear els seus propis programes i treballar amb senyals reals. Entre els diferents dispositius SDR disponibles comercialment, es va decidir fer servir el mòdul Adalm Pluto ja que era un dels dispositius pensat per a la docència i l'aprenentatge enfocat a estudiants d'enginyeria.

1.2 Objectius

Com s'ha comentat prèviament, aquesta és una primera avaluació dels dispositius SDR per tal de veure quin és el seu potencial per a ser aplicat a les pràctiques del GESST i per a posar a prova el mòdul Adalm Pluto.

L'objectiu principal d'aquest treball és veure les capacitats d'aquest dispositiu SDR i, a partir de la informació online proporcionada per la pròpia companyia Analog Devices i per Matlab, poder començar a generar senyals amb el mòdul, aplicar diverses modulacions i poder transmetre i rebre aquests senyals que es creen.

Per altra banda, es busca que aquest treball serveixi de pont per tal que, en un futur, es pugui aprofundir més en Software Defined Radio i, ja sigui amb el mateix Adalm Pluto o amb un altre dispositiu SDR, es segueixi avançant fins a treure el màxim rendiment que es pugui d'aquesta tecnologia per als futurs estudiants del GESST i els ajudi a veure els problemes de les comunicacions digitals amb casos reals i aporti un nou punt de vista més enllà de simulacions (mantenint la importància dels casos teòrics ja que ajuden a entendre els conceptes dins d'unes condicions establertes).

1.3 Estructura del Projecte

Per tal d'aconseguir complir tots els objectius del projecte, es van dividir les tasques a realitzar en diversos blocs per a anar assolint cadascuna de les parts del treball. S'ha organitzat aquest document de manera similar a com s'han realitzat les taques durant aquests mesos per a que sigui coherent amb el mètode de treball.

Ja que aquest treball ha de servir de pont per a seguir investigant més sobre els dispositius SDR com l'Adalm Pluto, necessita tenir unes bases ben establertes sobre la teoria i funcionament darrere d'aquesta tecnologia. Tot i que en alguna assignatura com Laboratori de Telecomunicacions havíem parlat i treballat amb SDR, aquest s'utilitzava com a 'caixa negra' i no s'entrava a treballar en el concepte de Software Defined Radio, cosa que si ha estat necessària en aquest projecte per tal d'avaluar aquest mòdul.

- El primer pas és entendre el concepte de SDR, el seu funcionament i veure quins dispositius hi ha actualment al mercat per tal de poder comparar les seves funcionalitats i veure els seus avantatges i inconvenients. A partir d'aquest punt cal centrar-se en l'Adalm Pluto en concret i veure quines són les seves característiques principals que el fan destacar envers els altres i aprofundir en la seva arquitectura i disseny.

- Per a acabar amb la part teòrica cal tenir ben clars els principis de la modulació, sincronisme i, més en concret, la modulació PSK, ja que juntament amb les seves variants és la que es farà servir al llarg del treball. També s'ha de tenir en compte un dels factors que més hem treballat en l'assignatura de comunicacions digitals: la probabilitat d'error.

- El segon gran bloc del treball és la part pràctica, aquí es busca provar les funcionalitats del mòdul i documentar-les clarament per a que sigui fàcil d'entendre pels alumnes que utilitzin l'Adalm Pluto per primer cop en una pràctica de laboratori. Es comença per la configuració bàsica del dispositiu, seguit de l'enviament de senyals simples per a entendre com funciona la comunicació, fins a arribar a la transmissió i recepció de senyals modulats reals mitjançant PSK.

2 Introducció a Software Defined Radio

Al llarg dels últims anys l'àmbit de les telecomunicacions ha estat reinventant-se constantment. Cada cop apareixen més eines que permeten millorar les comunicacions de manera molt lligada a les noves tecnologies digitals. És així com va sorgir el concepte de Software Defined Radio.

2.1 Concepte de SDR i Funcionament General

Els dispositius de Software Defined Radio (SDR) apareixen a partir dels anys 70 degut a l'interès per cercar de maneres d'incrementar l'eficiència de les comunicacions per ràdio i es caracteritzen per tenir algunes o totes les funcions de la capa física definides per software. La idea principal és poder unir el hardware de ràdio amb el processament digital de senyal.

Al llarg dels anys s'ha aconseguit passar de transceptors basats en hardware i que requerien implementar canvis modificant-se de manera física a nous sistemes de tractament de senyal amb la major part de la funcionalitat de banda base definida per software. Així, s'aconsegueix una millor flexibilitat i eficiència que permet realitzar simultàniament vàries tasques com transmissió, recepció i processament de dades, les quals poden ser modificades fàcilment canviant el software per a aconseguir noves funcionalitats[1].

La figura 1 mostra un diagrama de blocs simplificat de l'arquitectura d'un dispositiu SDR i permet obtenir una idea general del seu funcionament

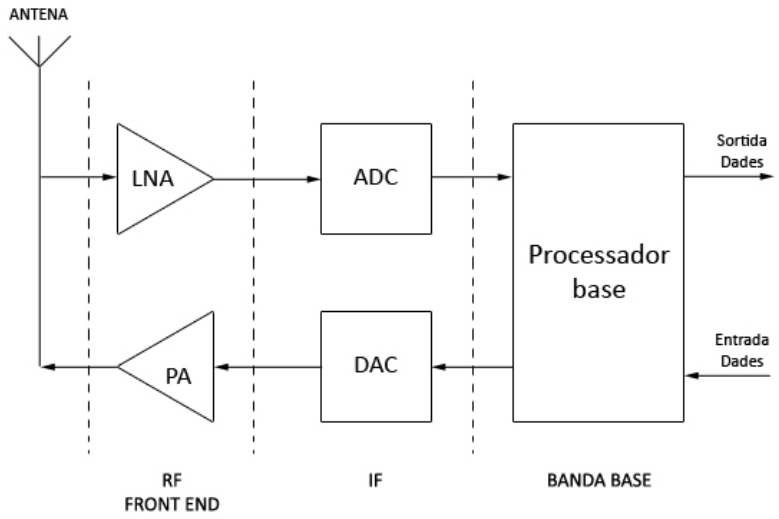

Figura 1. Diagrama de blocs simplificat d'una SDR

Per tal de dur a terme la transmissió, es genera un senyal a banda base a partir de les dades que arriben al processador. Un cop s'ha generat el senyal, el bloc de Freqüència Intermitja

(FI) s'encarrega d'aplicar totes les modificacions necessàries (modulació, filtratge...) abans de passar-lo pel conversor DAC i enviar el senyal analògic al bloc de radiofreqüència. En el tercer i últim bloc es passa el senyal pel mesclador per a desplaçar-la a la freqüència necessària per la radiocomunicació, per a finalment amplificar-la i enviar-la per l'antena.

El procés de recepció repeteix el mateix procés en el sentit invers, primer s'amplifica el senyal que es rep per l'antena i es fixa el nivell de soroll del receptor, i a través del mesclador es baixarà a una freqüència intermitja, ja que és molt complicat poder fer la conversió d'analògic a digital a altes freqüències.

En el bloc FI es farà la conversió, desmodulació i els passos necessaris per a transferir el senyal al processador per tal que aquest pugui extreure la informació [2].

2.2 Aplicacions de SDR

La teoria darrere dels dispositius SDR mostra com obren un nou món de possibilitats per a les comunicacions digitals però el principal dubte que genera és: quines aplicacions se li poden donar a Software Defined Radio?

Com passa amb totes les noves tecnologies de comunicacions, podem dividir les seves aplicacions entre l'àmbit militar i l'àmbit civil.

2.2.1 Àmbit Militar

Un dels primers projectes que va aparèixer en el món militar al voltant de l'any 2000 va ser el programa Joint Tactical Radio System (JTRS). Aquest projecte del Departament de Defensa d'Estats Units, el qual buscava substituir les ràdios que es feien servir per una nova família de ràdios programables per software que podrien proveir als soldats amb transmissió de veu, vídeo i de dades al camp de batalla.

Aquest nou sistema de comunicacions amb ràdios multicanal i multimode va reemplaçar les ràdios antigues i va permetre una gran flexibilitat en la generació de senyal, la seva encriptació i el seu processament. Gràcies a aquesta inversió es van poder substituir 750.000 ràdios velles amb 180.000 ràdios definides per software en els diversos vehicles, vaixells i avions de l'exèrcit [3].

2.2.2 Àmbit Civil

A part dels diversos programes militars, hi ha una gran quantitat d'usos per als dispositius SDR i que, poc a poc, es van incrementant juntament amb l'interès de la gent i l'aparició de nous productes al mercat cada cop més assequibles.

Una de les aplicacions més bàsiques és poder rebre transmissions de ràdio fent servir diverses aplicacions, tal com es pot veure a la figura 2, o creant el teu propi programa seguint els consells que pots trobar fàcilment de manera online.

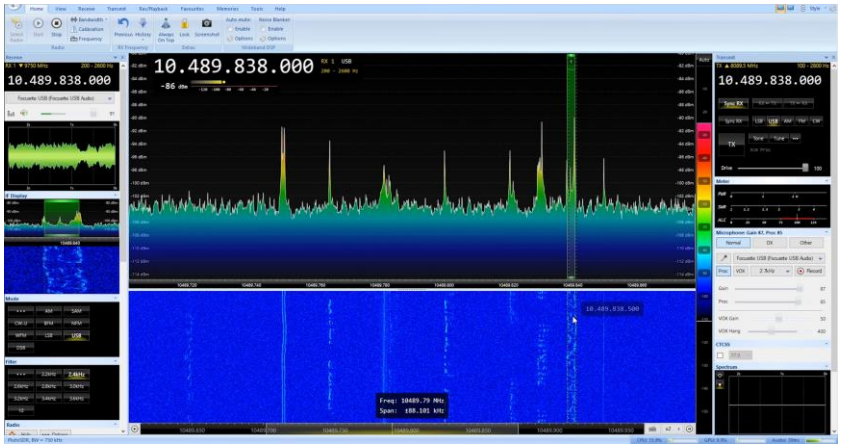

Figura 2. Recepció de ràdio mitjançant el programa SDR Console

Seguint la temàtica d'aquest treball, els dispositius SDR són una gran oportunitat per a aprendre i cada cop apareixen nous productes destinats a la docència. La gran versatilitat de la ràdio definida per software permet als alumnes d'enginyeria aplicar els coneixements que han après al llarg de les assignatures de la carrera i consolidar-los de manera pràctica mitjançant la programació [4].

Un dels exemples més clars és l'Adalm Pluto (veure figura 3), que està dissenyat per a l'aprenentatge actiu dels estudiants i disposa de bastanta informació i exemples per a agafar com a punt de partida per crear treballs de laboratori enfocats en cada una de les funcionalitats que el mòdul proporciona per a la transmissió, recepció i processat del senyal.

Figura 3. Mòdul Adalm Pluto

Per altra banda, una gran quantitat de radioaficionats han aprofitat la tecnologia SDR per a poder provar diversos modes de comunicació en múltiples bandes de freqüència fent servir Codi Morse, AM, FM i exprimint al màxim les capacitats d'aquests dispositius.

El SDR-1000 va ser un dels dispositius pioners que més es va utilitzar a partir de l'any 2002 que es basava en un DSP (Digital Signal Processor) per al filtratge, la modulació i la desmodulació, i feia servir la targeta de so de l'ordinador per als conversors ADC (Analog to Digital Converter) i DAC (Digital to Analog Converter). Seguint els passos de l'èxit amb el seu producte, la companyia FlexRadio Systems he seguit treballant en nous productes fins al dia d'avui amb SDR com la FLEX-6400, molt popular entre els radioaficionats [5], que es mostra a la figura 4.

Figura 4. Dispositiu SDR FLEX-6400

2.3 Instruments SDR

El món de les SDR es troba en constant evolució i últimament aquesta tecnologia ha anat guanyat molta popularitat i el mercat està ple de dispositius diferents. La competitivitat entre les empreses ha provocat que hi hagi un gran número de SDR per a triar, cada un amb els seus avantatges i inconvenients. A continuació es mostren les característiques principals dels dispositius més exitosos en el mercat i una comparació entre ells i el mòdul Adalm Pluto que s'ha fet servir per aquest treball.

2.3.1 HackRF ONE

La primera versió d'aquest dispositiu (veure figura 5) de la companyia Great Scott Gadfets va aparèixer al mercat el 2015, es tracta d'un transceptor que permet comunicació *half-duplex* i pot operar en un rang de freqüències bastant ample (d'1 MHz fins a 6 GHz). Va integrat amb Radio GNU i SDR# per tal de crear la interfície gràfica d'usuari. Aquest model destaca per haver cridat l'atenció de diversos hackers que van aconseguir sobrepassar la seguretat de comunicacions tàctiques militars i enviar senyals falses al GPS i també van aconseguir interceptar i reproduir els senyals de les claus per a obrir cotxes i portes de garatge [6-7].

Figura 5. Dispositiu SDR HackRF One

2.3.2 NooElec NESDR SMArt

La quarta versió del dispositiu RTL-SDR de la companyia nord-americana s'ha convertit en un gran èxit degut a les noves modificacions premium que han afegit respecte la versió prèvia. Han optat per invertir en un soroll de fase molt baix millorant el regulador de voltatge a RF, aconseguint reduir-lo 10 vegades. També s'ha millorat el rebuig a les interferències electromagnètiques i la dissipació de calor de la placa. Les millores d'aquesta última versió combinades amb el baix preu del mòdul l'han convertit en un dels productes més interessants del mercat de SDR [8]. Aquest SDR es pot veure a la figura 6.

Figura 6. Dispositiu RTL-SDR NoooElec NESDR SMArt

2.3.3 Flex-6700

La versió millorada de Flex-6400, que es comenta a l'apartat **2.2.2** es mostra a la figura7 i encapçala la llista de millors SDR de gamma alta amb un nou disseny més compacte. Aquest transceptor permet operar full-duplex amb les seves dues antenes podent emetre en una banda de freqüències i rebre en una altra. Treballa en un ampli rang de freqüències, amb un software compatible amb versions més antigues de la sèrie FLEX i amb la nova opció de poder accedir remotament a la ràdio des de qualsevol lloc fent servir un ordinador o un mòbil [9].

Figura 7. Dispositiu SDR FLEX-6700

2.3.4 PlutoSDR

El mòdul Adalm Pluto és una eina dissenyada per a l'aprenentatge actiu de la companyia Analog Devices. Està enfocat a l'àmbit de la docència per a que els estudiants d'enginyeria puguin entendre el funcionament de Software Defined Radio i de les Comunicacions Digitals mentre descobreixen les aplicacions pràctiques de la teoria de radiofreqüència.

L'Adalm Pluto combina un preu assequible amb unes molt bones prestacions i una gran flexibilitat en la programació sent compatible amb Matlab, Simulink, GNU Radio, C, C++, C# i fins i tot amb Python [10]. A més a més, fent una modificació de software es pot incrementar el rang de freqüències i passar de 325 MHz - 3.8 GHz a 70 MHz - 6 GHz [11].

2.3.5 Comparació de Característiques

En aquest apartat es resumeixen les propietats més destacables de cadascun dels productes SDR en una taula per a poder fer una ràpida comparació entre ells [12].

Dispositiu SDR	Rang de Freqüències	BW Màxim	Bits del ADC (Rx)	Bits del DAC (Tx)	Preu ($)
HackRF ONE	1 MHz - 6 MHz	20 MHz	8	8	299
NooElec NESDR SMArt	25 MHz - 1750 MHz	3.2 MHz	8	-	20.95
FLEX-6700	0.01 MHz - 73 MHz, 135 - 165 MHz	14 MHz	16	16	6999
Adalm Pluto	325 MHz - 3.8 GHz (70 MHz - 6 GHz)	20 MHz	12	12	148

2.4 Adalm Pluto

L'apartat anterior ha servit per a destacar les característiques principals del mòdul Adalm Pluto i els avantatges que aporta envers altres dispositius SDR del mercat i que són el motiu pel qual varem decidir fer aquesta primera aproximació al món de SDR amb el Pluto i no amb un dels seus competidors.

A diferència del bloc anterior en què es comentaven les propietats generals de cada dispositiu sense entrar gaire en detall, aquesta secció es centrarà en estudiar l'arquitectura darrere del mòdul i el seu funcionament intern.

2.4.1 Arquitectura i Disseny

El disseny de l'Adalm Pluto destaca per la seva senzillesa i fàcil portabilitat. Protegit per una carcassa de plàstic, el dispositiu s'alimenta mitjançant connexió USB i disposa de dues antenes GSM que permeten transmetre i rebre amb full duplex o half duplex en un rang de freqüències d'entre 70 MHz i 6 GHz, cada mòdul incorpora un cable SMA de 15 cm per a poder separar les dues antenes en cas que sigui necessari.

Pel que fa als components interns del dispositiu, utilitza Linux IIO framework (una subclasse de framework de Linux que està especialitzada en conversors ADC, DAC, mescladors, PLLs...) i es basa en el transceptor AD9363 que combina el front-end de RF amb una secció

de banda base flexible amb interfície digital configurable [13]. El diagrama de blocs de l'Adalm Pluto es mostra a la figura 8.

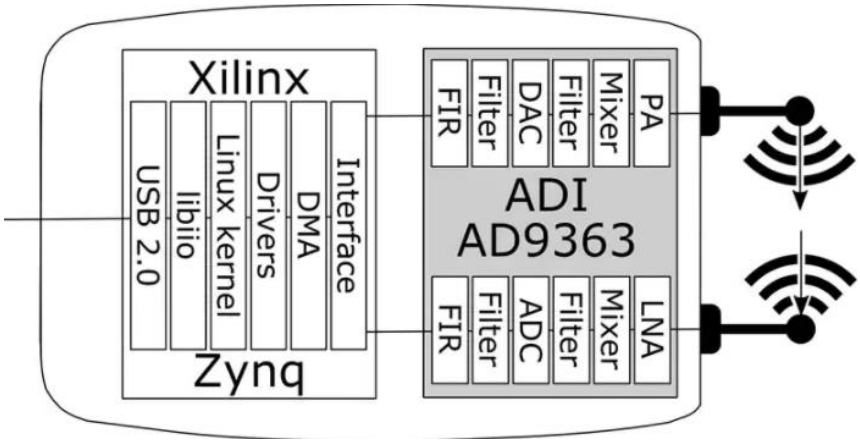

Figura 8. Diagrama de Blocs de l'Adalm Pluto

S'utilitzen dos connectors USB i un botó que es pot activar fent passar una agulla a través del forat que hi ha a la carcassa de plàstic com es mostra en la Figura 9.

Figura 9. Connectors USB i Botó de l'Adalm Pluto

El botó s'utilitza per a posar el dispositiu en recovery mode però la seva funcionalitat està definida per software, així que es pot modificar el seu funcionament [14].

Entrant en detall a la PCB del mòdul, es pot apreciar en la figura 10 el disseny simple que s'ha fet servir per aquest dispositiu. Des de la pròpia empresa destaquen que al crear l'Adalm Pluto buscaven mostrar als compradors com és l'aparença d'un disseny minimalista.

A part del transceptor AD9363 i dispositius d'alimentació cal destacar el Micron DDR3L amb una memòria de 512 MB que s'utilitza pels buffers i la RAM. En canvi, el Micron SPI Flash té una memòria de 16 MB i s'encarrega de l'arrencada del sistema [15].

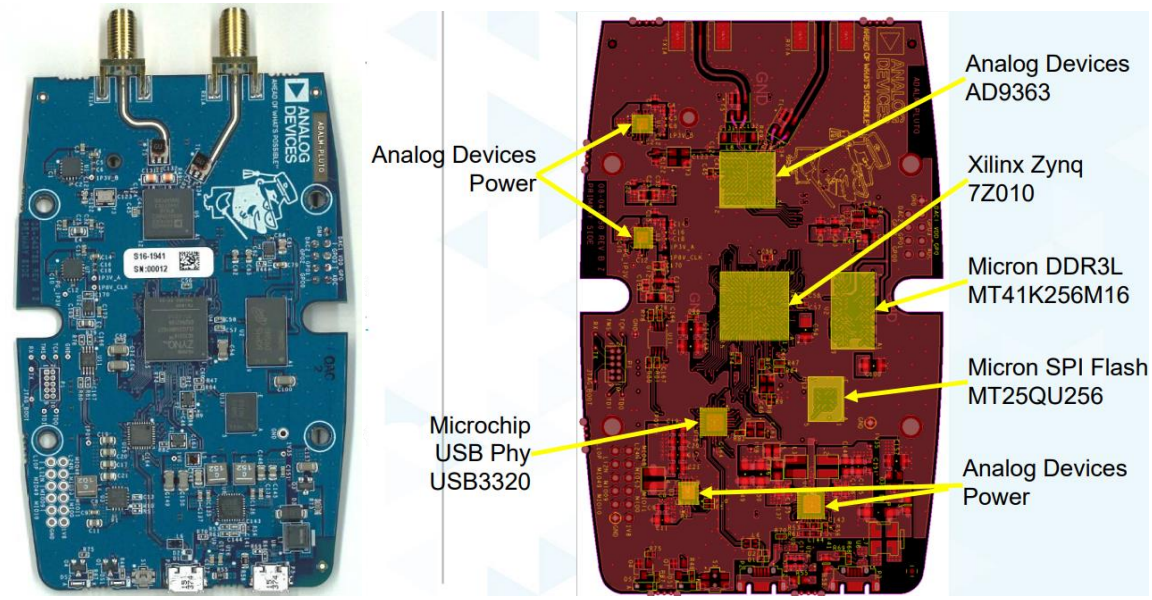

Figura 10. PCB interna de l'Adalm Pluto

Una de les parts més importants és el transceptor AD9363, que es mostra en la Figura 11 (es mostra el diagrama de blocs de l'AD9361 però al ser els dos de la família AD936X, la única diferència és el rang de freqüències). L'AD9363 disposa d'una banda de transmissió de 47 MHz fins a 6 GHz i una banda de recepció de 70 MHz fins a 6 GHz (un cop aplicada la modificació de software que es comenta a l'apartat 2.3.4), amb un ample de banda de canal variable entre 200 kHz i 20 MHz. Disposa de convertidors ADC i DAC de 12 bits i un control automàtic de correcció de quadratura, filtratge digital i correcció d'offset. Els transmissors fan servir una arquitectura de conversió directe per tal de generar modulacions precises i amb molt baix soroll. A més a més, porta integrats diferents PLLs per a la transmissió i per a la recepció. Tot això encapsulat en un chip de 10 mm x 10 mm [16-17].

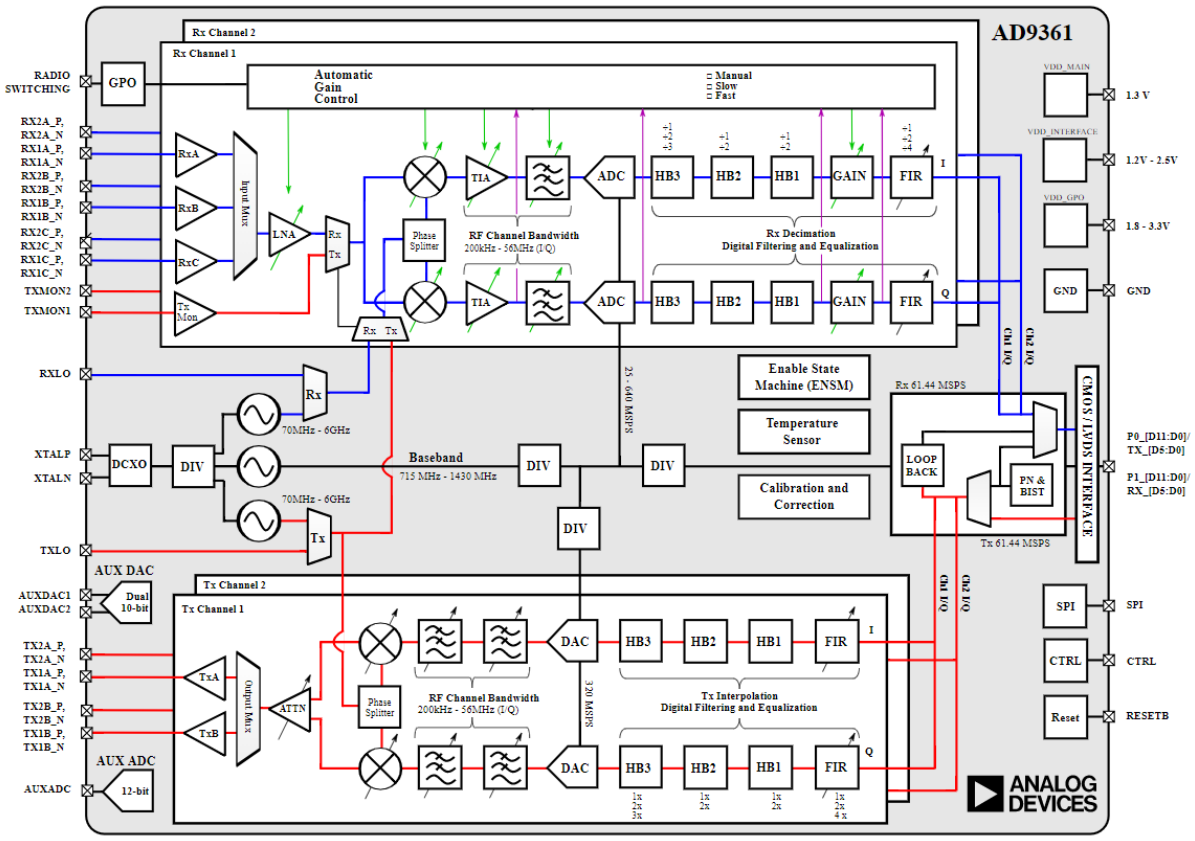

Figura 11. Transceptor AD936X

3 Introducció a la Modulació

L'objectiu principal d'un sistema de comunicacions és poder transferir informació d'un punt a un altre. Al llarg dels anys s'ha produït una gran evolució en la manera en què transmetem, rebem i processem aquesta informació.

La forma més bàsica de transmetre informació és la transmissió en banda base, on les dades s'envien directament sense ser modulades a través del canal. Aquesta forma d'enviar la informació provoca que ocupin la banda baixa de l'espectre radioelèctric i que només hi pugui haver una única comunicació. Per tant, limita les comunicacions simultànies sobre un mateix canal. La freqüència màxima que ocupa el senyal en l'espectre de freqüències marca el seu ample de banda (B).

A mesura que la tecnologia avançava cada cop ha anat incrementant la quantitat de senyals que s'envien (increment de nombre de serveis, de nombre d'usuaris...), fet que ha provocat que s'hagi de regular l'espectre radioelèctric per tal de no saturar el mitjà que la gran majoria de senyals comparteixen (l'aire).

La modulació consisteix en modificar els paràmetres (un o diversos) d'una ona, típicament sinusoïdal, que s'anomena portadora amb el senyal modulador que és el que conté la informació. Mitjançant la modulació, que porta implícita una translació de freqüència del senyal, s'aconsegueix ordenar l'espectre radioelèctric, facilitar la propagació del senyal, evitar interferències, disminuir la mida de les antenes (la mida és aproximadament proporcional a la longitud d'ona) i optimitzar l'ample de banda [18-19].

A l'hora de modular es distingeix entre dos grans estratègies: la modulació analògica i la modulació digital.

La modulació analògica s'aplica a senyals amb informació contínues en temps i en amplitud amb un nombre infinit de valors. Dins d'aquest bloc es troba la modulació analògica lineal, que consisteix en modificar l'amplitud de l'ona portadora com és el cas de la modulació AM (Modulació d'Amplitud), DBL (Doble Banda Lateral), BLU (Banda Lateral Única) i BLV (Banda Lateral Vestigial), i la modulació analògica angular, que manté l'amplitud però es modifica la freqüència (FM) o la fase (PM) de la portadora. Les tècniques de modulació analògica més conegudes son la AM i la FM ja que es fan servir a dia d'avui encara en la ràdio i en algun altra sistema. En la figura 12 es pot veure un exemple de modulació d'amplitud on, a partir de la informació de l'ona moduladora es modifica l'amplitud de la portadora fins a aconseguir el senyal modulat (color negre) [20-21]. No ens estendrem més en les modulacions analògiques ja que no són l'objectiu final d'aquest projecte, sinó que el que es vol és fer comunicacions amb modulacions digitals.

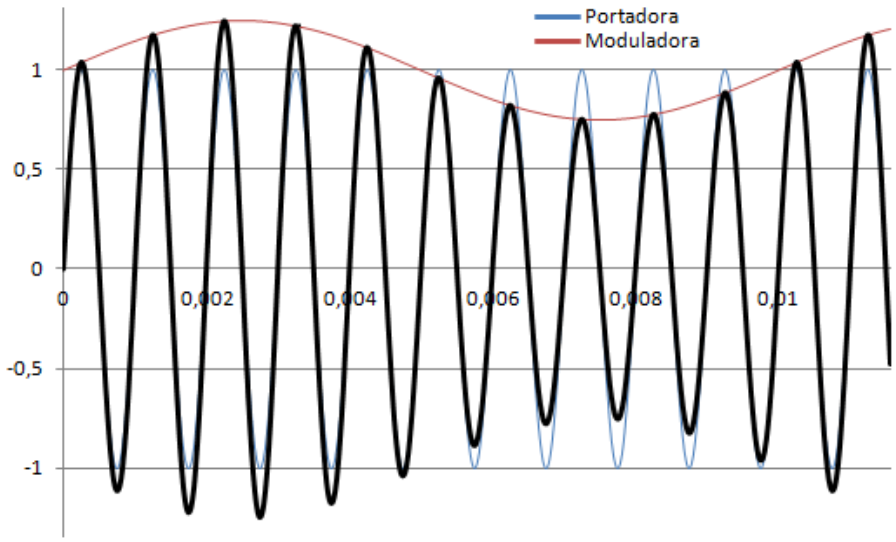

Figura 12. Modulació d'Amplitud en el domini temporal

3.1 Principis de la Modulació Digital

En l'apartat anterior s'expliquen breument els avantatges de la modulació i les diverses tècniques de modulació analògica que es fan servir a dia d'avui. Tot i que la modulació analògica encara és bastant popular en alguns camps degut a que és molt més simple d'aplicar, cada cop les noves tecnologies s'estan decantant per la modulació digital.

El fet de treballar amb senyals digitals fa que la comunicació sigui molt més robusta al soroll i a les interferències ja que, al treballar amb valors discrets, és molt més simple pel receptor discernir entre quin valor està rebent i permet reconstruir un senyal idèntic al que s'ha transmès (sempre i quan la distorsió del senyal no sigui prou gran com per provocar un error). També cal tenir en compte la importància de la seguretat en les comunicacions avui en dia i la modulació digital té el gran avantatge de permetre una fàcil encriptació de la informació a diferència de la modulació analògica.

Tot i això, la modulació digital també comporta certs inconvenients a l'hora de fer-la servir. Al fer servir la modulació digital, es necessita un sistema de sincronització (veure apartat **3.3**) i passar el senyal a través d'un conversor ADC i per un conversor DAC si es vol transmetre informació analògica. A més a més, els sistemes de comunicació digital requereixen d'un ample de banda major per a transmetre la mateixa informació que els analògics.

3.1.1 Modulació Digital Banda Base

Quan es treballa amb senyals digitals la informació que es transmet s'ordena en una seqüència de bits anomenada senyal en banda base. A més a més, un conjunt de *k* bits es pot ajuntar per a formar un *símbol*, generant un alfabet de *M* símbols seguint la fórmula següent:

$$M = 2^k \qquad (1)$$

Depenent del nombre de símbols que faci servir el sistema digital s'anomenarà sistema M-ari, sent el cas de k=1 el més simple (un símbol és el mateix que un bit) i es coneix pel nom de sistema binari. El senyal de banda base es pot enviar codificant els bits i associant-los a un pols elèctric, conegut com a Pulse Code Modulation. Així doncs, en la figura 13 es mostra un exemple de PCM fent servir un sistema binari en què un bit 1 s'associa a un pols alt (1 V) i un bit 0 a un pols baix (0 V) [22].

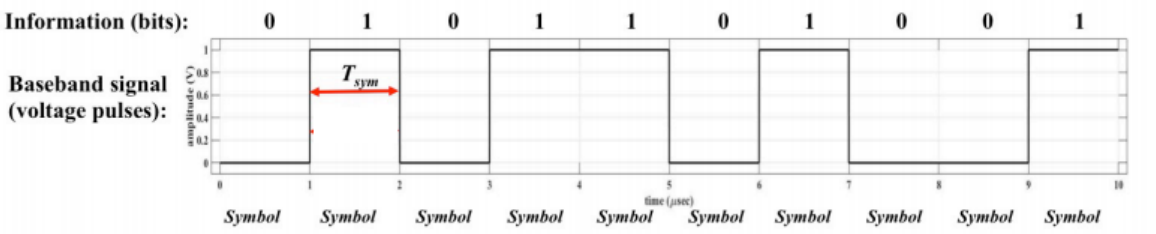

Figura 13. Exemple Modulació PCM

Un dels paràmetres més importants d'una transmissió és la velocitat de transmissió de bits (R_b) que es relaciona amb la velocitat de transmissió de símbols (R_{simbol}) amb el paràmetre *k*.

$$R_b = k \cdot R_{simbol} \qquad (2)$$

Finalment, la velocitat de transmissió de símbols és l'invers del temps que es tarda en transmetre un símbol (T_{simbol}).

$$R_{simbol} = \frac{1}{T_{simbol}} \qquad (3)$$

Per exemple, si el valor de T_{simbol} és de 1us i, com que es tracta d'un sistema binari, el valor de R_b i de R_{simbol} és de 10^6 bits/s o símbols/s [23].

3.1.2 Modulació Digital Pas Banda

Les modulacions pas banda tenen com a funció principal desplaçar el senyal que es transmet a una banda de freqüències que s'adequï a l'estàndard amb el qual es vol realitzar la comunicació, permetent d'aquesta forma l'existència de comunicacions simultànies.

Així com s'ha explicat en els punts anteriors, la modulació es basa en canviar algun dels paràmetres (amplitud, fase o freqüència) de l'ona portadora seguint la informació que porta l'ona moduladora. Per a expressar el senyal variant en el temps de forma matemàtica fem servir la següent expressió:

$$s(t) = A(t)\cos\left(\theta(t)\right) \qquad (4)$$

On A(t) és l'amplitud del senyal en un instant de temps i θ(t) és la fase. La fase variant en el temps es pot expressar amb la freqüència angular:

$$\theta(t) = \omega_0 \cdot t + \phi(t) \qquad (5)$$

I sabent que la freqüència angular és (on f_c és la freqüència central del radiocanal on es realitzarà la comunicació):

$$\omega_0 = 2 \cdot \pi \cdot f_c \qquad (6)$$

Arribem a l'expressió final [24]:

$$s(t) = A(t)\cos\left[2\pi f_c \cdot \mathrm{t} + \phi(t)\right] \qquad (7)$$

A partir d'aquest senyal es poden aplica diverses modulacions com Amplitude Shift Keying (ASK), Frequency Shift Keying (FSK), Phase Shift Keying (PSK)...

També cal distingir entre els diversos tipus de detecció que es fa servir en el receptor. Per una banda es pot fer servir detecció coherent, en què la fase de la portadora serveix per a detectar el senyal ja que des de recepció es té una referència de cada senyal que pot arribar i en la desmodulació es busca la correlació entre el senyal rebut amb cada una de les referències. Per altra banda, si no s'utilitza la fase de la portadora per a detectar el senyal es tracta de detecció no coherent.

3.2 Modulació PSK

En la modulació angular PSK es modifica la fase instantània de la portadora per una sèrie de valors discrets de possibles fases. Fent referència a l'expressió de l'apartat anterior, tenint en compte que E_b és l'energia de bit i que T_b és el temps de bit, la descripció matemàtica d'un senyal modulat amb PSK és:

$$\omega_0 = \sqrt{\frac{2E_b}{T_b}}\cos\left(\omega_0 t + \phi_i(t)\right) \qquad (8)$$

$\phi_i(t)$ representa cadascun d'aquests possibles valors de fase on el nombre de casos dependrà del nivell de modulació (M) i vindrà donat per la següent equació:

$$\phi_i(t) = \frac{2\pi(i-1)}{M}; \qquad i = 1, \ldots, M \qquad (9)$$

Si suposem el cas més simple (BPSK) podem assignar als 1 una fase de 0º i als 0 una fase de 180º (π), tal com es pot observar en la figura 14.

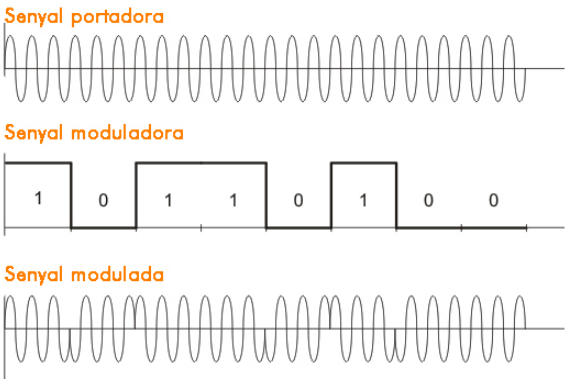

Figura 14. Modulació BPSK en el Domini Temporal

Una manera molt típica de representar els senyals modulats és a partir d'un diagrama fasorial (la llargada del fasor representa l'amplitud i la seva posició angular respecte els eixos representa la fase). Tot i això, lar representació més típica és el diagrama de constel·lació (es mostra la mateixa informació però enlloc de fer servir un fasor, es fa servir un punt, que és on apunta el fasor, tal com es pot veure a la figura 15 per una BPSK i la figura 16 per una QPSK) [25-26-27].

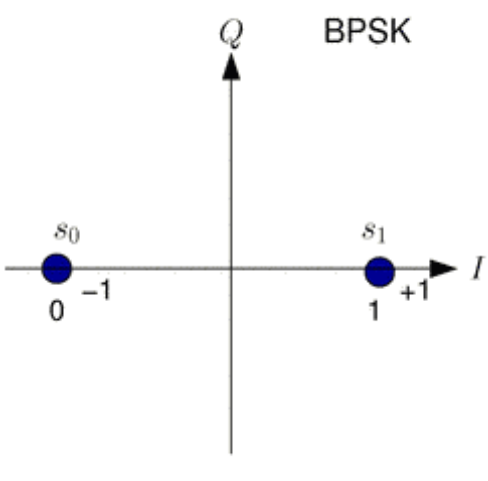

Figura 15. Constel·lació BPSK

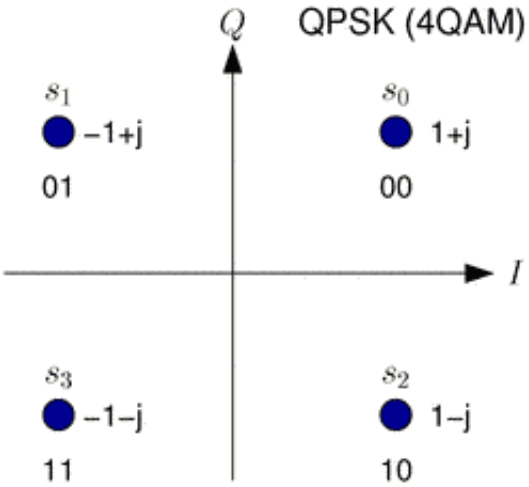

Figura 16. Constel·lació QPSK

En el cas de la modulació quaternària QPSK el número de símbols M és 4 i, per tant, es necessiten codificar quatre fases diferents. A continuació es mostra un diagrama de blocs d'un modulador QPSK bàsic:

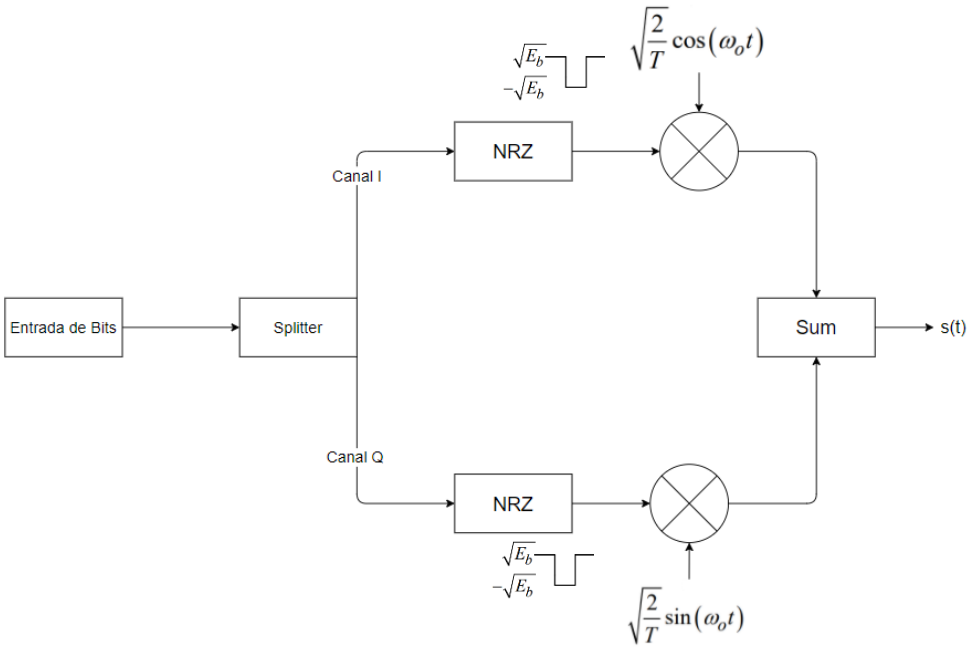

Figura 17. Modulador QPSK

Els bits que arriben per l'entrada es dividiran entre el canal de Fase (I) i el de Quadratura (Q) enviant a els bits parells a una branca i els imparells a l'altra. Al passar pel codificador NRZ els bits es converteixen en polsos on tant els 1ns com els 0s tenen energia associada, i quan arriben al mesclador es traslladen a la freqüència de l'ona portadora per a que finalment el senyal resultant sigui:

$$s(t) = \pm\sqrt{\frac{2E_b}{T}}\cos(\omega_0 t) \pm \sqrt{\frac{2E_b}{T}}\sin(\omega_0 t) \qquad (10)$$

I si s'expressa en funció del mòdul i fase s'obté:

$$s(t) = \sqrt{\frac{2E_s}{T}}\cos\left(\omega_0 t + (2n-1)\frac{\pi}{4}\right); \qquad n = 1, 2, 3, 4 \qquad (11)$$

A partir d'aquesta expressió i la constel·lació de la figura 16 es poden trobar els valors de les quatre fases diferents que s'han fet servir per codificar els bits d'entrada.

Entrada Binària	Fase de Sortida QPSK
00	45º
01	135º
10	315º
11	225º

3.3 Sincronisme

Els sistemes de comunicacions digitals requereixen d'un sistema de sincronització entre el transmissor i receptor per a detectar correctament el senyal un cop arriba, ja que normalment utilitzen rellotges diferents al estar a distàncies grans i sense connexió física. Aquest sistema s'ha d'encarregar de decidir quin és l'instant òptim de mostreig de cada símbol per tal de detectar el seu valor correctament i també ha de poder identificar on acaba una trama i on comença la següent per a dividir correctament la informació [28].

Els senyals que es fan servir per a la sincronització permetran recuperar la informació transmesa. Els tipus de senyals de sincronització més utilitzats són:

- **Senyals de sincronisme de trama:** s'utilitzen per a poder separar els diferents grups de bits i classificar-los per trames.
- **Senyals de sincronisme de bit:** marquen l'interval corresponent a cada un dels bits transmesos.
- **Senyals de sincronisme de portadora**: permeten recuperar paràmetres de l'ona portadora com la fase o la freqüència i són molt utilitzats en els sistemes de detecció coherent del senyal.

3.3.1 Codis Barker

Les seqüències o codis Barker són una successió finita de N valors que poden ser +1 o -1 que es fa servir com a senyal de sincronisme de trama entre el transmissor i el receptor en sistemes de comunicació digital [29].

Cadascuna de les possibles seqüències ha de complir tres propietats fonamentals basades en l'autocorrelació:

- La relació entre el lòbul principal i els lòbuls laterals ha de ser tan gran com es pugui
- Els lòbuls laterals han de tenir la mínima energia possible
- L'energia dels lòbuls laterals ha d'estar distribuïda uniformement.

Avui en dia només es coneixen nou seqüències diferents que compleixin aquestes condicions i es creu que no existeix cap altra codi Barker més [30]. Aquestes nou seqüències són les següents:

Known Barker codes

Length	Codes		Sidelobe level ratio
2	**+1 −1**	+1 +1	−6 dB
3	**+1 +1 −1**		−9.5 dB
4	+1 +1 −1 +1	+1 +1 +1 −1	−12 dB
5	+1 +1 +1 −1 +1		−14 dB
7	**+1 +1 +1 −1 −1 +1 −1**		−16.9 dB
11	**+1 +1 +1 −1 −1 −1 +1 −1 −1 +1 −1**		−20.8 dB
13	+1 +1 +1 +1 +1 −1 −1 +1 +1 −1 +1 −1 +1		−22.3 dB

Figura 18. Taula de Codis Barker

La sincronització amb l'Adalm Pluto es farà fent servir l'últim d'aquests codis.

3.4 Bit Error Rate (BER)

La Probabilitat d'Error de Bit ve donada per la relació entre el número de bits erronis i el número de bits transmesos. Aquest és un dels paràmetres quantitatius més importants en una comunicació digital ja que indica com de fiable és un sistema de comunicacions digital i sempre es busca reduir-lo al mínim. Cada protocol té un valor estandarditzat de probabilitat d'error per tal de garantir l'eficiència de la comunicació.

$$BER = \frac{nº\ de\ bits\ erronis}{nº\ bits\ transmessos} \qquad (12)$$

Quan es treballa amb senyals pas banda, també es pot parlar de la Probabillitat d'Error de Símbol (P_E) que en el cas de les modulacions M-PSK es relaciona amb el BER a través del número de símbols (M):

$$BER = \frac{P_E}{\log_2 M}; \qquad (si\ P_E \ll 1) \qquad (13)$$

També cal tenir en compte la relació entre l'Energia de bit i la densitat espectral del soroll (E_b/N_o) i que no s'ha de confondre amb la relació senyal-soroll (SNR) tot i que es troben relacionades a través de la expressió següent [31]:

$$\frac{E_b}{N_o} = \frac{S}{N} \cdot \left(\frac{W}{R_b}\right) = \frac{S}{N} \cdot \left(\frac{T_b}{T_s}\right) \qquad (14)$$

Hi ha vàries gràfiques que mostren com estan lligats els valors de BER i de E_b/N_o en cada cas de modulació [32]:

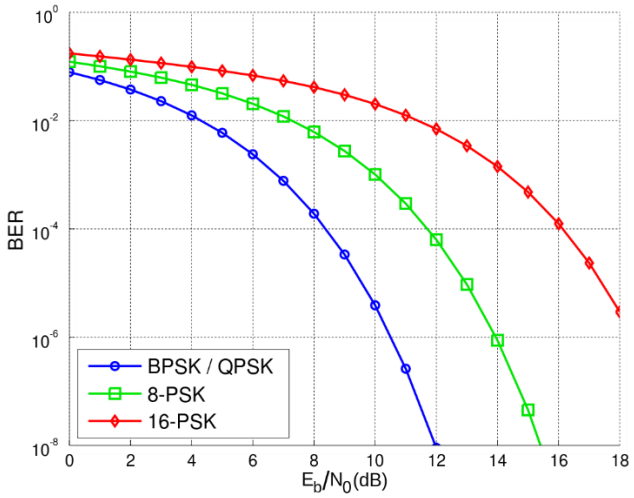

Figura 19. Corbes BER-Eb/No en Modulacions PSK

Per tal de calcular la Probabilitat d'Error de Bit s'utilitza la funció *Q(x)*. La funció *Q* es fa servir en estadística ja que mostra l'error en la cua d'una distribució gaussiana o, dit d'altra manera, la probabilitat que una variable aleatòria normal (gaussiana) obtingui un valor major que el de *x* [33].

$$Q(x) = \frac{1}{\sqrt{2\pi}} \cdot \int_x^{\infty} e^{(-u^2/2)}\, du \qquad (15)$$

Un cop definida la funció *Q*, l'expressió del BER en la modulació BPSK és:

$$BER = P_E = Q\left(\sqrt{\frac{2E_b}{N_o}}\right) \qquad (16)$$

Per a un cas general PSK de M símbols la Probabilitat d'Error de Símbol segueix l'expressió:

$$P_E(M) = 2Q\left(\sqrt{\frac{2E_s}{N_o}} \cdot sin\frac{\pi}{M}\right);\ E_s = E_b \cdot \log_2 M \qquad (17)$$

Finalment, si es concreta pel cas de QPSK ($M = 2^k = 4$):

$$P_E(4) = 2Q\left(\sqrt{\frac{2E_b * \log_2 4}{N_o}} \cdot sin\frac{\pi}{4}\right) = 2Q\left(\sqrt{\frac{4E_b}{N_o}} \cdot \frac{1}{\sqrt{2}}\right) \qquad (18)$$

$$BER = \frac{P_E(4)}{\log_2 4} = \frac{2}{\log_2 4} Q\left(\sqrt{\frac{2E_b}{N_o}}\right) = Q\left(\sqrt{\frac{2E_b}{N_o}}\right) \quad (19)$$

Així doncs, es pot confirmar com el BER d'un senyal modulat amb QPSK és el mateix que en el cas de BPSK, tal i com es mostra en la Figura 13.

4 Configuració i Posada en Marxa de l'Adalm Pluto

Al llarg d'aquest apartat es mostrarà com configurar l'Adalm Pluto des de zero amb les últimes actualitzacions disponibles i amb totes les funcionalitats que Matlab disposa per a aquest dispositiu per tal que qualsevol estudiant pugui començar a treballar amb el mòdul des de zero.

4.1 Configuració

La configuració de l'Adalm Pluto destaca per ser ràpida i intuïtiva, seguint els passos següents es pot aconseguir el dispositiu configurat correctament amb l'última versió.

El primer pas és connectar l'Adalm Pluto a l'ordinador mitjançant el cable USB que ens proporciona el mòdul. L'ordinador el detecta i apareix com a un dispositiu d'emmagatzematge de memòria amb una sèrie de fitxers ja instal·lats, tal com es mostra a la figura 20.

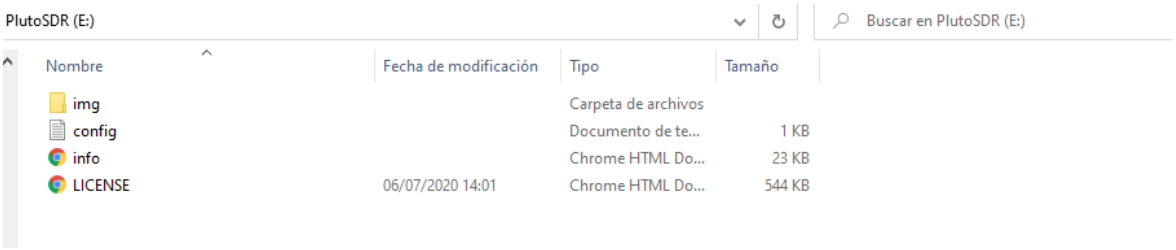

Figura 20. Emmagatzematge del mòdul Adalm Pluto

El fitxer més important és "info.html", aquest fitxer obrirà una pàgina web en el buscador amb la informació bàsica que es necessita. Juntament amb informació sobre el dispositiu i una sèrie d'enllaços d'interès, s'indica un enllaç per a descarregar els drivers i llibreries que es necessiten per a operar amb l'Adalm Pluto.

Un cop els drivers s'han descarregat es revisa l'apartat de firmware de la pàgina web (veure figura 21), allí s'indicarà si hi ha una nova versió per a actualitzar i, com es veu en la taula, es mostrarà quina versió té actualment el dispositiu (en aquest cas la versió és la v0.32).

Model	Analog Devices PlutoSDR Rev.B (Z7010-AD9363A)
Serial	1044739659930001faff07008b3e29236f
Build	v0.32
Linux	Linux pluto 4.19.0-119999-g6edc6cd #319 SMP PREEMPT Mon Jul 6 15:45:01 CEST 2020 armv7l GNU-Linux;1 core(s)
U-Boot	U-Boot PlutoSDR v0.20-PlutoSDR-00053-g89d0754 (Jan 14 2019 - 13:15:56 +0100)
FPGA	2019_r1-139-g847f
Root FS	2019.02.2-405-g1daa6
IIO	Library version: 0.21 (git tag: v0.21)

Figura 21. Informació sobre la versió de firmware

Si el firmware està actualitzat apareixerà un missatge confirmant que el dispositiu està treballant amb l'última versió disponible i, en cas contrari, proporcionarà un enllaç de descàrrega per a actualitzar el PlutoSDR, tal com es veu a les figures 22 i 23.

Figura 22. Missatge amb enllaç de descàrrega

Figura 23. Missatge de confirmació de la versió de firmware

A continuació cal descarregar l'última versió del firmware disponible (la descàrrega es fa en forma de fitxer .zip que, un cop descomprimit, es copia el fitxer "pluto.frm" i s'ha de col·locar dins de la carpeta d'emmagatzematge de l'Adalm Pluto). La resta de fitxers que es mostren a la figura 24 no són necessaris per a l'actualització.

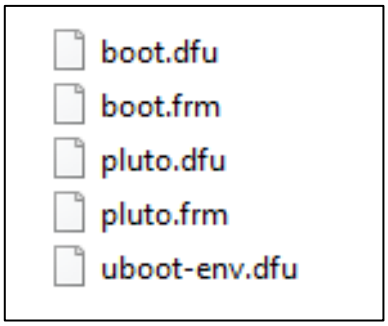

Figura 24. Fitxers de firmware

Un cop guardat el fitxer cal fixar-se tal com es mostra en la figura 25 en les dues opcions per a extreure el mòdul i seleccionar la segona (- *Expulsar PlutoSDR (E:)*), aquesta provoca que l'Adalm Pluto es reinicií i actualitzi el firmware. Aquest procés trigarà uns minuts (aproximadament fins a 5 min) i veurem que el LED1 farà pampallugues contínuament. És important no desconnectar l'alimentació durant l'actualització ja que podria provocar que l'Adalm Pluto no funcioni correctament [34].

Figura 25. Opcions per a reiniciar l'Adalm Pluto

Quan acabi el procés d'actualització el LED1 deixarà de fer pampallugues tan ràpid i l'ordinador tornarà a detectar l'Adalm Pluto. A més a més, el fitxer "pluto.frm" haurà desaparegut.

Per a comprovar que tot s'ha carregat bé es pot tornar a revisar el fitxer *.html* d'abans i hauria d'indicar que ja tenim l'última versió de firmware. Finalment es pot utilitzar la comanda: *iio_info -s* per veure com es detecta correctament el dispositiu, tal com es veu a la figura 26.

```
C:\Users\ASUS>iio_info -s
Library version: 0.21 (git tag: 565bf68)
Compiled with backends: xml ip usb serial
Unable to create Local IIO context : Function not implemented
Available contexts:
        0: 0456:b673 (Analog Devices Inc. PlutoSDR (ADALM-PLUTO)), serial=1044739659930001faff07008b3e29236f [usb:1.4.5]
        1: 192.168.2.1 (Analog Devices PlutoSDR Rev.B (Z7010-AD9363A)), serial=1044739659930001faff07008b3e29236f [ip:pluto.local]
```

Figura 26. Informació del Pluto mitjançant comandes

Es mostrarà la versió actual de la llibreria (v0.21) i, a més a més, el número de sèrie únic de cada mòdul i la URI que s'ha assignat al port USB on s'ha connectat (en aquest cas usb:1.4.5).

4.2 Matlab Communications Toolbox

Un cop ja es té l'Adalm Pluto operatiu i amb les últimes actualitzacions és necessari configurar Matlab per tal que permeti fer servir totes les comandes i funcions relacionades amb el mòdul.

A continuació s'ha d'obrir Matlab i dirigir-se a (veure figura 27):

Home --> Environment --> Add-Ons --> Get Hardware Support Packages i buscar el *Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio*.

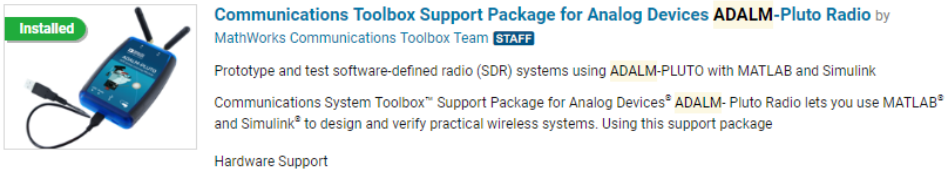

Figura 27. Toolbox per a PlutoSDR de Matlab

Cal descarregar el paquet i seguir les instruccions que detalla Matlab per a fer la instal·lació (aquest paquet de comunicacions va ser implementat en la versió de firmware v0.26 i no suporta versions anteriors, per això és important haver actualitzat l'Adalm Pluto prèviament).

A part d'incorporar un gran número de funcions relacionades amb Adalm Pluto, tal com es mostra a la figura 28 el propi toolbox permet comprovar el correcte funcionament tant de l'antena emissora com de la receptora i també indica quina URI d'usb té el mòdul (per defecte Matlab selecciona la direcció usb:0 i no hi haurà cap problema si només es fa servir

un Adalm Pluto; en cas de fer-ne servir dos o més es recomana fer servir cadascun amb un ordinador per separat o, com a mínim, crear una sessió de Matlab per a cadascun dels mòduls) [35].

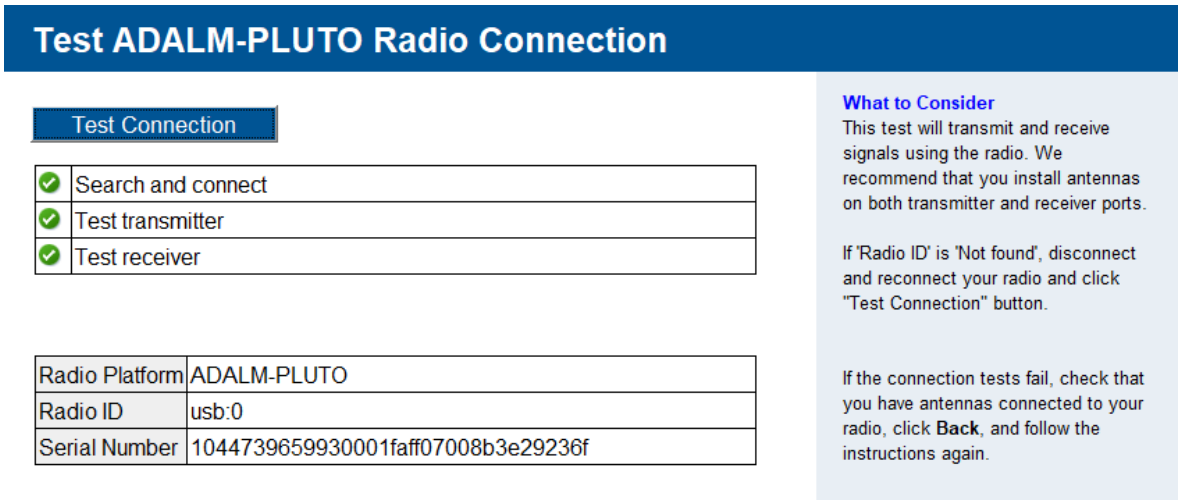

Figura 28. Resultat del test de les antenes del PlutoSDR

Un cop tot està correctament configurat, es pot començar a treballar amb l'Adalm Pluto. Un dels seus avantatges és que permet treballar tant amb Simulink com amb Matlab (*veure apartat 2.4.3*) però en aquest projecte s'ha decidit, juntament amb el tutor, treballar únicament amb Matlab ja que per a poder aplicar els resultats del treball en les pràctiques d'assignatures de la carrera és més pràctic pels alumnes seguir fent servir l'entorn amb el que estan més familiaritzats (Matlab), doncs al llarg del grau no es fa servir gairebé Simulink.

4.3 Primers Passos i Comunicació d'un To

L'objectiu del treball és utilitzar SDR per tal d'aplicar els coneixements teòrics de la carrera a casos pràctics real. De forma similar, un cop apresa la teoria darrere dels dispositius SDR i de les comunicacions digitals i amb l'Adalm Pluto connectat, el següent pas és passar a fer diferents proves pràctiques d'aplicació amb el dispositiu.

Per tal de comprovar el funcionament tant del mòdul Adalm Pluto, com dels objectes, les funcions i les llibreries incloses en el *Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio*, cal començar per les parts més bàsiques i anar augmentant la complexitat progressivament.

La funció principal d'un sistema de comunicacions és poder enviar i rebre dades, així doncs, el primer pas és generar el senyal que s'enviarà. L'entorn de Matlab facilita molt aquest aspecte ja que podem crear senyals de maneres molt diverses, ja sigui a partir d'un missatge (com en el cas de l'apartat 5), generant una seqüència aleatòria de bits per a enviar o directament utilitzant un senyal sinusoidal. En aquest cas es fa servir directament una ona sinusoidal (veure codi 1).

```
%% Generació del senyal
Fs = 30.72e6;           %freq de mostreig
Fc = 1e6;               %freq central del senyal
t = 1/Fs:1/Fs:ch_size/Fs;
amplitud = 1024;
sigR = sin(2*pi*Fc*t+0).*amplitud;
sigC = sin(2*pi*Fc*t+pi/2).*amplitud;
sig = complex(sigR,sigC); %senyal complex
```

Codi 1. Generació del senyal complex

Es genera un senyal complex a una freqüència central de 1 MHz, una amplitud de 1024 i una freqüència de mostreig f_s (Fs al codi) bastant més elevada per tal de complir amb el teorema de Nyquist:

$$fs \geq 2fmax \qquad (20)$$

On f_{max} és la component espectral de més alta freqüència del senyal a mostrejar. D'aquesta manera és segur poder reconstruir el senyal en recepció sense preocupar-se per l'aliasing. Per a poder enviar i rebre el senyal cal crear un objecte de sistema per a la transmissió i un objecte per a la recepció, tal com es mostra a codi 2.

```
%% Declaració Objectes PlutoSDR
rx = sdrrx('Pluto');
tx = sdrtx('Pluto');

rx.GainSource = 'AGC Fast Attack';
ch_size = 1e6;
rx.SamplesPerFrame = 1e6;
```

Codi 2. Declaració d'Objectes de Sistema

Quan es declara un nou objecte de sistema sense especificar cap paràmetre, s'utilitzen els valors per defecte mostrats a la figura 29 [36]:

```
rx = sdrrx('Pluto')

rx =

  comm.SDRRxPluto with properties:

                DeviceName: 'Pluto'
                   RadioID: 'usb:0'
             CenterFrequency: 2.4000e+09
                  GainSource: 'AGC Slow Attack'
              ChannelMapping: 1
           BasebandSampleRate: 1000000
              OutputDataType: 'int16'
            SamplesPerFrame: 3660
      ShowAdvancedProperties: false
```

Figura 29. Propietats per Defecte de l'Objecte de Recepció

Únicament s'han de modificar el paràmetre *GainSource* per a configurar-lo per a senyals que canvien de nivell de potència ràpidament i el *SamplesPerFrame* per a controlar el número de mostres que hi haurà en cada trama durant la comunicació.

Finalment, podem enviar el senyal a través de l'antena transmissora i rebre la informació a través de l'antena receptora fent servir el llistat d'instruccions que s'ha desenvolupat i que es mostren a Codi 3.

```
%% Transmit and receive with PlutoSDR
tx.transmitRepeat(sig.');
frames = 10;
cap = zeros(ch_size*frames,1);
prev = 0;
for frame = 1:frames
    % Call radio
    o = rx();
    % Save data
    indx = (frame-1)*ch_size+1 : frame*ch_size;
    cap(indx) = o;
    % Info
    s = sprintf('Frame %d of %d',frame,frames);
    fprintf(repmat('\b',1,prev));fprintf(s);prev = length(s);
end
fprintf('\n');
```

Codi 3. Transmissió i Recepció del senyal

Tal com s'observa a la figura 30, si es representa gràficament la part real i imaginària del senyal complex que es genera a la transmissió i es compara amb el senyal que tenim en recepció es pot veure com si bé són similars, aquest últim està distorsionat. Aquest efecte hem deduït que prové degut a diferents problemes associats a utilitzar el mateix Adalm Pluto per transmetre i rebre, i que no s'han pogut solucionar amb els diferents paràmetres que podíem configurar ni amb elements d'atenuació externs per evitar possibles saturacions per excés de potència en recepció.

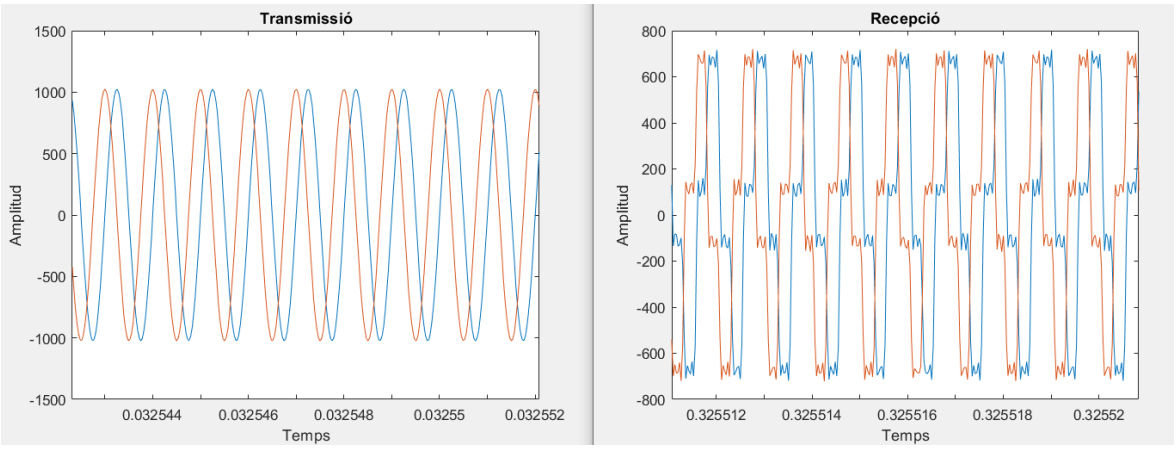

Figura 30. Senyal Transmès i Senyal Rebut

5 Transmissió del Senyal

A continuació, fent servir de referència els diversos exemples que Matlab proporciona per a treballar amb el dispositiu SDR es mostra l'objectiu principal del projecte, que és la comunicació d'informació entre dos Adalm Pluto mitjançant modulació QPSK. A diferència de l'apartat anterior farem servir dos dispositius. Cada dispositiu estarà connectat a un ordinador diferent per tal d'evitar problemes que deguts a la gestió dels ports USB de Matlab. Un Adalm Pluto s'encarregarà de la transmissió i l'altre de la recepció.

En aquest apartat s'explica tot allò relacionat amb la part de transmissió, generació i modulació del senyal. El codi es troba dividit en diverses parts, ja que cadascuna aporta una característica diferent a la transmissió.

Per realitzar la transmissió del senyal es faran servir les següents funcions i classes:

- **plutoradioqpsktransmiter_init:** Funció que inicialitza els paràmetres de la comunicació, les especificacions de les trames (mida de les trames, codi Barker per al sincronisme i el missatge de la trama), i els paràmetres de l'objecte de transmissió com poden ser el factor de roll-off, el tipus de filtre que es farà servir o la freqüència a la que es transmetrà.
- **runPlutoradioQPSKTransmitter:** En aquesta funció s'agafen els valors inicialitzats prèviament i es carreguen en l'objecte de transmissió de l'Adalm Pluto i comença la transmissió de dades. A diferència de recepció, el transmissor estarà constantment enviant la informació sense parar, mentre que el receptor un cop completi la lectura de la informació acabarà la seva execució.

- **QPSKTransmitter:** Aquesta classe conté la informació i funcions necessàries per a aplicar la modulació (QPSK) al senyal que es vol transmetre i defineix quines característiques tindrà la modulació (com podria ser afegir un offset de fase).

- **QPSKBitsGenerator:** Classe que serveix per generar els bits corresponents a cadascuna de les trames que s'enviaran.

5.1 Generació del Senyal

El primer objectiu de la transmissió és la generació del senyal i la configuració dels paràmetres de la comunicació.

A diferència de la comunicació d'un to, degut a l'increment de complexitat en aquest cas, cal tenir en compte la gran majoria (si no tots) els paràmetres dels objectes que creem per a controlar al màxim el sistema i no fer servir valors per defecte.

En la funció plutoradioqpsktransmitter_init es guarden aquests valors en la variable que se li passi (prmQPSKTransmitter) per a després aplicar-los durant la declaració dels objectes de transmissió, i que es mostren a codi 4. A cada paràmetre s'hi ha afegit el comentari per saber a què es correspon.

```
%% Parametres generals
SimParams.Rsym = 0.2e6;              % Velocitat de simbol Rs
SimParams.ModulationOrder = 4;       % Ordre de la modulacio (QPSK)
SimParams.Interpolation = 2;
```

```
SimParams.Decimation = 1;
SimParams.Tsym = 1/SimParams.Rsym;  % Temps de simbol Ts=1/Rs
SimParams.Fs   = SimParams.Rsym * SimParams.Interpolation; % Freq de
mostreig
```

Codi 4. Paràmetres generals

Cal establir els valors generals de la comunicació com poden ser la velocitat de símbols o l'odre de la modulació que es farà servir (en aquest cas QPSK). També cal assegurar-se de mantenir els valors en recepció per a recuperar el senyal correctament. Com a paràmetres generals també apareix el guany de l'antena transmissora, el factor de roll-off i la freqüència de la portadora.

Per tal de no generar bits aleatoris per a formar el senyal a transmetre, he fet servir una de les frases més conegudes en el món de la programació com a missatge: "Hello world", aquest missatge és la base de la trama que s'enviarà, i es mostra a codi 5.

```
%% Parametres i estructura de la trama
% Codi Barker per al sincronisme
SimParams.BarkerCode      = [+1 +1 +1 +1 +1 -1 -1 +1 +1 -1 +1 -1 +1];
SimParams.BarkerLength    = length(SimParams.BarkerCode);
% Es repeteix dos cops el codi com a capçalera
SimParams.HeaderLength    = SimParams.BarkerLength * 2;
% Missatge = 'Hello world 000\n'
SimParams.Message         = 'Hello world';
SimParams.MessageLength   = length(SimParams.Message) + 5;
% 5 = espai + 3 nº + salt de linia

% Nombre de missatges en cada trama
SimParams.NumberOfMessage = 100;
% Tamany = nº missatges * tamany missatge * 7 (nº de bits per char)
SimParams.PayloadLength   = SimParams.NumberOfMessage *
SimParams.MessageLength * 7;
% Tamany total de la trama (en simbols)
SimParams.FrameSize       = (SimParams.HeaderLength +
SimParams.PayloadLength) ...
    / log2(SimParams.ModulationOrder);
% Temps de trama = Tsimbol * Tamany de la trama (en simbols)
SimParams.FrameTime       = SimParams.Tsym*SimParams.FrameSize;
```

Codi 5. Paràmetres i estructura de la trama

La trama que s'ha generat té la següent estructura:

Codi Barker | Codi Barker | 'Hello world 000\n' | 'Hello world 001\n' | ... | 'Hello world 099\n'

Com s'explica en els apartats anteriors del treball els sistemes de comunicació digital requereixen de tècniques per a assegurar el sincronisme entre transmissor i receptor, en aquest cas s'afegeix dos cops el codi Barker de 13 elements (+ + + + + − − + + − + − +) com a capçalera de la trama per a poder separar les trames en recepció. S'afegeix el codi Barker dos cops per tal que un cop aplicada la modulació aquest s'enviï tant en fase com en quadratura. La resta de la trama és una seqüència del missatge "Hello world" junt amb un número que es va incrementant en cada nova instància del missatge i seguit d'un salt de línia.

El número i el salt de línia s'han incorporat únicament per tal de facilitar la visualització de la informació ja que el receptor mostrarà per pantalla el missatge a mesura que el va rebent.

Un cop generada la trama de prova, es converteixen a bits els caràcters per a seguir amb el procés de la modulació del senyal. Originalment, en els exemples que proporciona Matlab es fa servir l'objecte IntegerToBit que forma part del Communications Toolbox però en pròximes versions del programa s'eliminarà [37], tal com es mostra a la figura 31. Per aquesta raó s'ha canviat a la funció de2bi per tal d'evitar problemes de compatibilitat amb futures versions de Matlab (ja que l'objectiu és poder aplicar-ho en pràctiques de laboratori en un futur)[38].

Figura 31. Alerta de MathWorks en l'Objecte IntegerToBit del Communications Toolbox

5.2 Modulació PSK

La funció runPlutoradioQPSKTransmitter que es mostra a codi 6 rep tots els paràmetres inicialitzats de l'apartat anterior a través de la variable prmQPSKTransmitter i permet carregar els valors en els objectes de sistema que es creen.

```
% Create and configure the transmitter System object
    hTx = QPSKTransmitter(...
        'UpsamplingFactor',        prmQPSKTransmitter.Interpolation, ...
        'RolloffFactor',           prmQPSKTransmitter.RolloffFactor, ...
        'RaisedCosineFilterSpan',
prmQPSKTransmitter.RaisedCosineFilterSpan, ...
        'MessageBits',             prmQPSKTransmitter.MessageBits, ...
        'MessageLength',           prmQPSKTransmitter.MessageLength, ...
        'NumberOfMessage',
prmQPSKTransmitter.NumberOfMessage, ...
        'ScramblerBase',           prmQPSKTransmitter.ScramblerBase, ...
        'ScramblerPolynomial',
prmQPSKTransmitter.ScramblerPolynomial, ...
        'ScramblerInitialConditions',
prmQPSKTransmitter.ScramblerInitialConditions);
```

Codi 6. Declaració d'Objecte de la Classe QPSKTransmitter

L'objecte de transmissió hTx té dues finalitats principals, la primera és preparar el senyal i els bits d'informació que conté i la segona és aplicar la modulació QPSK.

5.2.1 Scrambler

En la funció anterior s'ha convertit el missatge de la trama d'informació en bits però abans d'aplicar la modulació es modifiquen els bits. Per tal de d'assegurar una distribució uniforme d'uns i zeros per a les funcions de *timing recovery* que s'utilitzen en la recepció es fa servir un aleatoritzador (scrambler) mitjançant l'objecte comm.Scrambler del Communications Toolbox i la classe QPSKBits Generator [39].

Un scrambler és bàsicament un dispositiu que modifica la seqüència de les dades que li arriben per una nova seqüència sense perdre informació i amb una menor probabilitat de que es repeteixin molts zeros o uns seguits [40].

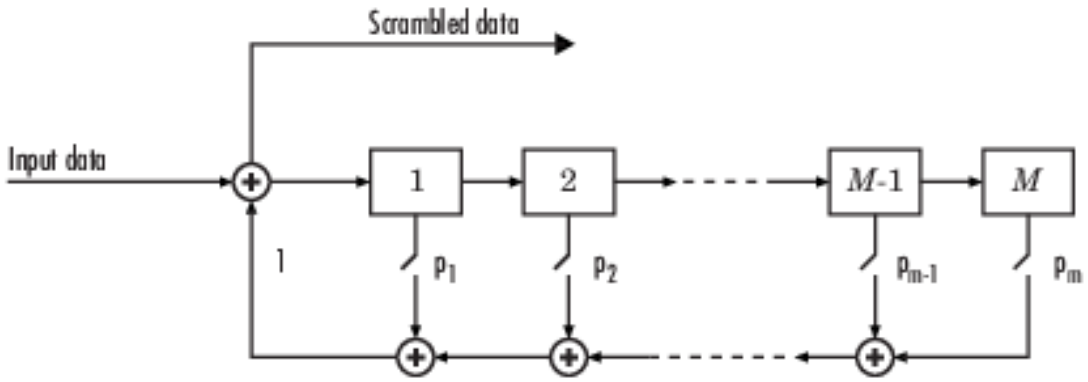

Figura 32. Diagrama de comm.Scrambler

Es crea l'objecte indicant que les dades que rebrà són de base 2 (ScramblerBase), s'utilitzarà el polinomi [1 1 1 0 1] per a definir l'estat (on o off) dels *switch* (ScramblerPolynomial) i les condicions inicials dels registres de l'*scrambler* [0 0 0 0 0] (ScramblerInitialConditions).

5.2.2 QPSK Modulator i Root-Raised-Cosine Filter (RRC)

Un cop les dades ja són uniformes, s'aplica la modulació QPSK amb l'objecte comm.QPSKModulator i amb els paràmetres que es mostren a codi 7.

```
% Creacio objecte per modulacio QPSK
obj.pQPSKModulator  = comm.QPSKModulator( ...
                'BitInput',                     true, ...
                'PhaseOffset',                  pi/4, ...
                'OutputDataType',               'double');
```

Codi 7. Declaració d'Objecte comm.QPSKModulator

A l'aplicar la modulació podem comprovar a la figura 33 el resultat amb l'objecte comm.ConstellationDiagram per a mostrar el diagrama de constel·lació del senyal modulat, verificant que efectivament es correspon a una QPSK tal com s'ha explicat a la teoria en apartats anteriors.

Es pot observar com tots els símbols generats estan concentrats en els 4 diferents valors típics de la modulació QPSK, amb una diferència de 90º entre símbols i amb l'offset inicial de fase de 45º.

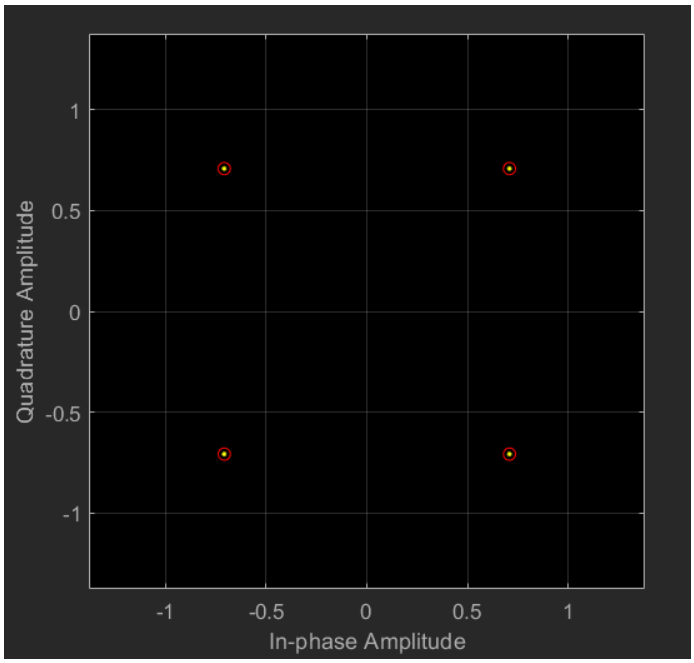

Figura 33. Constel·lació del Senyal Modulat QPSK

Amb la certesa que la modulació s'ha fet correctament el següent pas és fer passar el senyal modulat per un filtre conformador de polsos root-raised-cosine filter (RRC) per tal de tenir la distribució de potència del senyal transmès confinada en un ample de banda i generar el mínim de senyal fora d'aquest per minimitzar la interferència co-canal. Això es pot observar a codi 8. S'ha optat per aquest tipus filtre ja que no genera interferència intersimbòlica (ISI).

```
% Creacio objecte filtre RRC
obj.pTransmitterFilter = comm.RaisedCosineTransmitFilter( ...
                'RolloffFactor',                obj.RolloffFactor, ...
                'FilterSpanInSymbols',
obj.RaisedCosineFilterSpan, ...
                'OutputSamplesPerSymbol',       obj.UpsamplingFactor);
```

Codi 8. Declaració d'Objecte comm.QPSKModulator

Els filtres conformadors ens permeten reduïr la ISI i requerir d'un menor ample de banda (sempre per sobre de l'ample de banda de Nyquist, que és l'ample de banda mínim per a suportar la transmissió sense ISI). Els filtres rectangulars són inviables ja que produirien distribucions espectrals de banda il·limitada. Així doncs, s'utilitzen filtres conformadors de tipus cosinus realçat i el paràmetre més important d'aquests filtres és el factor de roll-off (en el nostre cas s'utilitza 0.5) que serveix per mesurar l'excés d'ample de banda que ocupa per sobre del mínim teòric de Nyquist. Com es pot veure en la figura 34 (on β = factor de roll-off) a mesura que disminueix el valor de β el filtre s'acosta a la forma d'un filtre rectangular.

També hem utilitzat un filtre root cosinus, tal com es realitza en algunes comunicacions, el qual s'ha d'aplicar un cop en el transmissor i un cop en el receptor (a l'aplicar-se dos cops, el seu comportament global és el mateix que el d'un cosinus realçat).

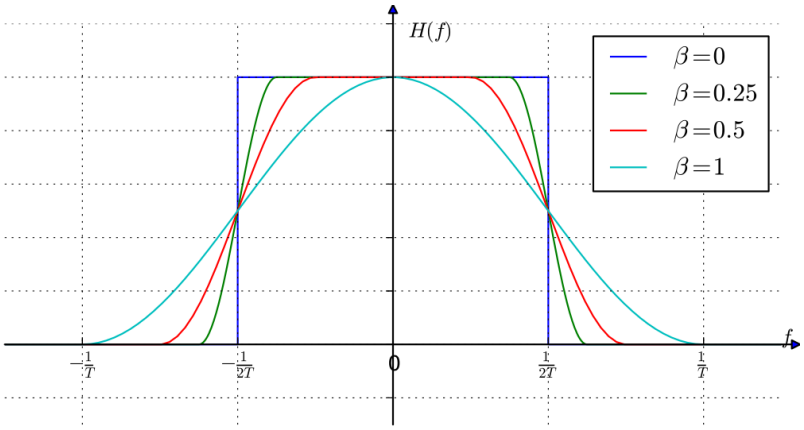

Figura 34. Funció de Transferència Root Cosinus Filter

Si observem a la figura 35 el diagrama de l'ull del senyal generat a la sortida del filtre conformador mitjançant la funció eyediagram de Matlab s'observa el resultat que ha tingut en el senyal (en la figura es mostra la forma després d'un filtre cosinus realçat i la forma després del root cosinus que es fa servir en el codi i que més tard al receptor s'ha de tornar a aplicar).

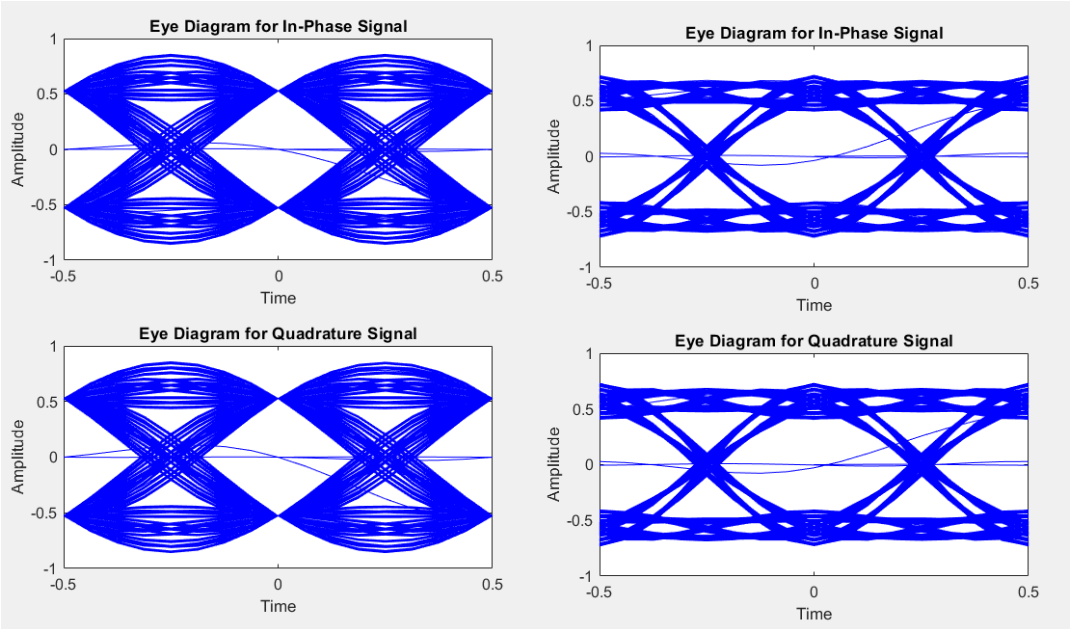

Figura 35. Diagrama de l'ull després del Filtre (esquerra cosinus realçat, dreta root cosinus)

5.3 Enviament del Senyal

Per tal d'enviar el senyal es crea un objecte de sistema de transmissió *sdrtx* propi del *Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio* (veure codi 9) que es connecta directament al hardware de la ràdio i es manté connectat fins que no es crida la funció *release()*.

```
% Creacio objecte de transmissio del Pluto
    radio = sdrtx('Pluto');
    radio.RadioID               = prmQPSKTransmitter.Address;
    radio.CenterFrequency       =
prmQPSKTransmitter.PlutoCenterFrequency;
    radio.BasebandSampleRate    =
prmQPSKTransmitter.PlutoFrontEndSampleRate;
    radio.SamplesPerFrame       = prmQPSKTransmitter.PlutoFrameLength;
    radio.Gain                  = prmQPSKTransmitter.PlutoGain;
```

Codi 9. Declaració d'Objecte de Transmissió

Es configuren tots els paràmetres que s'havien inicialitzat i s'indica la ID associada al port USB on es connecta l'Adalm Pluto i que en l'apartat 4.3 s'ha explicat com trobar-la. Per tant, ara ja tenim el primer Adalm Pluto transmetent un senyal que hem generat amb modulació QPSK.

6 Recepció del Senyal

L'Adalm Pluto que es dedica a la recepció s'encarrega de, un cop rebut el senyal enviat per l'altre dispositiu, aplicar tota una sèrie de processos de compensació de freqüència, sincronització de trames i resolució d'ambigüitat per tal de desmodular el senyal i poder regenerar els missatges que s'envien en la transmissió.

L'estructura del codi és bastant similar al cas de transmissió, on es creen una sèrie d'objectes de recepció a partir del *Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio* i, un cop s'inicialitzen els valors dels objectes, es configuren a l'Adalm Pluto.

Per a la recepció del senyal es faran servir les diferents funcions i classes:

- **plutoradioqpskreceiver_init:** Funció que s'encarrega d'inicialitzar els paràmetres de la comunicació d'acord amb els valors escollits en transmissió per a que concordin. També es torna a crear el missatge enviat amb el codi Barker per tal de poder comparar el senyal que rebem amb el transmès i calcular l'error que s'ha produït.
- **QPSKReceiver:** Aquesta classe és la base per a la desmodulació del senyal rebut, en les seves propietats es guarden els paràmetres que es faran servir per a la desmodulació i, al mateix temps inclou vàries funcions de Control de Guany Automàtic (AGC), compensació de freqüència, correcció dels errors de temps en els símbols, detecció del codi Barker implementant un algoritme de correlació i sincronització de trames.
- **runPlutoradioQPSKReceiver:** Funció per a configurar els valors preestablerts en la inicialització als objectes de recepció de l'Adalm Pluto, es desmodula el senyal fent servir les funcions de la classe QPSKReceiver i retorna el valor dels missatges rebuts i del càlcul de BER.

6.1 Objecte per a la Recepció

La funció plutoradioqpskreceiver_init s'encarrega del mateix procés d'inicialització que la seva contrapart en la transmissió. Els paràmetres que es guarden són els mateixos ja que necessitem que els valors siguin coherents amb els del transmissor, amb excepció de l'objecte de sistema de recepció. Es pot observar al codi 10.

```
%% Rx parameters
% Filtre i descrambler
SimParams.RolloffFactor     = 0.5;
SimParams.ScramblerBase     = 2;
SimParams.ScramblerPolynomial           = [1 1 1 0 1];
SimParams.ScramblerInitialConditions    = [0 0 0 0];
SimParams.RaisedCosineFilterSpan = 10;                  % SPAN en simbols

% Paràmetres pel Control de Guany Automatic (AGC)
SimParams.DesiredPower                  = 2;            % Potencia
desitjada(W)
SimParams.AveragingLength               = 50;
SimParams.MaxPowerGain                  = 60;



SimParams.MaximumFrequencyOffset        = 6e3;
% Look into model for details for details of PLL parameter choice.
```

```
% Refer equation 7.30 of "Digital Communications - A Discrete-Time
Approach" by Michael Rice.
K = 1;
A = 1/sqrt(2);
SimParams.PhaseRecoveryLoopBandwidth    = 0.01;         % Normalized loop
bandwidth for fine frequency compensation
SimParams.PhaseRecoveryDampingFactor    = 1;            % Damping Factor
for fine frequency compensation
SimParams.TimingRecoveryLoopBandwidth   = 0.01;         % Normalized loop
bandwidth for timing recovery
SimParams.TimingRecoveryDampingFactor   = 1;            % Damping Factor
for timing recovery
% K_p for Timing Recovery PLL, determined by 2KA^2*2.7 (for binary PAM),
% QPSK could be treated as two individual binary PAM,
SimParams.TimingErrorDetectorGain       = 2.7*2*K*A^2+2.7*2*K*A^2;
SimParams.PreambleDetectorThreshold     = 0.8;
```

Codi 10. Inicialització dels Paràmetres per l'objecte de recepció

Cadascun d'aquests paràmetres servirà per a un procés diferent durant el bloc de recuperació del senyal i desmodulació. En el següent apartat s'entrarà més en detall en els processos que s'apliquen per a recuperar el senyal.

6.2 Desmodulació i Recuperació del senyal

Per tal d'obtenir el missatge original el senyal haurà de passar per diversos passos per tal de recuperar el senyal, que haurà patit múltiples efectes negatius degut al propi sistema de comunicació i la transmissió per l'aire (soroll, distorsió, interferències...). Amb els valors de la funció plutoradioqpskreceiver_init es declara una instància de la classe QPSKReceiver anomenada *rx* que s'encarrega d'aplicar les modificacions al senyal rebut per a desxifrar el missatge. El senyal es rebrà a partir de l'objecte de recepció *sdrrx* que es connecta al hardware de l'Adalm Pluto fins que s'utilitzi la comanda *release()*.

Aquests mètodes de recuperació del senyal poden servir en unes possibles pràctiques per a que els alumnes vegin els processos a seguir un cop es treballa amb senyals reals i fent servir un canal que no es pot controlar (l'aire), així que a continuació s'explica el seu funcionament.

6.2.1 Control de Guany Automàtic (AGC)

Fent servir l'objecte comm.AGC que es mostra a codi 11, es crea un AGC que actua com a regulador del guany del receptor i contínuament adapta el seu valor per a mantenir un nivell de senyal constant a la sortida. D'acord amb els valors inicialitzats s'encarregarà de mantenir el nivell de senyal a la sortida a 2W però limitant el guany màxim a 60 dB [41].

```
obj.pAGC = comm.AGC( ...
       'DesiredOutputPower',        obj.DesiredPower, ...
       'AveragingLength',           obj.AveragingLength, ...
       'MaxPowerGain',              obj.MaxPowerGain);
```

Codi 11. Declaració de l'objecte d'AGC

6.2.2 *Root-Raised-Cosine Filter (RRC)*

Seguint l'explicació en l'apartat de transmissió, al fer servir un filtre conformador root cosinus es necessita aplicar el mateix filtre també en recepció (codi 12) amb l'objecte comm.RaisedCosinefilter i mantenint els mateixos valors que en l'emissió i, tot i que es pot delmar (decimate) el senyal descartant mostres del senyal que poden ser redundants, mantenim totes les mostres configurant el valor de DecimationFactor a 1 [42].

```
obj.pRxFilter = comm.RaisedCosineReceiveFilter( ...
       'RolloffFactor',            obj.RolloffFactor, ...
       'FilterSpanInSymbols',      obj.RaisedCosineFilterSpan, ...
       'InputSamplesPerSymbol',    obj.InputSamplesPerSymbol, ...
       'DecimationFactor',         obj.DecimationFactor);
```

Codi 12. Declaració de l'objecte RRC

6.2.3 *Coarse Frequency Compensator*

Fent servir un algoritme basat en la correlació i adaptat per a diversos tipus de modulació, aquest objecte que es mostra a codi 13 permet estimar i compensar l'offset en la freqüència de l'ona portadora en el senyal rebut. Aquest offset arriba fins a un valor màxim de 6 kHz [43].

```
obj.pCoarseFreqEstimator = comm.CoarseFrequencyCompensator( ...
       'Modulation',               'QPSK', ...
       'Algorithm',                'Correlation-based', ...
       'MaximumFrequencyOffset',   obj.MaximumFrequencyOffset, ....
       'SampleRate',               obj.SampleRate/obj.DecimationFactor);
```

Codi 13. Declaració de l'objecte Coarse Frequency Compensator

6.2.4 *Carrier Synchronizer*

El sistema detecta i compensa l'offset en freqüència i en fase de l'ona portadora (codi 14) sempre que el senyal faci servir modulacions amb una única ona portadora (com és el cas de BPSK, QPSK, OQPSK, 8PSK...). Quan s'indica que la modulació és una QPSK tindrà en compte que la variable ModulationPhaseOffset són pi/4 radians [44].

```
obj.pFineFreqCompensator = comm.CarrierSynchronizer( ...
       'Modulation',               'QPSK', ...
       'ModulationPhaseOffset',    'Auto', ...
       'SamplesPerSymbol',         obj.PostFilterOversampling, ...
       'DampingFactor',            obj.PhaseRecoveryDampingFactor, ...
       'NormalizedLoopBandwidth',  obj.PhaseRecoveryLoopBandwidth);
```

Codi 14. Declaració de l'objecte Carrier Synchronizer

6.2.5 Symbol Synchronizer

Aquest objecte (codi 15) s'encarrega de corregir la desviació que hi pot haver entre els rellotges del transmissor i el receptor fent servir els valors inicialitzats en l'objecte d'acord amb les especificacions quan es treballa amb QPSK. Cal destacar que per a assegurar que la sincronització dels símbols és completa el valor de NormalizedLoopBandwith ha de ser menor a 0.1 (tot i que pot prendre qualsevol valor entre 0 i 1) [45].

```
obj.pTimingRec = comm.SymbolSynchronizer( ...
        'TimingErrorDetector',      'Gardner (non-data-aided)', ...
        'SamplesPerSymbol',         obj.PostFilterOversampling, ...
        'DampingFactor',            obj.TimingRecoveryDampingFactor, ...
        'NormalizedLoopBandwidth',  obj.TimingRecoveryLoopBandwidth, ...
        'DetectorGain',             obj.TimingErrorDetectorGain);
```

Codi 15. Declaració de l'objecte Symbol Synchronizer

6.2.6 Preamble Detector

Gràcies al Preamble Detector es pot reconèixer i detectar quins símbols formen part del preàmbul de tota la seqüència de dades que va rebent i automàticament l'objecte s'encarrega de trobar en quin punt de la seqüència acaba el preàmbul [46]. Es pot observar a codi 16.

```
obj.pPrbDet = comm.PreambleDetector(obj.pModulatedHeader, ...
        'Input',                    'Symbol', ...
        'Threshold',                obj.PreambleDetectorThreshold);
```

Codi 16. Declaració de l'objecte Preamble Detector

Finalment, després d'aplicar aquest procés en recepció, s'aconsegueix que el senyal sigui el màxim semblant a aquell que s'ha modulat i transmès i, tot i que no és exactament a l'original, permet distingir perfectament el valor dels bits, tal com es pot observar amb el diagrama de l'ull següent:

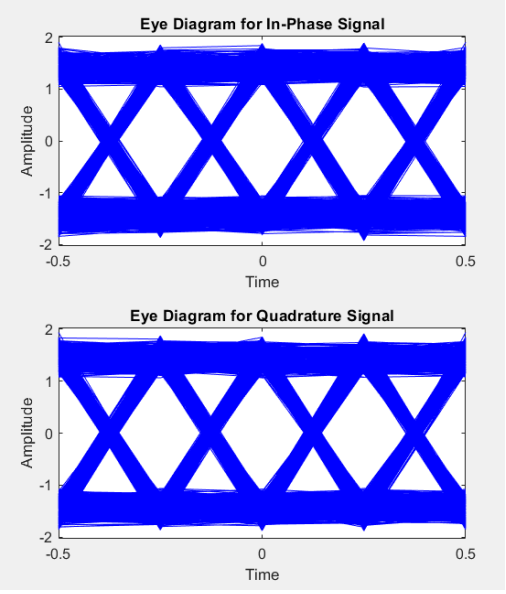

Figura 36. Senyal Recuperada

6.3 Missatge Resultant i Càlcul d'Error

Tal com es mostra a codi 17, utilitzant una instància de la classe interna QPSKDataDecoder es pot obtenir el missatge que arriba si es coneix l'estructura de les trames que hem generat en la transmissió i mostrar-lo per pantalla. De forma inversa a la transmissió s'aplica un desaleatoritzador (descrambler) fent servir el mateix polinomi base que en l'scrambler per a recuperar la seqüència original d'informació.

```
obj.pDataDecod = QPSKDataDecoder( ...
        'ModulationOrder',           obj.ModulationOrder, ...
        'HeaderLength',              obj.HeaderLength, ...
        'NumberOfMessage',           obj.NumberOfMessage, ...
        'PayloadLength',             obj.PayloadLength, ...
        'DescramblerBase',           obj.DescramblerBase, ...
        'DescramblerPolynomial',     obj.DescramblerPolynomial, ...
        'DescramblerInitialConditions', obj.DescramblerInitialConditions,
...
        'BerMask',                   obj.BerMask, ...
        'PrintOption',               obj.PrintOption);
```

Codi 17. Declaració de l'objecte QPSK Data Decoder

Un cop tenim les trames en l'ordre original es pot separar la capçalera i descartar-la, i convertir els bits restants en caràcters per a mostrar per pantalla el missatge obtingut. Al mateix temps, a partir del missatge original es fa un càlcul de la proporció de mostres errònies respecte les mostres totals per saber la probabilitat d'error d'aquesta comunicació. Això es pot observar a la figura 37 junt amb un altre exemple enviant el missatge "TFG Joaquim" enlloc de "Hello world".

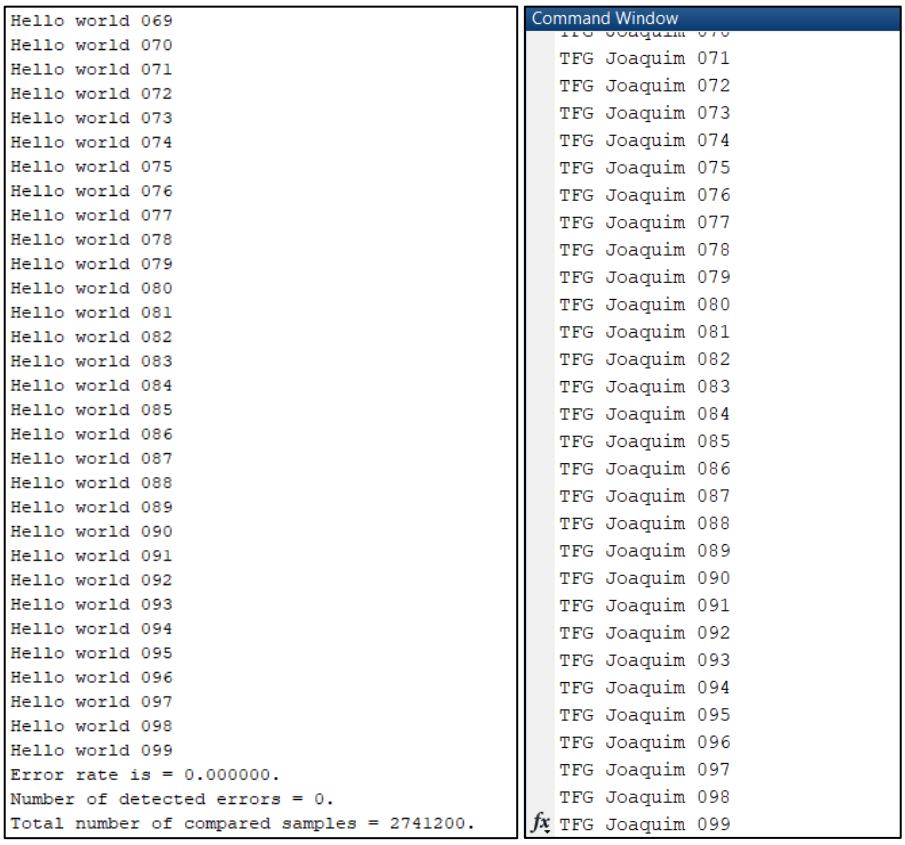

Figura 37. Missatge Rebut i càlcul d'Error

7 Conclusions

L'objectiu principal d'aquest treball ha sigut fer una primera aproximació als dispositius SDR per tal de comprendre el seu funcionament des de zero i avaluar les seves possibles aplicacions al món de l'ensenyament.

En primer lloc calia tenir en compte diversos productes del mercat per a comparar les seves característiques tècniques i decidir quin s'adapta millor a les necessitats dels alumnes. Finalment, es va escollir el mòdul Adalm Pluto ja que destacava per la combinació d'unes bones prestacions, un preu assequible i era un dispositiu dissenyat expressament per a l'aprenentatge actiu d'estudiants d'enginyeria. Així doncs, després d'haver realitzat el treball i veient els resultats obtinguts es pot dir que la decisió va ser la correcta, més enllà de pel seu bon funcionament i el seu baix cost, ha demostrat que és més que un simple dispositiu de Software Defined Radio si no que darrere seu hi ha tota una comunitat amb ganes d'aprendre i d'aprofitar al màxim els seus recursos. És fàcil trobar tota mena d'informació, guies i exemples de programes per a implementar o fins i tot pràctiques de laboratori que poden motivar als estudiants a seguir treballant amb el dispositiu pel seu compte fora de les sessions de laboratori.

També es buscava establir una comunicació real amb l'Adalm Pluto per aplicar els conceptes que s'han estudiat en assignatures com comunicacions digitals i al final s'ha pogut realitzar la comunicació aplicant una modulació QPSK. A mesura que el treball avançava també avançaven la complexitat i tot i que finalment només s'ha pogut fer servir un tipus de modulació, ens relaciona amb el tercer i últim objectiu que era servir de pont per a poder treure-hi més rendiment en un futur.

A més a més, la informació que s'ha aconseguit en aquest treball segurament pugui ajudar a altres estudiants que s'interessin en el tema per a que no hagin de començar de zero. Treballar amb el Pluto sembla que ha sigut un pas en la direcció correcta i té molt potencial per a fer-se servir en pràctiques de laboratori amb alumnes.

Referències

[1] Travis F. Collins, Robin Getz, Di Pu, Alexander M. Wyglinski, “Software-Defined Radio for Engineers”, Artech House, 2018

[2] Wikipedia, “Ràdio Definida per Software”, https://ca.wikipedia.org/wiki/R%C3%A0dio_Definida_per_Software. [ONLINE] 2/08/2021

[3] “Joint Tactical Radio System Programmable, Modular Communications”, https://www.globalsecurity.org/military/systems/ground/jtrs.htm. [ONLINE] 2/08/2021

[4] Youtube, “ADALM PLUTO Full Duplex Software Defined Radio” https://youtu.be/nlXiD-A2fA8. [ONLINE] 10/08/2021

[5] “SDR Transforms Amateur Radio”, https://www.electronicdesign.com/technologies/dsps/article/21762172/sdr-transforms-amateur-radio. [ONLINE] 10/08/2021

[6] Great Scott Gadgets, “HackRF One”, https://greatscottgadgets.com/hackrf/one/. [WEB OFICIAL] 14/08/2021

[7] Wikipedia, “HackRF One”, https://en.wikipedia.org/wiki/HackRF_One. [ONLINE] 14/08/2021

[8] Nooelec, “Nooelec NESDR SMArt” https://www.nooelec.com/store/sdr/sdr-receivers/nesdr-smart-sdr.html. [WEB OFICIAL] 14/08/2021

[9] FlexRadio, “FLEX-6700”, https://www.flexradio.com/products/flex-6700-signature-series-sdr-transceiver/. [WEB OFICIAL] 14/08/2021

[10] Analog Devices, “Adalm Pluto Overview”, https://wiki.analog.com/university/tools/pluto. [ONLINE] 15/08/2021

[11] Youtube, “ADALM PLUTO Frequency Expansion Modification”, https://www.youtube.com/watch?v=g1jIAh6fG_A&ab_channel=TechMinds. [ONLINE] 15/08/2021

[12] Wikipedia, “List of software-defines radios”, https://en.wikipedia.org/wiki/List_of_software-defined_radios. [ONLINE]

[13] Youtube, “Affordable RF tools, the ADALM-Pluto SDR”, https://youtu.be/qMeJS4gSIXo. [ONLINE] 17/08/2021

[14] Analog Devices, “ADALM-PLUTO Hardware”, https://wiki.analog.com/university/tools/pluto/hacking/hardware. [ONLINE] 17/08/2021

[15] Youtube, “Introduction to the ADALM-PLUTO SDR”, https://youtu.be/05nLPVJW9Uo. [ONLINE] 17/08/2021

[16] Analog Devices, “Adalm Pluto Transmit Architecture”, https://wiki.analog.com/university/tools/pluto/users/transmit. [ONLINE] 17/08/2021

[17] Analog Devices, “AD9361 Product Overview”, https://www.analog.com/en/products/ad9361.html#product-overview. [ONLINE] 17/08/2021

[18] Ramón Villarino, Antonio Lázaro, “Modulacions analògiques lineals”, Teoria del Tema 4 de l’assignatura Fonaments de Comunicacions II

[19] Ramón Villarino, Antonio Lázaro, “Modulacions analògiques angulars”, Teoria del Tema 5 de l’assignatura Fonaments de Comunicacions II

[20] “Síntesis de la modulación AM”, https://www.hispasonic.com/tutoriales/sintesis-27-modulacion-am-teoria-basica/39919. [ONLINE] 10/08/2021

[21] Ramón Villarino, “Introducció a les Comunicacions”, Teoria del Tema 1 de l’assignatura Fonaments de Comunicacions II

[22] David Girbau Sala, “Transmissió Digital Banda Base”, Teoria del Tema 3 de l’assignatura Comunicacions Digitals

[23] “Digital Modulations”, https://www.usna.edu/ECE/ec312/Lessons/wireless/EC312_Lesson_21_Digital_Modulation_Course_Notes.pdf. [PDF ONLINE], 20/08/2021

[24] David Girbau Sala, “Transmissió Digital Pas Banda”, Teoria del Tema 4 de l’assignatura Comunicacions Digitals

[25] Lou Frenzel, “Understanding Modern Digital Modulation Techniques”, https://www.electronicdesign.com/technologies/communications/article/21798737/understanding-modern-digital-modulation-techniques. [ONLINE] 20/08/2021

[26] Wikipedia, “Phase-shift keying”, https://en.wikipedia.org/wiki/Phase-shift_keying. [ONLINE] 20/08/2021

[27] “Modulación digital PSK o BPSK”, https://mundotelecomunicaciones1.blogspot.com/2014/10/modulacion-digital-psk-o-bpsk-binary.html. [ONLINE] 20/08/2021

[28] Agamenon, “Sistemas de comunicación digital” http://agamenon.tsc.uah.es/Asignaturas/ittt/td/apuntes/Presentacion%20tema%201.pdf [PDF ONLINE] 23/08/2021

[29] Youtube, “Understanding Barker Codes”, https://www.youtube.com/watch?v=ARspKn76TEU&ab_channel=RohdeSchwarz. [ONLINE] 23/08/2021

[30] Wikipedia, “Barker Code”, https://en.wikipedia.org/wiki/Barker_code. [ONLINE] 23/08/2021

[31] John Price, Terry Goble, “Telecommunications Engineer’s Reference Book”, Elsevier, 1993

[32] Wikipedia, “Bit error rate”, https://en.wikipedia.org/wiki/Bit_error_rate. [ONLINE] 23/08/2021

[33] Wikipedia, “Q-function”, https://en.wikipedia.org/wiki/Q-function. [ONLINE] 23/08/2021

[34] Youtube, “Adalm Pluto SDR Tutorial: Firmware Update”, https://youtu.be/NW7p7rib_rQ. [ONLINE] 16/07/2021

[35] MathWorks, “Communications Toolbox Support Package for Analog Devices ADALM-Pluto Radio” https://es.mathworks.com/help/supportpkg/plutoradio/index.html?s_tid=srchtitle_communications%20toolbox%20adalm%20pluto_4. [ONLINE] 14/08/2021

[36] MathWorks, “sdrrx”, https://www.mathworks.com/help/supportpkg/plutoradio/ref/sdrrx.html. [ONLINE] 25/08/2021

[37] MathWorks, “comm.IntegerToBit”, https://www.mathworks.com/help/comm/ref/comm.integertobit-system-object.html. [ONLINE] 25/08/2021

[38] MathWorks, “de2bi”, https://www.mathworks.com/help/comm/ref/de2bi.html. [ONLINE] 25/08/2021

[39] MathWorks, “commScrambler”, https://www.mathworks.com/help/comm/ref/comm.scrambler-system-object.html. [ONLINE] 25/08/2021

[40] Wikipedia, “Scrambler”, https://en.wikipedia.org/wiki/Scrambler. [ONLINE] 25/08/2021

[41] MathWorks, “comm.AGC”, https://es.mathworks.com/help/comm/ref/comm.agc-system-object.html. [ONLINE] 26/08/2021

[42] MathWorks, “comm.RaisedCosineReceiveFilter”, https://es.mathworks.com/help/comm/ref/comm.raisedcosinereceivefilter-system-object.html. [ONLINE] 26/08/2021

[43] MathWorks, “comm.CoarseFrequencyCompensator”, https://es.mathworks.com/help/comm/ref/comm.coarsefrequencycompensator-system-object.html. [ONLINE] 26/08/2021

[44] MathWorks, “comm.CarrierSynchronizer”, https://es.mathworks.com/help/comm/ref/comm.carriersynchronizer-system-object.html. [ONLINE] 26/08/2021

[45] MathWorks, “comm.SymbolSynchronizer”, https://es.mathworks.com/help/comm/ref/comm.symbolsynchronizer-system-object.html. [ONLINE] 26/08/2021

[46] MathWorks, “comm.PreambleDetector”, https://es.mathworks.com/help/comm/ref/comm.preambledetector-system-object.html. [ONLINE] 26/08/2021

Annexes

S'adjunta a continuació el codi que s'ha fet servir en el projecte per tal d'establir la comunicació del to i la comunicació amb un senyal QPSK

Annex A Codi per a generar la comunicació d'un to amb un sol Adalm Pluto

```
%% Comunicacio d'un to
clear all;


%% Declaració Objectes PlutoSDR
rx = sdrrx('Pluto');
tx = sdrtx('Pluto');

rx.GainSource = 'AGC Fast Attack';
rx.SamplesPerFrame = 1e6;
ch_size = 1e6;
rx.SamplesPerFrame = ch_size;


%% Generació del senyal
Fs = 30.72e6;       %freq de mostreig
Fc = 1e6;           %freq central del senyal
t = 1/Fs:1/Fs:ch_size/Fs;
amplitud = 1024;
sigR = sin(2*pi*Fc*t+0).*amplitud;
sigC = sin(2*pi*Fc*t+pi/2).*amplitud;
sig = complex(sigR,sigC); %senyal complex



%% Plot del senyal original
figure(1)
t = 1/Fs:1/Fs:ch_size/Fs;
plot(t,sigR,t,sigC);
xlabel('Temps');
ylabel('Amplitud');
xlim([t(end-300) t(end)])


%% Transmit and receive with PlutoSDR
tx.transmitRepeat(sig.');
frames = 10;
cap = zeros(ch_size*frames,1);
prev = 0;
for frame = 1:frames
    % Call radio
    o = rx();
    % Save data
    indx = (frame-1)*ch_size+1 : frame*ch_size;
    cap(indx) = o;
    % Info
    s = sprintf('Frame %d of %d',frame,frames);
    fprintf(repmat('\b',1,prev));fprintf(s);prev = length(s);
end
```

```
fprintf('\n');


%% Plot en la recepció
figure(2)
t = 1/Fs:1/Fs:frames*ch_size/Fs;
plot(t,real(cap),t,imag(cap));
xlabel('Temps');
ylabel('Amplitud');
xlim([t(end-300) t(end)])
```

Annex B Codi d'inicialització dels paràmetres de transmissió

```
function SimParams = plutoradioqpsktransmitter_init

%% Parametres generals de la comunicacio
SimParams.Rsym = 0.2e6;             % Velocitat de simbol Rs
SimParams.ModulationOrder = 4;      % Ordre de la modulacio (QPSK)
SimParams.Interpolation = 2;
SimParams.Decimation = 1;
SimParams.Tsym = 1/SimParams.Rsym;  % Temps de simbol Ts=1/Rs
SimParams.Fs   = SimParams.Rsym * SimParams.Interpolation; % Freq de
mostreig

%% Parametres i estructura de la trama
% Codi Barker per al sincronisme
SimParams.BarkerCode      = [+1 +1 +1 +1 +1 -1 -1 +1 +1 -1 +1 -1 +1];
SimParams.BarkerLength    = length(SimParams.BarkerCode);
% Es repeteix dos cops el codi com a capçalera
SimParams.HeaderLength    = SimParams.BarkerLength * 2;
% Missatge = 'Hello world 000\n'
SimParams.Message         = 'TFG Joaquim';
SimParams.MessageLength   = length(SimParams.Message) + 5;
% 5 = espai + 3 nº + salt de linia

% Nombre de missatges en cada trama
SimParams.NumberOfMessage = 100;
% Tamany = nº missatges * tamany missatge * 7 (nº de bits per char)
SimParams.PayloadLength   = SimParams.NumberOfMessage *
SimParams.MessageLength * 7;
% Tamany total de la trama (en simbols)
SimParams.FrameSize       = (SimParams.HeaderLength +
SimParams.PayloadLength) ...
    / log2(SimParams.ModulationOrder);
% Temps de trama = Tsimbol * Tamany de la trama (en simbols)
SimParams.FrameTime       = SimParams.Tsym*SimParams.FrameSize;


%% Parametres filtre i aleatoritzador
% factor de rolloff filtre root cosinus
SimParams.RolloffFactor     = 0.5;
% base del scrambler i polinomi
SimParams.ScramblerBase     = 2;
SimParams.ScramblerPolynomial           = [1 1 1 0 1];
SimParams.ScramblerInitialConditions    = [0 0 0 0];
SimParams.RaisedCosineFilterSpan = 10; %en simbols

%% Message generation
```

```
msgSet = zeros(100 * SimParams.MessageLength, 1);
for msgCnt = 0 : 99
    msgSet(msgCnt * SimParams.MessageLength + (1 :
SimParams.MessageLength)) = ...
        sprintf('%s %03d\n', SimParams.Message, msgCnt);
end
bits = de2bi(msgSet, 7, 'left-msb')';
SimParams.MessageBits = bits(:);

% parametres pel sdrtx
SimParams.PlutoCenterFrequency      = 915e6;
SimParams.PlutoGain                 = 0;
SimParams.PlutoFrontEndSampleRate   = SimParams.Fs;
SimParams.PlutoFrameLength          = SimParams.Interpolation *
SimParams.FrameSize;

% temps d'execucio
SimParams.FrameTime =
SimParams.PlutoFrameLength/SimParams.PlutoFrontEndSampleRate;
SimParams.StopTime  = 1000;
```

Annex C Codi d'inicialització dels paràmetres de transmissió

```
function runPlutoradioQPSKTransmitter(prmQPSKTransmitter)

persistent hTx radio
if isempty(hTx)
    % Configurem els parametres que hem guardat a prmQPSKTransmitter a
cada objecte
    hTx = QPSKTransmitter(...
        'UpsamplingFactor',             prmQPSKTransmitter.Interpolation,
...
        'RolloffFactor',                prmQPSKTransmitter.RolloffFactor,
...
        'RaisedCosineFilterSpan',
prmQPSKTransmitter.RaisedCosineFilterSpan, ...
        'MessageBits',                  prmQPSKTransmitter.MessageBits,
...
        'MessageLength',                prmQPSKTransmitter.MessageLength,
...
        'NumberOfMessage',
prmQPSKTransmitter.NumberOfMessage, ...
        'ScramblerBase',                prmQPSKTransmitter.ScramblerBase,
...
        'ScramblerPolynomial',
prmQPSKTransmitter.ScramblerPolynomial, ...
        'ScramblerInitialConditions',
prmQPSKTransmitter.ScramblerInitialConditions);

    % Creem l'Objecte de transmissio lligat al hardware de la radio per a
enviar el senyal
    radio = sdrtx('Pluto');
    radio.RadioID               = prmQPSKTransmitter.Address;
    radio.CenterFrequency       =
prmQPSKTransmitter.PlutoCenterFrequency;
    radio.BasebandSampleRate    =
prmQPSKTransmitter.PlutoFrontEndSampleRate;
```

```
    radio.SamplesPerFrame       = prmQPSKTransmitter.PlutoFrameLength;
    radio.Gain                  = prmQPSKTransmitter.PlutoGain;
end

currentTime = 0;
disp('Inici de la Transmissió')

    % Temps d'execució de la transmissió
while currentTime < prmQPSKTransmitter.StopTime

    data = step(hTx);

    % Transmetem les dades
    step(radio, data);

    % Actualitzem el temps que ha passat
    currentTime = currentTime+prmQPSKTransmitter.FrameTime;
end

if currentTime ~= 0
    disp('Transmissió finalitzada')
end

release(hTx);
release(radio);

end
```

Annex D Codi per a aplicar els objectes de modulació, aleatorització i filtratge del senyal

```
classdef (StrictDefaults)QPSKTransmitter < matlab.System

    properties (Nontunable)
        UpsamplingFactor = 2;
        ScramblerBase = 2;
        ScramblerPolynomial = [1 1 1 0 1];
        ScramblerInitialConditions = [0 0 0 0];
        RolloffFactor = 0.5
        RaisedCosineFilterSpan = 10
        NumberOfMessage = 10
        MessageLength = 16
        MessageBits = []
    end

    properties (Access=private)
        pBitGenerator
        pQPSKModulator
        pTransmitterFilter
        pMessage = 'TFG Joaquim';
        pHeader = [+1 +1 +1 +1 +1 -1 -1 +1 +1 -1 +1 -1 +1];
    end

    methods
        function obj = QPSKTransmitter(varargin)
            setProperties(obj,nargin,varargin{:});
        end
    end
```

```
    methods (Access=protected)
        function setupImpl(obj)
            %fem servir la classe de matlab QPSKBitsGenerator per a
convertir el missatge a bits i aplicar aleatorització
            obj.pBitGenerator = QPSKBitsGenerator( ...
                'NumberOfMessage',              obj.NumberOfMessage, ...
                'MessageLength',                obj.MessageLength, ...
                'MessageBits',                  obj.MessageBits, ...
                'ScramblerBase',                obj.ScramblerBase, ...
                'ScramblerPolynomial',          obj.ScramblerPolynomial,
...
                'ScramblerInitialConditions',
obj.ScramblerInitialConditions);
            % Objecte per a la modulacio QPSK del senyal a partir dels
bits i amb un offset de fase de pi/4
            obj.pQPSKModulator  = comm.QPSKModulator( ...
                'BitInput',                     true, ...
                'PhaseOffset',                  pi/4, ...
                'OutputDataType',               'double');
            obj.pTransmitterFilter = comm.RaisedCosineTransmitFilter( ...
                'RolloffFactor',                obj.RolloffFactor, ...
                'FilterSpanInSymbols',
obj.RaisedCosineFilterSpan, ...
                'OutputSamplesPerSymbol',       obj.UpsamplingFactor);
        end

        function transmittedSignal = stepImpl(obj)
            [transmittedBin, ~] = obj.pBitGenerator();                  %
apliquem la modulacio a la informacio en forma de bits
            modulatedData = obj.pQPSKModulator(transmittedBin);
            transmittedSignal = obj.pTransmitterFilter(modulatedData); %
filtre conformador de tipus root-cosinus
        end

        function resetImpl(obj)
            reset(obj.pBitGenerator);
            reset(obj.pQPSKModulator );
            reset(obj.pTransmitterFilter);
        end

        function releaseImpl(obj)
            release(obj.pBitGenerator);
            release(obj.pQPSKModulator );
            release(obj.pTransmitterFilter);
        end

        function N = getNumInputsImpl(~)
            N = 0;
        end
    end
end
```

Annex E Inicialització dels paràmetres de la recepció

```
function SimParams = plutoradioqpskreceiver_init

%% Parametres generals de la comunicacio
SimParams.Rsym = 0.2e6;             % Velocitat de simbol Rs
SimParams.ModulationOrder = 4;      % Ordre de la modulacio (QPSK)
SimParams.Interpolation = 2;
SimParams.Decimation = 1;
SimParams.Tsym = 1/SimParams.Rsym;  % Temps de simbol Ts=1/Rs
SimParams.Fs   = SimParams.Rsym * SimParams.Interpolation; % Freq de
mostreig

%% Parametres i estructura de la trama
% Codi Barker per al sincronisme
SimParams.BarkerCode      = [+1 +1 +1 +1 +1 -1 -1 +1 +1 -1 +1 -1 +1];
SimParams.BarkerLength    = length(SimParams.BarkerCode);
% Es repeteix dos cops el codi com a capçalera
SimParams.HeaderLength    = SimParams.BarkerLength * 2;
% Missatge = 'Hello world 000\n'
SimParams.Message         = 'TFG Joaquim';
SimParams.MessageLength   = length(SimParams.Message) + 5;
% 5 = espai + 3 n° + salt de linia

% Nombre de missatges en cada trama
SimParams.NumberOfMessage = 100;
% Tamany = n° missatges * tamany missatge * 7 (n° de bits per char)
SimParams.PayloadLength   = SimParams.NumberOfMessage *
SimParams.MessageLength * 7;
% Tamany total de la trama (en simbols)
SimParams.FrameSize       = (SimParams.HeaderLength +
SimParams.PayloadLength) ...
    / log2(SimParams.ModulationOrder);
% Temps de trama = Tsimbol * Tamany de la trama (en simbols)
SimParams.FrameTime       = SimParams.Tsym*SimParams.FrameSize;

%% Rx parameters
% Filtre i descrambler
SimParams.RolloffFactor     = 0.5;
SimParams.ScramblerBase     = 2;
SimParams.ScramblerPolynomial           = [1 1 1 0 1];
SimParams.ScramblerInitialConditions    = [0 0 0 0];
SimParams.RaisedCosineFilterSpan = 10;                  % SPAN en simbols

% Paràmetres pel Control de Guany Automatic (AGC)
SimParams.DesiredPower                  = 2;            % Potencia
desitjada(W)
SimParams.AveragingLength               = 50;
SimParams.MaxPowerGain                  = 60;
SimParams.MaximumFrequencyOffset        = 6e3;
% Look into model for details for details of PLL parameter choice.
% Refer equation 7.30 of "Digital Communications - A Discrete-Time
Approach" by Michael Rice.
K = 1;
A = 1/sqrt(2);
SimParams.PhaseRecoveryLoopBandwidth    = 0.01;
SimParams.PhaseRecoveryDampingFactor    = 1;
SimParams.TimingRecoveryLoopBandwidth   = 0.01;
SimParams.TimingRecoveryDampingFactor   = 1;
% K_p for Timing Recovery PLL, determined by 2KA^2*2.7 (for binary PAM),
% QPSK could be treated as two individual binary PAM,
```

```
% 2.7 is for raised cosine filter with roll-off factor 0.5
SimParams.TimingErrorDetectorGain       = 2.7*2*K*A^2+2.7*2*K*A^2;
SimParams.PreambleDetectorThreshold     = 0.8;

%% Message generation and BER calculation parameters
msgSet = zeros(100 * SimParams.MessageLength, 1);
for msgCnt = 0 : 99
    msgSet(msgCnt * SimParams.MessageLength + (1 :
SimParams.MessageLength)) = ...
        sprintf('%s %03d\n', SimParams.Message, msgCnt);
end
bits = de2bi(msgSet, 7, 'left-msb')';
SimParams.MessageBits = bits(:);

% For BER calculation masks
SimParams.BerMask = zeros(SimParams.NumberOfMessage *
length(SimParams.Message) * 7, 1);
for i = 1 : SimParams.NumberOfMessage
    SimParams.BerMask( (i-1) * length(SimParams.Message) * 7 + ( 1:
length(SimParams.Message) * 7) ) = ...
        (i-1) * SimParams.MessageLength * 7 + (1:
length(SimParams.Message) * 7);
end
% Paràmetres de l'objecte de recpecio del pluto sdrrx
SimParams.PlutoCenterFrequency      = 915e6;
SimParams.PlutoGain                 = 30;
SimParams.PlutoFrontEndSampleRate   = SimParams.Fs;
SimParams.PlutoFrameLength          = SimParams.Interpolation *
SimParams.FrameSize;

% temps d'execucio de la recepcio
SimParams.PlutoFrameTime = SimParams.PlutoFrameLength /
SimParams.PlutoFrontEndSampleRate;
SimParams.StopTime = 10;
```

Annex F Recepció del senyal i decodificació per a obtenir el missatge original

```
function BER = runPlutoradioQPSKReceiver(prmQPSKReceiver, printData)

% configurem els parametres inicialitzats al receptor
persistent rx radio;
if isempty(rx)
    rx  = QPSKReceiver(...
        'ModulationOrder',
prmQPSKReceiver.ModulationOrder, ...
        'SampleRate',                           prmQPSKReceiver.Fs, ...
        'DecimationFactor',
prmQPSKReceiver.Decimation, ...
        'FrameSize',
prmQPSKReceiver.FrameSize, ...
        'HeaderLength',
prmQPSKReceiver.HeaderLength, ...
        'NumberOfMessage',
prmQPSKReceiver.NumberOfMessage, ...
        'PayloadLength',
prmQPSKReceiver.PayloadLength, ...
```

```
        'DesiredPower',
prmQPSKReceiver.DesiredPower, ...
        'AveragingLength',
prmQPSKReceiver.AveragingLength, ...
        'MaxPowerGain',
prmQPSKReceiver.MaxPowerGain, ...
        'RolloffFactor',
prmQPSKReceiver.RolloffFactor, ...
        'RaisedCosineFilterSpan',
prmQPSKReceiver.RaisedCosineFilterSpan, ...
        'InputSamplesPerSymbol',
prmQPSKReceiver.Interpolation, ...
        'MaximumFrequencyOffset',
prmQPSKReceiver.MaximumFrequencyOffset, ...
        'PostFilterOversampling',
prmQPSKReceiver.Interpolation/prmQPSKReceiver.Decimation, ...
        'PhaseRecoveryLoopBandwidth',
prmQPSKReceiver.PhaseRecoveryLoopBandwidth, ...
        'PhaseRecoveryDampingFactor',
prmQPSKReceiver.PhaseRecoveryDampingFactor, ...
        'TimingRecoveryDampingFactor',
prmQPSKReceiver.TimingRecoveryDampingFactor, ...
        'TimingRecoveryLoopBandwidth',
prmQPSKReceiver.TimingRecoveryLoopBandwidth, ...
        'TimingErrorDetectorGain',
prmQPSKReceiver.TimingErrorDetectorGain, ...
        'PreambleDetectorThreshold',
prmQPSKReceiver.PreambleDetectorThreshold, ...
        'DescramblerBase',
prmQPSKReceiver.ScramblerBase, ...
        'DescramblerPolynomial',
prmQPSKReceiver.ScramblerPolynomial, ...
        'DescramblerInitialConditions',
prmQPSKReceiver.ScramblerInitialConditions,...
        'BerMask',                              prmQPSKReceiver.BerMask,
...
        'PrintOption',                          printData);

    % creem l'objecte de recepcio del pluto lligat al hardware fins la
crida del release()
    radio = sdrrx('Pluto');
    radio.RadioID               = prmQPSKReceiver.Address;
    radio.CenterFrequency       = prmQPSKReceiver.PlutoCenterFrequency;
    radio.BasebandSampleRate    =
prmQPSKReceiver.PlutoFrontEndSampleRate;
    radio.SamplesPerFrame       = prmQPSKReceiver.PlutoFrameLength;
    radio.GainSource            = 'Manual';
    radio.Gain                  = prmQPSKReceiver.PlutoGain;
    radio.OutputDataType        = 'double';
end

% escoltem els senyals
currentTime = 0;
BER = [];
rcvdSignal = complex(zeros(prmQPSKReceiver.PlutoFrameLength,1));

while currentTime <  prmQPSKReceiver.StopTime
    % rebem el senyal per radio
    rcvdSignal = radio();

    % decodifiquem el missatge per a mostrarlo per pantalla
```

```
    [~, ~, ~, BER] = rx(rcvdSignal);

    currentTime=currentTime+(radio.SamplesPerFrame /
radio.BasebandSampleRate);
end

release(rx);
release(radio);
```

Annex G Desmodulació

```
classdef (StrictDefaults)QPSKReceiver < matlab.System

    properties (Nontunable)
        ModulationOrder = 4;
        SampleRate = 200000;
        DecimationFactor = 1;
        FrameSize = 1133;
        HeaderLength = 13;
        NumberOfMessage = 20;
        PayloadLength = 2240;
        DesiredPower = 2
        AveragingLength = 50
        MaxPowerGain = 20
        RolloffFactor = 0.5
        RaisedCosineFilterSpan = 10
        InputSamplesPerSymbol = 2
        MaximumFrequencyOffset = 6e3
        PostFilterOversampling = 2;
        PhaseRecoveryLoopBandwidth = 0.01;
        PhaseRecoveryDampingFactor = 1;
        TimingRecoveryDampingFactor = 1;
        TimingRecoveryLoopBandwidth = 0.01;
        TimingErrorDetectorGain = 5.4;
        PreambleDetectorThreshold = 8;
        DescramblerBase = 2;
        DescramblerPolynomial = [1 1 1 0 1];
        DescramblerInitialConditions = [0 0 0 0];
        BerMask = [];
        PrintOption = false;
    end

    properties (Access = private)
        pAGC
        pRxFilter
        pCoarseFreqEstimator
        pCoarseFreqCompensator
        pFineFreqCompensator
        pTimingRec
        pPrbDet
        pFrameSync
        pDataDecod
        pMeanFreqOff
        pCnt
    end

    properties (Access = private, Constant)
        pUpdatePeriod = 4
```

```
        pModulatedHeader = sqrt(2)/2 * (-1-1i) * [+1; +1; +1; +1; +1; -1;
-1; +1; +1; -1; +1; -1; +1];
        pMessage = 'Hello world';
        pMessageLength = 16;
    end

    methods
        function obj = QPSKReceiver(varargin)
            setProperties(obj,nargin,varargin{:});
        end
    end

    methods (Access = protected)
        function setupImpl(obj, ~)
            %objecte de control de guany
            obj.pAGC = comm.AGC( ...
                'DesiredOutputPower',       obj.DesiredPower, ...
                'AveragingLength',          obj.AveragingLength, ...
                'MaxPowerGain',             obj.MaxPowerGain);
            %filtre root cosinus a la recpecio
            obj.pRxFilter = comm.RaisedCosineReceiveFilter( ...
                'RolloffFactor',            obj.RolloffFactor, ...
                'FilterSpanInSymbols',      obj.RaisedCosineFilterSpan,
...
                'InputSamplesPerSymbol',    obj.InputSamplesPerSymbol,
...
                'DecimationFactor',         obj.DecimationFactor);
            %compensadors de freq i fase
            obj.pCoarseFreqEstimator = comm.CoarseFrequencyCompensator(
...
                'Modulation',               'QPSK', ...
                'Algorithm',                'Correlation-based', ...
                'MaximumFrequencyOffset',   obj.MaximumFrequencyOffset,
....
                'SampleRate',
obj.SampleRate/obj.DecimationFactor);

            obj.pCoarseFreqCompensator = comm.PhaseFrequencyOffset( ...
                'PhaseOffset',              0, ...
                'FrequencyOffsetSource',    'Input port', ...
                'SampleRate',
obj.SampleRate/obj.DecimationFactor);

            obj.pMeanFreqOff = 0;

            obj.pCnt = 0;
            % objecte de sincronitzacio de portadora
            obj.pFineFreqCompensator = comm.CarrierSynchronizer( ...
                'Modulation',               'QPSK', ...
                'ModulationPhaseOffset',    'Auto', ...
                'SamplesPerSymbol',         obj.PostFilterOversampling,
...
                'DampingFactor',
obj.PhaseRecoveryDampingFactor, ...
                'NormalizedLoopBandwidth',
obj.PhaseRecoveryLoopBandwidth);
            % sincronització de simbols
            obj.pTimingRec = comm.SymbolSynchronizer( ...
                'TimingErrorDetector',      'Gardner (non-data-aided)',
...
```

```
                'SamplesPerSymbol',         obj.PostFilterOversampling,
...
                'DampingFactor',
obj.TimingRecoveryDampingFactor, ...
                'NormalizedLoopBandwidth',
obj.TimingRecoveryLoopBandwidth, ...
                'DetectorGain',             obj.TimingErrorDetectorGain);

            obj.pPrbDet = comm.PreambleDetector(obj.pModulatedHeader, ...
                'Input',                    'Symbol', ...
                'Threshold',
obj.PreambleDetectorThreshold);

            obj.pFrameSync = comm.internal.examples.FrameSynchronizer(
...
                'OutputFrameLength',        obj.FrameSize, ...
                'PreambleLength',           obj.HeaderLength / 2);
            % desmodulacio qpsk i desaleatoritzador de bits
            obj.pDataDecod = QPSKDataDecoder( ...
                'ModulationOrder',          obj.ModulationOrder, ...
                'HeaderLength',             obj.HeaderLength, ...
                'NumberOfMessage',          obj.NumberOfMessage, ...
                'PayloadLength',            obj.PayloadLength, ...
                'DescramblerBase',          obj.DescramblerBase, ...
                'DescramblerPolynomial',    obj.DescramblerPolynomial,
...
                'DescramblerInitialConditions',
obj.DescramblerInitialConditions, ...
                'BerMask',                  obj.BerMask, ...
                'PrintOption',              obj.PrintOption);
        end

        function [RCRxSignal, timingRecSignal, fineCompSignal, BER] =
stepImpl(obj, bufferSignal)

            AGCSignal = obj.pAGC(bufferSignal);
% AGC control
            RCRxSignal = obj.pRxFilter(AGCSignal);
% Pass the signal through

% Square-Root Raised Cosine Received Filter
            [~, freqOffsetEst] = obj.pCoarseFreqEstimator(RCRxSignal);
% Coarse frequency offset estimation
            % average coarse frequency offset estimate, so that carrier
            % sync is able to lock/converge
            freqOffsetEst = (freqOffsetEst + obj.pCnt *
obj.pMeanFreqOff)/(obj.pCnt+1);
            obj.pCnt = obj.pCnt + 1;            % update state
            obj.pMeanFreqOff = freqOffsetEst;

            coarseCompSignal = obj.pCoarseFreqCompensator(RCRxSignal,...
                -freqOffsetEst);
% Coarse frequency compensation
            timingRecSignal = obj.pTimingRec(coarseCompSignal);
% Symbol timing recovery

            fineCompSignal = obj.pFineFreqCompensator(timingRecSignal);
% Fine frequency compensation
```

```
            [prbIdx, dtMt] = obj.pPrbDet(fineCompSignal);
% Detect frame header

            [symFrame, isFrameValid] = obj.pFrameSync(fineCompSignal, ...
                prbIdx, dtMt);
% Frame synchronization

            BER = obj.pDataDecod(symFrame, isFrameValid);

        end

        function resetImpl(obj)
            reset(obj.pAGC);
            reset(obj.pRxFilter);
            reset(obj.pCoarseFreqEstimator);
            reset(obj.pCoarseFreqCompensator);
            reset(obj.pFineFreqCompensator);
            reset(obj.pTimingRec);
            reset(obj.pPrbDet);
            reset(obj.pFrameSync);
            reset(obj.pDataDecod);
            obj.pMeanFreqOff = 0;
            obj.pCnt = 0;
        end

        function releaseImpl(obj)
            release(obj.pAGC);
            release(obj.pRxFilter);
            release(obj.pCoarseFreqEstimator);
            release(obj.pCoarseFreqCompensator);
            release(obj.pFineFreqCompensator);
            release(obj.pTimingRec);
            release(obj.pPrbDet);
            release(obj.pFrameSync);
            release(obj.pDataDecod);
        end

        function N = getNumOutputsImpl(~)
            N = 4;
        end
    end
end
```