Disseny d'una API per SDR

Jordi Castillo Valles

Resum– Aquest treball presenta una proposta d'una interfície de programació d'aplicacions (API) per treballar amb ràdios definides per software (SDR). S'han considerat les diverses SDRs del mercat per tal de fer una API generalista, fent que l'usuari final tingui un nivell d'abstracció superior. Aquest projecte no és només una proposta teòrica sinó que s'ha implementat aquesta API per treballar amb la SDR AD9361. Les proves s'han pogut fer gràcies a l'institut de ciències de l'espai on tenen una SDR basada en aquest xip. L'API ha sigut completament implementada i s'ha comprovat de forma satisfactòria que totes les funcionalitats funcionen correctament. Aquesta API ha sigut implementada en C ja que les SDRs solen utilitzar-se en projectes de teledetecció en forma de sistemes encastats, i considerem que és el llenguatge més adient per aquest tipus de projectes.

Paraules clau– API, Handler, SDR, SDR virtual, BSP

Abstract– This paper presents a proposal of an application programming interface (API) intended to aim the development of systems that work with software defined radios (SDR). The various SDRs that are available in the market have been considered in order to design an universal API, thanks to this the user of the API will have a higher level of abstraction. This project is not only a theoretical work but the API has been implemented to work with the AD9361 SDR. The tests have been made possible by the Institut de Ciències de l'Espai, where all teh funcionality intended has been checked. This API has been implemented in C because the vast majority of SDRs are used in remote sensing projects as embedded systems. We have considered that c is the most adequate language for this type of appliacations.

Keywords– API, Handler, SDR, virtual SDR, BSP

1 Introducció - Context del treball

En el món de les telecomunicacions fa bastant temps, si es volia transmetre/rebre en una freqüència determinada s'utilitzava un circuit específicament dissenyat per a dur aquesta tasca. Més endavant van aparèixer les arquitectures dels transmissors/receptors superheterodins. Aquest nou plantejament del sistema de comunicació permetia enviar/rebre senyals a diferents freqüències necessitant modificar només un paràmetre del sistema. Aquest fet va ser un gran avanç, ja que permetia fer sistemes complexos on es treballava a diferents freqüències sense necessitat de fer circuits grans, l'arquitectura superheterodina ens aporta una gran escalabilitat. Aquesta arquitectura es va combinar amb el concepte de microcontrolador, de forma que es crea un sistema on mitjançant el software podem controlar alguns paràmetres de la nostra ràdio.

Una SDR és això, un xip amb el que ens podem comunicar que conté una ràdio amb una arquitectura que ens permet modificar molts dels seus paràmetres, com el guany, la freqüència de mostreig, l'amplada de banda, la freqüència de treball o l'atenuació. Aquests dispositius són ideals pel desenvolupament de prototips ja que la seva gran flexibilitat els permeten adaptar-se a qualsevol requeriment de l'aplicació.

El problema és que els projectes solen tenir una duració llarga, de forma que els dissenyadors de SDRs tenen temps de treure nous models. Això dificulta el desenvolupament d'una aplicació que es pugui emprar en diferents models de SDRs. Per tant, considerem interessant desenvolupar una API que elimini aquesta problemàtica a l'usuari de la SDR. Aquest és l'objectiu principal del treball a desenvolupar.

Aquest treball es desenvolupa conjuntament amb el Institut de Ciències de l'Espai i l'Institut d'Estudis Espacials de Catalunya, qui ens proporcionarà un sistema encastat amb una SDR basada en el xip AD9361 per a poder realitzar el treball.

• E-mail de contacte: jordi.castillo@autonoma.cat
• Menció realitzada: Enginyeria de Computadors
• Treball tutoritzat per: Màrius Montón Macian (departament de microelectrònica i sistemes electrònics)
• Curs 2022/23

2 OBJECTIUS

El nostre objectiu és desenvolupar una API que no només emmascari la dificultat de controlar una SDR a baix nivell, sinó que també elimini el problema d'haver de treballar tenint en consideració el hardware sobre el qual s'executarà aquest treball, de forma que es pugui canviar el hardware sense haver de modificar el software. Com és una API que ajudarà al desenvolupament àgil de sistemes amb SDRs, també s'inclouran funcions de test per permetre l'experimentació ràpida.

Com que aquest treball està pensat per a ser utilitzat per diversos projectes, sempre s'haurà de tenir en compte l'opinió dels usuaris finals a l'hora de dissenyar els diferents aspectes del sistema. Per tant, el feedback serà imprescindible en aquest treball i sempre haurà de prevaldre davant altres opinions. També cal tenir en compte en quins llenguatges desenvoluparan els seus projectes, en el món dels sistemes encastats C i C++ són els reis, motiu pel qual desenvoluparem l'API en el llenguatge C, ja que el codi C és compatible amb C++, però codi escrit en C++ no té per què ser compatible amb C.

Finalment, cal discutir que l'objectiu més important és tenir una API funcional al final del desenvolupament del projecte, ja que tot el que hem discutit anteriorment està molt bé, però fa falta tenir alguna cosa que entregar.

3 METODOLOGIA I PLANIFICACIÓ

Per poder plantejar el projecte el primer pas és identificar quina metodologia de treball utilitzarem [1]. Hi ha vàries alternatives, totes amb els seus avantatges e inconvenients, en el nostre cas ens hem decantat per una metodologia de tipus cascada [2]. Aquesta metodologia divideix el treball en tasques lineals, és a dir, fins que no s'acaba la primera tasca no es pot començar amb la segona. Això té sentit amb el que volíem plantejar, però té un gran problema, aquest mètode assoleix que a l'inici del projecte sabem exactament què volem fer i quan trigarem a fer-ho. Això és un gran problema ja que no podem saber quan trigarem a desenvolupar un codi que tampoc sabem exactament que ha de fer. La solució va ser acceptar aquest gran defecte i fer una tasca molt gran (de fet massa gran com per a considerar el diagrama de Gantt correcte) on aplicaríem una metodologia incremental. Això implica que el projecte té una estructura rígida que garantís que al final del projecte s'hagués fet el suficient treball com per presentar-lo però tindríem la flexibilitat de no haver de planificar unes tasques que no sabem si existeixen o no.

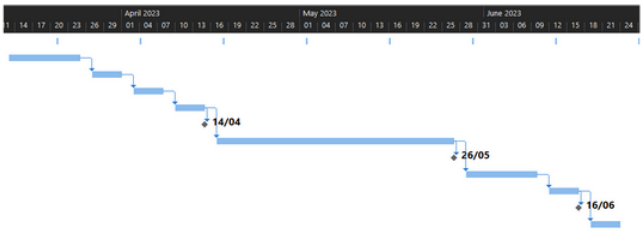

Fig. 1: Diagrama de Gantt del projecte

La planificació s'ha fet amb el programari Microsoft Project i la podem observar en la imatge 1. Com podem observar, el treball es divideix en 3 grans blocs:

- El primer gran bloc es basa en fer una proposta de l'API a desenvolupar. És imprescindible que es defineixi de la millor forma possible què es desenvoluparà, ja que és relativament fàcil modificar la funcionalitat de l'API en la fase de planificació però és extremadament complexe fer-ho en la fase de desenvolupament. El primer pas serà fer un estudi de mercat per identificar les diferents SDRs i les seves característiques. El següent pas serà plantejar un primer document on s'exposarà l'arquitectura i la funcionalitat de l'API. Finalment, aquest document serà posat en comú amb els potencials usuaris de l'API per tal d'obtenir feedback i fer les modificacions adients. D'aquesta forma obtindrem una proposta finals sobre el que s'ha de programar.

- El procediment serà un desenvolupament àgil del software, on es començarà per fer una versió molt simple de l'API que sigui funcional i, a mesura que es validi el correcte funcionament de les funcions programades, s'aniran desenvolupant noves funcions. Així, la primera versió de l'API només es connectarà amb la SDR, després podrem modificar alguns paràmetres, després transmetre...

- Finalment, amb l'API desenvolupada haurem de crear documentació del codi i del projecte. Aquesta fase pot semblar trivial, però una bona documentació en una API és imprescindible, ja que si hi han moltes funcions pot ser complex per a un nou usuari aprendre a utilitzar-la. Per a garantir aquest últim punt tornarem a demanar feedback dels usuaris.

4 PROPOSTA DE L'API

En aquesta secció discutirem el procediment que s'ha seguit per a fer la proposta de l'API i presentarem les funcions més importants a conèixer.

En aquesta fase del treball volíem acabar amb un document on es recollissin el conjunt de funcions a desenvolupar en la propera fase del projecte. Per a fer-ho començarem fent un estudi de mercat per entendre quines funcions són necessàries incloure en l'API, encara que no siguin possibles implementar en el nostre cas particular, després discutirem la primera versió de L'API i, finalment, explicarem les funcions que els possibles usuaris de l'API ens indiquen que faltaria afegir.

4.1 Estudi de mercat

Quan vam fer l'estudi de mercat vam començar a veure moltes SDRs de diferents fabricants, però vam decidir englobarles en 2 categories.

El primer tipus que vam trobar són les SDRs que són un dongle USB i solen tenir una configuració bastant limitada dels seus paràmetres. Aquestes SDRs s'han de configurar mitjançant la modificació de registres interns, que varien en funció del dispositiu. Aquest és un cas on el nostre projecte resulta excel·lent, ja que permetria desenvolupar aplicacions sense haver de hardcodejar quins registres es modifiquen i com es modifiquen. Aquests dongles solen estar basats en el xip RTL2832U i el R820T2. [3]

El segon tipus, són SDRs que es combinen amb FPGAs (mitjançant el bus PCiEm mPCiE o similars), de forma que

el processador que vol controlar la SDRr s'ha de comunicar mitjançant una connexió IP. En aquest cas la configuració de la SDR es realitza mitjançant una comunicació TCP que envia ordres a la FPGA, i és en aquesta FPGA on hi ha un conjunt de drivers que transformen aquests missatges TCP en ordres a la SDR en si. Un altre cop, el nostre projecte resulta excel·lent a l'hora de facilitar el desenvolupament d'aplicacions, ja que cada SDR necessita la seva pròpia biblioteca per enviar aquests missatges TCP/IP. Alguns exemples d'aquest tipus de SDR són els xips d'Analog Devices [4], que es controlen amb la biblioteca IIO, la BladeRF [5] que és una SDR que es controla amb la biblioteca libbladeRF o la LimeSDR [6] amb la biblioteca SoapySdr.

Com podem observar sembla que hi ha més varietat de SDR en les que no s'utilitza connexió USB. Això és degut al fet que els dongles USB són considerades SDRs per amateurs i radioaficionats i no per a un ús professional. Això implica que les SDR professionals són les més cares i no són assequibles pel públic en general. Tot i això, hi ha hagut iniciatives per intentar que els particulars puguin aconseguir SDR de qualitat professional a un cost més reduït. XTRX [7] és una SDR que va sorgir com un projecte de crowdfunding que es controla amb la biblioteca libxtrx. IotSDR [8] és un altre SDR finançat mitjançant un crowdfunding, però no té una biblioteca genèrica compatible amb molts llenguatges de programació (no té suport de C, llenguatge que farem servir per desenvolupar la nostra API).

Aquesta gran varietat de SDRs fan que les seves funcionalitats siguin molt diferents entre elles. Quan es treballi amb l'API caldrà tenir en compte que algunes funcions podran no tenir cap funcionalitat o podran tenir una funcionalitat reduïda en funció de la plataforma física en si. Per exemple, podríem tenir un dongle USB que només rebi per un únic port, per un únic canal a una amplada de banda determinada. Això faria que l'usuari no pugui determinar cap d'aquests paràmetres, ja que estarien fixats per la SDR, però sí que podria fer un port d'un programa desenvolupat per un altre SDR on aquests paràmetres fossin necessaris ser configurats sense haver de fer modificacions addicionals.

Aquesta idea d'una API genèrica no és nova, el problema és que normalment no són tan genèriques. Per exemple, Osmocom [9] té una API per certs dispositius RTL. El problema d'aquest tipus d'APIs és que són APIs fetes per radioaficionats i, per tant, es centren en les SDR del tipus USB dongle, que s'assemblen molt entre elles. La nostra API vol ser el màxim genèric possible per a que al programador li sigui indiferent treballar amb una SDR RTL o amb una SDR d'Analog Devices, aquest és el punt on brilla aquest projecte, ja que no hi ha res similar en el mercat.

Durant aquesta anàlisi de mercat hem comprovat que valoren molt el fet que les SDR tinguin una biblioteca per Gnu Radio. GnuRadio és un software de programació gràfica per a entorns SDR, és molt similar al famós Node-Red. Seria ideal poder afegir compatibilitat amb GnuRadio a la nostra API, com de moment no sabem si tindrem temps per a fer-ho ho descartem.

4.2 Proposta inicial

Aquesta fase consisteix a fer una proposta de l'estructura i les funcions de l'API. La primera versió consistia en un handler que representa l'API i diverses funcions per configurar-la. El problema d'aquesta implementació és que va en contra de la filosofia de la mateixa API. El nostre objectiu és crear una SDR virtual universal que és la que modifica i controla l'usuari. Si carreguem la configuració sempre directament a la SDR real, és com si no existís aquesta SDR virtual. Per tant, una modificació que vam fer des de la primera versió respecte a l'última va ser afegir un altre handler que representi la configuració de la SDR. D'aquesta forma, tindrem dos Handlers que representaran la SDR virtual i la configuració desitjada.

Respecte a les funcions, trobem 5 grans blocs en els quals podem agrupar totes les funcions: (i) Funcions de configuració: Són els setters i els getters que ens permeten modificar els següents paràmetres: ubicació SDR real, port a configurar, canal a utilitzar, freqüència de mostreig, freqüència del oscilador local, amplada de banda, guany de recepció i atenuació del transmissor. (ii) Funcions de control: Permeten controlar la nostra SDR virtual i la communicació amb la SDR real. (iii) Funcions de transmissió/recepció: Permeten a l'usuari transmetre/rebre senyals en la SDR real mitjançant la SDR Virtual. Aquests senyals han d'estar descomposts en I i Q i tenen valors de tipus float (IEEE 754) en el rang de 0 a 1. (iv) Funcions de test: Permeten a l'usuari fer alguna prova per a comprovar que la SDR funciona correctament. De moment, en aquesta proposta l'usuari pot enviar un to, calcular el punt de compressió d'1dB del receptor mitjançant un transmissor de la pròpia SDR i trobar el punt d'intercepció de tercer ordre del receptor mitjançant un transmissor de la pròpia SDR. (v) Funcions de debug: Permeten mostrar en el terminal la configuració feta i que vol dir el codi d'error que hagi retornat una funció.

En aquesta tasca hem hagut de fer diverses versions de les propostes, ja que hi havien alguns errors de plantejament i faltaven funcions per garantir una completesa de l'API. També hem fet dues minitasques que han col·laborat en l'endarreriment de la tasca. La primera és que hem fet dos petits exemples de com s'hauria d'utilitzar l'API. D'aquesta forma l'usuari final pot tenir un petit codi de guia per saber com s'han d'usar les funcions que hi ha. La segona mini tasca va ser la creació de la documentació de l'API mitjançant Doxygen. Doxygen és una eina de codi obert que permet documentar el codi mentre s'està escrivint mitjançant l'ús de comentaris especials. D'aquesta forma quan demanem feedback dels usuaris finals ho tindran més fàcil per entendre la filosofia de l'API i ens podran donar un feedback més valuós.

A continuació podem veure com s'organitza la nostra API en la figura 2. Com podem observar la idea és que desde el programa que genera l'usuari ell interactua principalment amb el handler de la configuració. A continuació es passa aquest handler a la SDR virtual, en aquest moment el programa de l'usuari pot interactura amb la SDR virtual per encendre-la o apagar-la, però res més. Un cop s'activa aquesta SDR virtual, la nostra llibreria pren el control del programa e interactua amb la llibreria IIO creant un IIO context. Aquest context permet al nostre software comunicar-se amb la SDR. A continuació, mitjançant el IIO context podem modificar certs camps de dades que es troben en el kernel del host de la SDR. Finalment, és aquest kernel qui es comunica amb la SDR i fa les modificacions necessàries.

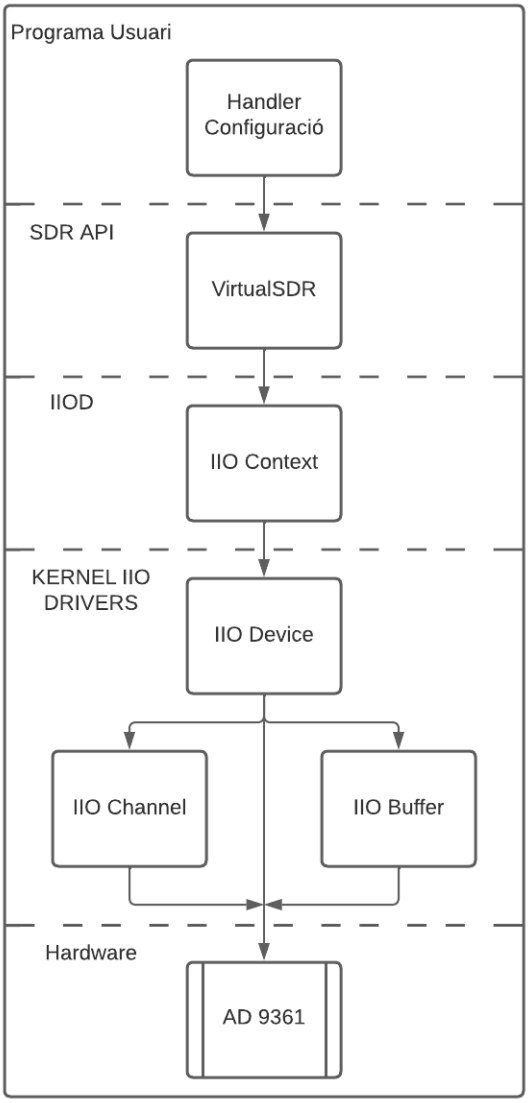

Fig. 2: Arquitectura de l'API

4.3 Proposta final

Amb l'API discutida en l'anterior apartat vam consultar amb els potencials usuaris quines millores proposaven. En general, eren petites funcions on es completava l'API però sense afegir una funcionalitat nova massa complexa. Tot i així, faltava una funcionalitat imprescindible per afegir, el "Board Support Package" (BSP), aquesta funcionalitat permet a la nostra API tenir més coneixement sobre el hardware sobre el qual s'executarà, cosa imprescindible ja que el plantejament del projecte era dissenyar una API universal, és a dir, dissenyar una biblioteca que funcionés en qualsevol hardware.

També vam afegir certes funcions que permetessin a l'usuari obtenir dades del sistema. Més específicament, vam afegir funcions per saber l'estat dels ports i per conèixer els paràmetres dels PLLs. D'aquesta forma l'usuari podria tenir accés a funcions que li donessin feedback del sistema.

Finalment, vam afegir funcions relacionades amb els possibles filtres del sistema. En el cas del xip sobre el que provaríem l'API no hi ha cap opció de controlar aquest filtre, però com volem fer una API que pugui servir per a totes les SDRs del mercat, vam decidir afegir aquesta funcionalitat.

5 Implementació de l'API

L'API s'ha desenvolupat de forma incremental, i els grups de funcions desenvolupats han sigut:

- Sistema de BSP
- Configuració del handler de SDR_Config
- Carga del handler de la configuració al handler de la SDR
- Connexió de la SDR virtual a la real per carregar paràmetres de la configuració
- Configuració de les funcions per transmetre/rebre dades
- Connexió de la SDR virtual a la real per transmetre/rebre dades
- Afegir funcionalitats de fitxers (guardar dades rebudes a un fitxer, transmetre les dades d'un fitxer, guardar i carregar la configuració actual a un fitxer)
- Afegir funcionalitats addicionals, com les funcions de test.

La implementació de l'API s'ha fet sobre el xip AD9361, així que és interessant que discutim la estructura de la SDR, tal com podem veure en la figura 3.

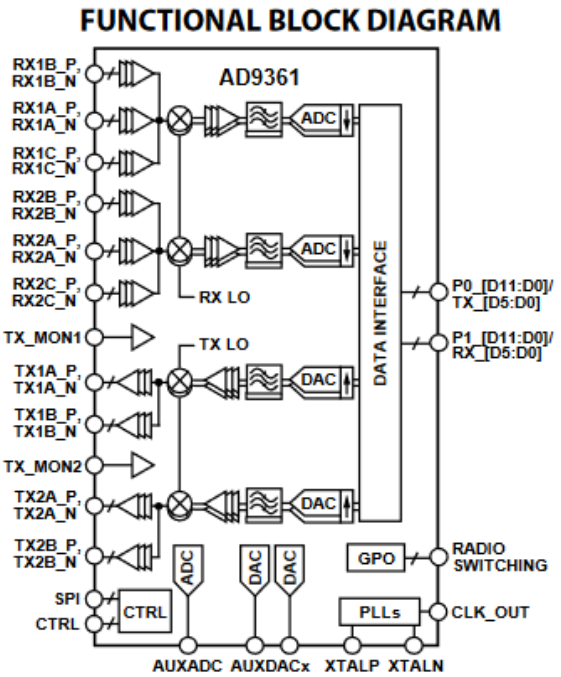

Fig. 3: Diagrama de blocs de la SDR

Com podem observar, l'estructura entre els receptors i transmissors és idèntica. Això simplificarà molt la tasca de programació. En aquests canals hi han un conjunt de blocs que són els que haurem de configurar. Tota aquesta gestió es farà mitjançant la biblioteca IIO d'analog devices [10]. Aquesta biblioteca funciona com una capa addicional d'abstracció, el nostre programa utilitzarà unes funcions

que faran crides externes a un mòdul del kernel del SO del sistema on es troba la SDR.

Un detall important a comentar és el fet de quines unitats utilitzarem per programar l'API. Hem decidit utilitzar les unitats del Sistema Internacional sempre que fos possible, és a dir, l'usuari haurà de treballar en Hz, en dB, i en mostres per segons.

5.1 Sistema de BSP

En la nostra API hem de tenir un sistema de BSP, això vol dir que hem de crear un fitxer que quan es llegeix, el nostre sistema crei una configuració per una SDR determinada. És a dir, el fitxer tindrà informació sobre el hardware de la SDR real i crearà una configuració per una SDR virtual a mesura.

La idea no és només que es pugui configurar paràmetres com el nombre de ports/ canals, sinó que ens permeti anar més enllà. Per exemple, fixem-nos en el següent esquema de la SDR sobre la que farem les proves:

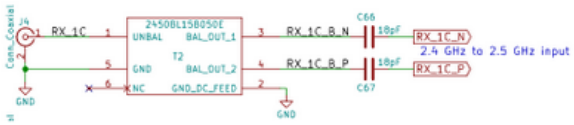

Fig. 4: Balun de la SDR amb la que es treballarà

Com podem observar, la SDR treu el senyal mitjançant dues línies de transmissió que, al passar per un balun, arriben al port de recepció en sí. Les característiques del balun ens limiten a quina freqüència podem rebre, en aquest cas ens limita a un rang de 2.4 GHz a 2.5 GHz (bàsicament a WI-FI). Això implica que el hardware està dissenyat per treballar a una freqüència determinada i, encara que el software ens ho podria permetre, no hauríem de treballar fora d'aquest rang. Aquesta és la funcionalitat més interessant de la BSP que hem implementat, per a cada paràmetre que es pot modificar es determina un valor mínim i un valor màxim per tal que l'usuari es vegi obligat a treballar en el rang que el hardware exigeix.

La idea, és que en les funcions on l'usuari modifica la configuració de la SDR, saltaran missatges d'error cada cop que intenti utilitzar un valor no suportat pel hardware.

El primer pas per dissenyar aquest sistema va ser decidir quin tipus de format d'arxiu utilitzaríem pel BSP. La primera opció va ser JSON, ja que és un format molt fàcil de llegir per l'usuari. El problema va ser que quan vam començar a plantejar l'estructura de l'arxiu, ens vam adonar que tampoc era tan clar per l'usuari. Al final, com l'usuari ha d'introduir uns valors mínims i màxims per cada paràmetre a configurar per cada port, la solució més senzilla era la d'implementar un sistema basat en arxius csv. Al final, tota la informació que el BSP necessita no deixa de ser una taula, on el nombre de columnes està clarament definit, motiu pel qual és un format fàcil d'implementar i relativament còmode per l'usuari.

Aquesta funció va ser la primera a implementar, ja que no només defineix uns paràmetres necessaris per la resta de funcions sinó que també defineix el com estructurarem la memòria del nostre programa. Com no sabem el nombre de línies que tindrà el nostre fitxer vam plantejar 3 opcions diferents que comentem a continuació:

1. Obrir el fitxer un cop per saber el nombre de línies i poder crear un array on guardar aquesta informació.
2. Afegir una linea inicial on s'indiqui el nombre de files del BSP i crear l'array a partir d'aquest valor.
3. Ús d'una llista encadenada, on no fa falta saber el nombre d'elements de l'arxiu.

D'aquestes 3 opcions, es va decidir utilitzar la tercera. La primera opció es va descartar, ja que obrir un fitxer per comptar el nombre de línies sembla una opció poc elegant, mentre que la segona es va descartar perquè la nostra filosofia a l'utilitzar un fitxer csv era que fos un fitxer simple de crear per l'usuari, si comencéssim a afegir línies "irregulars" a un fitxer que l'usuari final ha de crear, tindria més sentit utilitzar un fitxer tipus JSON.

L'usuari ha d'omplir un fitxer CSV on cada fila indicarà les característiques que defineixen cada canal de la SDR. El nombre de columnes amb les dades venen predefinides per nosaltres, és a dir, cada fila ha de tenir un format concret per a que pugui ser interpretat correctament. Actualment, no hi ha cap mecanisme que verifiqui si el format de l'arxiu és correcte o no, motiu pel qual és imprescindible que l'usuari de l'API crei correctament l'arxiu seguint el format que podem observar en la imatge 5, en aquest cas és un exemple d'una SDR amb dos canals de recepció i un canal de transmissió.

	A	B	C	D	E	F	G	H	I	J	K
1	Channel	Type	MinFS	MaxFS	MinFrec	MaxFrec	MinAmp	MaxAmp	MinBw	MaxBw	Number of Ports
2	A		0	20000000	40000000	5000	5000000000	20	90	1000000	16000000	2
3	B		0	20000000	30000000	5000	5000000000	20	90	2000000	32000000	1
4	A		1	20000000	30000000	2000	5000000000	10	90	1000000	16000000	2

Fig. 5: Exemple d'un fitxer BSP vist en format Excel

Juntament amb les funcions del BSP també vam afegir una funció pereliminar la memòria dinàmica del handler i per mostrar per la terminal la configuració actual de la SDR. Amb aquestes funcions creades, les vam provar per comprovar que tot funcionés correctament. Vam observar que el BSP es carregava bé i que la memòria dinàmica s'alliberava correctament.

5.2 Configuració del handler de SDRConfig

En aquest apartat vam haver d'implementar un conjunt de setters i getters per poder configurar la SDR. En aquesta fase, realment la majoria de les funcions tenen una estructura molt similar, en general és recórrer una llista fins a trobar el/els nodes que l'usuari ens indica i llegir o escriure el valor.

Com hem discutit anteriorment, el BSP ens força a fer que el valor que l'usuari vulgui introduir estigui dins d'un rang. La proposta d'aquesta API és que quan l'usuari vol afegir un valor per sota del mínim en realitat s'afegeixi el valor mínim, i si el valor està per sobre del màxim, afegirà el màxim. Aquest canvi es notifica a l'usuari mitjançant el codi que retorna cada funció, l'usuari pot saber a què es refereix cada codi d'error mitjançant una funció anomenada printSdrError. D'aquesta forma és molt fàcil treballar des de la terminal i funcionés comprovar que tot hagi anat correctament.

Amb aquestes funcions desenvolupades ja podem començar a desenvolupar les funcions relacionades amb la SDR virtual.

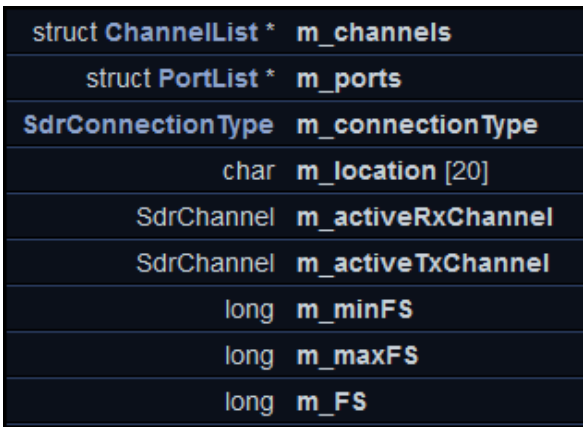

Fig. 6: Struct del handler de la configuració

5.3 Càrrega del handler de la configuració al handler de la SDR

Podríem pensar que al final, la funció de carregar la configuració només és copiar-se l'adreça de memòria del handler o que només s'ha de copiar tot el handler, però no és veritat. El handler està guardant dades que no són necessàries en la SDR (com per exemple els valors màxims i mínims que ens dona el BSP o els canals que no s'utilitzaran). Això significa que hi hauran dades que sí que es copiaran i altres que no.

La funció en si no deixa de ser recórrer la llista del Handler i anar guardant els valors en una llista interna de la SDR virtual. Aquesta funció té molt a veure amb la del handler de la configuració, així que també tindrem les dues funcions auxiliars per netejar la memòria dinàmica i per veure per terminal quina configuració s'ha carregat.

Amb aquestes funcions addicionals codificades vam procedir a fer diverses proves per veure que tot funcionés correctament. Totes les proves funcionaven correctament, la informació que s'habia de traspassar d'un handler a l'alre es copiava de forma correcta. També vam comprovar que les mesures implementades per evitar que la carga continuada de handlers ens crees problemes a l'hora de treballar amb memòria dinàmica funcionessin correctament.

Amb aquest apartat finalitzat ja tenim el handler de la configuració completament desenvolupat, motiu pel qual ara és un bon moment per discutir l'estructura del handler, podem veure a la imatge 6 com està codificat el struct del handler [11]. Com podem observar, el struct guarda el tipus de connexió i on es troba la SDR per poder fer la connexió amb la SDR real, guarda les dades relacionades amb la freqüència de mostreig, quin canals estàn actius i dues llistes. La primera llista guarda informació sobre els canals mentre que la segona llista guarda informació sobre cada port.

5.4 Connexió de la SDR virtual a la real per carregar paràmetres de la configuració

Un cop ja tenim les funcions necessàries perquè l'usuari pugui configurar la SDR hem de programar la funció InitSDR perquè la SDR virtual es connecti a la real i carregui la configuració. Per a poder parlar amb la SDR real necessitem utilitzar una llibreria que el fabricant ens proporciona. En el nostre cas treballarem amb una SDR basada en el xip d'Analog Devices AD9361, motiu pel qual treballarem amb la llibreria IIO, si utilitzessim un altre xip hauríem d'utilitzar una altre llibreria. Aquesta llibreria ens permet fer crides remotes al kernel del host de la SDR.

El primer que ha de fer la funció és establir la connexió entre el nostre programa i la SDR. Un cop fet això, haurem de fer vàries crides a diverses funcions per obtenir la adreça de memòria de cada paràmetre a configurar en la SDR real i, amb aquestes direccions, ja podrem configurar la SDR real.

5.5 Configuració de les funcions per transmetre/rebre dades

Aquestes funcions permeten a l'usuari enviar i rebre dades, sigui de forma continuada o no. Per a poder dissenyar aquestes funcions s'ha de tenir en compte el fet que cada SDR treballa a resolucions i potències diferents (tenen diferents ADCs i DACs), motiu pel qual si volem que l'API sigui tan genèrica com sigui possible hem d'establir un criteri de com es defineixen aquests senyals. El nostre plantejament és que l'usuari ha d'indicar el senyal a transmetre descompost en fase i quadratura (I i Q) amb un rang de valors des del 1 fins al -1. D'aquesta forma l'usuari només s'ha de centrar a triar quanta potència vol transmetre (o quanta ha rebut) d'acord amb el valor màxim (1) sense necessitat que sàpiga com internament la SDR processa les dades.

Buffer Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AD9361 Bit	11	10	9	8	7	6	5	4	3	2	1	0	X	X	X	X

Fig. 7: Formati binari del buffer de l'AD9361

En el cas de l'AD9361, els ADCs i els DACs són de 12 bits, on el que es fa és ignorar els 4 bits menys significants d'un uint16t, tal com podem veure a la taula 7. La nostra funció haurà d'adaptar els senyals tenint en compte el funcionament intern del sistema. Aquesta problemàtica només la tenim a l'hora de transmetre les dades, no de rebre-les. Aquest és un exemple de perquè volem crear una API que generi una capa d'abstracció addicional a l'usuari final. Si ens fixem en altres SDRs podem observar com aquest patró es repeteix. Per exemple, BladeRF [5] reb les dades en un format de 11 bits més signe (exactament igual que en el AD9361). Aquest format de dades l'anomenen SC16 Q11. En canvi, Soapy SDR requereix que l'usuari envii les dades en format de número complex.

Finalment, hem de modificar la funció de startSDR per a afegir aquesta nova funcionalitat d'enviar i rebre dades. El que hem de fer és crear un buffer en la SDR on guardarem les dades, i el nostre programa es comunicarà amb aquest buffer per copiar els valors. Cal destacar que aquest buffer pot ser circular o no, en cas que vulguem enviar dades de forma contínua.

Aquestes funcions van ser programades mentre no podíem accedir a la SDR real per fer cap mena de prova, motiu pel qual, tenien els mateixos errors que vam apreciar en la configuració de la SDR, amb algun altre error addicional. Un cop solucionats els errors que sabem que teníem ja era moment de provar si tot funcionava correctament en la SDR real.

5.6 Connexió de la SDR virtual a la real per transmetre/rebre dades

A continuació vam fer diversos tests per comprovar que es transmetes les dades de forma correcta, vam fer un test enviant un to e intentant rebre'l. Quan vam executar la funció, vam poder comprovar que tot funcionava correctament. Vam comprovar que podíem utilitzar tots els canals de la SDR de forma correcta. Això vol dir que ara ja tenim la major part de l'API desenvolupada. Només ens queda desenvolupar les funcionalitats relacionades amb fitxers i les addicionals.

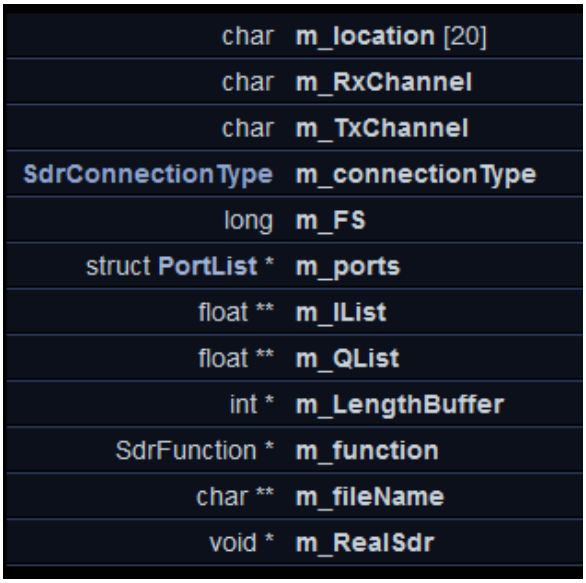

Fig. 8: Struct de la SDR virtual

Ara és el millor moment per discutir el struct que representa la SDR virtual ja que les noves funcions no afegiran res més. El struct que discutirem ara el podem observar en la imatge 8. Com podem observar [11], mantenim el tipus de connexió, la localització de la SDR, els canals actius i la freqüència de mostreig que trobem al handler de la configuració 6, encara que s'han processat la localització de la SDR i els canals actius a un altre format. Si seguim comentant les diferències amb el handler de la configuració, podem veure que la llista de canals ara no existeix. Això és degut a que la llista dels canals només existeix per a que el BSP pugui funcionar correctament. En canvi, la llista dels ports és una còpia exacta de la llista del handler de la configuració.

A continuació, hem de discutir els següents 3 paràmetres del struct, que són dues matrius de floats i un array d'ints. Aquets paràmetres són els buffers on es guardaran les dades a transmetre i les dades a rebre. És interessant destacar que en realitat no tenim dues matrius sinó dos arrays de punters de tipus float. Això és important ja que quan volem rebre les dades l'usuari haurà de crear ell el buffer i donar el punter a la nostra API. D'aquesta forma l'usuari final no haurà de necessitar tenir en compte funcions addicionals per gestionar aquests buffers interns.

Finalment, hem de discutir el punter de tipus void que trobem en el handler. Aquest punter no té un tipus assignat ja que és el punter auxiliar on es guardaran tots els paràmetres necessaris per gestionar la SDR real. La idea és que qualsevol altre SDR necessitarà els paràmetres anteriorment mencionats, però sempre hi hauran alguns que dependran de cada SDR. La idea és que en la implementació de l'API es declararà un struct amb tots els paràmetres necessaris per a funcionar. En el cas de l'AD9361 només necessitarem emmagatzemar dos paràmetres, però en altres situacions podríen ser molts més.

5.7 Afegir funcionalitats de fitxers

Amb la major part de l'API desenvolupada ens disposem a discutir els dos tipus de funcions relacionades amb fitxers, les que ens permeten rebre o transmetre dades des de fitxers i les que ens permeten guardar la configuració de l'API. Per mantenir la consistència en relació amb el BSP desenvolupat, aquests fitxers seran en format CSV.

Les funcions relacionades amb la transmissió i la recepció de dades són implementacions molt similars a les que hem discutit anteriorment, amb l'única diferència que afegim un pas addicional per treballar amb els fitxers. En el cas de la transmissió de dades, el nou codi haurà de crear uns buffers i omplir-los amb les dades que hi hagin en el fitxer que indiqui l'usuari. En canvi, a nivell de recepció, el que hem de fer és esperar a haver rebut dades per després guardar-les en un fitxer en format CSV.

Ara podem discutir com guardem la configuració creada per l'usuari. En aquest cas estem parlant d'un fitxer que l'usuari no ha de crear, motiu pel qual podem acceptar guardar la informació en un format que sigui poc intuïtiu. Seguirem amb la idea d'utilitzar un fitxer CSV on volcarem totes les dades que tingui el handler de la configuració.

Aquestes funcions tal i com estan implementades obren el fitxer i suposen que és el format correcte, no hi ha programat cap mecanisme que informi a l'usuari si el format és incorrecte i eviti una inicialització incorrecta. És important tenir en compte aquest factor per tal d'evitar errors.

5.8 Afegir funcionalitats addicionals, com les funcions de test

En aquest apartat discutirem les últimes funcions a desenvolupar de l'API. Les principals funcions a desenvolupar son les anomenades funcions de test. A priori podrien semblar les més complexes d'implementar, però com ja tenim tota l'API implementada en realitat ens trobem amb unes funcions que criden internament a la resta de funcions, però no s'ha de desenvolupar una gran quantitat de codi. En aquestes funcions farem ús de la FFT, utilitzarem una implementació recursiva molt simple, ja que la idea no és donar unes funcions optimitzades al màxim sinó que volem donar unes funcions que puguin servir per fer proves ràpides per comprovar que tot funciona correctament.

A continuació cal discutir les funcions de sincronisme del sistema. Hem de tenir una forma de consultar l'estat de cada port de la SDR, si està en "ON"o si està en "OFF". En el nostre cas, hem decidit que un port estarà en "ON"si es troba transmetent de forma contínua i es trobarà en "OFF"en cas contrari. Això és degut a que quan rebem dades sempre les rebrem totes de cop, motiu pel qual sempre s'haurien de trobar en OFF aquests ports.

Seguidament hem de discutir les funcions que no podem implementar ja que el xip AD9361 no suporta aquestes funcionalitats. La primera funció és la que ens permet obtenir informació sobre els PLLs de la placa. En aquest cas no és

possible accedir a les dades d'aquests sistemes. El segon grup de funcions son respecte el tipus de filtre del sistema. En aquest cas, podem saber que el filtre del sistema correspon a un butterworth d'ordre 3, però no podem modificar el filtre en si, per tant, són funcions que no donarem suport.

Finalment, cal destacar les funcions que alliberen tota la memòria dinàmica dels handlers, les funcions per mostrar per pantalla les configuracions dels handlers i la funció per indicar quin codi d'error ha tornat cada funció. Aquestes funcions s'han desenvolupat en conjunt de la resta de l'API, però realment no podíem afirmar que les hem completat fins que haguéssim completat la programació de l'API.

6 Conclusions

Finalment, podem veure que tenim una API completament funcional desenvolupada. Un dels grans reptes és el fet d'intentar generalitzar el màxim possible el disseny d'una biblioteca. Encara que vam fer un estudi de mercat, i que a primera vista sembla que no hi ha masses SDRs al mercat, hi ha diferències prou notables com perquè s'hagin de fer concessions. Aquesta API pretén ser una interfície molt senzilla per a poder fer programes amb SDRs sense necessitat de tenir un gran nombre de coneixements de com funciona el sistema en si, motiu pel qual pot ésser que depenent de què vol fer l'usuari final, pugui preferir treballar directament sobre la SDR. Tot i això, la idea és que la nostra proposta doni eines suficients per a poder ser utilitzada en un gran nombre de casos, sobretot pot ser una gran eina per a fer proves de conceptes o experiments.

Futures millores d'aquest projecte poden ser optimitzar el codi o ampliar la seva funcionalitat. També es podria optar per buscar un altre SDR per programar el suport a aquesta nova plataforma de forma que puguem comprovar que el nostre disseny realment és el prou genèric. Una millora futura interessant seria integrar la nostra API amb GNU ràdio, un entorn de desenvolupament per aplicacions de telecomunicacions, d'aquesta forma podem apropar el nostre projecte als usuaris finals donant-lis una millor experiència.

Evidentment, la conclusió final més important és el fet que podem fer aplicacions per SDRs sense tenir la necessitat de tenir un gran coneixement sobre la SDR en si. El fet de simplificar l'experiència a l'usuari final és un factor clau a l'hora de desenvolupar qualsevol aplicació. Independentment del projecte, és molt comú tenir una fase de prova de conceptes i prototipatge, i aquesta API és ideal per a aquest fi. De fet, la documentació d'aquesta API s'ha enviat a un grup de nano-satelits de la UPC i a un grup d'interferometria GNNS per a que analitzin si els hi agrada aquesta proposta i, en cas afirmatiu, que la puguin utilitzar.

Agraïments

Agrair a Màrius Montón per l'ajuda i els consells que m'ha donat durant el desenvolupament de l'API, en especial durant la fase de la proposta d'API, on al principi les coses no van sortir com esperava.

Agrair a Dani de l'Institut de Ciències de l'Espai pel seus esforços quan la SDR de l'ICE no funcionava correctament.

Agraïments a Serni Ribó per haber donat feedback sobre l'API.

Referències

[1] Santander: Metodologías de desarrollo de software: ¿qué son? [En línia]. Disponible: https://www.becas-santander.com/es/blog/metodologias-desarrollo-software.html

[2] Instituto Europeo de Postgrado, "Meto- dología Waterfall: Modelo de gestión de proyectos en cascada", [En línia]. Disponible: a:https://www.iep.edu.es/metodologia-waterfall

[3] Reddit, L·Low-cost software defined radio (RTL2832 SDR) community"[En línia]. Disponible: https://www.reddit.com/r/RTLSDR/

[4] Analog Devices Wiki, "AD-FMCOMMS2-EBZ User Guide"[En línia]. Disponible: https://wiki.analog.com/resources/eval/user-guides/ad-fmcomms2-ebz

[5] NUAND, "BladeRF"[En línia]. Disponible: https://www.nuand.com/

[6] Lime Microsistems, L·LimeSDR"[En línia]. Disponible: https://limemicro.com/products/boards/limesdr/

[7] XTRX, "XTRX A fairwaves tiny SDR"[En línia]. Disponible: https://xtrx.io/

[8] Crowd Supply, ïotSDR"[En línia]. Disponible: https://www.crowdsupply.com/embedinn/iotsdr

[9] Osmocon, "rtl-sdr"[En línia]. Disponible: https://osmocom.org/projects/rtl-sdr/wiki/Rtl-sdr

[10] Analog Devices, "the libIIO documentation"[En línia]. Disponible: https://analogdevicesinc.github.io/libiio/

[11] Jordi Castillo, "Virtual SDR API"[En línia]. Disponible: https://jordi273.github.io/

Apèndix

A.1 On trobar l'API

L'API es pot trobar en el següent enllaç de github: https://github.com/Jordi273/TFG-SDR En cas que l'enllaç no funcionés és l'usuari Jordi273 en la carpeta TFG-SDR.