UNIVERSITAT POLITÈCNICA DE CATALUNYA

Circuit DDS en format XBee

9 de juny de 2014

Dissertació que presenta Joan Martínez Domene
sota la direcció de Dr. Eng. Pere Palà Schönwälder
per assolir el grau d'Enginyer en Sistemes TIC.

A la meva filla Valèria.
Amb el seu somriure m'ajuda a creure
que un món millor és possible.

Índex

Agraïments

Vull agrair l'ajuda rebuda per tot el departament del DIPSE
durant aquests anys, en que els alumnes de les TIC
ens hem pogut sentir que formàvem part de quelcom important.

Especialment he d'agrair al tutor de projecte, Pere Palà, per el suport donat,
les idees i l'enfoc que m'ha aportat en el bon desenvolupament d'aquest.

També vull agrair a tots els meus companys de la primera promoció
del Grau en Enginyeria de Sistemes TIC, per la seva amistat, compromís
i col·laboració al llarg d'aquests quatre anys.

I molt especialment vull agrair tot el suport, paciència i comprensió
que m'ha donat Pilar, que ha fet possible aquest treball.

Gràcies a tots.

Abstract

This document contains the technical and constructive report for a microcontrolled DDS oscillator circuit. The onboard microcontroller allows the programming of the oscillator in a versatile way, using an instruction set close to the user's language. The circuit is compatible with XBee footprint. Moreover the document also contains the user documentation and examples for proper use.

The device is intended to be a basic building block for a wide range of applications, thus easing the process of electronic system's engineering. It exhibits remarkable degrees of flexibility, which makes it amenable to a wide range of possible applications.

Resum

El present document recull, per una banda, la memòria tècnica i constructiva d'un circuit oscil·lador basat en DDS i gestionat per un microcontrolador, el qual permet la programació d'aquest oscil·lador de manera versàtil, utilitzant un seguit de instruccions properes al usuari-programador. El circuit és compatible amb el *footprint* dels mòduls XBee. Per altra banda també consta de la documentació d'usuari i els exemples adequats per al seu ús.

El dispositiu té la finalitat de contribuir a l'enginyeria de sistemes electrònics, aportant una peça habitual en múltiples aplicacions i dotar-la de la flexibilitat suficient per tal d'adaptar-se al major ventall d'aplicacions possible.

El document està estructurat de la següent manera:

Introducció i estat de l'art En aquest capítol s'exposen els diferents criteris, tècnics i socials que influeixen en l'estat de l'art tecnològic i d'on neix la necessitat de poder disposar d'estructures o blocs de disseny comuns i amb formats estàndard que facilitin la tasca de d'aprenentatge, disseny i fabricació.

Descripció del conjunt Aquí es desenvolupa el concepte de funcionament general que hi ha en el dispositiu. El capítol dona una visió d'allò que s'espera d'aquest dispositiu des d'un punt de vista tècnic.

Maquinari Aquest capítol justifica els elements destacats del projecte, quina és la seva funció en el sistema i com és el seu comportament.

Programa Dins d'aquest capítol es troba la relació que hi ha entre els diferents mòduls de programa que formen el *firmware* del dispositiu. Es comenten les parts de codi destacables. Per la seva importància, es descriu el comportament d'execució de les funcions-comanda i de les execucions de codi als esdeveniments externs. Aquests dos elements són fonamentals per entendre l'arquitectura del projecte.

Manual d'usuari El contingut d'aquest capítol coincidiria amb el manual de que disposaria l'usuari. El capítol s'ha redactat tenint en compte l'àmplia varietat de persones que poden estar interessades en el dispositiu. Per tant s'ha escollit una organització dels apartats de forma acurada. Especialment útil és l'apartat dedicat a les comandes.

Aplicacions En aquest capítol es veuen uns pocs exemples de les possibilitats d'ús que aquest dispositiu pot permetre. Es descriuen les aplicacions de manera simple, ja que únicament tenen una finalitat il·lustrativa en el context de la documentació del projecte. Tot i així es pot veure el potencial del dispositiu.

Conclusions El capítol descriu breument els objectius que s'han assolit amb la realització d'aquest projecte i les conclusions que es desprenent del treball realitzat.

Futures línies de treball Aquí es detallen aquelles línies de treball que, de forma no esperada, s'han obert durant el desenvolupament del dispositiu i condueixen el projecte a properes ampliacions.

Annexos Per el seu interès s'ha afegit en el present projecte el codi font de programa, suficientment comentat. Així com el disseny de la placa realitzada i el full de característiques del circuit DDS emprat. També s'hi pot trobar els components utilitzats i el seu cost.

3

1. Introducció i Estat de l'Art

El sector industrial electrònic ha tingut des de fa dècades una gran importància en les nostres vides, tant des de el punt de vista de l'electrònica domèstica, fins a les aplicacions militars, passant per tots els camps intermitjos com per exemple la indústria, la medicina, el transport o els serveis. A dia d'avui encara hi ha persones que pensen i declaren que aquesta indústria és el futur, però en realitat és el nostre present des de fa temps. Per situar el projecte en el seu àmbit cal tenir en compte diferents aspectes.

1.1. Àmbit tecnològic

En un ventall tant ampli de camps d'aplicació, els productes i aplicacions electròniques són incomptables. No és possible ni tant sols intentar fer un breu recompte d'equips i dispositius amb els que ens podem trobar diàriament al llarg d'un dia qualsevol. De totes maneres aquests equips moltes vegades tenen parts que són comunes entre dispositius molt diferents. Potser no són exactament idèntiques, ni tant sols serveixen a la mateixa finalitat, però conceptualment són el mateix. Aquest és el cas dels *oscil·ladors*.

Els circuits oscil·ladors[2] són aquells que són capaços de convertir un corrent continu en un corrent que varia de forma periòdica en el temps (corrent periòdic); aquestes oscil·lacions poden ser sinusoïdals, quadrades, triangulars, etc., depenent de la forma que tingui l'ona produïda. Un oscil·lador d'ona quadrada se sol denominar multivibrador i de fet, només s'anomena oscil·ladors als que funcionen en base al principi d'oscil·lació natural que formats per una bobina L (inductància) i un condensador C (capacitància), mentre que els altres se'ls assignen noms especials.

Un oscil·lador electrònic és, conceptualment, un amplificador que el seu senyal d'entrada es pren de la pròpia sortida a través d'un circuit de realimentació. En la seva visió clàssica un oscil·lador es pot considerar que està compost per:

- Un circuit el desfasament que depèn de la freqüència. Per exemple:
  - Oscil·lador elèctric (LC) o electromecànic (quars).
  - Retard de fase RC o pont de Wien.
- Un element amplificador
- Un circuit de realimentació.

Existeixen multitud de tipus d'oscil·lador, com per exemple:

- Oscil·lador RC
- Oscil·lador RF

- Oscil·lador LC
  - Oscil·lador Colpitts
    - Oscil·lador Seiler
    - Oscil·lador Clapp
  - Oscil·lador Hartley
  - Oscil·lador Vackar
- Oscil·lador Pierce
- Oscil·lador de cristall
- Oscil·lador de pont de Wien
- Oscil·lador de cavitat
- VCO
- PLL
- DDS

Tal com s'ha comentat anteriorment, l'oscil·lador pot tenir diferents aplicacions en els diferents camps en els que els sistemes electrònics intervenen. Per cada camp d'aplicació, l'enginyer electrònic decideix quin és el tipus d'oscil·lador que més s'adapta a las característiques que es requereixen per l'aplicació final a la que ha de pertànyer.

En aquest punt s'han de valorar costos, fiabilitat, precisions, marges de treball i qualsevol altre paràmetre de decisió usual per l'enginyer de sistemes.

El present projecte vol centrar-se en un disseny particular d'oscil·lador, donada l'àmplia utilització d'aquest circuit en el món de l'electrònica. Es pretén aportar, en el seu disseny, un sistema suficientment flexible per tal d'adaptar-se a diferents aplicacions i responent a l'idea de crear una eina versàtil i universal a tenir en compte. El fet de pretendre universalitzar una eina d'aquestes característiques no és una tasca simple ni trivial, ja que, tal com s'ha comentat, intervenen multitud de factors que caracteritzen la tria del dispositiu adequat per cada rang d'aplicacions.

1.2. Àmbit tecno-antropològic

La complexitat o no de fabricació de l'electrònica a facilitat en alguns moments i empitjorat en altres l'accés al coneixement i l'ús dels sistemes electrònics per l'us pedagògic, per la fabricació de prototips o bé simplement per hobby.

Fa unes dècades era fàcil aconseguir esquemes electrònics i kits per realitzar muntatges. Aquests muntatges, en els temps en van aparèixer, estaven amb línia amb els productes electrònics. Es podien construir elements electrònics amb característiques similars als productes de mercat.

Més tard l'evolució de l'electrònica, sobretot l'aparició de microprocessadors en dispositius d'ús comú, va posar a els productes electrònics de mercat per davant dels dispositius que podien ser muntats en un taller d'una escola o en el garatge d'un aficionat, amb l'excepció d'alguns casos d'èxit molt populars.

L'aparició de Internet va permetre posar en contacte persones de llocs distants, compartint informació. Al mateix temps que la tecnologia ha sigut fabricada en rangs de producció massius que han fet abaratir els costos de productes electrònics i dels seus components. Al barrejar l'intercanvi de coneixement i facilitat d'adquisició de components i dispositius electrònics, s'aconsegueix retornar al mateix punt en què es fàcil poder experimentar, aprendre i treballar amb sistemes electrònics, amb una complexitat similar a la que es pot trobar dins dels dispositius electrònics d'ús comú que el mercat ens ofereix. En molts casos els nivells de prestacions entre prototips i equips de mercat poden ser diferents però, en essència, la tecnologia que implica és la mateixa.

Són molt coneguts en els àmbits tècnics els diferents moviments que s'han creat entorn el concepte de *Programari Lliure*, com ara els *Open Source Software* o el *Free Software*. El mateix concepte aplicat a equips i dispositius existeix des de fa moltes dècades. L'intercanvi i publicació de esquemes i instruccions detallades no és un fet nou. No obstant l'etiqueta que defineix el fet de publicar de forma oberta les fonts necessàries per la fabricació, és de creació molt més recent. La definició de *dispositiu de fonts obertes* o OSHW (de les sigles en anglès *Open Source Hardware*) s'estén més enllà del que són pròpiament dispositius electrònics[3], abastant qualsevol tipus d'objecte físic, susceptible de ser fabricat, com per exemple els models tridimensionals per ser manufacturats per tècniques de impressió 3D.

Enfocant el concepte de *Open Source Hardware* concretament al món de l'electrònica apareixen multitud de plataformes[4] i dispositius sota aquest gran paraigües. Entre ells podem nombrar alguns com ara:

OLinuXino[5] és un projecte de Software i Hardware Oberts i de baix cost basat en Linux (A la Figura 1.1). Aquest sistema és un ordinador en una placa i característiques industrials amb GPIO's que pot treballar en el rang de temperatura -25º a +85ºC.

Beagle Board[6] és un ordinador en una placa (A la Figura 1.2) que incorpora un processador OMAP3530 720MHz ARM Cortex-A8, extensions NEON and VFP acceleració addicional, gràfics POWERVR™, vídeo d'alta resolució i capacitat de realitzar stremming amb un dispositiu media player portàtil. Treballa amb les funcionalitats completes d'un ordinador tradicional en un encapsulat minúscul.

Arduino[7] és una plataforma electrònica de fonts obertes (A la Figura 1.3)per a desenvolupament, bassada en Hardawre i Software flexible i fàcil d'utilitzar. Està enfocat a artistes, dissenyadors, aficionats i en l'àmbit de la educació. Inicialment els dispositius electrònics d'Arduino s'han construït al voltant de microcontroladors de la familia AVR de Atmel. En l'actualitat estan apareixen noves plataformes Arduino amb una tecnologia diferent.

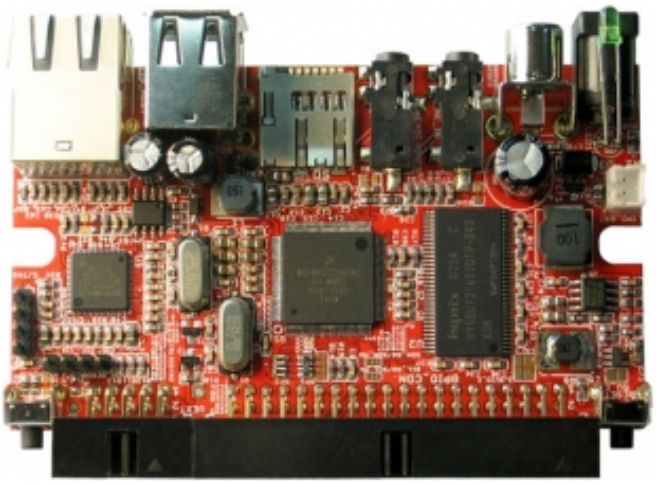


Figura 1.1.: Single Computer Board OLinuXino

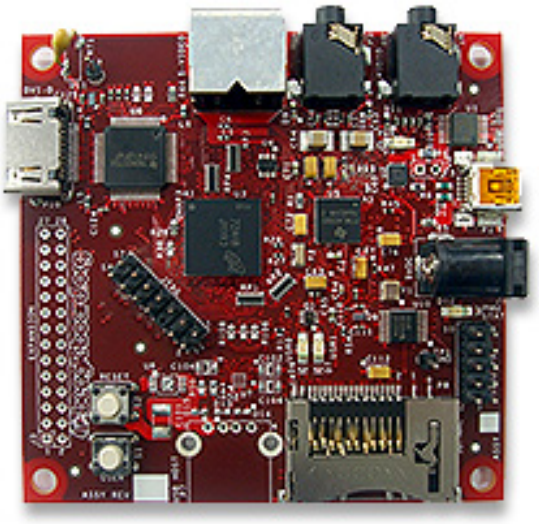


Figura 1.2.: Single Computer Board Beagle Board

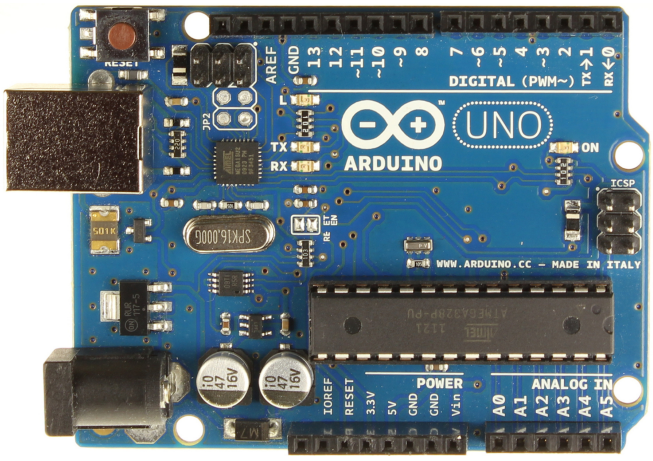


Figura 1.3.: Placa de la plataforma Arduino

Al fer públiques les fonts d'aquests projectes, han aparegut multitud de fabricants que han agafat els dissenys i han fabricat les rèpliques d'aquests dispositius. Segurament el cas d'Arduino és el més replicat[8]. Per exemple podem trobar Roboduino, Seeeduino, Brasuino, Diavolino, Japanino o Freeduino per nombrar-ne alguns.

Des de el punt purament econòmic de la organització, això poden ser algunes vendes que deixen de fer. Per altre banda, s'obté una massificació d'usuaris que saben fer anar aquesta plataforma i poden aportar valor afegit al moviment Arduino.

Un dels punts favorables es que també es genera una economia de productes compatibles amb Arduino, com són per exemple les anomenades *Shields*. Les *Shields* són plaques electròniques que s'adapten al format Arduino i es connecten a aquest per tal d'oferir algun tipus de funcionament especific o ampliació del Hardware. Per exemple la Libelium Xbee Shield (A la Figura 1.4), GeekOnFire GPRS/GSM Shield (A la Figura 1.5) o DFRobot LCD Shield per anomenar algunes de les 317 que es poden trobar en Arduino Shield List [9].

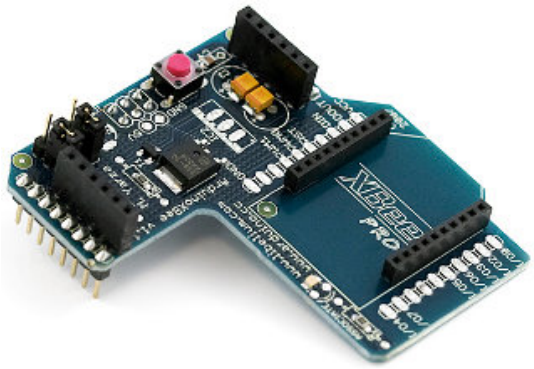


Figura 1.4.: Libelium Xbee Shield, fabricada a Saragossa

Figura 1.5.: GeekOnFire GPRS/GSM Shield

L'existència del moviment *OSHW* i del col·lectiu *Arduino* proveeix d'un entorn favorable per la creació del Treball Final de Grau al que es refereix aquesta memòria, tant des de el punt de vista del propi procés de disseny i creació, pel qual s'ha utilitzat una plataforma Arduino, com des del punt de vista d'acceptació del projecte en els col·lectius que requereixin d'un dispositiu oscil·lador com el proposat. La compatibilitat del format utilitzat amb determinades *Shields* existents fa que aquest dispositiu sigui atractiu i de fàcil integració dins dels projectes d'aplicació. Al mateix temps aquest projecte aporta un nou conjunt de funcionalitats, no disponibles fins al moment actual.

1.3. L'arquitectura de sistemes

Els canvis de les darreres èpoques també han influït en la visió que l'enginyer de sistemes té sobre el hardware disponible, en el moment en que ha fer front a un nou disseny. De manera clàssica el enginyer de sistemes dedicava el 70% de les seves hores de treball en crear prototips físics que esdevenien productes hardware acabats. Actualment es requereix que existeixin mòduls estàndard que serveixin pel seu fi. Aquests mòduls han de tenir molta flexibilitat per donar resposta a un bon nombre de camps d'aplicació.

Siguen d'aquesta manera el mercat ha de proveir dels recursos necessaris a l'arquitectura de sistemes, per tal que l'enginyer pugui desenvolupar models de forma àgil i que aquests models puguin esdevenir productes finals, si pot ser utilitzant els mateixos recursos estàndard.

Els estàndards ho són per diferents naturaleses: per legislacions, per regularitzacions, per recomanacions tècniques o *de facto*.

Quan una determinada característica, no trivial, en un producte es adoptada per multitud d'altres productes de forma intencionada, aquesta característica esdevé un estàndard de facto. Aquests estàndards tenen bona acceptació en el mercat perquè són de fàcil assimilació amb productes que ja posseeixen característiques compatibles. La forma i patillatge dels mòduls *XBee* de la companyia *Digi* són un bon exemple. Podem veure aquesta a la Figura 1.6

Figura 1.6.: Forma i pinout dels mòduls XBee

Donada la bona acceptació del format físic *XBee*, en el present projecte s'ha optat per un disseny amb la mateixa forma física que la dels esmentats mòduls. D'aquesta manera es busquen les sinergies i complicitats dels productes ja existents en el mercat, oferint una característica innovadora en aquest format de mòduls.

1.4. Qualitat vs. preu

En la recerca de dispositius similars al que es proposa en aquest projecte, s'han trobat dos tipus de plaques disponibles: Per una banda els fabricants de circuits integrats posen en el mercat determinades plaques de avaluació (Evaluation Boards). Per altra banda, es poden trobar petites plaques muntades a xina.

Les Evaluation Board són plaques fabricades amb gran qualitat i destinades a la realització de proves en laboratori, cobrint l'espai de disseny previ a la creació dels primers prototipus d'un determinat producte. Aquestes plaques disposen de connexions i punts de prova per facilitar la mesura, comprovant la idoneïtat d'un determinat circuit i les seves limitacions. Siguen aquest l'objectiu, les plaques d'avaluació es presenten amb un format molt espaiat, ja que les característiques dimensionals no és un aspecte característic a considerar per l'àmbit d'aplicació que tenen. A la Figura 1.7 podem veure l'aspecte d'una d'aquestes plaques. En concret es tracta de la placa *AD9837 Evaluation Board* de la companyia Analog Devices[10]. Aquesta és una Evaluation Board d'un circuit DDS molt similar a l'utilitzat en aquest projecte.

Les plaques d'avaluació degut al camp d'aplicació que tenen i sobretot a la seva baixa demanda per un determinat us específic, són costoses i això provoque que quedin allunyades dels àmbits de disseny i producció no industrial. El cost de la placa del la Figura 1.7 està al voltant de 60€. Normalment es poden acompanyar de documentació i el fabricant proporciona suport, amplia informació i Notes d'Aplicació a través de la pròpia web.

En els darrers anys s'ha donat un fenomen d'expansió de la fabricació de tecnologia a la xina. Es fabrica electrònica a costos molt baixos i s'exporta a tot el món amb molta facilitat. Aquesta

Figura 1.7.: Placa d'avaluació del DDS AD9837

situació, junt a l'increment de comunitats al voltant de la creació de productes tecnològics, com s'ha apuntat en l'apartat 1.2, han donat lloc a que des de la xina proliferi la fabricació de petits mòduls de tot tipus destinats a la fabricació de ginys, interconnectant aquest mòduls entre si, o a un altre tipus de plataforma o muntatge. A la Figura 1.8 hi ha un exemple del que es pot trobar. En aquest cas també es tracta d'una placa basada en un circuit DDS. Es desconeix de qui és el disseny ni on s'ha fabricat, ja que aquest mòdul es pot trobar a través de diferents llocs webs i la font original queda enterbolida.

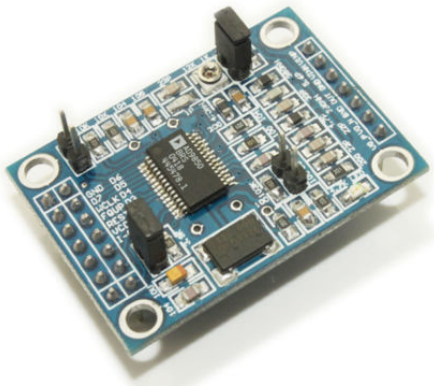


Figura 1.8.: Mòdul DDS AD9850, fabricat a xina

El cost d'aquesta placa pot estar entre 7 i 10€. Si be el cost és un factor molt interessant per aquests productes, la baixa qualitat de fabricació i la manca de garantia per part del fabricant/venedor és un altre element a tenir en compte. A més, aquest tipus de producte no té cap mena de valor afegit, com per exemple una documentació en bones condicions o suport post venda. Aquests mòduls són molt interessant per els aficionats o en el desenvolupament de parts poc costoses d'equips prototipus. Però per el sector productiu i industrial té poc interès donada la seva manca de suport i qualitat.

Donats aquests aspectes, podem concloure que en l'actualitat no existeix una solució amb prou garanties perquè resulti interessant en l'àmbit professional i prou econòmica per tal que sigui assequible per a qualsevol aficionat, estudiant o petit taller.

El dispositiu, objectiu d'aquest Treball Final de Grau, pot donar resposta també en aquest punt concret, oferint una interfície d'usuari i una secció de comandes ben documentada. Tot això suposa un valor afegit en contra d'uns mòduls (tant les Evaluation Boards com els mòduls fabricats a xina) que no posseeixen aquestes característiques. Els costos també poden reduir-se en relació a les Evaluation Boards i potser significativament, en cas que es pugues platejar una fabricació massiva.

2. Descripció de Conjunt

2.1. Característiques

El propòsit d'aquest projecte és desenvolupar un mòdul electrònic, que implementa un oscil·lador per Síntesi Digital Directa (o *DDS*) microcontrolat. El fet de disposar d'un microcontrolador que gestiona el treball del oscil·lador digital, permet facilitar el seu ús sense haver de renunciar a la flexibilitat que el propi dispositiu DDS proporciona. És a dir, afegim una capa d'interfície per tal que l'usuari implementi les seves aplicacions de forma simple, sense preocupar-se de com es gestionat el diàleg entre el microcontrolador i el propi oscil·lador.

El disseny respecta la forma i disposició de terminals que es pot trobar en un dispositiu *XBee*. Aquest factor proporciona una compatibilitat amb altres dispositius adaptats per allotjar a tercers que respecten aquest *outline* característic i que es converteix en un estàndard *de facto*.

Al respectar el factor de forma dels mòduls *XBee* i com es tracta d'un mòdul que en el seu cor incorpora un DDS com a característiques principals, aquest projecte el podem anomenar com a *DDSBee*.

Com característiques del sistema podem destacar:

- Generació de tres formes d'ona integrades: Sinusoïdal, triangular i quadrada.
- El sistema està preparat per poder definir altres formes d'ona a través del firmware.
- Marge de freqüències entre 0 i 12,5MHz que pot ampliar-se amb filtrat extern sobre les freqüències de àlies.
- Salts de freqüència de 1 Hz. amb un arrodoniment del DDS inferior a 0.1Hz.
- Salts de fase de 1^{o} amb un arrodoniment del DDS inferior a 0.09^{o} .
- Resolució del DAC del DDS 10 bits.
- velocitat d'activació per interrupció externa (mode ASK): 18.5us.
- Interfície d'usuari còmode.
- Més de 30 comandes implementades per la gestió del DDS i els seus modes.
- Funció d'ajuda integrada.
- comunicació UART d'alta velocitat (115200 bd) amb desviació de velocitat inferior a 1.4% . La velocitat pot ser ajustada per firmware.

2.2. Modes de funcionament

El DDSBee es pot gestionar de dos maneres diferents:

Per comandes: El circuit respon a les ordre introduïdes des d'un port sèrie UART. La comunicació pot venir des d'un ordinador o des de qualsevol equip en funcionament autònom. El DDSBee té un comportament similar al que es podia trobar en un mòdem telefònic, el qual responia a les comandes AT. En el nostre cas, les comandes que utilitza són instruccions pròpies creades *ad-hoc*. Aquestes comandes són acrònims del anglès d'allò que la funció realitza. Com per exemple *SRFR0* que significaria *Set and Run Frequency Register 0*. És a dir, programa el registre de freqüència número zero i posa'l en funcionament de forma immediata. Per consultar el llistat complet de les comandes implementades, s'ha de veure l'apartat 5.6 en el Manual d'usuari.

Per interrupcions: El circuit pot funcionar de manera autònoma. No obstant, s'ha de haver fixat algun mode de funcionament de forma prèvia, seleccionat a través de l'us de comandes. Utilitzant les interrupcions, el DDSBee respon als canvis d'estat dels terminals prèviament definits, amb accions corresponents al mode de funcionament seleccionat. És a dir determinada acció sobre un pin del circuit pot generar un canvi en el comportament del DDS. Amb aquesta tècnica es pot controlar l'activació o no de la sortida i efectuar canvis en els registres activats. Això permet establir modes de funcionament del circuit com si fos un Modulador. Cal habilitar el control extern a través de les comandes corresponents, abans de fer ús de les interrupcions.

Els modes de funcionament es configuren a través de les comandes i es poden executar indistintament a través de comandes o a per mitjà de les interrupcions, depenent de allò que ens interessi per cada cas i cada aplicació.

Els modes de funcionament són:

Mode ASK: [11] Aquest és el mode de modulació per desplaçament d'amplitud. Més concretament seria una modulació *OOK (On/Off Keying)* ja que el que fa es donar tota la modulació o res. A la Figura 2.1 es pot veure com el DDSBee realitza aquesta modulació digital. Per el nostre circuit aquest mode contempla l'us d'un terminal (pin 9), el qual al ser posat en estat alt habilita la sortida del registre de freqüència número 1. En cas de estar en estat baix, deshabilita la sortida del DDS.

Aquest mode també pot ser utilitzat simplement com a habilitador o deshabilitador de la sortida, per alguna aplicació que necessiti interrompre la sortida de manera controlada.

La limitació d'aquest mode està en la velocitat de commutació, seguint el terminal d'interrupció. La figura 2.2 mostra el retard que hi ha entre l'actuació d'un polsador i l'arrencada de la sortida. Com podem veure aquest temps és de 18.5us.

Mode FSK: El mode FSK implementa la modulació per desplaçament de freqüència. Això vol dir que utilitza dos freqüències diferents per indicar si el valor es un '1' o bé un '0'. En aquest mode, el DDSBee fa un intercanvi dels seus dos registres de freqüència interns. Els valors dels dos registres són fixats en el moment de establir el mode de funcionament. A la Figura 2.3 es pot veure com el DDSBee realitza aquesta modulació digital. A més dels mode FSK genèric s'han creat dues comandes per tal de programar de manera instantània dos modes FSK específiques que han sigut molt populars:

Mode Bell-202: [12] Aquest és un mode específic de modulació FSK. Bell 202 utilitza un to de 1200 Hz per indicar una marca (típicament un '1' binari) i 2200 Hz per indicar un espai (típicament un '0' binari). La Figura 2.3 correspon exactament a aquest mode FSK.

Mode Bell-103: [13] Aquest és un mode específic de modulació FSK. Bell 103 utilitza un to de 1270 Hz per indicar una marca (típicament un '1' binari) i 1070 Hz per indicar un espai (típicament un '0' binari).

Mode PSK: [14] PSK vol dir modulació per desplaçament de fase. Aquest és un esquema de modulació digital que transmet dades mitjançant el canvi, o la modulació de la fase d'un senyal de referència (l'ona portadora). En el DDSBee aquest mode requereix de tres paràmetres: freqüència portadora, desplaçament de fase 0 i desplaçament de fase 1. Donat que únicament tenim dos símbols a indicar en aquest mode (fase 0 o fase 1), el mode s'anomena 2PSK o també BPSK de *Binary Phase Shift Keying*.

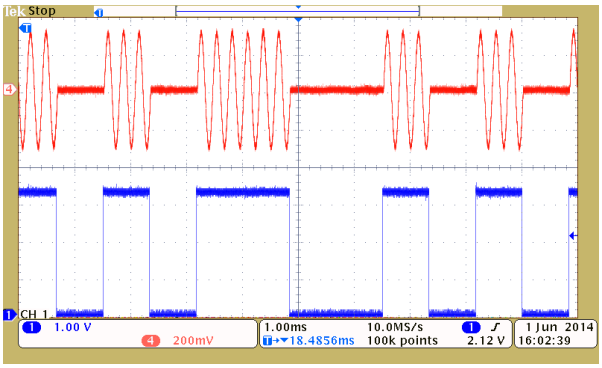


Figura 2.1.: Modulació ASK/OOK

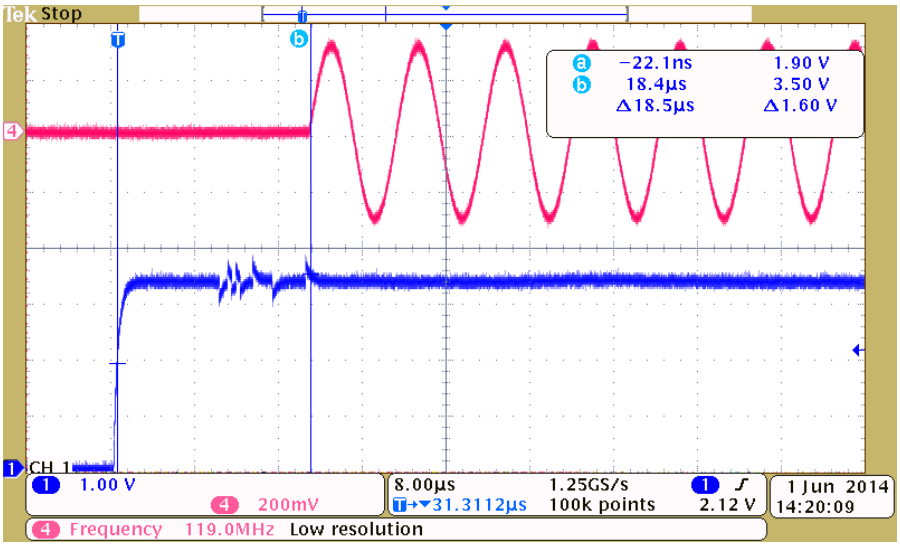


Figura 2.2.: Temps que triga en activar-se la sortida en mode ASK al accionar-se un polsador

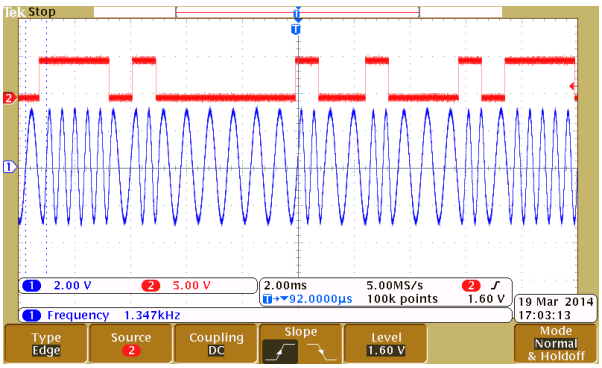


Figura 2.3.: Modulació FSK

3. Maquinari

3.1. El microcontrolador AVR

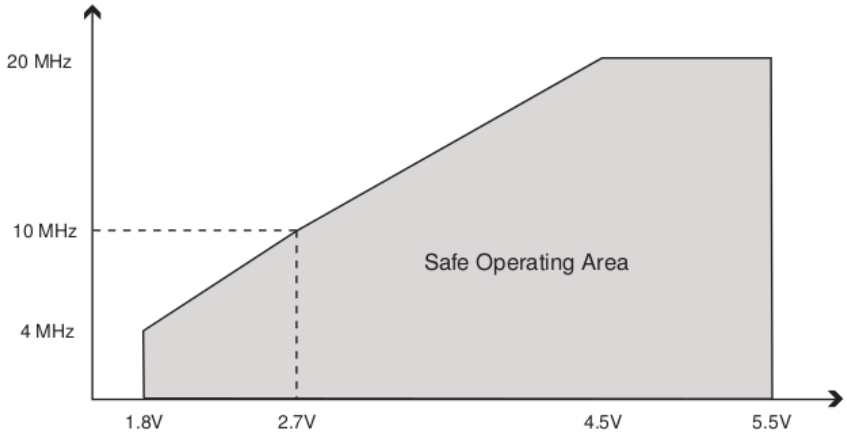
Els AVR's són una família de microcontroladors fabricats per Atmel, especialment dissenyats per a ser programats amb una compilació de codi C i obtenir un resultat el més optim possible. Existeixen diversos components dins d'aquesta família amb diferents capacitats de memòria RAM i FLASH, diferent amplada de bit, diferent nombre d'entrades, sortides i diversitat de tipus de perifèrics integrats.
Des de fa uns anys aquesta família s'ha popularitzat enormement per què han estat escollits per formar el nucli de la plataforma de desenvolupament de *hardware lliure Arduino*. Aquesta plataforma a portat la programació de microcontroladors a un gran públic que ara pot crear les seves pròpies aplicacions d'una manera ràpida i fàcil.

El dispositiu en que es plantejarà el present treball és el *Atmega328*.
Més concretament la seva versió en encapsulat TQFP (Thin Quad Flat Package) de 32 pins que ens permet la integració d'aquest dispositiu en una àrea petita de la superfície de la placa. A més aquest dispositiu ens ofereix un bon nombre de característiques que fan d'ell el dispositiu adequat per aquest projecte. Podem comentar algunes d'aquestes característiques:

- Disponibilitat d'espais de memòria diferents que permet una gran flexibilitat de gestió del programa: RAM per les dades d'execució, E2PROM per mantenir configuracions i FLASH per programa i constants del tipus cadena de caràcters.
- Molt bona capacitat de memòria: 2KBytes de RAM, 1KByte de E2PROM i 32Kbytes de Flash.
- Interfície de comunicació SPI. Disposa de 4 línies de gestió del Bus específiques: MOSI, MISO, CLK i SS (Veure l'apartat 3.1.2)
- Interfície de comunicació sèrie UART.
- Moltes possibilitats de generar interrupcions externes i gestió ràpida d'aquestes interrupcions.

3.1.1. Oscil·lador del microcontrolador

Segons les especificacions de la família AVR , la freqüència màxima tolerable de treball del microcontrolador està molt relacionada amb la tensió d'alimentació d'aquest. El nostre dispositiu està dissenyat per tal de poder funcionar en el rang de 3 fins a 5V. No obstant, degut a que el seu disseny està pensat per funcionar allà on un mòdul XBee pugui fer-ho. Donat això i tenint en compte que la nostra alimentació vindrà fixada per el que esperem tenir en els mòduls XBee, que serà de 3.3V, podem determinar la freqüència de treball basant-nos en la següent gràfica de la figura 3.1.1, extreta de la documentació tècnica del microcontrolador **??**:



Tal com es mostra a la figura, la corba de la freqüència màxima contra la tensió d'alimentació és lineal entre $1.8V < VCC < 2.7V$ i entre $2.7V < VCC < 4.5V$.

D'aquesta manera podem determinar que la freqüència màxima del microcontrolador en aquestes condicions serà la que indica l'equació 3.1

$$F_{max} = 10MHz + (3.3V - 2.7V) * \frac{20MHz - 10Mhz}{4.5V - 2.7V} = 13.33MHz \tag{3.1}$$

Per tal de trobar un oscil·lador estàndard de forma fàcil i que resulti econòmic, s'ha decidit utilitzar un oscil·lador ceràmic a 10MHz (MURATA - CSTCC10M0G53), amb bona tolerància i estabilitat en freqüència. L'elecció de aquest oscil·lador ens condiciona molts factors del projecte: com les velocitats de comunicació, tant les internes per SPI com les externes per UART. També ens condiciona la velocitat de resposta a interrupcions o el consum del propi circuit.

3.1.2. El Bus SPI

El bus SPI (de l'anglès *Serial Peripheral Interface*) és un estàndard de comunicacions, utilitzat principalment per a la transferència d'informació entre circuits integrats en equips electrònics. Inclou una línia de rellotge, dada entrant, dada sortint i un pin de selecció de xip, que connecta o desconnecta l'operació del dispositiu amb què desitja comunicar-se. D'aquesta manera, aquest estàndard permet compartir les línies de rellotge i de transmissió i recepció de dades [17].

SPI és un estàndard de bus sèrie creat per Motorola i compatible amb els productes de silici de diversos fabricants. Les interfícies de comunicació SPI estan disponibles en els processadors populars. Es tracta d'un enllaç de dades en sèrie síncron que opera a full duplex (els senyals de dades es mouen en ambdues direccions a la vegada).

Molts sistemes digitals tenen perifèrics amb els que es comuniquen a unes velocitats que no necessàriament han de ser elevades en excés. Els avantatges d'un bus sèrie, com el SPI, el I2C, el UART o d'altres, és que minimitza el nombre de terminal i la grandària dels circuit integrat que els utilitzen. Això redueix el cost de fabricar muntar i provar l'electrònica. Un bus de perifèrics sèrie és l'opció més flexible quan molts tipus diferents de perifèrics han de ser controlats. L'arquitectura consisteix en senyals de rellotge, data in, data out i xip select per a cada un dels circuits integrats que han de sers controlats. Quasi qualsevol dispositiu digital

pot ser controlat amb esta combinació de senyals. Els dispositius es diferencien en un número predicible de formes. Uns lligen la dada quan el rellotge puja altres quan el rellotge baixa. Alguns ho lligen en el flanc de pujada del rellotge i altres en el flanc de baixada. Escriure és quasi sempre en la direcció oposada de la direcció de moviment del rellotge. Alguns dispositius tenen dos rellotges. Un per a capturar o mostrar les dades i l'altre per al dispositiu intern.

Els avantatges més remarcables són:

- Comunicació Full Dúplex
- Major velocitat de transmissió que amb I2C o SMBus
- Protocol flexible en què es pot tindre un control absolut sobre els bits transmesos
- No està limitat a la transferència de blocs de 8 bits
- Elecció de la grandària de la trama de bits, del seu significat i propòsit
- La seva implementació en maquinari és extremadament simple
- Consumix menys energia que I2C o que SMBus degut que posseïx menys circuits (incloent les resistències pull-up) i estos són més simples
- No cal arbitratge o mecanisme de resposta davant de fallades
- Els dispositius esclaus usen el rellotge que envia el mestre, no necessiten per tant el seu propi rellotge
- No és obligatori implementar un transceptor (emissor i receptor), un dispositiu connectat pot configurar-se perquè només envie, només reba o ambdós coses al mateix temps
- Usa molts menys terminals en cada xip/connector que una interfície paral·lel equivalent
- Com a màxim un únic senyal específic per a cada esclau (senyal SS), els altres senyals poden ser compartides

Els desavantatges més remarcables són:

- Consumeix més terminals en cada xip que I2C, inclús en la variant de 3 fils
- El adreçament es fa per mitjà de terminals específiques (senyalització fora de banda) a diferència del que ocorre en I2C que se selecciona cada xip per mitjà d'una direcció de 7 bits que s'envia per les mateixes línies del bus
- No hi ha control de flux per maquinari
- No hi ha senyal d'assentiment. El mestre podria estar enviant informació sense que estiguera connectat cap esclau i no es donaria compte de res
- No permet fàcilment tindre diversos mestres connectats al bus
- Només funciona en les distàncies curtes a diferència de, per exemple, RS-232, RS-485, o Bus CA

Tant la SPI com I2C proporcionen un bon suport per a la comunicació amb els dispositius perifèrics lents que s'accedeix de forma intermitent. Les EEPROM i els rellotges de temps real són exemples d'aquests dispositius. Però SPI s'adapta millor que I2C per a aplicacions que són considerades com fluxos de dades (a diferència de la lectura i escriptura sobre adreces de llocs en un dispositiu esclau). Un exemple estàndard d'una aplicació és la comunicació de dades entre el microprocessador o processadors digitals de senyals. Una altra és la transferència de dades de convertidors analògic a digital.

SPI també pot arribar a velocitats de dades significativament majors que I2C. Les interfícies compatibles amb SPI, solen anar a desenes de megahertzs. SPI realment guanya major eficiència en les aplicacions que s'aprofiten de la seva capacitat duplex, com ara la comunicació entre un çòdec"(codificador-descodificador) i un processador de senyal digital, que consisteix a enviar simultàniament mostres d'entrada i sortida.

3.1.3. EL Bus SPI en funcionament

Els dispositius SPI es comuniquen mitjançant una relació Master-Slave. A causa de la seva falta d'un sistema d'adreçament integrat en el dispositiu de fàbrica, SPI requereix més esforç i més recursos de maquinari que I2C, quan es tracta de treballar amb més d'un esclau. Però SPI tendeix a ser més simple i més eficient que el I2C en una configuració de aplicacions punt a punt (de mestre únic, a un sol esclau). Això és degut a una única raó: la manca d'adreçament de dispositius significa menys despeses. Aquest és el nostre cas.

Com s'ha comentat, els dispositius es comuniquen utilitzant una relació Master-Slave, en la qual el mestre inicia la trama de dades. Quan el mestre genera un rellotge i selecciona un dispositiu esclau, les dades poden ser transferits a qualsevol o en totes dues direccions simultàniament. De fet, pel que fa a SPI, les dades es transfereixen sempre en ambdues direccions. Depèn dels dispositius mestres i esclaus per saber si un byte rebut és significatiu o no. Així, un dispositiu ha de descartar el byte rebut en un mode de "Sols escriptura"o generar un byte fals en un mode "Sols lectura".

SPI Especifica quatre senyals:

- clock (SCLK)
- master data output, slave data input (MOSI)
- master data input, slave data output (MISO)
- slave select (SS)

En 3.1 podem veure aquests senyals en configuració d'un únic esclau. Un dispositiu esclau està seleccionat quan el mestre afirma el seu senyal de CSS.

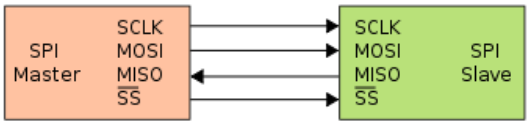


Figura 3.1.: Connexió Master-Slave Simple

Si hi ha diversos dispositius esclaus, l'amo d'esclaus genera un senyal de selecció per separat per a cada esclau. Aquestes relacions es mostren en 3.2.

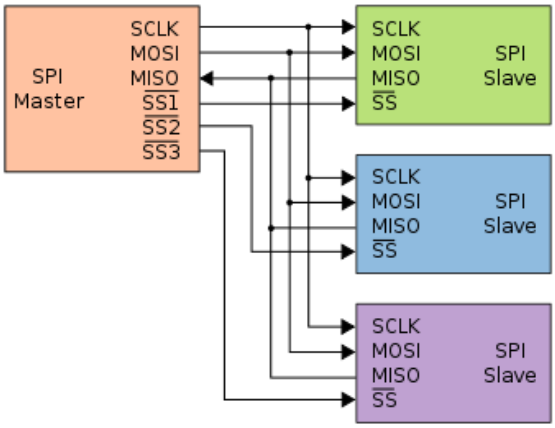


Figura 3.2.: Connexió Master amb tres cmponents esclaus

El mestre genera senyals de selecció d'esclaus amb entrades entrades/sortides de propòsit general o qualsevol altra lògica. Consisteix en l'antiga tècnica de bit-banging i pot ser molt sensible. S'ha de temporitzar en relació amb les altres senyals i assegurar, per exemple, que no canviarà la línia de selecció a la meitat d'un frame.

Si bé SPI no descriu d'una manera específica els sistemes de diversos mestres, alguns dispositius compatibles SPI disposen d'alguns senyals addicionals que fan que les implementacions d'aquest tipus sigui possible. No obstant això, és complicat i innecessàri en general, pel que no es fa sovint.

Un parell de paràmetres anomenat polaritat del rellotge **(CPOL)** i la fase del rellotge **(CPHA)** determinar els flancs del senyal de rellotge en el qual les dades són extrets i mostrejats. Cada un dels dos paràmetres té dos estats possibles, el que permet quatre combinacions possibles, *totes les quals són incompatibles un amb l'altre*. Així que un parell de mestre / esclau ha d'utilitzar els mateixos valors de paràmetres de parell per comunicar-se. Si s'utilitzen diversos esclaus que es fixen en diferents configuracions, el mestre haurà de reconfigurar-se a si mateix cada vegada que necessiti comunicar-se amb un esclau diferent.

3.1.4. En una capa superior

SPI no té un mecanisme de reconeixement per confirmar la recepció de les dades. De fet, sense un protocol de comunicació, el mestre SPI no té coneixement de si un esclau existeix. SPI tampoc ofereix control de flux. Si es necessita un control de flux per maquinari, es pot fer alguna cosa fora de la SPI.

Els esclaus poden ser considerats com dispositius d'entrada / sortida del mestre. SPI no especifica un determinat protocol d'alt nivell per al diàleg mestre-esclau. En algunes aplicacions, un protocol d'alt nivell no és necessari i només les dades en brut s'intercanvien. Un exemple d'això és una interfície a un còdec simple. En altres aplicacions, un protocol de nivell superior, com ara un protocol de comandament-resposta, pot ser necessari. Recordeu que el mestre ha d'iniciar els *frames*, tant per a la seva comanda, com per la resposta de l'esclau.

Tant SPI com I2C ofereixen un bon suport per a la comunicació amb dispositius de baixa velocitat, però SPI s'adapta millor a aplicacions amb dispositius de transferència *streams* de dades.

La capacitat de les comunicacions full-duplex amb SPI i les taxes de dades (que van fins a diversos megabits per segon) constitueixen, habitualment, una estructura de control de les sol·licituds individuals d'esclaus extremadament simple i eficient per el cas de mestre únic. D'altra banda, pot ser difícil per aplicar més d'un esclau, per la seva falta de incorporat en el tractament i la complexitat només creix a mesura que augmenta el nombre d'esclaus.

Lluny de ser només un simple "port de bytes", SPI és sovint una solució elegant per a les necessitats de comunicació modestes. També es pot fer servir com una plataforma a partir de la qual crear protocols de nivell superior.

Des de el punt de vista del nostre projecte, el protocol SPI està infrautilitzat, ja que el circuit DDS utilitzat amb el que el microcontrolador es comunica, no té sortida de dades. Per això el bus SPI estarà treballant sempre usant la línia MOSI. La línia MISO queda desconnectada per la comunicació amb el DDS.

3.1.5. Modes de Funcionament de SPI

Existeixen quatre modes de funcionament del Bus SPI, d'acord amb la configuració de la senyal de rellotge *(CLK)* pel que respecta a la seva fase i la seva polaritat en relació a les dades serie.

Aquests quatre modes de funcionament queden resumits en la figura 3.3. Tal com es veu a la figura 3.3, el senyal $\overline{SS}$ es generat per el dispositiu mestre (MASTER) i selecciona a l'esclau (SLAVE)com a dispositiu perifèric amb el qual es treballarà.

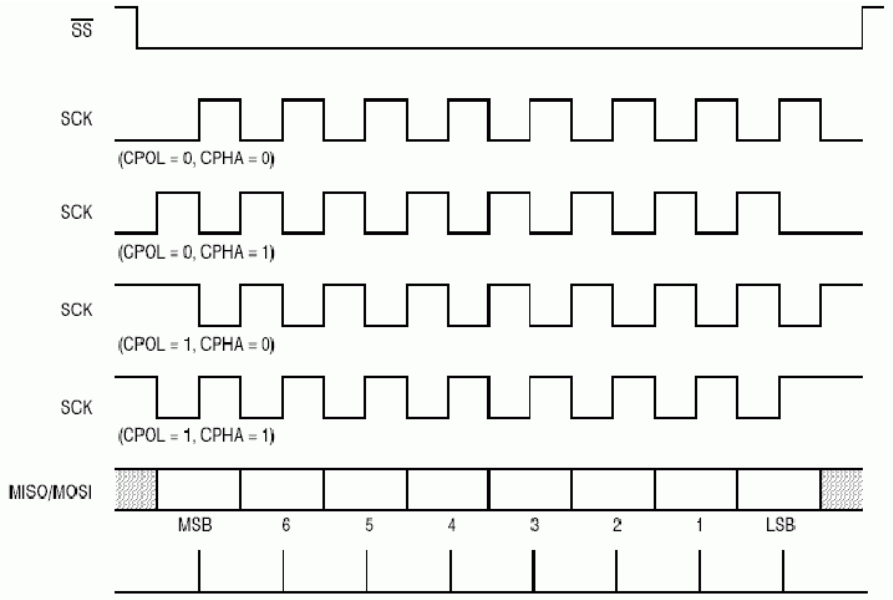


Figura 3.3.: Modes de funcionament del Bus SPI

A partir de aquí tenim els quatre modes de treball resumits en la taula.

	CPOL=0	**CPOL=1**
CPHA=0	Estat Repòs=0 Flanc Repòs a Actiu	Estat Repòs=1 Flanc Repòs a Actiu
CPHA=1	Estat Repòs=0 Flanc Actiu a Repòs	Estat Repòs=1 Flanc Actiu a Repòs

3.1.6. Programació del microcontrolador

Per tal de descarregar el programa compilat cap a el nostre dispositiu s'utilitzarà també el Bus SPI. S'ha previst en el disseny de placa l'accés al bus intern a través de terminals del dispositiu. Concretament són necessaris els terminals que s'especifiquen en la taula 3.1.6

Pin DDSBee	**Denominació**
1	VCC
5	RESET
6	SCK
7	MISO
8	MOSI
10	GND

Aquests pins en el DDSBee es connecten, per mitjà d'un adaptador per poder utilitzar un programador ISP, del tipus AVRISP, STK500 o similar. El connexionat que utilitzen aquests programadors és el que s'esmenta a la figura 3.4.

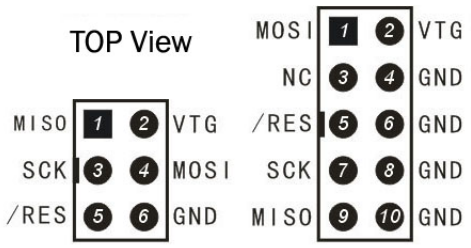


Figura 3.4.: pinout habitual per els programadors ISP

3.2. El DDS

3.2.1. Introducció al DDS

Les sigles *DDS* volen dir *Síntesi Digital Directa* (del anglès Direct Digital Synthesis). Un DDS és un generador de forma d'ona al qual se li pot variar la freqüència. És a dir, es un oscil·lador variable. Els oscil·ladors DDS són elements encara força desconeguts en el món de l'enginyeria en general.

Explicat de forma breu, la principal diferència entre el *DDS* i altres oscil·ladors (per exemple els PLL) és que en els primers la forma d'ona de sortida (generalment un sinus) està emmagatzemada en una memòria ROM interna. Si la ROM conté una forma d'ona sinusoïdal aquesta també es pot anomenar SIN ROM. L'accés a les dades en la memòria ROM es realitza a través d'un comptador que adreça la memòria a partir d'una senyal de rellotge (CLK_{Master}). Aquest comptador proporciona la informació digital de fase a través de l'adreçament de la ROM. Les dades de la ROM, que conte la forma d'ona, són portades a un convertidor Digital a Analògic (DAC), de manera que es produeix un tipus d'efecte *"sampling"* de la forma d'ona que s'extreu per la sortida.

Aquesta és una visió molt simple del DDS. Amb la tècnica plantejada per canviar el valor de la freqüència seria necessari canviar la freqüència del rellotge d'entrada CLK_{Master} o reprogramar la memòria ROM. Això ens plantejaria nous problemes. Per exemple com controlar la freqüència del rellotge d'entrada de manera precisa en un rang ampli de treball.

Si introduïm un *Acumulador de Fase* dins del control digital, l'arquitectura es transforma en un Oscil·lador Controlat Numèricament (o NCO), que és la base de la tècnica de DDS. L'acumulador de fase permet determinar de quina manera es llegeixen els valors ubicats en la memòria ROM i d'aquesta manera es pot modificar la freqüència de sortida proporcionada. Això es fa comparant el rellotge i el valor de la freqüència desitjada per incrementar el registre de fase.

Si suposem que s'adreça cada un dels valors ubicats en la memòria ROM. Això produeix la mínima freqüència, ja que en cada clock del rellotge CLK_{Master} es llegeix cada una de les dades. En canvi si únicament accedim a dos valors de la memòria, obtenim la màxima freqüència possible per aquell dispositiu, que serà:

$$F_{max} = \frac{CLK_{Master}}{2} \tag{3.2}$$

Aquesta tècnica permet tenir una reconstrucció de la forma d'ona molt precisa a on, de forma general, la seva precisió depèn només de la profunditat de bits en que s'ha emmagatzemat aquesta forma i no depèn de la freqüència que es vol obtenir. Això possibilita un disseny del

filtrat posterior molt més simple i efectiu.
[20] [19] [22]

Tot i la limitació de la equació 3.2, existeix la possibilitat de treballar amb algun dels àlies que es poden generar i que proporcionarien freqüències per sobre de la freqüència nominal. El DDS proveeix d'una sortida mostrejada, l'espectre del qual conté la suma de la freqüència fonamental de sortida més freqüències imatge (o àlies) que apareixen en múltiples de la freqüència de rellotge i de la suma de freqüència de sortida. Podem veure a la Figura 5.3 la representació d'aquest espectre [18].

Per exemple, si es disposa d'un DDS amb un rellotge de sistema de 120 MHz, el seu límit màxim de freqüència estaria en 60 MHz. Si es desitja obtenir a partir d'aquest un rang de freqüències entorn als 220MHz. necessitaríem un filtre a la sortida del DDS sintonitzat en aquesta freqüència i generar freqüències al voltant dels 20MHz. D'aquesta manera s'obte que la tercera freqüència imatge ens cau dins del rang del filtre, es a dir:

$$2f_c - f_{out} = 2 * 120 - 20 = 220MHz \tag{3.3}$$

Així l'usuari pot estendre el rang de treball del circuit.

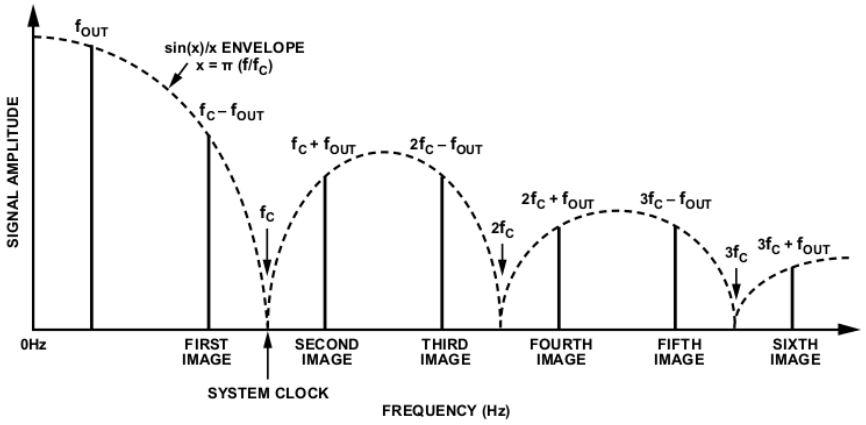


Figura 3.5.: Espectre de sortida del DAC

3.2.2. El DDS AD9833 de Analog Devices

El DDS que s'utilitza en el present projecte és el *AD-9833*[21] del fabricant de circuits integrats electrònics *Analog Devices*, per la seva simplicitat de terminals i components externs. El seu *data sheet* s'inclou com annexe en la secció A.4 per el seu interès. Seguidament es mostren les característiques més rellevants:

- Fase i freqüència programables digitalment.
- Baix consum: únicament 12.65mW a 3V d'alimentació
- Rang de freqüències de 0 a 12,5MHz
- 28 bits de resolució en freqüència proporciona una resolució millor de 0.1 Hz a 25MHz de senyal de rellotge (millor de 0.004 Hz a 1MHz)

- Formes d'ona Sinusoïdal i triangular amb una amplitud de $610mV_{pp}$ i senyal quadrat amb amplitud de V_{cc}
- Ampli rang d'alimentació de 2.3 fins a 5.5V
- No requereix de components externs.
- Interfície SPI amb 3 terminals (FSYNC, SCLK i SDATA).
- Ampli rang de temperatura : $-40C$ fins $+105C$.
- Opció de apagat.
- Encapsulat MSOP (Micro Small Outline Package) de 10 terminals.
- Qualificat per aplicacions en el sector de l'automoció.

3.2.3. Programació

El DDS AD-9833, al igual que altres DDS de la mateixa família [1], una vegada s'alimenta ha de ser resetejat, si es vol que es mantinguin a zero els registres apropiats . Per evitar sortides espúries del Convertidor Analògic a Digital mentre s'està inicialitzant, convé mantenir el bit de RESET a 1 mentre el component es prepara per començar a generar una sortida.
El bit de RESET no posa a zero la fase, ni la freqüència ni el registre de control. Aquests registres són susceptibles de contenir dades no vàlides i, per tant, han de ser posades a un valor conegut. Aleshores, el bit de RESET ha de ser posat a zero per tal de començar a generar la sortida. La sortida apareixerà vuit cicles de MCLK després de posar el RESET a 0.

Seguidament veiem un exemple de programació que il·lustrarà el funcionament de cada un dels registres implicats.

Suposem que volem programar una freqüència de sortida de 1562.5Hz i el AD9833 està connectat a un oscil·lador de 25MHz, que serà el senyal de rellotge en els aquest projecte. El codi que cal programar en el registre de freqüència està donat per la formula 3.4

$$FreqReg = \frac{f_{OUT} * 2^{28}}{f_{MCLK}} \tag{3.4}$$

En el nostre exemple això equival a dir:

$$FreqReg = \frac{1562.5 * 2^{28}}{25 * 10^6} \tag{3.5}$$

$$= 16777\,decimal = 4189\,Hex = 0100\,0001\,1000\,1001\,bin \tag{3.6}$$

Donat això, la seqüència que cal transmetre cap a el DDS és:

[1] Per exemple AD9837, AD9834 i AD9838

Hexadecimal	binari
0x2100	0010 0001 0000 0000
0x4189	0100 0001 1000 1001
0x4001	0100 0000 0000 0001
0xC000	1100 0000 0000 0000
0x2000	0010 0000 0000 0000

La descripció en detalla dels valors que es transmeten en el bus SPI és la següent:
0x2100-Registre de Control

- DB13 es posa a 1. Això permet la carrega completa del registre de Freqüència en dos cicles de escriptura consecutius. La primera escriptura conté els 14 LSB's i la segona els 14 MSB's.
- El bit de RESET (DB8) es posat a 1. Aquesta acció posa a zero registres interns que correspon a una sortida del DAC a mitja escala.

0x4189 -Registre de Freqüència 0 LSB

- DB15 i DB14 són establerts a 0 i 1, respectivament, el que indica la adreça del registre de freqüència 0
- Els 14 bits restants corresponent als 14 bits MENYS significatius de la dada. 0x0189 = 00 0001 1000 1001

0x4001-Registre de Freqüència 0 MSB

- DB15 i DB14 són posats a 0 i 1, respectivament, el que indica la adreça del registre de freqüència 0
- Els 14 bits restants corresponent als 14 bits MES significatius de la dada. Això equival a posar el bit DB14 de la paraula obtinguda en la formula 3.5 com a primer bit (bit 0) del registre de les MSB's. 00 0000 0000 0001

0xC000-Registre de Fase 0

- DB15, DB14 i DB13 són carregats amb 110. DB12 no importa el valor. Que és l'adreça per el Registre de Fase 0.
- La resta dels 12 bits estan posats a 0 en aquest cas.

0x2000-Surt de Reset

- Extingim la senyal de RESET i apareixerà la senyal en el DAC.

4. Programa

En aquest apartat es descriurà el funcionament general de la programació realitzada. Els programes s'han realitzat per complet amb llenguatge de programació *C*. S'han dividit les funcionalitats en diferents mòduls organitzats de forma jeràrquica des de el punt de vista de la seva interacció amb el hardware. En els nivells més baixos es troben els mòduls encarregats de la gestió de registres i d'altres estructures simples. Mentre es va pujant de nivell existeix cada vegada més una virtualització de les accions a més baix nivell, tot mantenint el major grau d'independència entre mòduls no adjacents. D'aquesta forma s'obté una estructura compacta alhora que molt versàtil.

Per més detall s'han afegit en els annex de la present memòria A.1, els llistats complets dels fitxers font dels mòduls utilitzats per la programació del dispositiu presentat.

4.1. Distribució de mòduls

A la figura 4.1 es pot observar la distribució i relació que hi ha entre els diferents mòduls que interactuen en aquest projecte.

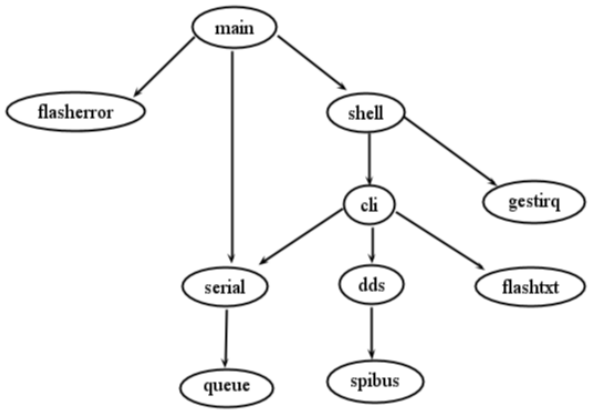


Figura 4.1.: Distribució i relació entre mòduls

Els mòduls que intervenen són:

main Aquest mòdul té la finalitat de inicialitzar el sistema i esperar comandes des de el port sèrie.

flasherror Aquest mòdul té la funcionalitat de deixar lliure la memòria SRAM del AVR, forçant a mantenir les cadenes de text constants dels missatges d'error, en memòria de programa Flash. Dins del mòdul s'implementen les definicions necessàries i les funcions macros, per tal de que l'usuari pugui cridar a determinades funcions i adreces de memòria Flash de

forma simple, obviant el fet que es tingui d'utilitzar l'accés específic cap a aquest espai de memòria.

serial Aquest mòdul és l'encarregat de configurar apropiadament i fer l'accés al port sèrie.

queue Aquest mòdul implementa una cua per fer les transferències a través del port sèrie UART.

shell Aquest mòdul abstrau el concepte de interfície bàsica, fent la inicialització de les variables que han de contenir les comandes i per altra banda el pas de comandes cap a el mòdul que ha d'interpretar-les.

cli Aquest és el mòdul de interfície per línia de comandes, Aquí s'ha implementat la funció que ha d'executar les comandes, així com totes les funcions-comanda que pot executar el dispositiu, incloent-hi comandes de programació, de comportament, d'ajut i d'altres. El mòdul inclou un mecanisme per mantenir ocultes determinades funcions. Aquest fet és especialment útil per disposar d'un conjunt d'instruccions d'ús avançat, de test, de control o bé de debugging. Aquestes funcions poden estar o no documentades per tal de mantenir-ne la privacitat del seu ús per part dels usuaris estàndards.

flashtxt Aquest mòdul té la funcionalitat de facilitar la programació de noves funcions-comanda i deixar lliure la memòria SRAM del AVR, forçant a mantenir les cadenes de text constants, en memòria de programa Flash. Dins del mòdul s'implementen les definicions necessàries i les funcions macros, per tal de que el sistema pugui cridar a determinades funcions i adreces de memòria Flash de forma simple, obviant el fet que es tingui d'utilitzar l'accés específic aquest espai de memòria.

dds En aquest mòdul estan definides les comandes especifiques i genèriques amb que treballa el circuit oscil·lador DDS.

spibus Aquest mòdul implementa les funcions de gestió i transferència bàsiques necessàries per realitzar la comunicació en un bus SPI. El mòdul conté funcions de lectura i escriptura, tant en mode Master com en mode Slave, tot i que en aquest dispositiu únicament són usades les instruccions de escriptura en mode Master, ja que la comunicació que es realitza és en tot moment de forma unidireccional, enviant comandes des de el dispositiu Mestre (microcontrolador) cap a el dispositiu Esclau (DDS).

gestirq Dins del mòdul *gestirq* es porta a terme la inserció de funcions en els vectors d'interrupcions i la habilitació i desactivació de les interrupcions. Les interrupcions externes juguen un paper bàsic en l'ús d'aquest dispositiu en sistemes de comunicació, ja que permet els canvis de registres de forma àgil de manera externa, donant lloc a modulacions per interrupció de portadora, en fase o bé en freqüència.

4.2. Descripció de Programa: Procés d'arrancada

En el moment de posada en marxa del dispositiu, aquest realitza la funció de configuració de tots els mòduls. A la figura 4.2 es mostra el diagrama del procés d'inicialització. Seguidament tenim la descripció d'aquest procés.

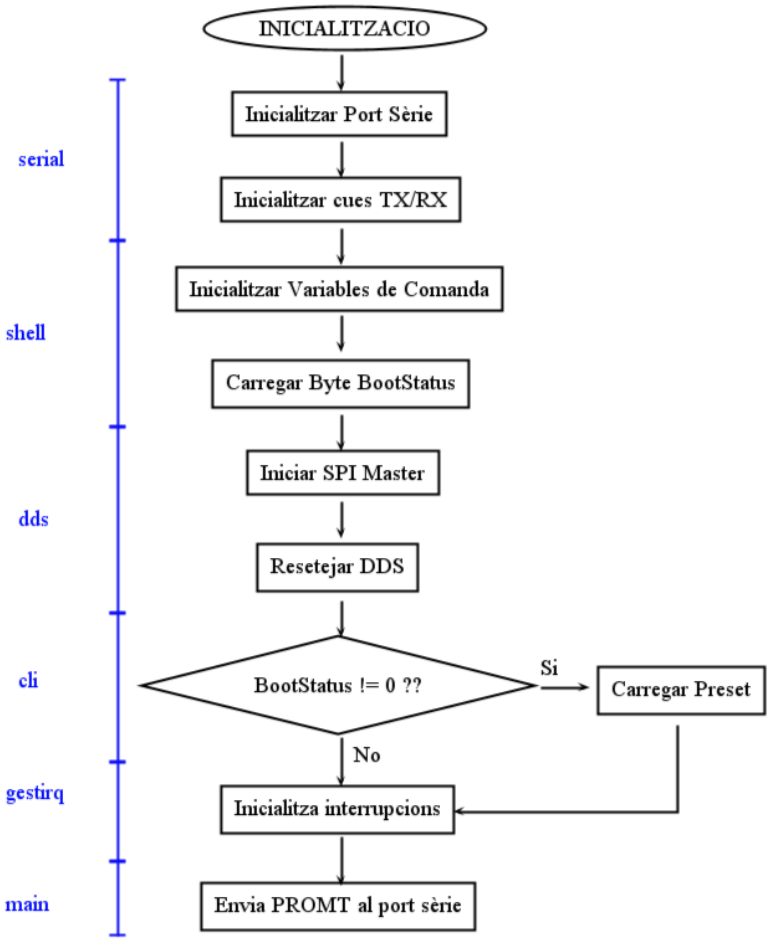


Figura 4.2.: Diagrama del procés d'inicialització

En primer lloc, el proces d'arrancada inicialitza el port sèrie amb els paràmetres de comunicació adequats que, en principi són a 115200 bd,. 8 bits de dades, sense paritat i 1 bit de stop (115200, 8, N, 1). Els valors necessaris per configurar el registre de velocitat són calculats en el moment del pre-processat, gràcies a l'ús del mòdul *util/setbaud* [1]. La part encarregada de configurar aquests paràmetres és la que és mostra a continuació:

```
    UBRR0H = UBRRH_VALUE;
    UBRR0L = UBRRL_VALUE;
    UCSR0A = 32;
#if USE_2X
    UCSR0A |= (1 << U2X0);
#else
    UCSR0A &= ~(1 << U2X0);
#endif
```

També cal reservar l'espai per les cues de dades de la transmissió i la recepció de dades per el port sèrie. Aquestes cues són implementades per el mòdul *queue*, però des de el mòdul sèrie es criden les cues que intervenen en la comunicació.

Seguidament es reserva l'espai per allotjar les variables de comanda. Aquestes variables són

les que emmagatzemaran el nombre de arguments capturats pel port sèrie i els propis arguments incloent la comanda mateixa com a argument 0.

Per tal de poder arrancar el dispositiu en la seva configuració per defecte o bé amb algun preset pre-carregat, es necessari consultar l'estat del Boot d'arrancada del propi dispositiu, que es guardarà en la posició de memòria EEPROM senyalada per BOOTVAR_BYTE.

Després cal reiniciar el DDS, així que abans cal configurar el bus SPI en mode Master. El Bus SPI es configura per tal de poder treballar a la màxima velocitat disponible. El fet de poder treballar com més ràpid millor ens ha de permetre disminuir al màxim el temps de canvi entre esdeveniments que afectin al DDS. Per exemple l'activació de la sortida o el canvi d'un registre per un altre.

Per el AVR la màxima velocitat de transmissió de dades per SPI és una quarta part de la freqüència del seu rellotge. Durant les proves de desenvolupament el SPI s'ha fet anar a 4MHz, mentre que en la aplicació final la velocitat màxima que es pot obtenir és de 2.5MHz de rellotge de SPI.

La inicialització del mòdul *cli* s'encarrega de comprovar l'estat del *Boot Status*, prèviament carregat des de la memòria EEPROM. Si aquest determina que s'ha de carregar una inicialització prèvia, es recuperen els valors emmagatzemats en memòria EEPROM i programa els registres del DDS. En cas que no es tingui de carregar cap preset d'usuari, el programa continua endavant, deixant el DDS amb el Reset realitzat prèviament.

Seguidament cal configurar l'estat inicial de les interrupcions que s'utilitzaran i habilitar les interrupcions generals del sistema.

Finalitzant el procés de configuració, només cal enviar el PROMT del sistema pel port sèrie, indicant que tot està a punt per funcionar. En arribar a aquest punt el programa queda a l'espera.

4.3. Descripció de Programa: Interpretació de comandes

Una vegada el dispositiu s'ha inicialitzat, aquest entre en estat de espera de comandes per tal de ser interpretades. El procediment per realitzar la interpretació de comandes és el que es detalla en el diagrama de la figura 4.3. Seguidament tenim la descripció d'aquest procés.

El programa principal del dispositiu està a l'espera de capturar missatges per el port sèrie del dispositiu. Aquests missatges han d'estar finalitzats amb una marc especial (centinella), que en el nostre cas serà el caràcter no imprimible de retorn de línia (\n). Una vegada el procés de captura de missatges té una cadena de caràcters finalitzada amb el centinella, delega el procés de interpretar la cadena a les capes inferiors i espera que les capes inferiors li retornin un resultat. El resultat serà interpretat com a missatge d'error de les capes inferiors. d'aquesta manera podem tenir diferents codis d'error:

error: 0 Ens diu que no hi ha hagut cap error en les capes de interpretació i execució de les comandes inferiors. També retorna aquest codi en cas d'enviar una cadena buida

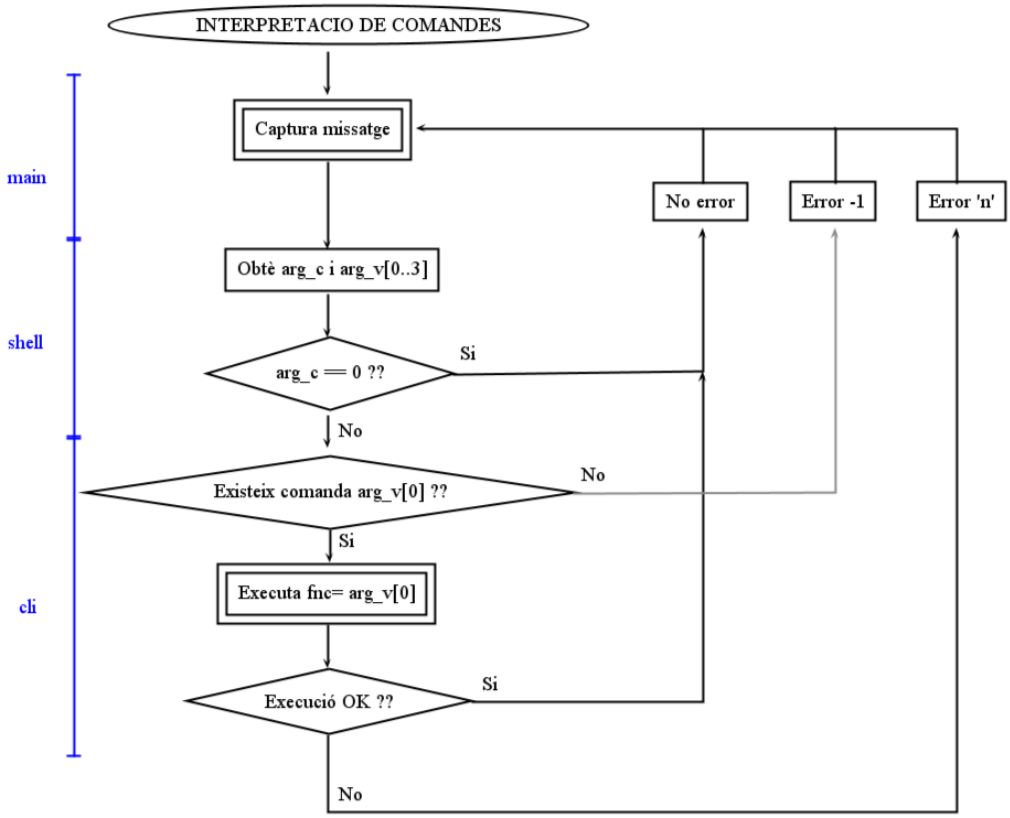


Figura 4.3.: Diagrama del procés d'interpretació de comandes

a la capa inferior (mòdul *shell*). Aquesta situació no retorna cap missatge a l'usuari, únicament presenta un nou PROMT.

error: -1 Aquest missatge ens diu que la comanda que s'ha passat del mòdul *shell* al mòdul *cli* no està dins del repertori de les comandes programades. Aquesta situació retorna a l'usuari el missatge *"Unknown command"*.

error: -2 El missatge ens diu que el nombre de paràmetres que s'han passat a la funció dins del mòdul *cli* no coincideix amb l'esperat. Això pot implicar tant una mancança de paràmetres com un excés dels mateixos. Aquesta situació retorna a l'usuari el missatge *"Wrong parameters"*.

error: -3 Aquest codi d'error informa que almenys un dels arguments que acompanyen a la comanda està fora del rang definit per la funció específica. Aquesta situació retorna a l'usuari el missatge *"Out of range"*.

error: -4 El error informa que hi ha un paràmetre no permès, ja que intenta accedir a un espai de memòria EEPROM no habilitat. Aquesta situació retorna a l'usuari el missatge *"Not allowed"*.

La traducció humanament llegible dels codis d'errors es realitza a través de missatges predefinits dins del mòdul *flasherror*.

La funció *process_command*, dins del mòdul *shell*, pren com a origen la cadena de caràcters lliurada per la capa superior i la tokenitza, per obtenir el nom de la comanda, els seus arguments i fa un recompte de quants elements a obtingut en total. Aquestes dades són deixades en les variables globals arg_v[ ] i arg_c.

En el cas que el comptador d'elements *arg_c* sigui zero, la funció *process_command*, retorna el control a la funció de captura de missatge. En cas contrari realitza una crida a la funció *exec_command*, ubicada en el mòdul *cli*.

La funció *exec_command* pren la cadena de caràcters ubicada en *arg_v[0]* i la compara amb les cadenes de caràcters de les funcions programades. Aquestes cadenes de caràcters amb les que realitza la comparació estan ubicades en memòria flash de programa. Per gestionar aquestes cadenes, existeix el mòdul *flashtxt*, que conté les variables i funcions macro per poder accedir a les àrees de programa per llegir i comparar dades.

Si la funció *exec_command* no troba coincidència amb la comanda rebuda, aleshores retorna el control a la funció que l'ha cridat, passant-l'hi el codi d'error número -1. Encas que si que trobi coincidència, *exec_command* comprova que existeixi una funció associada a la comanda i executa aquesta funció associada, tal com es mostra en el següent fragment de codi:

```
int8_t exec_command(const uint8_t * command) {
  /* Funció que rep un apuntador a una cadena de
     caracters i explora totes les comandes integrades
     en el CLI. Si existeix una funció associada
     a la comanda entrada, la executa inmmediatament.
     En cas contrari retorna el error (-1) per marcar
     la funció com desconeguda.  */

  uint8_t i;
  if (*command==0) return 0;
  for (i=0; i<NUM_CMNDS; i++){
    if (!CMP_F_CMND(command, i))
      {
  if (commands[i].fnc) {
    return commands[i].fnc();
  }
      }
  }
  return -1;
}
```

En el cas que existeixi la funció relativa a la comanda i aquesta funció sigui executada, el primer que fa la funció, en els casos que és necessari, és comprovar que el nombre d'arguments de que disposa associats a la comanda són exactament els que necessita. Això ho fa comprovant el contingut de la variable *arg_c*. Si la funció no disposa del nombre necessari de paràmetres, retorna el control a la funció *exec_command* amb el codi d'error corresponent (-2).

Si el nombre de paràmetre són suficients però almenys un d'ells està fora de rang també retorna el control cap a la funció que l'ha cridat, sense haver finalitzat l'execució de la funció i amb el codi d'error número -3.

Si la comanda sol·licitada fa accessos a la memòria EEPROM, ja sigui de lectura o escriptura, es comprova que aquest accés sigui a un espai de memòria permès. Actualment s'han definit únicament 3 espais de memòria numerats del 1 al 3 i habilitats per ser utilitzats en presets d'usuari. Si l'accés s'intenta fer fora del marge permès es produeix el error -4 i queda avortada l'execució de la instrucció.

Si finalment la funció s'executa correctament, aquesta retorna un codi d'error zero (sense cap error). Aquest valor és mogut des de les capes inferiors fins al mòdul *main* on serà interpretat i donarà la funció com a executada i mostra el PROMPT de programa.

4.4. Descripció de Programa: Executar funcions del DDS

Aquest apartat defineix com és l'execució de les funcions del DDS, que són aquelles cridades per comandes d'usuari i que afecten directament al contingut de almenys un dels registres interns del circuit integra DDS. Aquestes funcions responen al diagrama de la Figura 4.4

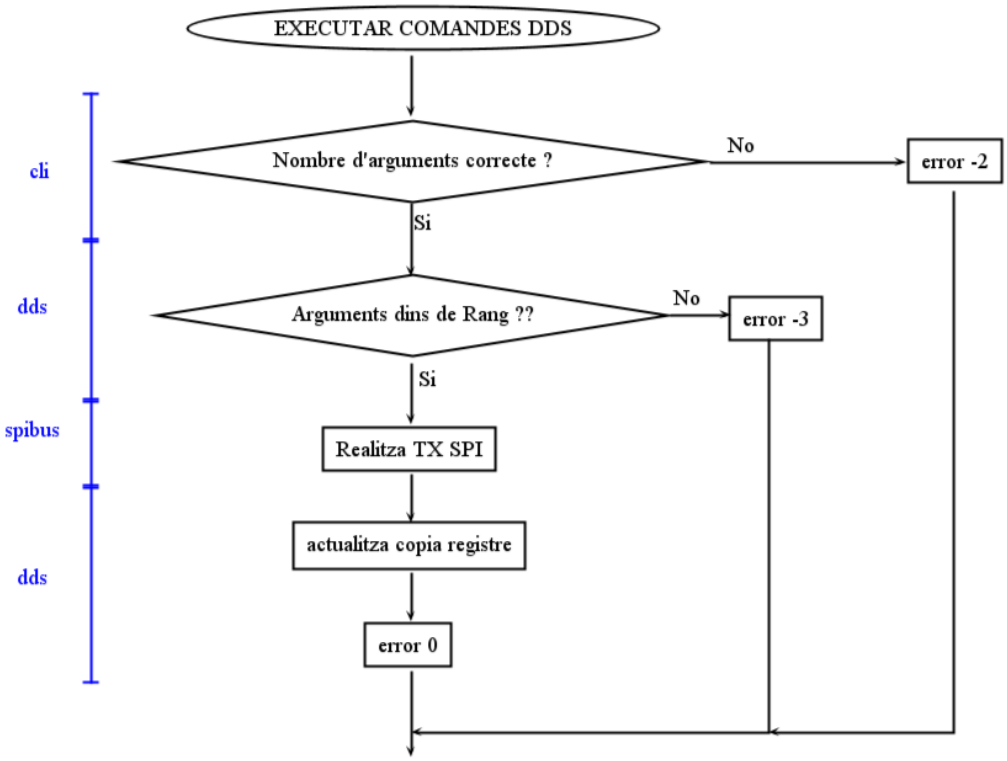


Figura 4.4.: Diagrama del procés d'execució de funcions del DDS

Quan arriba una d'aquestes comandes al mòdul *cli* i la funció *exec_command* delega la execució d'aquesta a la corresponent funció associada, el primer que fa aquesta es comprovar si el nombre de paràmetres lliurats és el correcte. Si la funció no disposa del nombre necessari de paràmetres, retorna el control a la funció *exec_command* amb el codi d'error corresponent.

Si no, la funció delega sobre la corresponent funció dins del mòdul *dds* l'execució. Dins d'aquest mòdul es comprovarà la validesa dels paràmetres (si n'hi ha). Si almenys un d'ells està fora de rang retorna el control cap a la funció que l'ha cridat en el mòdul *cli*, retornant codi d'error número -3.

Si la funció pot ser executada, formata adequadament les dades per el bus SPI i transfereix la informació al DDS a través d'aquest bus. En el moment de fer-s'he la transferència de dades, es realitza una copia del que s'ha enviat al registre dins d'una variable. Això es necessari ja que el circuit DDS no te cap mecanisme per permetre la lectura directe dels seus registres. Per el correcte funcionament del nostre dispositiu es necessari conèixer en tot moment el contingut dels registres del DDS i per tant s'han de realitzar copies d'aquest registre.

4.5. Descripció de Programa: Configurar els terminals externs

El dispositiu d'aquest projecte permet manipular els registres, sense haver de connectar-se al port sèrie. Això es possible si prèviament s'han configurat els terminals externs del dispositiu. D'aquesta manera es pot aconseguir commutar entre dos registres del DDS o deshabilitar-ne la sortida.

Per realitzar aquestes funcions de manera el més ràpida possible i asincrónicament, es necessari configurar els terminals per acceptar interrupcions externes i configurar les rutines del servei d'interrupció (ISR) associades a cada terminal. Si l'assignació de funcionalitats diferents en una mateixa ISR es realitza de forma dinàmica, aleshores es possible donar múltiples usos a un mateix terminal extern. Això pot permetre canviar la funció programada, sense la necessitat de refer el connexionat extern.

El diagrama de la figura 4.5 mostra el procés que s'utilitza per fer això.

Abans de descriure el diagrama de la figura 4.5 cal entendre la estratègia seguida per executar diferents funcions amb un mateix terminal extern.
S'han configurat un tipus de variable que és un apuntador de funció, tal com es descriu en la següent línia de codi:

```
typedef void (*nexus_t)(void);
```

Les funcions de servei d'interrupció són molt simples, ja que únicament consulten l'estat del terminal i realitzen una crida a una funció del tipus definit prèviament, d'acord al valor obtingut del terminal. Les següents línies de codi donen un exemple d'una d'aquestes funcions:

```
ISR(INT0_vect){
  /* Utilitza la interrupció INT0
     per seleccionar un dels dos
     registres de freqüència.
     Modulació FSK.              */
  if(PIND & (1<<PIND2)) int0_high();
  else int0_low();
}
```

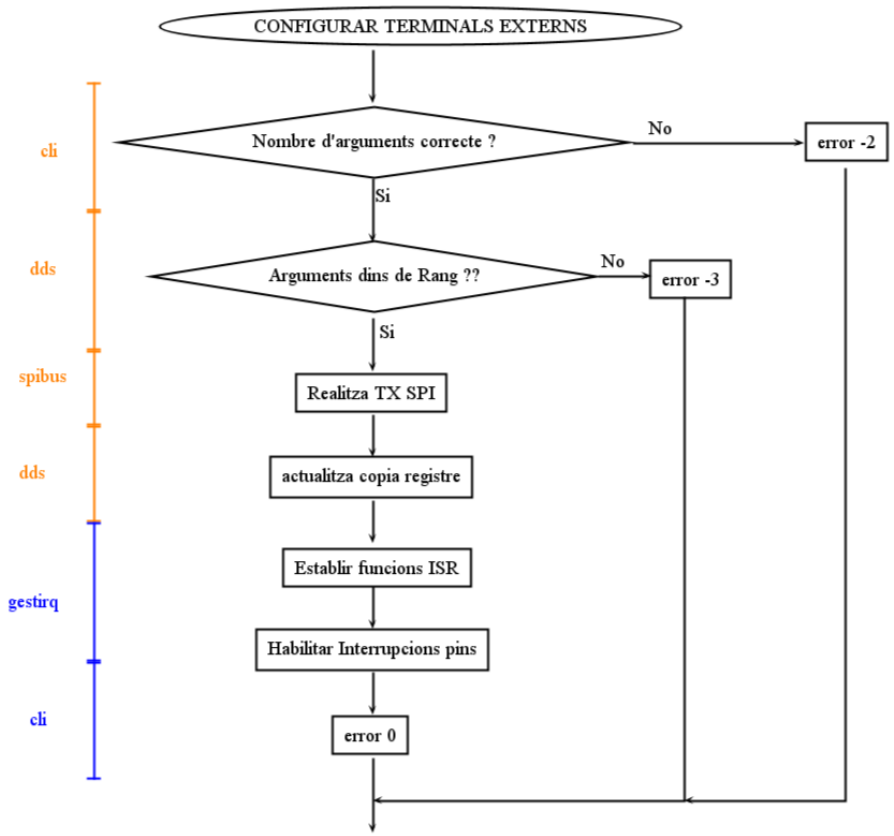


Figura 4.5.: Diagrama del procés de configuració per executar funcions per interrupcions externes

Aquestes funcions, en principi són variables buides, a les que cal donar un valor per ser emprades. Aquest és el mecanisme descrit per el diagrama de la figura 4.5.

La part part de la figura 4.5 marcada en color taronja és idèntica a la que es descriu en l'apartat 4.4. La part del diagrama marcat en blau és la secció que s'encarrega pròpiament de configurar les interrupcions.

En primer lloc es crida a una funció del mòdul *gestirq* que l'únic que fa és donar un valor a les variables del mòdul, usades per alguna de les funcions ISR. El següent és un exemple de com es fa això:

```
void set_INT0(nexus_t fl, nexus_t fh){
  int0_high=fh;
  int0_low=fl;
}
```

Per últim, cal habilitar les interrupcions necessàries per actuar sobre aquests terminals.

```
void INT0_enable(bool active){
  /* habilita o deshabilita INT0
     de EIMSK */
```

```
    if(active) EIMSK |= 0x01;
    else EIMSK &= 0xFE;
  }
```

4.6. Descripció de Programa: Executant funcions per els terminals externs

Una vegada s'han definit les funcions associades a les interrupcions externes segons s'ha vist en l'apartat 4.5, només cal veure com actuen aquestes interrucions una vegada es produeixen. A la figura 4.6 podem veure el diagrama de com succeeix aquest fet.

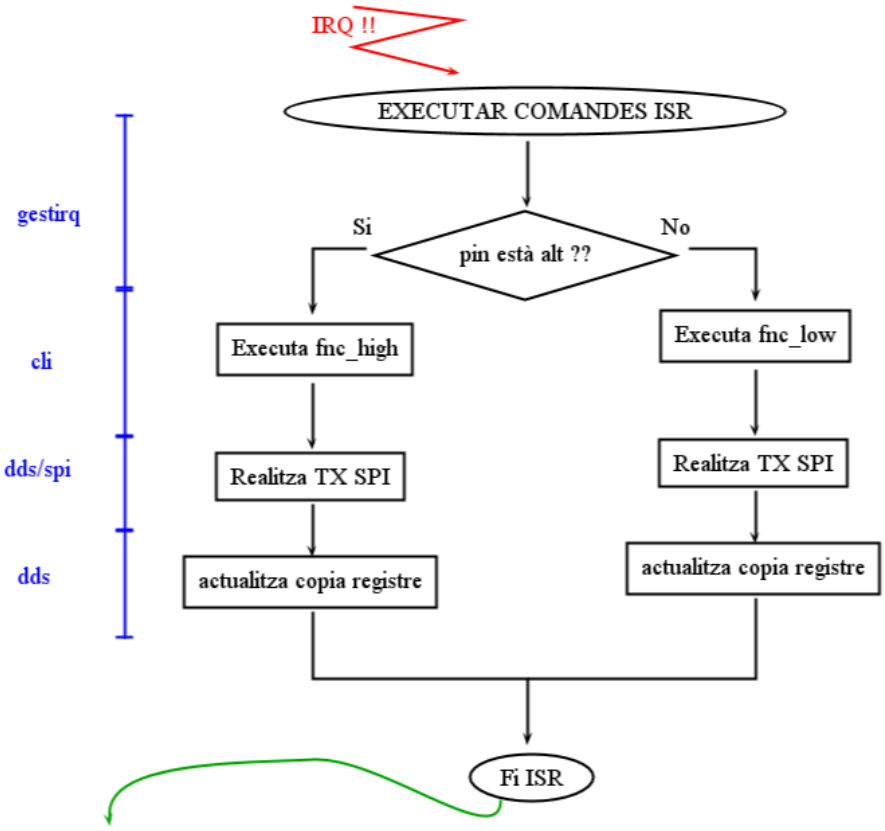


Figura 4.6.: Diagrama del procés de execució de funcions per interrupcions externes

Tal com s'ha comentat anteriorment, les funcions de servei d'interrupció són molt simples, ja que únicament consulten l'estat del terminal i realitzen una crida a una funció del tipus definit prèviament, d'acord al valor obtingut del terminal. Les següents línies de codi donen un exemple d'una d'aquestes funcions:

```
ISR(INT0_vect){
  /* Utilitza la interrupció INT0
     per seleccionar un dels dos
```

```
        registres de freqüència.
        Modulació FSK.              */
    if(PIND & (1<<PIND2)) int0_high();
    else int0_low();
}
```

Per tant es aquí on es determina quina de les dues branques de la ISR s'ha d'executar.

El salt a la funció a executar és automàtic. No cal consultar en quin mode s'està treballant ni quins valors s'han de programar, ja que tot això s'ha definit de forma prèvia i en conseqüència la ISR salta a executar la funció prevista en el moment de la seva configuració. Aquestes funcions són les mateixes que estan associades a comandes introduïdes per el port sèrie, però amb l'avantatge que no cal comprovar l'estat dels errors. L'execució és instantània.

Al final de 'execució de la funció la ISR retorna el control del flux del programa allà on l'havia agafat de forma sobtada.

Aquest és un mecanisme molt flexible que aporta molt valor afegit al dispositiu i el dota d'unes característiques molt interessants per a futures millores i ampliacions.

4.7. Descripció de Programa: Executant altres funcions

La resta de funcions no llistades anteriorment tenen un abast local. Són funcions que realitzen canvis en variables o fan lectura d'elles i no requereixen de més comentaris. Per ampliar la informació es aconsellable revisar el codi font de les funcions.

Si que cal descriure la funció associada a la comanda d'ajuda (HELP). Aquesta funció es mostra a continuació:

```
static int8_t help_fnc(void){
  /*  Explora totes les comandes i retorna
      el text de informació associat a cada
      una d'elles. Aquesta funció incorpora
      un mecanisme per tal de poder ocultar
      algunes comandes al usuari. Això permet
      incorporar comandes de control intern
      i debugging
  */
  char txt_help[50];
  char buff_cmd[7];
  char buff_hlp[40];
  char txt_help_format[11];
  GET_FLASH(txt_help_format,HELP_FRMT);
  for (uint8_t i=0; i<NUM_CMNDS; i++){
    GET_F_HELP(buff_hlp,i);
    if (*buff_hlp != 0){
      GET_F_CMND(buff_cmd,i);
      sprintf(txt_help, txt_help_format, buff_cmd, buff_hlp);
      serial_print(txt_help);
    }
  }
  return 0;
```

```
}
```

Primer de tot, aquesta funció reserva l'espai necessari per emmagatzemar variables de text, que és la matèria prima amb la que treballa aquesta funció. Després va a buscar en la memòria Flash de programa i carrega el format de text amb el que és presentarà l'ajuda en pantalla. Seguidament inicia un bucle per totes les comandes emmagatzemades en la memòria Flash de programa cercant dins dels camps d'ajuda no buits i imprimeix la informació del nom de comanda més el text d'ajuda associat a cada una de les comandes.

Si es programa una funció determinada, associada a un nom de comanda, però el text d'ajuda es deixa en blanc, aleshores aquesta comanda no apareix al executar l'ajuda. Aquest mecanisme permet instal·lar funcions en el firmware que queden ocultes a l'usuari. Aquestes funcions poden tenir la seva importància per realitzar controls o reservar un conjunt de instruccions avançades. Aquesta opció és desenvolupada en aquest projecte.

Les cadenes de text utilitzades en el mòdul *cli* són emmagatzemades en memòria Flash de programa i es gestionen en el mòdul *flashtxt*, juntament amb les funcions definides a través de macros, per tal de gestionar les cadenes de caràcters amb una certa agilitat i legibilitat del codi.

4.8. Organització i creació de comandes

És important entendre l'estructura d'implementació de les comandes, tant per entendre l'organització de mòduls associats a aquestes cmandes com per la creació de noves funcionalitats que poden ampliar el rang d'aplicació del dispositiu.

La implementació de les comandes està formada per tres parts:

Nom de la comanda: Aquest és l'acrònim de la funció. Es una cadena de text constant, la qual es comparada amb la informació introduïda per l'usuari a través del port sèrie UART. Tots els noms de comanda estan definits en una estructura de *array* de manera que la funció *exec_command( )* pot recórrer un a un els elements cercant coincidències. Tant la cadena de text constant, com l'estructura de *array* estan definides en el mòdul *flashtxt*, on s'estableix que aquests quedin ubicats en la memòria Flash de programa.

Ajuda de comanda: Aquesta és una cadena de text constant que conté l'ajuda associada a cada una de les instruccions. Aquesta informació s'obté en resposta a la comanda HELP. Tots els texts d'ajuda estan definits en una estructura de tipus *array* de manera que la funció *help( )* pot recórrer un a un els elements e imprimir-los al costat del nom de cada comanda. Tant la cadena de text constant, com l'estructura de *array* estan definides en el mòdul *flashtxt*, on s'estableix que aquests quedin ubicats en la memòria Flash de programa. En el cas que es vulgui mantenir una funció oculta de la vista dels usuaris, només cal deixar la línia de Ajuda de comanda en blanc.

Funció associada: La funció associada és aquella que s'executa al instanciar la comanda. Aquestes funcions són del tipus *static int8_t funcio_comanda(void)* i estan ubicades a dins del mòdul *cli*. Per tant són funcions locals a aquest mòdul i són executades per la funció *exec_command( )*, tal com s'ha vist en el apartat 4.3. Totes les funcions associades a comandes també estan agrupades en una estructura de tipus *array*. Aquest array és en realitat un conjunt d'apuntadors a funcions.

Al tenir l'arquitectura de les comandes repartida en tres estructures diferents es important mantenir la relació entre elles, utilitzant els mateixos índex en tots els casos. Únicament si l'organització de les comandes es correcta la interpretació d'aquestes serà la esperada.

Seguidament es mostra el cas de la comanda *ENFR0* com exemple de l'estructura emprada.

En primer lloc cal definir el contingut de les dues constants de text: el nom i l'ajuda.

```
const char cmnd2[] PROGMEM="ENFR0";
const char htxt2[] PROGMEM="Drive freq.reg. 0 to output";
```

Després cal incorporar els apuntadors a les cadenes de caràcters dins de les estructures de llistat de comandes i llistat de text d'ajuda. En la següent porció de codi podem veure que els apuntadors *htxt2* i *cmnd2* queden inclosos dins d'aquestes llistes.

```
PGM_P const cmnd_lst[] PROGMEM =
{
  cmnd1,  cmnd2,  cmnd3,  cmnd4,
  cmnd5,  cmnd6,  cmnd7,  cmnd8,
  cmnd9,  cmnd10,  cmnd11,  cmnd12,
  cmnd13,  cmnd14,  cmnd15,  cmnd16,
  cmnd17,  cmnd18,  cmnd19,  cmnd20,
  cmnd21,  cmnd22,  cmnd23,  cmnd24,
  cmnd25,  cmnd26,  cmnd27,  cmnd28,
  cmnd29,  cmnd30,  cmnd31,  cmnd32,
  cmnd33,  cmnd34, cmnd35
};

PGM_P const htxt_lst[] PROGMEM =
{
  htxt1,  htxt2,  htxt3,  htxt4,
  htxt5,  htxt6,  htxt7,  htxt8,
  htxt9,  htxt10,  htxt11,  htxt12,
  htxt13,  htxt14,  htxt15,  htxt16,
  htxt17,  htxt18,  htxt19,  htxt20,
  htxt21,  htxt22,  htxt23,  htxt24,
  htxt25,  htxt26,  htxt27,  htxt28,
  htxt29,  htxt30,  htxt31,  htxt32,
  htxt33,  htxt34, htxt35
};
```

Per les funcions-comanda del mòdul *cli*, primer es declara el seu prototip:

```
static void enable0(void);
```

Després cal ubicar la funció dins de l'estructura de funcions-comanda:

```
const cli_command_t commands[]={
       /* Comandes d'actuació */
  {disable_out},
  {enable0},
```

i per últim s'ha de definir la funció:

```
static void enable0(void){
  /* Funció que selecciona el registre
```

```
        de freqüència 0 i habilita la sortida */
    DDS_Active();
    DDS_Start_0();
    DDS_Idle();
  }
```

Aquest mètode d'organització en principi pot semblar confús, però en contrapartida fa possible alliberar la memòria de dades SRAM d'estructures de text constants. Si es mantinguessin les dades actuals sense aquesta estructura, les dades de text omplirien la memòria SRAM, mentre que gràcies a l'us d'aquesta estructura l'ocupació de SRAM està al voltant del 15%, deixant lloc per fer créixer la pila del sistema o implementar noves funcionalitats.

5. Manual d'usuari

Aquest capítol és la primera versió del manual d'usuari. En el manual es repetiran alguns dels conceptes bàsics que ja s'han apuntat dins de diferents apartats de la descripció tècnica i es donarà una visió pràctica sobre l'ús del dispositiu, objecte d'aquest projecte.

5.1. Introducció

Aquest dispositiu té el propòsit de proporcionar al usuari la possibilitat de generar de forma àgil diferents formes d'ona, en un rang ampli de freqüències i amb un cost assequible.

Donat que té una bona resposta a l'estabilitat al canviar d'una freqüència a un altre i no necessita temps de recuperació i, a més, aquesta commutació entre dos freqüències o dos fases és molt ràpida permet ser utilitzat com a modulador de dades.

Degut a la seva forma característica, que adopta el format que es troba en els mòduls XBee i al cor del seu oscil·lador que li proporciona un molt bon funcionament (el DDS), s'ha decidit batejar a aquest amb el nom de *DDSBee*.

El DDSbee implementa una interfície de comunicació sèrie del tipus UART. Aquest tipus de comunicació sèrie és un mètode pràcticament universal de intercanvi de dades entre molts dispositiu, ja sigui de forma directa o bé a través de adaptadors de comunicacions; com pot ser el cas dels circuits integrats MAX232 [23] per els ports RS232, o bé els circuits integrats FT232RL [24]per enllaçar amb comunicacions USB.
Aquests dispositius poden ser:

- microprocessadors.
- microcontroladors (AVR, PIC, ...)
- ordinadors (a través o no de adaptadors)

Aquests els podrem trobar per tot arreu, ja sigui de manera independent, com els ordinadors de sobretaula o portàtils, com també els podem trobar formant part de altres dispositius de qualsevol propòsit:

- Equips de comunicacions (mòdems, enrutadors, ADSL o AP-wifi)
- Equips industrials (equips de control)

Així doncs l'ús del port UART per part del DDSBee obra una porta a poder ser connectat a multitud de altres equips i dispositius, podent, d'aquesta forma, cobrir un ampli ventall d'aplicacions possibles allà on sigui necessari poder disposar d'un oscil·lador amb bona precisió i resolució.

També cal destacar del DDSBee el seu factor de forma. Tal com s'ha dit amb anterioritat, el circuit s'ha dissenyat sobre la base del layout extern dels mòduls XBee. Aquests mòduls de comunicacions RF són àmpliament usats, això fa que existeixin molts dispositius que proveeixin de connectivitat envers aquests mòduls. Donat això, fabricant de altres dispositius han adoptat aquest layout per realitzar els seus propis mòduls, el que està provocant que aquest encapsulat extern es converteixi en un estàndard per la realització de mòduls similars. El mòdul DDSBee aprofita aquest fet per aportar una nova solució, no contemplada per cap dels mòduls existents que segueixen la forma dels XBee.

5.2. Abast d'aquest manual d'usuari

Aquest manual pretén descriure el *què* i el *com* del DDSBee per tal que l'usuari en faci un correcte ús i descobreixi el potencial d'aquest mòdul que, en essència, es molt simple.

Després de llegir aquest manual l'usuari ha de ser capaç de:

- Conèixer la estructura hardware del dispositiu.
- Saber connectar el mòdul amb un equip i accedir a les seves comandes.
- Poder seleccionar modes de funcionament.
- Saber que existeixen comandes per usuaris avançats.
- Poder desenvolupar funcions basant-nos en els exemples d'us.

5.3. Descripció física

El mòdul té un format extern del tipus XBee, el qual el fa apte per utilitzar-lo amb interfícies de connexió d'aquest tipus.

Descrit breument, el DDSBee és un oscil·lador DDS microcontrolat. Per tant té dues part clarament definides, cada un d'aquests té la seva circuiteria associada ubicada en una de les cares del mòdul.

- El microcontrolador (AVR Atmega328)
- L'osicl·lador DDS(AD9833)

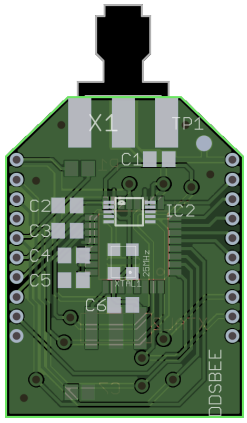
En el mòdul existeixen dos nivells de comunicació diferents:

Externa UART: La comunicació entre l'exterior i el mòdul es fa a través de la connexió UART del microcontrolador. Aquesta és accessible des de els terminals 2 i 3 del mòdul, de la mateixa manera que ho són de manera estàndard en els mòduls XBee.

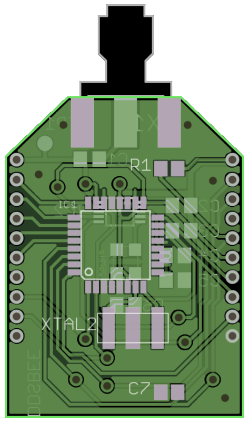
Interna SPI: Entre el microcontrolador i el DDS la comunicació es unidireccional i es realitza per mitjà d'un bus de comunicacions SPI. A través d'aquest bus el microcontrolador programa els diferents registres existents dins del DDS i produeix els canvis en la sortida.

La part própia del DDS incorpora un cristall oscil·lador de referència amb una freqüència de $25MHz$. Això produeix que el valor màxim de freqüència programable del DDSBee sigui de la meitat ($12.5MHz$). En realitat d'un Hertz menys (12.499.999 Hz.). No obstant això cal tenir en compte que cada vegada que el volem pujar la freqüència de sortida en la que ha de treballar l'oscil·lador,la resolució de la forma d'ona disminueix. És a dir, per $12.5MHz$ només disposem de dos bits de resolució, un a cada semiperíode. Amb lo que quan utilitzem l'oscil·lador per freqüències elevades les formes d'ona triangular i sinusoïdal, no poden ser molt exactes. Al menys sense disposar d'una recuperació eficient de la forma d'ona de forma externa.

L'aspecte de la placa és la que es mostra a la figura 5.1, on es poden veure els dissenys de les cares superior i inferior (*top* i *bottom*, respectivament).



(a) Top



(b) Bottom

Figura 5.1.: Vista del disseny de la placa de circuit imprès

DDSBee proporciona dues sortides diferents.

- La sortida principal està situada en un connector SMA a la part frontal del mòdul. Aquesta sortida està desacoblada en continua a través d'un condensador. Aquesta sortida és apta per ser usada amb etapes de amplificació de àudio o RF que requereixin un nivell

zero de tensió continua a l'entrada. Degut a la presència del condensador de desacoblament aquesta sortida podria distorsionar algunes senyal de molt baixa freqüència,per exemple un senyal quadrada per sota de $10Hz$.

- La sortida secundaria està situada en el pin número 20 del mòdul i proveeix la sortida directa des de el DDS intern. Aquesta sortida té un nivell mitjà de tensió al voltant de $310mV$. Aquesta sortida no està desacoblada en continua, per tant es vàlida per senyals de transició lenta o bé per a mantenir un nivell de tensió constant a la sortida, si s'atura el DDS en un determinat moment.

5.3.1. Descripció dels terminals

Els terminals del dispositiu es descriuen a continuació, a la figura 5.2

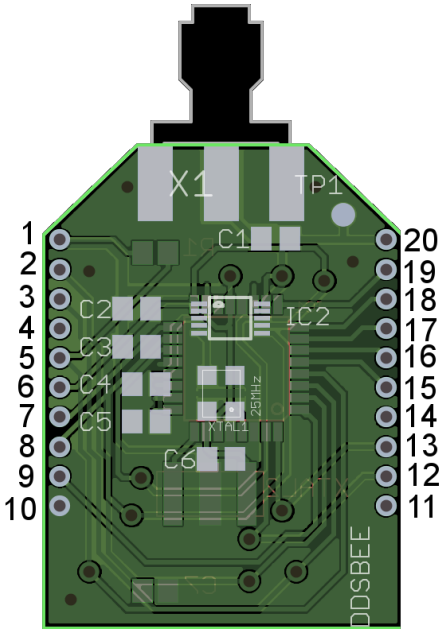


Figura 5.2.: Llistat de terminals del DDSBee

Pin DDSBee	Denominació	Pin DDSBee	Denominació
1	VCC	20	F_OUT
2	DOUT	19	AD1/DIO1
3	DIN	18	AD2/DIO2
4	N.C.	17	AD3/DIO3
5	RESET	16	DIO6/SDA
6	SCK	15	DIO5/SCL
7	MISO	14	N.C.
8	MOSI	13	REG_CHG
9	EN_OUT	12	DIO7
10	GND	11	N.C.

Alimentació: El DDSBee està a limentat entre els terminals 1 (VCC) i 10 (GND). L'alimentació pot tenir valors entre 3 i 5V (Típicament alimentat a 3.3V).

DOUT: Sortida de dades des de el DDSBee. Les dades s'envien a 115200 bds. amb el format de 8 bits de dades un bit de stop i sense paritat.

DIN: Entrada de dades cap a el DDSBee. Cal finalitzar cada línia amb la retorn de línia (n), sense retorn de carro.

F_OUT: Sortida del DDSBee directa. Aquesta sortida incorpora la tensió contínua (300 mV) al contrari de la sortida principal a través del connector SMA (Veure la descripció de la secció anterior 5.3).

EN_OUT: Terminal de control extern per habilitar o deshabilitar la sortida.

REG_CHG: Terminal d'intercanvi de registres. Segons l'estat de aquest terminal s'escull internament entre els dos registres de fase o els dos registres de freqüència, depenent del mode de treball en cada cas.

RESET: Posant aquest terminal en estat baix es fa un reset del sistema i aquest es reiniciat. També s'utilitza per programar el firmware.

Programació de Firmware: El dispositiu es programa amb un ISP, que està distribuït en el terminals del DDSBee 6 (SCK), 7 (MISO), 8 (MOSI) i el RESET del terminal 5. També és necessari proveir el sistema d'alimentació (VCC i GND).

Terminals de reserva: Són terminals que en la present versió de Firmware no tenen definida cap aplicació però estan connectats internament, per tal de proveir futures ampliacions. Aquests terminals són el 12 (DIO7), el 15 (DIO5), el 16 (DIO6), el 17 (DIO3), el 18 (DIO2) i el 19 (DIO1).

Terminals no connectats (N.C.): Aquests terminals no tenen cap connexió interna amb el DDSBee. Són el 4, el 11 i el 14.

5.4. Utilització del mòdul

El DDSBee es pot gestionar de dos maneres diferents:

Per comandes: El circuit respon a les ordre introduïdes des d'un port sèrie UART. La comunicació pot venir des d'un ordinador o des de qualsevol equip en funcionament autònom.

Per interrupcions: El circuit pot funcionar de manera autònoma. No obstant, s'ha de haver fixat algun mode de funcionament de forma prèvia, seleccionat a través de l'us de comandes. Utilitzant les interrupcions, el DDSBee respon als canvis d'estat dels terminals prèviament definits amb accions corresponents al mode de funcionament seleccionat. Cal habilitar el control extern a través de les comandes corresponents, abans de fer ús de les interrupcions.

El DDSBee pot ser utilitzat sense necessitat de configurar cap mode de funcionament determinat, programant en cada moment el comportament del dispositiu. Per exemple establint la freqüència d'un registre i posant-lo en marxa al mateix temps (comanda SRFR0) i deshabilitant la sortida per tancar-lo (Comanda DSOUT). Aquest comportament pot ser adequat per alguns usos (Veure l'apartat 5.7.1). D'altra banda, el DDSBee incorpora modes de treball que facilitant les accions i es pot realitzar la mateixa tasca únicament canviant l'estat d'un determinat terminal, ja sigui de manera manual o bé a través de circuiteria externa.

Els modes de funcionament es configuren a través de les comandes i es poden executar indistintament a través de comandes o a per mitjà de les interrupcions, depenent de l'aplicació que l'usuari vulgui fer en cada cas.

Els modes de funcionament són:

Mode ASK: [11] Aquest és el mode de modulació per desplaçament d'amplitud. Més concretament seria una modulació *OOK (On/Off Keying)* ja que el que fa es donar tota la modulació o res. Per el nostre circuit aquest mode contempla l'us d'un terminal (pin 9), el qual al ser posat en estat alt habilita la sortida del registre de freqüència número 1. En cas de estar en estat baix, deshabilita la sortida del DDS.

Aquest mode, seria l'utilitzat en el cas que simplement es necessites un control per la sortida, per alguna aplicació que necessiti habilitar o inhabilitar el senyal generat.

Mode FSK: El mode FSK implementa la modulació per desplaçament de freqüència. Això vol dir que utilitza dos freqüències diferents per indicar si el valor es un '1' o bé un '0'. En aquest mode, el DDSBee fa un intercanvi dels seus dos registres de freqüència interns. Els valors dels dos registres són fixats en el moment de establir el mode de funcionament. A més dels mode FSK genèric s'han creat dues comandes per tal de programar de manera instantània dos modes FSK específiques que han sigut molt populars:

Mode Bell-202: [12] Aquest és un mode específic de modulació FSK. Bell 202 utilitza un to de 1200 Hz per indicar una marca (típicament un '1' binari) i 2200 Hz per indicar un espai (típicament un '0' binari).

Mode Bell-103: [13] Aquest és un mode específic de modulació FSK. Bell 103 utilitza un to de 1270 Hz per indicar una marca (típicament un '1' binari) i 1070 Hz per indicar un espai (típicament un '0' binari).

Mode PSK: [14] PSK significa modulació per desplaçament de fase. En el DDSBee també es podria anomenar 2PSK o bè BPSK de *Binary Phase Shift Keying*, ja que únicament hi ha

la capacitat de transmetre dos símbols. Aquest és un esquema de modulació digital que transmet dades mitjançant el canvi, o la modulació de la fase d'un senyal de referència (l'ona portadora). En el DDSBee aquest mode requereix de tres paràmetres: freqüència portadora, desplaçament de fase 0 i desplaçament de fase 1.

5.5. Per usuaris avançats

Si bé el DDSBee està dissenyat per ser gestionat d'una manera simple a través de les comandes genèriques que es descriuen en l'apartat 5.6 i acceptant les limitacions en freqüència pel que fa a la resolució emprada en generar les formes d'ona, els usuaris avançats poden experimentar amb més profunditat les característiques d'aquest dispositiu. Aquest és un capítol per a valents!

Comanda RAW: Aquesta és una comanda que fa transparent l'enviament de dades entre el microcontrolador i el circuit DDS. RAW proporciona un pont de comunicació entre el port sèrie i el Bus SPI. És a dir, permet enviar informació directament des de port sèrie i programar així el DDS.

RAW accepta un màxim de tres paràmetre de cop, aquest paràmetres són grups de 16 bits, codificats en format DECIMAL. Aquest fet és molt important, ja que usualment els valors enviats són interpretats com a hexadecimals o bé binaris. Com exemple suposem que volem enviar la dada 0x2000 en hexadecimal (0010000000000000 en binari). Primer cal convertir aquest valor a decimal:

$$0x2000 \rightarrow 8192$$

Una vegada tenim el nombre que volem en decimal el posem com a paràmetre a la comanda RAW.

```
> RAW 8192
```

Aquest paràmetre serà enviat a través del Bus SPI de forma directa. La comanda RAW no modifica registres temporals, ni memòries en execució, per tant cal tenir en compte que les modificacions que es realitzin amb RAW no són reflectides en les condicions de programa actuals.

Filtratge de sortida: La freqüència màxima que ofereix el mòdul és de 12.5MHz. No obstant a aquesta freqüència únicament ofereix dos punts per generar la forma d'ona. L'usuari ha d'entendre que difícilment aquesta forma d'ona s'assemblarà a una ona sinusoïdal, ni triangular. Els usuaris que vulguin experimentar poden utilitzar filtres adequats per tal de centrar el la sortida en el valor de freqüència desitjat.

Igualament existeix la possibilitat de treballar amb algun dels àlies que es poden generar i que proporcionarien freqüències per sobre de 12.5MHz. El DDS proveeix d'una sortida mostrejada, l'espectre del quals conté la freqüència fonamental de sortida més freqüències imatge (o àlies) que apareixen en múltiples de la freqüència de rellotge (25MHz.) i de la freqüència de sortida. Podem veure a la Figura 5.3 la representació d'aquest espectre [18].

Si es desitja, per exemple, obtenir un rang de freqüències entorn als 28MHz. necessitaríem un filtre a la sortida del DDS sintonitzat en aquesta freqüència i generar freqüències al voltant dels 3MHz. D'aquesta manera s'obte que la segona freqüència imatge ens cau dins del rang del filtre, es a dir:

$$f_c + f_{out} = 25 + 3 = 28MHz$$

Així l'usuari pot estendre el rang de treball del circuit.

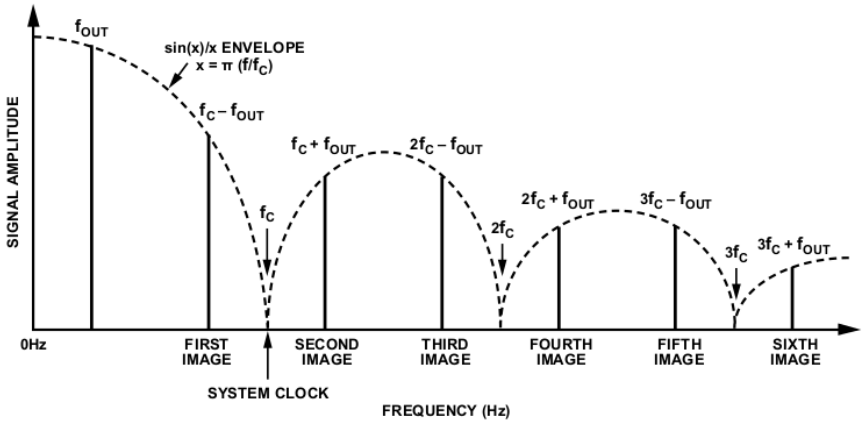


Figura 5.3.: Espectre de sortida del DAC

5.6. Interfície d'usuari i Comandes

5.6.1. Interfície d'usuari

El dispositiu ofereix al usuari dues vies de interactuar amb ell. Per una banda hi ha la interfície d'usuari i per altra tenim el control extern del dispositiu, a través dels seu pins. Per poder actuar per mitjà d'aquesta segona via cal, primerament, haver establert el mode de funcionament adequat a través de la interfície.

La interfície d'usuari requereix d'una connexió física de port sèrie a nivell UART. Aquesta comunicació s'estableix a 115200 bds,. 8 bits de dades, sense paritat i 1 bit de stop (115200, 8, N, 1).

Les comandes tenen el següent model:

`CMND (⟨ PAR1 ⟩ ⟨ PAR2 ⟩ ⟨ ... ⟩)`

Totes les comandes són introduïdes amb lletres majúscules i nombres. Tots els paràmetres són nombre sencers i positius. Per tant els valors de freqüència s'expressen en unitats de Hertz i els valors de graus han d'expressar-se en unitats de grau sexagesimals (^{o}). Per altres paràmetres, cal consultar la documentació referent a la comanda concreta.

El nombre de paràmetres no és fix per totes les comandes i moltes d'elles no requereixen de cap paràmetre. Per tant cal consultar la documentació referent a cada comanda específicament, per tal de comprovar la seva sintaxi específica.

La separació entre comandes i paràmetres o entre els propis paràmetres és realitza per mitjà d'un o més espais.

La comanda es tanca amb un simple salt de línia (\n). El tancament de línies de comanda amb retorns de carro i retorns de línia apareixen com a errors de comanda desconeguda (veure la secció de codis d'error).

5.6.2. Codis d'error

La interfície d'usuari, dins del seu interpret d'ordres, incorpora un una sèrie de missatges d'error que informen a l'usuari de que ha pogut fallar en un determinat moment. El mecanisme en el qual es generen aquests missatges proveeix al dispositiu d'una capa de seguretat que per tal d'impedir comportament no esperats.

Els missatges d'error implementats són:

Unknown command: L'interpret de comandes ens retorna aquest missatge en el cas que no trobi la comanda introduïda.

Wrong parameters: Les comandes retornen aquest error, si no s'introdueix el nombre exacte de paràmetres requerits en cada funció.

Out of range: Les comandes retornen aquest missatge en el cas que algun dels valors introduïts en les funcions està fora dels límits permesos per aquell paràmetre.

Not allowed: S'ha intentat accedir a alguna adreça de memòria EEPROM no permesa.

5.6.3. Llistat de comandes Implementades

Format del llistat de comandes

Seguidament es presenten totes les instruccions implementades en el dispositiu. Aquestes presenten el següent format:

CMND
Exemple of a Command

Un exemple de comanda

Model:

CMND ⟨ PAR1 ⟩ (⟨ PAR2 ⟩ opt.)

Exemple d'ús:

```
> CMND 1234
```

Aquest és un exemple de com són presentades les diferents instruccions dins d'aquest manual. En el requadre de la part superior tenim el nom de la comanda, en aquest exemple *CMND*. En el següent requadre trobem l'origen del acrònim que genera el nom de la comanda. Seguidament tenim la descripció breu en català d'allò que fa la comanda.

El model ens mostre el ús genèric de la comanda. Si les comandes incorporen paràmetres, aquests és mostren entre parèntesis en angle. Si algun dels paràmetres és opcional apareixerà entre parèntesis corbats o amb el vocable *opt.*.

L'exemple d'ús mostra una possible utilització de la comanda amb l'ús (o no) de paràmetres numèrics.

Per últim es troba l'apartat que estas llegint, el qual es una descripció acurada del ús de la comanda.

COMANDES D'ACTUACIÓ

El subconjunt de les comandes d'actuació, produeixen una acció instantània en la sortida del DDS, ja sigui per posar-lo en marxa, aturar-lo o seleccionar el registre que afecta a la sortida.

DSOUT

Disable Output

Deshabilita la sortida del oscil·lador, aturant-lo immediatament.

Model:

```
DSOUT
```

Exemple d'ús:

```
> DSOUT
```

Aquesta comanda permet parar la sortida del oscil·lador de forma immediata, independentment del registre activat prèviament.
La comanda no necessita cap paràmetre addicional.

ENFR0

Enable Frequency Register 0

Activa la sortida del oscil·lador amb el valor de FR0

Model:

```
ENFR0
```

Exemple d'ús:

```
> ENFR0
```

Aquesta comanda activa la sortida del oscil·lador de manera immediata amb el valor de freqüència programat en el registre de freqüència 0.
Aquesta comanda no requereix cap paràmetre addicional.

ENFR1

Enable Frequency Register 1

Activa la sortida del oscil·lador amb el valor de FR1

Model:

```
ENFR1
```

Exemple d'ús:

```
> ENFR1
```

La comanda *ENFR1* activa la sortida del oscil·lador de manera immediata amb el valor de freqüència programat en el registre de freqüència 1.
Aquesta comanda no requereix cap paràmetre addicional.

ENPHR0

Enable Phase Register 0

Habilita l'ús del registre de fase 0

Model:

```
ENPHR0
```

Exemple d'ús:

```
> ENPHR0
```

La comanda permet aplicar a la sortida l'increment de fase marcat en el registre de fase 0.
Aquesta comanda no requereix cap paràmetre.

ENPHR1

Enable Phase Register 1

Habilita l'ús del registre de fase 1

Model:

```
ENPHR1
```

Exemple d'ús:

```
> ENPHR1
```

La comanda permet aplicar a la sortida l'increment de fase marcat en el registre de fase 1. Aquesta comanda no requereix cap paràmetre.

SRFR0

Set and Run Frequency Register 0

Estableix el valor del registre de freqüència 0, intern del DDS i habilita la seva sortida immediatament.

Model:

```
SRFR0 ⟨ freq ⟩
```

Exemple d'ús:

```
> SRFR0 137500
```

Aquesta funció permet posar el DDS en funcionament de forma immediata, establint el valor de sortida del registre de freqüència número 0.
Els valors límit per el paràmetre ⟨ freq ⟩ poden estar compresos entre 0 i $12500000Hz$. Les unitats emprades són Hertz.

SRFR1

Set and Run Frequency Register 1

Estableix el valor del registre de freqüència 1, intern del DDS i habilita la seva sortida immediatament.

Model:

`SRFR1 ⟨ freq ⟩`

Exemple d'ús:

```
> SRFR1 2200
```

Aquesta funció permet posar el DDS en funcionament de forma immediata, establint el valor de sortida del registre de freqüència número 1. Els valors límit per el paràmetre ⟨ freq ⟩ poden estar compresos entre 0 i 12500000Hz. Les unitats emprades són Hertz.

COMANDES DE PROGRAMACIÓ

El subconjunt de comandes de programació tenen la missió de canviar el valor dels registres. Aquest canvi afectarà a la sortida sempre que el registre modificat estigui activat. En cas contrari únicament es canvia el valor del registre i es queda en espera de ser activat, per mitjà de la comanda d'actuació corresponent, per exemple.

SFR0

Set Frequency Register 0

Estableix el valor del registre de freqüència 0, intern del DDS.

Model:

`SFR0 ⟨ freq ⟩`

Exemple d'ús:

```
> SFR0 137500
```

Aquesta funció permet establir el valor de sortida del registre de freqüència número 0. Els valors límit per el paràmetre ⟨ freq ⟩ poden estar compresos entre 0 i 12500000Hz. Les unitats emprades són Hertz.

SFR1

Set Frequency Register 1

Estableix el valor del registre de freqüència 1, intern del DDS.

Model:

`SFR1 ⟨ freq ⟩`

Exemple d'ús:

```
> SFR1 2200
```

Aquesta funció permet establir el valor de sortida del registre de freqüència número 1. Els valors límit per el paràmetre ⟨ freq ⟩ poden estar compresos entre 0 i $12500000Hz$. Les unitats emprades són Hertz.

SPHR0

Set Phase Register 0

Estableix el valor del registre de fase 0.

Model:

`SPHR0 ⟨ fase ⟩`

Exemple d'ús:

```
> SPHR0 180
```

La comanda estableix el valor que s'establirà en el registre de fase 0. La comanda requereix d'un paràmetre. El valor s'expressa en unitats de grau i pot establir-se entre 0 i 359.

SPHR1

Set Phase Register 1

Estableix el valor del registre de fase 1.

Model:

`SPHR1 ⟨ fase ⟩`

Exemple d'ús:

```
> SPHR1 270
```

La comanda estableix el valor que s'establirà en el registre de fase 1. La comanda requereix d'un paràmetre. El valor s'expressa en unitats de grau i pot establir-se entre 0 i 359.

SINE

Select Sine Waveform

Selecciona la funció Sinus

Model:

`SINE`

Exemple d'ús:

```
> SINE
```

Aquesta comanda selecciona que la forma d'ona en la sortida sigui sinusoïdal. L'amplitud de la forma d'ona en la sortida és de uns $600mV$. La funció no activa la sortida si aquesta està inhabilitada. Enregistra la forma d'ona que s'activarà en el moment en que la sortida sigui habilitada.La comanda no requereix de cap paràmetre addicional.

SQR
Select Square Waveform

Selecciona la funció quadrada

Model:

`SQR`

Exemple d'ús:

```
> SQR
```

Aquesta comanda selecciona que la forma d'ona en la sortida sigui rectangular. L'amplitud de la forma d'ona en la sortida és de uns $3.3V$. La funció no activa la sortida si aquesta està inhabilitada. Enregistra la forma d'ona que s'activarà en el moment en que la sortida sigui habilitada.La comanda no requereix de cap paràmetre addicional.

TRI
Select Triangle Waveform

Selecciona la funció Triangle

Model:

`TRI`

Exemple d'ús:

```
> TRI
```

Aquesta comanda selecciona que la forma d'ona en la sortida sigui triangular. L'amplitud de la forma d'ona en la sortida és de uns $600mV$. La funció no activa la sortida si aquesta està inhabilitada. Enregistra la forma d'ona que s'activarà en el moment en que la sortida sigui habilitada.La comanda no requereix de cap paràmetre addicional.

COMANDES DE COMPORTAMENT

Les comandes de comportament són el subconjunt d'instruccions que estableixen els modes de funcionament del dispositiu. Realitzen configuracions completes per si mateixes i poden ser combinades per obtenir-ne un resultat concret. Per exemple combinar un mode de funcionament de modulació determinat amb la habilitació del control extern. ·

També s'hi troben classificades en aquest subconjunt les instruccions que estableixen de quina forma ha de activar-se el dispositiu, al ser posat en marxa.

EEOC

Enable External Output Control

Habilita el control de sortida externament

Model:

```
EEOC
```

Exemple d'ús:

```
> EEOC
```

Aquesta comanda permet que el control sobre la sortida es realitzi de manera externa. Aquesta funció permet deixar preparat el dispositiu per ser manipulat per mitjà de controls externs. La comanda no requereix de cap paràmetre addicional.

DEOC

Disable External Output Control

Deshabilita el control de sortida extern

Model:

```
DEOC
```

Exemple d'ús:

```
> DEOC
```

Aquesta comanda desactiva la possibilitat de que el control sobre la sortida es realitzi de manera externa. Amb aquesta funció es garanteix que el control sobre la sortida serà únicament per mitjà de comandes enviades per el port sèrie UART. La comanda no requereix de cap paràmetre addicional.

SFSC

Set Frequency Shift Control

Habilita el control extern de canvi de freqüència

Model:

```
SFSC
```

Exemple d'ús:

```
> SFSC
```

La comanda SFSC permet que externament es pugui manipular quin dels dos registres interns del DDS (FR0 o FR1) s'envia com a control de sortida. Aquesta comanda és la que es pot utilitzar per preparar el dispositiu per treballar com a modulador FSK. La comanda no requereix de cap paràmetre addicional.

SPSC

Set Phase Shift Control

Habilita el control extern de canvi de fase

Model:

```
SPSC
```

Exemple d'ús:

```
> SPSC
```

La comanda SFSC permet que externament es pugui manipular quin dels dos registres interns del DDS (FR0 o FR1) s'envia com a control de sortida. Aquesta comanda és la que es pot utilitzar per preparar el dispositiu per treballar com a modulador FSK. La comanda no requereix de cap paràmetre addicional.

SFSK

Set FSK Mode

Estableix el mode de funcionament com a modulador FSK

Model:

SFSK ⟨ $\texttt{freq}_0$ ⟩ ⟨ $\texttt{freq}_1$ ⟩

Exemple d'ús:

```
> SFSK  800 1500
```

Amb aquesta comanda es programen a la vegada els dos registres de freqüència FR0 i FR1, deixant el dispositiu completament preparat per poder ser gestionat com a modulador FSK. Els dos paràmetres ⟨ freq_0 ⟩ i ⟨ freq_1 ⟩ són expressats en Hertz i tenen una resolució de $1Hz$. Els valors vàlids per aquests paràmetres estan compresos entre 0 i $12500000Hz$.

SPSK

Set PSK Mode

Estableix el mode de funcionament com a modulador PSK

Model:

SPSK

Exemple d'ús:

```
> SPSK
```

Amb aquesta comanda es programen a la vegada els dos registres de freqüència PHR0 i PHR1, deixant el dispositiu completament preparat per poder ser gestionat com a modulador PSK.

SASK

Set ASK Mode

Estableix el mode de funcionament com a modulador ASK

Model:

`SASK ⟨ freq ⟩`

Exemple d'ús:

```
> SASK  3100
```

Amb aquesta comanda es programa el registre de freqüència FR0, deixant el dispositiu preparat per poder ser gestionat com a modulador Per manipulació de Amplitud ASK. El paràmetre $\langle freq \rangle$ s'expressa en Hertz i té una resolució de $1Hz$. Els valors vàlids per aquest paràmetre estan compresos entre 0 i $12500000Hz$.

B202

Set Bell-202 Mode

Estableix el mode de funcionament del estàndard Bell-202

Model:

`B202`

Exemple d'ús:

```
> B202
```

Amb aquesta comanda es programen a la vegada els dos registres de freqüència FR0 a $1200Hz$ i FR1 a $2200Hz$, deixant el dispositiu completament preparat per poder ser gestionat com a modulador FSK en l'estàndard Bell-202. La comanda no requereix de cap paràmetre addicional.

B103

Set Bell-103 Mode

Estableix el mode de funcionament del estàndard Bell-103

Model:

```
B103
```

Exemple d'ús:

```
> B103
```

Amb aquesta comanda es programen a la vegada els dos registres de freqüència FR0 a $1070Hz$ i FR1 a $1270Hz$, deixant el dispositiu completament preparat per poder ser gestionat com a modulador FSK en l'estàndard Bell-103. La comanda no requereix de cap paràmetre addicional.

RBOOT

Reset On Boot

Reset de paràmetres al posar-se en marxa

Model:

```
RBOOT
```

Exemple d'ús:

```
> RBOOT
```

Aquesta funció permet establir que en el moment d'arrancada del dispositiu, en el seu registre de control, hi haurà els paràmetres per defecte (valor 0x2100):

- Forma d'ona de sortida Sinus.
- Sortides desactivada.
- Seleccionat registre de fase 0.
- Seleccionat registre de freqüència 0.

Els registres de freqüència i fase del DDS mantindran el darrer contingut amb els quals s'hagin programat.
Aquesta funció no requereix de cap paràmetre addicional.

PBOOT
Preset On Boot

Carrega conjunt de paràmetres al posar-se en marxa

Model:

`PBOOT ⟨ num. (opt) ⟩`

Exemple d'ús:

```
> PBOOT
```

Aquesta funció permet establir que en el moment d'arrancada, el dispositiu carregui els paràmetres prèviament emmagatzemats a través de la funció STORE. La comanda s'encarrega de informar al inicialitzador que, després d'una posada en marxa ha de trobar els paràmetres en memòria no volàtil EEPROM. Si dins no s'ha realitzat de manera prèvia una crida a la comanda *STORE*, es poden obtenir resultats inesperats.
El paràmetre ⟨ num. ⟩ és opcional. Aquest paràmetre permet seleccionar entre diferents configuracions d'usuari pre-carregades. Si no s'introdueix aquest paràmetre, el seu valor per defecte és 1.

ALTRES COMANDES

Aquí s'hi troben les comandes que proveeixen informació a l'usuari i les que no poden ser classificades en cap dels grups anteriors.

HELP
Shows Help Text

Mostra pel terminal una ajuda breu sobre les comandes

Model:

`HELP`

Exemple d'ús:

```
> HELP
```

Per mitjà de la comanda HELP obtenim per el terminal, un llistat de les comandes bàsiques i una breu descripció de la seva funcionalitat. Aquesta ajuda no substitueix al manual de comandes, ni informa de les comandes més avançades o ocultes que el dispositiu pugui disposar. L'ajuda pretén ser un recordatori ràpid sobre la denominació de cada comanda. No es necessari cap més paràmetre en aquesta comanda.

STORE

Stores Actual Status in EEPROM

Guarda l'estat actual del dispositiu dins de memòria EEPROM

Model:

`STORE ⟨ num. (opt) ⟩`

Exemple d'ús:

```
> STORE 1
```

Aquesta instrucció emmagatzema una copia dels valors de freqüències fases corresponents a cadascun dels registres. També conserva una copia literal del contingut del registre de control del DDS. Totes aquestes dades són desades dins de memòria EEPROM no volàtil. Utilitzant aquesta funció, conjuntament amb la comanda *PBOOT*, permet que l'equip adquireixi una configuració per defecte només posar-se en marxa. Si es realitza una crida a la comanda *STORE* mentre la sortida està activa i després s'executa la comanda *PBOOT*, al posar en marxa el dispositiu aquest activarà la sortida que hi havia en el moment de haver executat *STORE*.
El paràmetre ⟨ num. ⟩ és opcional. Aquest paràmetre permet disposar de fins a tres configuracions d'usuari pre-carregades. El rang del paràmetre va de 1 fins a 3. Altres valors de paràmetre retornen el missatge de error *"Not Allowed"*. Si no s'introdueix aquest paràmetre, el seu valor per defecte és 1.

LOAD

Load a Memory Preset into DDS

Recupera un preset d'usuari prèviament guardat en la memòria EEPROM

Model:

`LOAD ⟨ num. (opt) ⟩`

Exemple d'ús:

```
> LOAD 3
```

Aquesta comanda permet recuperar els valors emmagatzemats en espais específics de la memòria EEPROM, en forma de presets d'usuari.

A través d'aquesta funció es ràpid el poder configurar opcions habituals en el dispositiu.

El paràmetre ⟨ num. ⟩ és opcional. Aquest paràmetre permet disposar de fins a tres configuracions d'usuari pre-carregades. El rang del paràmetre va de 1 fins a 3. Altres valors de paràmetre retornen el missatge de error *"Not Allowed"*. Si no s'introdueix aquest paràmetre, el seu valor per defecte és 1.

SBOOT

Status of Boot

Retorna l'estat en que s'inicialitza el dispositiu

Model:

`SBOOT`

Exemple d'ús:

```
> SBOOT
```

Aquesta comanda retorna, a través d'un missatge, quin és l'estat del procés Boot. Aquest missatge ens el donarà d'acord a si s'ha executat prèviament la comanda *PBOOT* o *SBOOT*. El missatge retornat pot ser:

Boot: default si es fa un reset al arrencar el dispositiu.

Boot: preset ⟨ num. (opt) ⟩ si prèviament s'ha executat la comanda *PBOOT*, on *num* correspon al número de memòria programada (de 1 a 3).

Aquesta comanda no requereix de cap paràmetre.

COMANDES AVANÇADES

El següent subconjunt de comandes requereixen d'un major grau de coneixement sobre el circuit DDS, per tal de ser útil al usuari. És per això que aquestes comandes són considerades avançades. Sent així, aquestes instruccions no apareixen documentades en la comanda *HELP* del dispositiu. Es recomana no fer-n'he ús si no es te clar l'abast de la funció.

CTRL

Get Control Register

Retorna el registre de control

Model:

```
CTRL
```

Exemple d'ús:

```
> CTRL
```

Aquesta comanda permet visualitzar la copia del contingut que hi ha en el registre de control del DDS. Aquest contingut es presenta en format hexadecimal. Per tal de entendre el seu significat cal llegir acuradament el Data Sheet del circuit integrat DDS AD9833. La comanda *CTRL* no admet cap paràmetre.

RAW
Put Raw Data into the DDS

Programa els registres del DDS directament

Model:

RAW ⟨ data1 ⟩ (⟨ data2 ⟩ opt.) (⟨ data3 ⟩ opt.)

Exemple d'ús:

```
> RAW 8192
```

Atenció!! No utilitzar aquesta comanda sense entendre el significat dels valors introduïts com a paràmetre. Cal llegir acuradament el Data Sheet del circuit integrat DDS AD9833.

La comanda *RAW* és una instrucció que permet enviar per el port SPI intern les dades de 16 bits passades com a paràmetres.

Atenció!! Les dades lliurades a la comanda han d'estar en format decimal. En el cas del exemple, s'ha volgut enviar el valor 0x2000 i primer cal convertir-lo en un valor decimal: 8192. No intentar introduir valors hexadecimals.

La comanda *RAW* necessita, com a mínim, un paràmetre i un màxim de tres.

5.7. Exemples de programació

Les següents línies d'aquest document mostren com realitzar programacions del DDSBee, explicant la utilitat de cada línia. Les accions mostrades en cada una de les seccions, s'han d'entendre com una seqüència d'accions per tal de que tingui sentit els resultats que es mostren.

5.7.1. Exemples simples

Si connectem la sortida a un sistema de àudio o a un oscil·loscopi podrem comprovar el resultat.

Establir el valor de freqüència 1520Hz. en el registre 0 i posar la sortida en marxa amb aquest valor

```
> SRFR0 1520
>
```

La sortida s'activa immediatament generant una ona sinusoïdal de 1520Hz.

Establir el valor de freqüència 2300 Hz. en el registre 1, sense posar-lo en marxa

```
> SFR1 2300
>
```

La sortida continua inalterada. Continuem obtenint un sinus a la sortida amb la freqüència de 1520Hz. De totes maneres el registre de freqüència número 1 ha quedat programat i llest per ser utilitzat.

Deshabilitar la sortida.

```
> DSOUT
>
```

La sortida s'atura de forma immediata.

Mostra el text d'ajuda de les comandes.

```
> HELP
   DSOUT:   Disables Output
   ENFR0:   Drive freq.reg. 0 to output
   ENFR1:   Drive freq.reg. 1 to output
  ENPHR0:   Drive phase reg. 0 to output
  ENPHR1:   Drive phase reg. 1 to output
   SRFR0:   Set and run frequency register 0
   SRFR1:   Set and run frequency register 1
    SFR0:   Set frequency register 0
    SFR1:   Set frequency register 1
   SPHR0:   Set phase register 0
   SPHR1:   Set phase register 1
    SINE:   Select sine waveform
     SQR:   Select square waveform (3.3V)
     TRI:   Select triangle waveform
    EEOC:   Enable external output control
    DEOC:   Disable external output control
    SFSC:   Set frequency shift control
    SPSC:   Set pahse shift control
    SFSK:   Set FSK mode
    SPSK:   Set PSK mode
    SASK:   Set ASK mode
    B202:   Set Bell-202 mode
    B103:   Set Bell-103 mode
    HELP:   Shows help text
   STORE:   Saves actual status
    LOAD:   Load a preset
   RBOOT:   Reset device on Boot (defaul)
   PBOOT:   Load preset on Boot
   SBOOT:   Gets the Boot status
>
```

Retorna per el port sèrie el text d'ajuda predefinit per les comandes.

Habilitar el registre de freqüència número 1.

```
> ENFR1
>
```

La sortida s'activa immediatament, amb el valor programat anteriorment (2300 Hz.)

Canvia la forma d'ona a triangular.

```
> TRI
>
```

Manté la sortida a 2300 Hz., però la forma d'ona ha canviat i ara és triangular.

Estableix el mode de funcionament per l'estàndard Bell-202.

```
> B202
>
```

Aquest mode de funcionament atura la sortida de manera instantània i configura automàticament els dos registres de freqüència FR0 i FR1 amb els valors de 2200 Hz i 1200 Hz, respectivament.

Habilitar el registre de freqüència número 0.

```
> ENFR0
>
```

La sortida s'activa immediatament, amb el valor programat anteriorment (2200 Hz.)

5.7.2. Exemples intermedis

Comprovem l'estat del Boot del sistema.

```
> SBOOT
Boot: default
>
```

Ens indica que el dispositiu s'inicia en mode desconnectat, que és el mode per defecte.

Establir un mode FSK personalitzat.

```
> SFSK 5200 1100
>
```

Aquest mode estableix que el registre de freqüència 0 proporcionarà una freqüència de 5200 Hz. i el registre 1 una freqüència de 1100 Hz.

Habilitar el registre de freqüència número 1.

```
> ENFR1
>
```

La sortida s'activa immediatament, amb el valor programat anteriorment en el registre 1 (1100 Hz.).

Emmagatzema l'actual estat del DDSBee en una memòria.

```
> STORE 2
>
```

Emmagatzema els valors del estat actual del DDSBee dins de la memòria EEPROM número 2. Aquesta opció guarda els valors dels registres inclús l'estat en que es troben (en marxa o no).

Programa el sistema perquè es posi en funcionament amb els paràmetres emmagatzemats.

```
> PBOOT 2
>
```

Estableix que la propera vegada que el mòdul es posi en marxa ho farà amb els paràmetres que contingui la memòria d'usuari EEPROM número2.

Comprovar el mode d'arrancada de nou.

```
> SBOOT
Boot: preset #2
>
```

Ens indica que el dispositiu s'inicia en mode de preset d'usuari número 2.

Si ara desconnectem l'equip i el tornem a posar en funcionament ens trobarem que arrenca directament amb el valor programat anteriorment en el registre 1 (1100 Hz.) que estava habilitat en el moment que es va emmagatzemar.

6. Aplicacions

En aquest capítol són presentats una sèrie de propostes per la realització d'aplicacions, basades amb el circuit oscil·lador DDS microcontrolat propòsit d'aquest projecte.

Abans de entrar en el detall de cada una de les aplicacions, cal comentar una característica que s'ha pogut comprovar en la realització de les aplicacions. Es tracta de la tolerància en la velocitat de transmissió sèrie que poden presentar alguns dispositius i que pot ser una font d'errors en la transferència de dades cap a el nostre dispositiu. Segons els càlculs previs realitzats per tal d'avaluar el comportament del circuit de transmissió sèrie, l'error teòric és de $1,37\%$, mentre que el error empíric que s'ha mesurat està entre 1.19% i 1.28%. Aquest error és raonable per obtenir una comunicació de forma correcta.

No obstant això, la comunicació amb d'altres dispositius pot necessitar ser ajustada en precisió, ja que podem trobar errors de transmissió en d'altres dispositius que superin el 5%.

Aquest problema s'ha trobat al realitzar la comunicació amb un Arduino UNO, prèviament programant la seva velocitat a través del seu IDE.

Per fer qualsevol tipus d'aplicació els muntatges a realitzar són extraordinàriament simples, ja que la complexitat s'ha delegat al DDSBee.

6.1. Oscil·lador de precisió amb Arduino

Per realitzar amb Arduino i DDSBee un sistema que pugui generar un senyal en un marge de freqüències reduït i que sigui modificable manualment, simplement cal disposar d'una placa XBee Shield i afegir un potenciòmetre multi-volta amb el cursor connectat a una entrada analògica (en aquesta aplicació d'exemple serà la A0) i els altres dos terminals entre positiu i negatiu. Aquest muntatge és el que es mostra a la figura 6.1

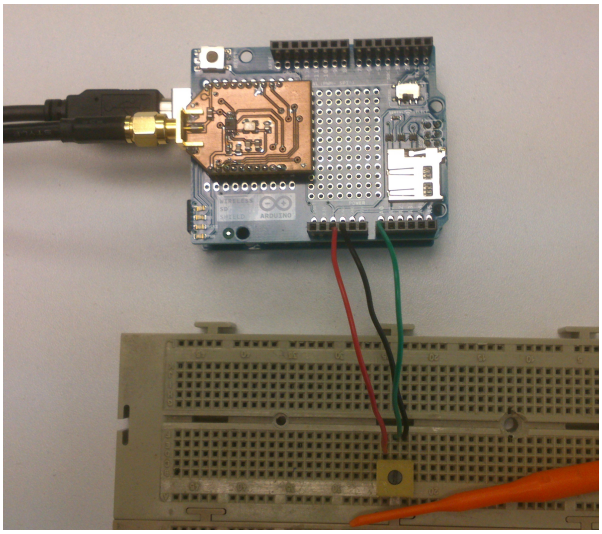


Figura 6.1.: Muntatge simple amb DDSBee i Arduino per fer un oscil·lador controlat manualment amb precisió

Per aquesta aplicació de Arduino s'ha escollit realitzar un programa en el propi IDE de Arduino, utilitzant els recursos i simplicitat que aquesta plataforma ofereix. El codi que s'utilitzarà és el següent:

```
/*****************************
  Oscil lador de precissió
-----------------------------
 Utilitzant un potenciòmetre
multi-volta connectat en
l'entrada analògica A0, es
pot ajustar amb precissió la
freqüència de sortida en el
rang: F_MIN fins F_MAX.
*****************************/
#define F_MIN 1000
#define F_MAX 2000

int value, ant_value;
char my_msg[30];

void setup(){
  /* el valor de Serial.begin
     pot requerir ser ajustat
     amb cura */
  Serial.begin(115200);
  delay(1000);
  serial_tx("SINE");
  delay(1000);
}


void loop(){
  value=analogRead(0);
  value=map(value,0,1023,F_MIN,F_MAX);
  if (value!=ant_value){
```

```
      ant_value=value;
      sprintf(my_msg,"SRFR0 %d", value);
      serial_tx(my_msg);
      delay(200);
      }
      delay(50);
  }

  void serial_tx(char * my_text){
    /* Rutina d'enviament per el port sèrie */
    Serial.print(my_text);
    Serial.print("\n");
  }
```

En la funció *setup()* l'únic que es fa és configurar la velocitat del port sèrie i establir que la forma d'ona serà sinusoïdal. La funció *loop()* s'encarregarà de llegir contínuament el valor de l'entrada analògica, convertir el valor obtingut en un valor de freqüència i comparar-lo amb l'anterior valor. En cas que siguin valors diferents, actualitza el valor de freqüència a través de enviar la comanda *SRFR0*.

6.2. Emissor de telegrafia en 137kHz

Existeix la possibilitat d'implementar un petit transmissor de telegrafia per experimentar dins d'alló que es conneix com a *LowFER* [25](Low-Frequency Experimental Radio). En el cas del nostre exemple es proposa fer un transmissor de telegrafia per la banda de radio aficionats de 137kHz[26].

Per fer-ho, en aquest cas el funcionament és autònom i únicament cal pre-definir el mode de treball. En el nostre cas la configuració és la següent:

```
> SASK 136900
> EEOC
> STORE
> PBOOT
```

Aquesta configuració garanteix l'estat dels registres del DDSBee en quant es posi en marxa de nou.

EL muntatge que cal fer és molt simple i nomes cal incorporar una manipulador de telegrafia (o bé un polsador) i una resistència de pull-down (en l'exmple és de 47K) connectats en el terminal 9 del DDSBee, com es mostra en la imatge 6.2.

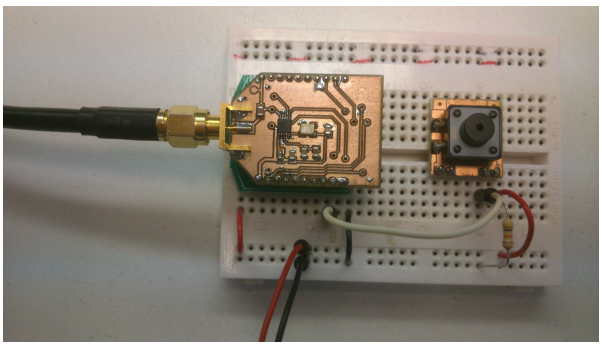


Figura 6.2.: Muntatge simple amb DDSBee per realitzar un senzill transmissor LowFER

Està clar que només canviant el valor amb el qual es parametritza la comanda *SASK* podem alterar la freqüència on es transmet.

6.3. Transmissor AFSK i desmodulació amb l'aplicació minimodem

El programa minimodem és una petita aplicació que permet desmodular diferents tipus de senyals modulats en freqüència, entre ells: Bell-103, RTTY, NOAA-SAME, Caller-ID i Bell-202.

Utilitzarem aquesta aplicació en el mode Bell-202 per provar el funcionament del DDSBee com a modulador. Primer cal configurar el DDSBee amb les següents instruccions:

```
> SFSK 1200 2200
> EEOC
> SFSC
```

En aquestes condicions ja tenim la configuració bàsica per tal de posar en marxa el dispositiu.

Per activar la sortida del DDSBee s'ha de posar el terminal 9 (EN_OUT) en una transició a estat alt.

Utilitzant un adaptador USB a UART TTL[1] connectarem la sortida de dades d'aquest adaptador a l'entrada REG_CHG (terminal 13) del DDSBee.

També cal connectar la sortida del mòdul DDSBee cap a l'entrada de micròfon del ordinador. El muntatge que s'ha realitzat és el que apareix a la següent fotografia.

[1] UMFT230XB-01 del fabricant FTDI Chip http://www.ftdichip.com

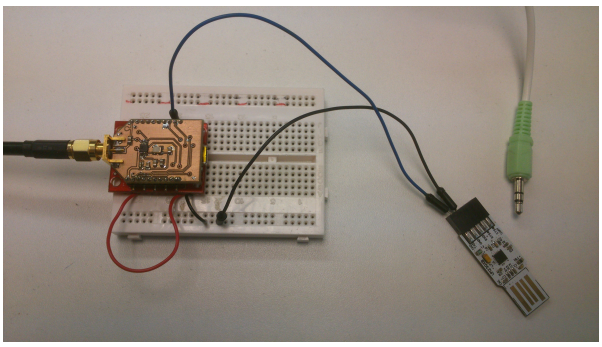


Figura 6.3.: Muntatge amb DDSBee per realitzar un modulador AFSK i desmodular amb ordinador i programa minimodem

En aquestes condicions obrim un terminal de control sèrie[2] configurat a 1200 bps (és la velocitat estàndard de Bell-202) i posem en marxa el programa minimodem amb la instrucció *minimodem -r 1200*. Si enviem text des de el terminal de control sèrie es rep des de el minimodem i el mostra per pantalla de la següent forma.

```
joan@portatil-asus:~$ minimodem -r 1200
### CARRIER 1200 @ 1200.0 Hz ###
Un carro carregat de rocs, corria per la carretera.

### NOCARRIER ndata=52 confidence=4.605 ampl=0.301 bps=1200.00 (rate perfect
    ) ###
```

A més d'aquesta petita prova, s'ha fet una transferència completa d'un fitxer de text de més de 22 KBytes, sense cap error.

6.4. Pràctiques de transmissió de dades a freqüències ultrasòniques

El món de les transmissions ultrasòniques és molt interessant des de el punt de vista educatiu:

- Són econòmiques de transmetre. Només cal aire com a via de propagació de les ones.
- Al estar per sobre dels llindars de la oïda humana, no generen molèsties dins d'un grup que estigui efectuant pràctiques. En contra del que passaria si les freqüències es trobessin dins del rang de la oïda.

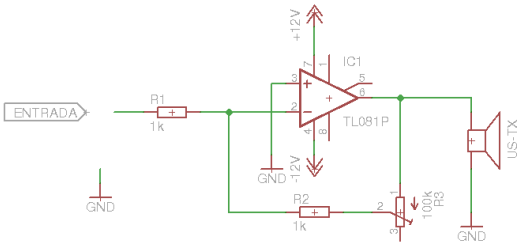
Per contra d'aquestes característiques existeix el problema de poder-les generar amb per exemple, un ordinador, ja que els filtres de sortida de les targetes de àudio retallen fortament les altes freqüències.

Per generar freqüències en el rang del ultrasons es poden utilitzar els DDSBee, que poden adaptar-se a practiques de laboratori.

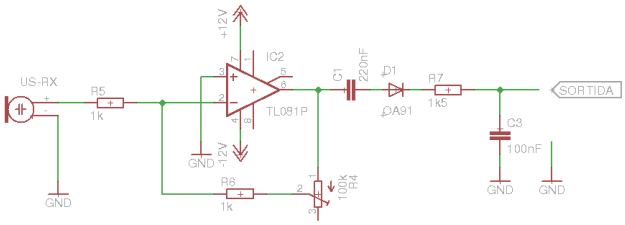
L'aplicació que és recull en aquest apartat descriu la transmissió de dades modulades en ASK utilitzant ultrasons. Per fer aquesta aplicació, s'ha necessitat utilitzar circuiteria externa per realitzar un emissor per una banda i un receptor en l'altra. Per els dos circuits s'han construït dos amplificadors de guany variable. En el amplificador que ha d'anar a la part de receptor se

[2]Es pot utilitzar qualsevol. Per aquesta prova s'ha utilitzat el *Cutecom*

li ha afegit un senzill detector d'envolupant. A la Figura 6.4a es mostra el circuit transmissor i a la figura 6.4b el circuit receptor.



(a) Transmissor d'ultrasons



(b) Receptor d'ultrasons

Figura 6.4.: Circuits transmissor i receptors d'ultrasons.

A l'entrada del circuit transmissor es connectaria el DDSBee amb la següent configuració:

```
> SASK 40000
> EEOC
```

Els transductors *US-TX* i *US-RX* són els que es poden veure a la figura 6.5. Aquests estan centrats en 40 kHz. i tenen un ample de banda estret. Per això i donada la simplicitat del circuit desmodulador, s'ha triat poder treballar amb una modulació ASK.

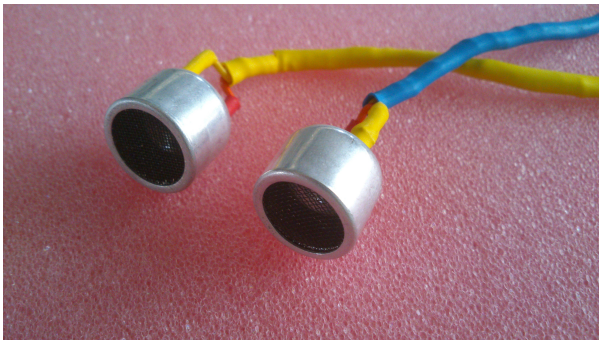


Figura 6.5.: Transductors d'ultrasons per 40kHz.

Els senyals obtinguts dels processos de transmissió i recepció són els que es mostren a la figura 6.6.

- El senyal *BIT*, són les dades que entren directament al terminal de control del DDSBee (pin 9), provinents de la transmissió d'un arxiu per port sèrie.

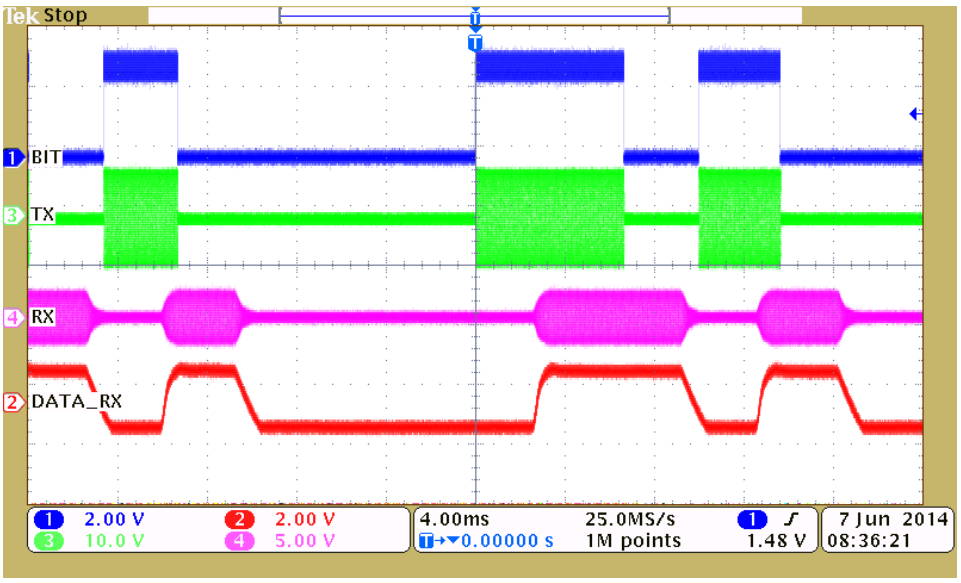


Figura 6.6.: Transmissió i recepció d'ultrasons

- El senyal *TX*, correspon a les dades modulades en amplitud per el DDSBee i amplificades.
- El senyal *RX*, és el senyal que s'obté en el receptor, després de ser amplificat 100 vegades.
- El *DATA_RX* és la senyal desmodulada amb el detector d'envolupant. La sortida del receptor pot anar directament a un terminal receptor de dades o bé fer passar la senyal de sortida per un circuit comparador i així acondicionar la forma d'ona de forma més rectangular.

Podem veure el petit desfasament temporal que hi ha entre l'etapa de transmissió i la de recepció. Aquest fet es deu a la lenta propagació de les ones acústiques en l'aire i la distància que hi havia entre emissor i receptor en el moment de fer aquesta aplicació.

7. Conclusions

El projecte que s'ha realitzat ha complert amb els objectius marcats en el seu inici. Per una banda, s'ha pogut integrar un sistema DDS i un microcontrolador en una placa de la mida d'un circuit XBee esdevenint compatible amb la distribució de terminals d'aquests mòduls. Per una altra banda s'han creat una interfície d'usuari molt còmode que pot contribuir a un desenvolupament àgil de productes electrònics diversos, ja que el camp d'aplicació dels oscil·ladors és molt extens.

És important remarcar el joc de comandes implementades i ben documentades en el manual d'usuari 5.6. Aquestes proveeixen d'una gran possibilitat de funcions per tal de programar el comportament del dispositiu. El present projecte descriu eficientment com s'ha estructurat la programació del dispositiu. D'aquesta forma resulta senzill poder ampliar el conjunt de comandes que es poden executar.

Per últim s'han presentat diferents exemples en el manual d'usuari (secció 5.7) i aplicacions d'ús (capítol 6) que mostren algunes possibilitats en les quals el dispositiu es susceptible de ser utilitzat. Sabent que una flor no fa primavera i que aquestes aplicacions són només una mostra del que el concepte *DDSBee* pot fer, aquest projecte pot contribuir en diferents camps (educatiu, enginyeria, indústria o el món "geek") d'una forma àgil i eficient.

Els objectius personal també han quedat satisfets, ja que s'han superat les dificultats - tant tècniques com personals- que han sorgit durant el desenvolupament d'aquest projecte. Igualment s'ha obert una porta nova, que condueix a fer evolucionar a aquest projecte i poder ampliar el concepte del DDSBee, tal com es detalla en el següent capítol.

8. Futures línies de treball

Durant el desenvolupament d'aquest projecte han aparegut moltes idees no previstes en el moment de marcar els objectius. Algunes d'aquestes s'han pogut anar afegint al resultat que s'ha obtingut, i d'altres han quedat a la reserva, pendents de poder realitzar futures millores. En concret i a nivell tècnic es proposen seguir les següents línies de treball:

Implementació de formes d'ona per Firmware: Es poden ampliar per software les formes d'ona que pot generar el dispositiu, ja que s'ha previst la possibilitat de comprovar la sortida a través d'un terminal ADC del microcontrolador i d'aquesta manera es pot controlar els canvis de fase del DDS i generar formes d'ona *a la carta*. Per exemple, dents de serra a partir d'una forma d'ona triangular, canviant la fase 180^{o} en el moment en que es detecti un canvi de sentit.

Accelerar el procés d'atenció a les interrupcions: Cal optimitzar l'execució de les interrupcions externes per tal de poder fer modulacions a més alta velocitat de la que actualment es pot generar. Aquesta línia de millora futura s'ha de afrontar tant des de el punt de vista de l'optimització de codi com de l'acceleració del microcontrolador. Accelerar el rellotge del microcontrolador fins al límit possible, aporta una major velocitat del procés d'atenció a la interrupció (guardar valors a la pila i passar el control a la ISR), com també es millora la transmissió de dades entre el microcontrolador i el oscil·lador DDS.

Script per muntar noves comandes: Per tal d'estalviar memòria de dades SRAM, ha estat necessària una re-organització del codi del projecte i les funcions-comanda queden aïllades dels texts d'ajuda i del propi nom de la comanda a executar. S'ha d'implementar un script que permeti a un programador, escriure de forma lineal les funcions a incorporar i en el moment de fer la programació del firmware del dispositiu que aquestes funcions quedin distribuïdes en les corresponents ubicacions dins dels mòduls de manera automàtica.

Investigar Ús com a Desmodulador: El gran potencial del DDS és la facilitat d'ús que ofereix per ser utilitzat com a modulador. Però per tal de poder tenir un complert sistema MODEM seria necessari disposar d'un sistema capaç de desmodular senyals codificades. S'ha d'obrir una línia d'investigació per tal de veure si és possible i, si és així, fins quin punt es pot utilitzar les entrades analògiques del propi microcontrolador per fer-ne ús per aquest propòsit. Aquesta opció ja s'ha previst en el disseny físic, implementant accés a entrades analògiques directament.

Contemplar la possibilitat de treballar per sobre dels 12.5MHz: A través del ús dels àlies de freqüència es possible generar senyals per sobre dels 12.5MHz. Per poder utilitzar senyals en aquests rangs de freqüència s'ha de contemplar en la programació del dispositiu, així com implementar un filtratge passa-alt o passa-banda efectiu. Aquest filtratge pot ser intern o extern, en cas que per manca d'espai no sigui realitzable.

Experimentar amb d'altres sistemes DDS: Les restriccions econòmiques en quant al cost del producte final i de disponibilitat d'espai físic en el format emprat, van condicionar en certa manera el treballar amb un DDS amb una freqüència màxima de 12.5MHz. En el mercat existeixen altres circuits oscil·ladors DDS que arriben al rang de UHF. Sense les restriccions de cost i espai anteriorment esmentades, serà interessant experimentar en el rang de VHF i UHF, dotant al circuits d'interfícies com la realitzada en aquest treball.

En un altre ordre de coses, una empresa de nova creació de Vilaseca està interessada en aquest dispositiu per ser afegit en un equip industrial de futura construcció, a més s'està estudiant la possibilitat de comercialitzar aquest producte i ja s'han iniciat converses amb l'empresa *Libelium*[1] de Saragossa.

[1] Libelium Comunicaciones Distribuidas S.L. http://www.libelium.com/

A. annexos

A.1. Fonts de programari

A.1.1. Programa principal

Mòdul main

Aquest mòdul té la finalitat de inicialitzar el sistema i esperar comandes des de el port sèrie.

```
#include <stdio.h>
#include <stdbool.h>
#include <avr/pgmspace.h>

#include "serial.h"
#include "shell.h"
#include "flasherror.h"

#define PROMT "> "              // Estableix promt de la línia de comandes
#define CENTINELLA '\n'         // Estableix caràcter utilitzat com a
    centinella
#define LEN_MSG_MAX 30          // Estableix la mida màxima dels missatges


uint8_t msg_entrada[LEN_MSG_MAX];
uint8_t * pnt_msg_e=msg_entrada;
uint8_t msg_processat[LEN_MSG_MAX];
uint8_t * pnt_msg_p=msg_processat;
uint8_t punter=0;

static void setup(void);
static void loop(void);

int main(void){
  /*******************************
        PROGRAMA PRINCIPAL
        ------------------
    Estructura el funcionament del
   programa en la funció setup() i
   la funció loop()
  *******************************/
  setup();
  while(true) loop();
  return 0;
}

static void setup(void){
  /***************************
         CONFIGURACIONS
         --------------
    Configura les característiques de
```

```
    comunicació i els diferents mòduls
    per treballar amb ells.
    ***************************/
    serial_init();
    shell_init();
    serial_print(PROMT);
}

static void loop(void){
    /***************************
      PROGRAMA CÍCLIC PRINCIPAL
      -------------------------
    ***************************/

    int8_t value;
    char buffer[20];

    if(serial_can_read()){
      uint8_t c= serial_get();
      if(c!=CENTINELLA){
        msg_entrada[punter++] = c;
      }else{
        msg_entrada[punter] = 0;
        value=process_command(pnt_msg_e);
        if(value<0){
    value=1+~value;
    GET_ERROR(buffer,value);
    serial_print(buffer);
        }
        punter=0;
        serial_print(PROMT);
      }
    }
}
```

Mòdul de missatges de error en Flash

Aquest mòdul té la funcionalitat de deixar lliure la memòria SRAM del AVR, forçant a mantenir les cadenes de text constants dels missatges de error, en memòria de programa Flash.

Dins del mòdul s'implementen les definicions necessàries i les funcions macros, per tal de que el usuari pugui cridar a determinades funcions i adreces de memòria FLASH de forma simple, obviant el fet que es tingui d'utilitzar l'accés específic aquest espai de memòria.

```
#ifndef _FLASHERROR_
#define _FLASHERROR_


  /* Aquestes estructures es deixen en memòria
     de programa FLASH i no han de  ocupar
     memòria SRAM de dades*/



#define GET_ERROR(punter,ierror)  strcpy_P(punter,(PGM_P)pgm_read_word(&(
    error_msg[ierror])))
```

```
      /*   Codis d'error    */
const char error_0[] PROGMEM="No error";
const char error_1[] PROGMEM="Unknown command";
const char error_2[] PROGMEM="Wrong parameters";
const char error_3[] PROGMEM="Out of range";
const char error_4[] PROGMEM="Not allowed";

PGM_P const error_msg[]PROGMEM =
{
    error_0,
    error_1,
    error_2,
    error_3,
    error_4
};



#endif
```

A.1.2. Gestió del port sèrie

Dins d'aquest apartat hi han dos mòduls diferents. Per una banda el mòdul del port sèrie i per altra el mòdul que gestiona la cua del port.

Mòdul serial

Aquest mòdul és l'encarregat de fer l'accés al port sèrie.

```
/*
*    This library is free software; you can redistribute it and/or
*    modify it under the terms of the GNU Lesser General Public
*    License as published by the Free Software Foundation; version
*    2.1 of the License.
*
*    This library is distributed in the hope that it will be useful,
*    but WITHOUT ANY WARRANTY; without even the implied warranty of
*    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
*    Lesser General Public License for more details.
*
*    You should have received a copy of the GNU Lesser General Public
*    License along with this library; if not, write to the Free Software
*    Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301
   USA
*


 ************************************************
 *  Nom:  Serial
 *
 *  Autor:  Joan Martínez
```

```
 *
 *  Descripcio: Mòdul implementació control sèrie
 *
 ************************************************/



#ifndef SERIAL_H
#define SERIAL_H

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

#define CENTINELLA '\n'

void serial_init(void);
uint8_t serial_get(void);
void serial_put(const uint8_t c);
bool serial_can_read(void);
uint8_t serial_print(const uint8_t *msg);
uint8_t serial_write(const uint8_t *msg);

#endif
```

```
#include <util/setbaud.h>
#include <avr/interrupt.h>
#include <avr/io.h>
#include <avr/sfr_defs.h>

#include "queue.h"
#include "serial.h"



//-------------------------------------------------------------

static queue_t recieve_q;
static queue_t transmit_q;

//-------------------------------------------------------------

void serial_init(void){
  /*Inicialitza el modul i deixa la UART a punt per enviar/rebre
    caracters de 8 bit, amb 1 bit  d stop , sense paritat
    i en mode asíncron.
    La velocitat de transmissió es defineix de forma automàtica
    a través de les macros que incorpora el mòdul <util/setbaud>
    i el arxiu Makefile que defineix els parametres de velocitat
    de CPU i velocitat de BAUD  */

   UBRR0H = UBRRH_VALUE;
   UBRR0L = UBRRL_VALUE;
   UCSR0A = 32;
#if USE_2X
   UCSR0A |= (1 << U2X0);
#else
   UCSR0A &= ~(1 << U2X0);
```

```
#endif

  UCSR0C = _BV(UCSZ01) | _BV(UCSZ00);                          //Mode,
      paritat,bits d'estop i mida
  UCSR0B = _BV(RXEN0) | _BV(TXEN0) | _BV(RXCIE0) | _BV(UDRIE0);//Enable rx i
       tx amb interrupció de rx i UDRE
  queue_empty(&recieve_q);
  queue_empty(&transmit_q);
  UCSR0B |= _BV(RXCIE0);     //inicialment habilitem interrució de recepció
  UCSR0B &= ~(_BV(UDRIE0));  //inicialment deshabilitem interrupció de
      transmissió
}

//------------------------------------------------------------

uint8_t serial_get(void){
  /*Retorna un byte llegit del port sèrie. Es bloqueja indefinidament
    fins que hi ha un caràcter disponible per a ser llegit*/
  uint8_t r;
  while(queue_is_empty(&recieve_q)){;}
  r = queue_front(&recieve_q);
  queue_dequeue(&recieve_q);
  return r;
}

//------------------------------------------------------------

void serial_put(const uint8_t c){
  /*Envia un byte pel port sèrie. En cas que estigui ocupat enviant
    una altra dada, es bloqueja fins que  l enviament  en curs acaba */
  while(queue_is_full(&transmit_q)){;}
  queue_enqueue(&transmit_q, c);
  if (!(UCSR0B & _BV(UDRIE0))) UCSR0B |= _BV(UDRIE0);

}

//------------------------------------------------------------

bool serial_can_read(void){
  /*Retorna true si hi ha un caràcter disponible per a ser llegit*/
  return !queue_is_empty(&recieve_q);
}

//------------------------------------------------------------
uint8_t serial_print(const uint8_t *msg){
  /*******************************
     Funció que crida a la funció
    'serial_put()' per cada un dels
    elements continguts en l'apuntador
    *msg, fins trobar el caràcter \0.
    Retorna el nombre de caràcters
    enviats, sense comptar CENTINELLA.
  *******************************/
  uint8_t i;
  for(i=0; msg[i]!=0;i++)
    serial_put(msg[i]);
  serial_put(CENTINELLA);
```

```
  return i;
}

uint8_t serial_write(const uint8_t *msg){
  /*****************************
     Funció que crida a la funció
   'serial_put()' per cada un dels
   elements continguts en l'apuntador
   *msg, fins trobar el caràcter \0.
   Retorna el nombre de caràcters
   enviats. Permet concatenar sortides
   per el port serie.
  ******************************/
  uint8_t i;
  for(i=0; msg[i]!=0;i++)
    serial_put(msg[i]);
  return i;
}



//------------------------------------------------------------

ISR(USART_RX_vect){
  /*Rutina d'atenció a la interrupció de recepció.
   Encua el byte rebut a la cua de recepció.*/
  queue_enqueue(&recieve_q, UDR0);
}


ISR(USART_UDRE_vect){
  /*Rutina d'atenció a la interrupció de registre de dades buit.
   Desencua el primer byte de la cua de transmissió i l'envia.*/
  uint8_t r;
  r = queue_front(&transmit_q);
  queue_dequeue(&transmit_q);
  UDR0 = r;
  if (queue_is_empty(&transmit_q)) UCSR0B &= ~(_BV(UDRIE0));
}
```

Mòdul queue

Aquest mòdul implementa una cua per fer les transferencies a través del port sèrie UART.

```
/*
*    This library is free software; you can redistribute it and/or
*    modify it under the terms of the GNU Lesser General Public
*    License as published by the Free Software Foundation; version
*    2.1 of the License.
*
*    This library is distributed in the hope that it will be useful,
*    but WITHOUT ANY WARRANTY; without even the implied warranty of
*    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
*    Lesser General Public License for more details.
*
*    You should have received a copy of the GNU Lesser General Public
```

```
*    License along with this library; if not, write to the Free Software
*    Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301
    USA
*


 ************************************************
 *  Nom:  QUEUE
 *
 *  Autor:  Joan Martínez
 *
 *  Descripcio: Implementació de cues
 *
 ************************************************/



#ifndef QUEUE_H
#define QUEUE_H

#include <stdbool.h>
#include <inttypes.h>


#define MIDA_BUFF 30

typedef struct
{
  uint8_t queue[MIDA_BUFF];
  uint8_t writePointer;
  uint8_t readPointer;
  uint8_t cont;
} queue_t;

void queue_empty(queue_t *const q);
bool queue_is_empty(const queue_t *const q);
bool queue_is_full(const queue_t *const q);
void queue_enqueue(queue_t *const q, uint8_t v);
void queue_dequeue(queue_t *const q);
uint8_t queue_front(const queue_t *const q);

#endif
```

```
#include <stdbool.h>
#include <inttypes.h>
#include <avr/io.h>
#include <util/atomic.h>
#include "queue.h"

void queue_empty(queue_t *const q){
    q->readPointer= 0;
    q->writePointer=0;
    q->cont=0;
}

bool queue_is_empty(const queue_t *const q){
```

```
  //return (q->writePointer==q->readPointer);   //TODO: He de comprobar si
      esta full???
  return ((q->cont)==0);
}

bool queue_is_full(const queue_t *const q){

  return (q->cont==MIDA_BUFF-1);
}

void queue_enqueue(queue_t *const q, uint8_t v){
  ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
  {
    if (!queue_is_full(q)) {

      q->queue[q->writePointer]=v;
      /*Si al incrementar el punter sobrepasem la capacitat torna un zero*/
      //++(q->writePointer)==MIDA_BUFF? 0: ++(q->writePointer);
      int a=(q->writePointer)+1;
      a==MIDA_BUFF ? (q->writePointer)=0: ((q->writePointer)+=1);
      (q->cont)+=1;
    }
  }
}

void queue_dequeue(queue_t *const q){
  ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
  {
    if (!queue_is_empty(q)) {
      /*Si al incrementar el punter sobrepasem la capacitat torna un zero*/
      //++(q->readPointer)==MIDA_BUFF? 0: ++(q->readPointer);
      int b=(q->readPointer)+1;
      b==MIDA_BUFF ? (q->readPointer)=0: ((q->readPointer)+=1);
      (q->cont)-=1;
    }
  }
}

uint8_t queue_front(const queue_t *const q){
  ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
  {
    if (queue_is_empty(q))
      return 0;
    else
      return q->queue[q->readPointer];
  }
}
```

A.1.3. Gestió de comandes

Dins d'aquest apartat hi han dos mòduls diferents. Per una banda el mòdul del Shell, que implementa un pas de comandes i per altra el mòdul que gestiona la execució de les comandes, anomenat cli.

Mòdul shell

Aquest mòdul implementa la inicialització de les variables que han de contenir les comandes i per altra banda el pas de comandes cap a el mòdul que ha d'interpretar-les.

```
/*
*    This library is free software; you can redistribute it and/or
*    modify it under the terms of the GNU Lesser General Public
*    License as published by the Free Software Foundation; version
*    2.1 of the License.
*
*    This library is distributed in the hope that it will be useful,
*    but WITHOUT ANY WARRANTY; without even the implied warranty of
*    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
*    Lesser General Public License for more details.
*
*    You should have received a copy of the GNU Lesser General Public
*    License along with this library; if not, write to the Free Software
*    Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301
    USA
*

 ************************************************
 *  Nom:  SHELL
 *
 *  Autor:  Joan Martínez
 *
 *  Descripcio:   Aquest mòdul implementa les eines
 *              per tal de interpretar les comandes
 *              lliurades al microcontrolador
 ************************************************/

#ifndef _SHELL_
#define _SHELL_



void shell_init(void);
int8_t process_command(const uint8_t *command);

#endif
```

```
#include <stdio.h>
#include <ctype.h>
#include <avr/eeprom.h>

#include "shell.h"
#include "cli.h"
#include "gestirq.h"

char comanda[8];
uint32_t arg_lst[3];
int arg_c;
void * arg_v[4];

static uint8_t boot_device(void);
```

```
int8_t process_command(const uint8_t *command){
  if (*command==0) return 0;
  arg_c=0;
  arg_c=sscanf(command,"%7s %lu %lu %lu",arg_v[0],arg_v[1],arg_v[2],arg_v
      [3]);
  if (!arg_c) return 0;
  else{
    return exec_command(arg_v[0]);
  }
}

void shell_init(void){
  arg_v[0]=comanda;
  arg_v[1]=&arg_lst[0];
  arg_v[2]=&arg_lst[1];
  arg_v[3]=&arg_lst[2];
  uint8_t bootv=boot_device();
  cli_init(bootv);
  gestirq_init();
}

static uint8_t boot_device(void){
  /* Llegeix el contingut de la posició BOOTVAR_BYTE
     de la E2PROM. Si el valor és 0 carrega
     el mode de funcionament per defecte.
     Si no, carrega el preset. */
  uint8_t bootvar = eeprom_read_byte((uint8_t*)BOOTVAR_BYTE);
  if(bootvar==255){
    /* Aquesta part asegura que l'equip funcionarà
       despres de ser flasexat, asegurant el valor 0
       dins del BOOTVAR_BYTE */
    eeprom_write_byte((uint8_t*)BOOTVAR_BYTE, 0);
    bootvar=0;
  }
  return bootvar;
}
```

Mòdul cli

En aquest mòdul hi han implementat la funció que ha d'executar les comandes, així com totes les funcions-comanda que pot executar, incloent-hi comandes de programació, de comportament, d'ajut i d'altres.

El mòdul inclou un mecanisme per mantenir ocultes determinades funcions. Aquest fet és especialment útil per disposar d'un conjunt d'instruccions d'ús avançat, de test, de control o bé de debugging. Aquestes funcions poden estar o no documentades per tal de mantenir-ne la privacitat del seu ús per part dels usuaris estàndards.

```
/*
*    This library is free software; you can redistribute it and/or
*    modify it under the terms of the GNU Lesser General Public
*    License as published by the Free Software Foundation; version
*    2.1 of the License.
*
*    This library is distributed in the hope that it will be useful,
*    but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
*     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
*     Lesser General Public License for more details.
*
*     You should have received a copy of the GNU Lesser General Public
*     License along with this library; if not, write to the Free Software
*     Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301
    USA
*

 ************************************************
 *  Nom:  CLI
 *
 *  Autor:  Joan Martínez
 *
 *  Descripcio:   Aquest mòdul implementa les comandes,
 *              tal com s'han de executar.
 *              (COMMAND LINE INTERFACE)
 ************************************************/

#ifndef _CLI_
#define _CLI_

/* Defineix la posició on s'emmagatzema
   el procediment de boot del dispositiu */
#define BOOTVAR_BYTE 511

void cli_init(const uint8_t inici);
int8_t exec_command(const uint8_t * command);

#endif
```

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <avr/io.h>
#include <util/delay.h>
#include <avr/eeprom.h>
#include <avr/pgmspace.h>

#include "flashtxt.h"
#include "cli.h"
#include "serial.h"
#include "dds.h"

/* Definició dels desplaçaments (SHIFT's) en que
   es troben els diferents valors en una EEPROM,
   a partir d'una adreça base*/
#define SFT_FR0 0
#define SFT_FR1 4
#define SFT_PH0 8
#define SFT_PH1 10
#define SFT_CTRL 12
#define SPAN_M 20
/* Nombre màxim de presets permessos */
#define PRST_MAX 3

extern int arg_c;
```

```
extern void * arg_v[4];


/* Llista de comandes  */

typedef int8_t (*operation_t)(void);

typedef struct{
  operation_t fnc;
}cli_command_t;

/*  Comandes que s'executen com a funcions privades a aquest mòdul */
static int8_t command_setrunfr0(void);
static int8_t command_setrunfr1(void);
static int8_t command_setfr0(void);
static int8_t command_setfr1(void);
static int8_t command_setph0(void);
static int8_t command_setph1(void);
static int8_t command_sine(void);
static int8_t command_square(void);
static int8_t command_triangle(void);
static int8_t command_enableControl(void);
static int8_t command_disableControl(void);
static int8_t command_setfsk(void);
static int8_t command_setask(void);
static int8_t command_setpsk(void);
static int8_t command_setqam(void);
static int8_t command_bell202(void);
static int8_t command_bell103(void);
static int8_t reset_on_init(void);
static int8_t preset_on_init(void);
static int8_t status_boot(void);
static int8_t help_fnc(void);
static int8_t command_store(void);
static int8_t command_load(void);

/* Funcions ocultes i de test */
static int8_t set_ddsraw(void);
static int8_t print_controlreg(void);
static int8_t nofares(void);
static int8_t command_test1(void);
static int8_t command_test2(void);
static int8_t close_encounters(void);

/* Comandes/Funcions interrupció */
static void frequency0(void);
static void frequency1(void);
static void enable0(void);
static void enable1(void);
static void disable_out(void);
static void start_out(void);
static void phase0(void);
static void phase1(void);
static void qam0_0(void);
static void qam0_1(void);
static void qam1_0(void);
static void qam1_1(void);
```

```
static int8_t load_from_e2prom(uint8_t inici);

const cli_command_t commands[]={
        /* Comandes d'actuació */
  {disable_out},
  {enable0},
  {enable1},
  {phase0},
  {phase1},
  {command_setrunfr0},
  {command_setrunfr1},

        /* Comandes de programació */
  {command_setfr0},
  {command_setfr1},
  {command_setph0},
  {command_setph1},
  {command_sine},
  {command_square},
  {command_triangle},

        /* Comandes de comportament */
  {command_enableControl},
  {command_disableControl},
  {nofares},
  {nofares},
  {command_setfsk},
  {command_setpsk},
  {command_setask},
  {command_setqam},   // EXPERIMENTAL:  "Set QAM Mode"
  {command_bell202},
  {command_bell103},

         /* Altres Comandes */
  {help_fnc},
  {command_store},
  {command_load},
  {reset_on_init},
  {preset_on_init},
  {status_boot},

        /* Comandes ocultes i  per el procés de debugging   */
  {command_test1},
  {command_test2},
  {close_encounters},
  {print_controlreg},
  {set_ddsraw},

  //Afegir nous comandaments aqui.
  {NULL}
};

/**********************
 *  Funcions-Comanda  *
 **********************/
```

```
static int8_t command_enableControl(void){
  /* Comanda que habilita l'execució
     d'interrupcions per INT0 */
  INT0_enable(true);
  return 0;
}

static int8_t command_disableControl(void){
 /* Comanda que deshabilita l'execució
     d'interrupcions per INT0 */
  INT0_enable(false);
  return 0;
}

static int8_t command_sine(void){
  /* Estableix que la sortida sigui senoidal */
  DDS_Active();
  DDS_Sine();
  DDS_Idle();
  return 0;
}
static int8_t command_square(void){
  /* Estableix que la sortida sigui cuadrada */
  DDS_Active();
  DDS_Square();
  DDS_Idle();
  return 0;
}

static int8_t command_triangle(void){
  /* Estableix que la sortida sigui triangular */
  DDS_Active();
  DDS_Triangle();
  DDS_Idle();
  return 0;
}

static int8_t command_setrunfr0(void){
  /* Comprova que la funció està acompanyada d'un
     únic argument i posa en marxa el registre de
     freqüència 0 amb el valor passat com a paràmetre */
  if (arg_c==2){
    DDS_Active();
    int8_t value=DDS_SetFreq_0(*(uint32_t *)arg_v[1]);
    DDS_Start_0();
    DDS_Idle();
    return value;
  }
  else
    return -2;
}

static int8_t command_setfr0(void){
  /* Comprova que la funció està acompanyada d'un
     únic argument i programa el registre de
     freqüència 0 amb el valor passat com a paràmetre */
  if (arg_c==2){
```

```
    DDS_Active();
    int8_t value=DDS_SetFreq_0(*(uint32_t *)arg_v[1]);
    DDS_Idle();
    return value;
  }
  else
    return -2;
}

static int8_t command_setrunfr1(void){
  /* Comprova que la funció està acompanyada d'un
     únic argument i posa en marxa el registre de
     freqüència 1 amb el valor passat com a paràmetre */
  if (arg_c==2){
    DDS_Active();
    int8_t value=DDS_SetFreq_1(*(uint32_t *)arg_v[1]);
    DDS_Start_1();
    DDS_Idle();
    return value;
  }
  else
    return -2;
}

static int8_t command_setfr1(void){
  /* Comprova que la funció està acompanyada d'un
     únic argument i programa el registre de
     freqüència 1 amb el valor passat com a paràmetre */
  if (arg_c==2){
    DDS_Active();
    uint8_t value=DDS_SetFreq_1(*(uint32_t *)arg_v[1]);
    DDS_Idle();
    return value;
  }
  else
    return -2;
}

static int8_t command_setph0(void){
  /* Comprova que la funció està acompanyada d'un
     únic argument i programa el registre de
     fase 0 amb el valor passat com a paràmetre */
  if (arg_c==2){
    DDS_Active();
    uint8_t value=DDS_SetPhase_0(*(uint16_t *)arg_v[1]);
    DDS_Idle();
    return value;
  }
  else
    return -2;
}

static int8_t command_setph1(void){
  /* Comprova que la funció està acompanyada d'un
     únic argument i programa el registre de
     fase 1 amb el valor passat com a paràmetre */
  if (arg_c==2){
```

```
      DDS_Active();
      uint8_t value=DDS_SetPhase_1(*(uint16_t *)arg_v[1]);
      DDS_Idle();
      return value;
    }
    else
      return -2;
}

static int8_t command_test1(void){
  /* Funció mutant de debugging  */

  return 0;
}

static int8_t command_test2(void){
  /* Funció mutant de debugging  */

  return 0;
}

static int8_t command_bell202(void){
  /* Programa els registres de freqüència amb
     els valors nominals del estàndard BELL-202 */
  DDS_Active();
  DDS_Reset();
  DDS_SetFreq_0(1200);
  DDS_SetFreq_1(2200);
  DDS_Idle();
  return 0;
}

static int8_t command_bell103(void){
  /* Programa els registres de freqüència amb
     els valors nominals del estàndard BELL-103 */
  DDS_Active();
  DDS_Reset();
  DDS_SetFreq_0(1070);
  DDS_SetFreq_1(1270);
  DDS_Idle();
  return 0;
}

static int8_t command_setfsk(void){
    /* Comprova que la funció està acompanyada de dos
       arguments i programa els dos registres de freqüència
       amb els valors lliurats com a paràmetres.
       A més estableix la funció a la que han de contestar
       els terminals de control per interrupció i habilita
       aquestes interrupcions.
       Retorna (-2) en cas que no tinguem tots els arguments.
    */
  if (arg_c==3){
    DDS_Active();
    DDS_Reset();
    int8_t value = DDS_SetFreq_0(*(uint32_t *)arg_v[1]);
    value = value | DDS_SetFreq_1(*(uint32_t *)arg_v[2]);
```

```
      DDS_SetPhase_0(0);
      DDS_SetPhase_1(0);
      DDS_Idle();

      set_INT0(disable_out, start_out);
      set_INT1(frequency1, frequency0);

      INT0_enable(true);
      INT1_enable(true);
      // TODO: Quedarà a deshabilitar totes les interrupcions externes que no
          siguin necessàries
      return value;
    }
    else
      return -2;
}

static int8_t command_setpsk(void){
    /* Comprova que la funció està acompanyada de dos
       arguments i programa els dos registres de fase
       amb els valors lliurats com a paràmetres.
       A més estableix la funció a la que han de contestar
       els terminals de control per interrupció i habilita
       aquestes interrupcions.
       Retorna (-2) en cas que no tinguem tots els arguments.
    */
  if (arg_c==3){
    DDS_Active();
    DDS_Reset();
    int8_t value = DDS_SetPhase_0(*(uint16_t *)arg_v[1]);
    value = value | DDS_SetPhase_1(*(uint16_t *)arg_v[2]);
    DDS_Idle();
    set_INT0(disable_out, enable0);
    set_INT1(phase1, phase0);

    INT0_enable(true);
    INT1_enable(true);
    // TODO: Quedarà a deshabilitar totes les interrupcions externes que no
        siguin necessàries
    return value;
  }
  else
    return -2;
}



static int8_t command_setqam(void){
    /* Comprova que la funció està acompanyada d'un
       únic argument i programa el registres de freqüència 1
       amb el valor lliurat com a paràmetre.
       A més estableix la funció a la que ha de contestar
       el terminal de control per interrupció i habilita
       aquesta interrupció.
       Retorna (-2) en cas que no tinguem tots els arguments.
    */
  if (arg_c==2){
```

```
      DDS_Active();
      DDS_Reset();
      int8_t value = DDS_SetFreq_1(*(uint32_t *)arg_v[1]);
      DDS_Phase_0();
      DDS_Idle();
      // Estableix la funció dels pins de interrupció
      set_INT0(qam0_1, qam0_0);
      set_INT1(qam1_1, qam1_0);
      // Habilita els pins de interrupció
      INT0_enable(true);
      INT1_enable(true);
      // TODO: Quedarà a deshabilitar totes les interrupcions externes que no
          siguin necessàries
      return value;
    }
    else
      return -2;
  }



  static int8_t command_setask(void){
      /* Comprova que la funció està acompanyada d'un
         únic argument i programa el registres de freqüència 1
         amb el valor lliurats com a paràmetre.
         A més estableix la funció a la que ha de contestar
         el terminal de control per interrupció i habilita
         aquesta interrupció.
         Retorna (-2) en cas que no tinguem tots els arguments.
      */
    if (arg_c==2){
      DDS_Active();
      DDS_Reset();
      int8_t value = DDS_SetFreq_1(*(uint32_t *)arg_v[1]);
      DDS_SetPhase_1(0);
      DDS_Idle();
      // Estableix la funció dels pins de interrupció
      set_INT0(disable_out, enable1);
      // Habilita els pins de interrupció
      INT0_enable(true);
      INT1_enable(false);
      // TODO: Quedarà a deshabilitar totes les interrupcions externes que no
          siguin necessàries
      return value;
    }
    else
      return -2;
  }

  static int8_t help_fnc(void){
    /*  Explora totes les comandes i retorna
        el text de informació associat a cada
        una d'elles. Aquesta funció incorpora
        un mecanisme per tal de poder ocultar
        algunes comandes al usuari. Això permet
        incorporar comandes de control intern
        i debugging
```

```
    */
    char txt_help[50];
    char buff_cmd[7];
    char buff_hlp[40];
    char txt_help_format[11];
    GET_FLASH(txt_help_format,HELP_FRMT);
    for (uint8_t i=0; i<NUM_CMNDS; i++){
      GET_F_HELP(buff_hlp,i);
      if (*buff_hlp != 0){
        GET_F_CMND(buff_cmd,i);
        sprintf(txt_help, txt_help_format, buff_cmd, buff_hlp);
        serial_print(txt_help);
      }
    }
    return 0;
}

static int8_t print_controlreg(void){
    /* Instrucció que retorna el contingut de
       la copia del registre de control del DDS.
       La copia retornada no té perque coincidir
       exactament amb el contingut real del
       registre del DDS en el cas que s'hagi
       utilitzat la comanda set_ddsraw() prèviament.
       Aquesta funció està associada a una
       comanda oculta.
    */
    char txt[9];
    char cntrl_format[9];
    GET_FLASH(cntrl_format,CTREG_FRMT);
    sprintf(txt,cntrl_format, DDS_GetCtlReg());
    serial_print(txt);
    return 0;
}

static int8_t set_ddsraw(void){
    /* Aquesta funció permet programar el DDS de
       manera directa. Permet introduir un nombre
       variable de arguments fins a un màxim del
       espai que es tingui reservat per arguments.
       Els arguments lliurats han de estar en
       format decimal.
       Aquesta funció està associada a una
       comanda oculta.
    */
    if (arg_c>1){
      DDS_Active();
      for (int i=1; i<arg_c; i++){
        DDS_Master16bTransmit(*(uint32_t *)arg_v[i]);
      }
      DDS_Idle();
      return 0;
    }
    else
      return -2;
}
```

```
static int8_t nofares(void){
  /* Funció 'dummy' per farcir. Te com a objectiu
     poder-la assignar a les comandes que encara
     no tenen una funció definida
  */
  char dtxt[19];
  GET_FLASH(dtxt,DUMMY_MSG);
  serial_print(dtxt);
  return 0;
}

 static int8_t close_encounters(void){
  /* Petita funció oculta de
     prova. Una broma, un ou
     de pascua
  */
  DDS_Active();
  DDS_Reset();
  DDS_SetFreq_0(2349);
  DDS_Start_0();
  DDS_Idle();
  _delay_ms(300);

  DDS_Active();
  DDS_Reset();
  DDS_SetFreq_0(2637);
  DDS_Start_0();
  DDS_Idle();
  _delay_ms(300);

  DDS_Active();
  DDS_Reset();
  DDS_SetFreq_0(2093);
  DDS_Start_0();
  DDS_Idle();
  _delay_ms(300);

  DDS_Active();
  DDS_Reset();
  DDS_SetFreq_0(1046);
  DDS_Start_0();
  DDS_Idle();
  _delay_ms(300);

  DDS_Active();
  DDS_Reset();
  DDS_SetFreq_0(1568);
  DDS_Start_0();
  DDS_Idle();

  _delay_ms(1200);
  disable_out();
  return 0;
 }

static void enable0(void){
  /* Funció que selecciona el registre
```

```
      de freqüència 0 i habilita la sortida */
  DDS_Active();
  DDS_Start_0();
  DDS_Idle();
}

static void enable1(void){
  /* Funció que selecciona el registre
      de freqüència 1 i habilita la sortida */
  DDS_Active();
  DDS_Start_1();
  DDS_Idle();
}

static void disable_out(void){
  /* Aquesta funció deshabilita la sortida */
  DDS_Active();
  DDS_Reset();
  DDS_Idle();
}

static void frequency0(void){
  /* Funció que selecciona el registre
      de freqüència 0, sense modificar l'estat
      d'activació de la sortida */
  DDS_Active();
  DDS_Freq_0();
  DDS_Idle();
}

static void frequency1(void){
  /* Funció que selecciona el registre
      de freqüència 1, sense modificar l'estat
      d'activació de la sortida */
  DDS_Active();
  DDS_Freq_1();
  DDS_Idle();
}

static void phase0(void){
  /* Funció que selecciona el registre
      de fase 0, sense modificar l'estat
      d'activació de la sortida */
  DDS_Active();
  DDS_Phase_0();
  DDS_Idle();
}

static void phase1(void){
  /* Funció que selecciona el registre
      de freqüència 1, sense modificar l'estat
      d'activació de la sortida */
  DDS_Active();
  DDS_Phase_1();
  DDS_Idle();
}
```

```
static void start_out(void){
  /* Funció que habilita la sortida, sense
     modificar cap registre. Per tant respecta
     las condicions de freqüència i fase prèvies */
  DDS_Active();
  DDS_Start();
  DDS_Idle();
}

static void qam0_0(void){
  static uint8_t fase;
  INT1_enable(false);

  if(PIND & (1<<PIND3)) fase=0;
  else fase=180;

  DDS_Active();
  DDS_SetPhase_0(fase);
  DDS_Idle();

  INT1_enable(true);
}

static void qam0_1(void){
  static uint8_t fase;
  INT1_enable(false);

  if(PIND & (1<<PIND3)) fase=270;
  else fase=90;

  DDS_Active();
  DDS_SetPhase_0(fase);
  DDS_Idle();

  INT1_enable(true);
}

static void qam1_0(void){
  static uint8_t fase;
  INT0_enable(false);

  if(PIND & (1<<PIND2)) fase=90;
  else fase=0;

  DDS_Active();
  DDS_SetPhase_0(fase);
  DDS_Idle();

  INT0_enable(true);
}

static void qam1_1(void){
  static uint8_t fase;
  INT0_enable(false);

  if(PIND & (1<<PIND2)) fase=270;
  else fase=180;
```

```
  DDS_Active();
  DDS_SetPhase_0(fase);
  DDS_Idle();

  INT0_enable(true);
}

static int8_t reset_on_init(void){
  eeprom_update_byte((uint8_t*)BOOTVAR_BYTE, 0);
  return 0;
}

static int8_t preset_on_init(void){
  uint8_t inici;
  if (arg_c>2) return -2;
  else if (arg_c==2 && (*(uint32_t *)arg_v[1]>PRST_MAX || *(uint32_t *)arg_v
      [1]<1)) return -4;
  else{
    if (arg_c==1) inici=1;
    else inici=*(uint32_t *)arg_v[1];
    eeprom_update_byte((uint8_t*)BOOTVAR_BYTE, inici);
    return 0;
  }
}


static int8_t status_boot(void){
  char stboot[19];
  char txt[19];
  const uint8_t bootvar = eeprom_read_byte((uint8_t *)BOOTVAR_BYTE);
  if (bootvar){
    GET_FLASH(stboot,PRESETBOOT_MSG);
    sprintf(txt,stboot,bootvar);
  }
  else {
    GET_FLASH(txt,DEFAULTBOOT_MSG);
  }
  serial_print(txt);
  return 0;
}

static int8_t command_store(void){
  uint8_t inici;
  if (arg_c>2) return -2;
  else if (arg_c==2 && (*(uint32_t *)arg_v[1]>PRST_MAX || *(uint32_t *)arg_v
      [1]<1)) return -4;
  else{
    if (arg_c==1) inici=1;
    else inici=*(uint32_t *)arg_v[1];

    uint32_t freq0 = DDS_GetFreqR0();
    uint32_t freq1 = DDS_GetFreqR1();
    uint16_t fase0 = DDS_GetPhaseR0();
    uint16_t fase1 = DDS_GetPhaseR1();
    uint16_t ctreg = DDS_GetCtlReg();
    eeprom_update_dword((uint32_t*)(SPAN_M*(inici-1)+SFT_FR0), freq0);
```

```
      eeprom_update_dword((uint32_t*)(SPAN_M*(inici-1)+SFT_FR1), freq1);
      eeprom_update_word((uint16_t*)(SPAN_M*(inici-1)+SFT_PH0), fase0);
      eeprom_update_word((uint16_t*)(SPAN_M*(inici-1)+SFT_PH1), fase1);
      eeprom_update_word((uint16_t*)(SPAN_M*(inici-1)+SFT_CTRL), ctreg);
      return 0;
    }
  }


  static int8_t command_load(void){
   uint8_t inici;

   if (arg_c>2) return -2;
   else if (arg_c==2 && (*(uint32_t *)arg_v[1]>PRST_MAX || *(uint32_t *)arg_v
       [1]<1)) return -4;
   else
     {
       if (arg_c==1) inici=1;
       else inici=*(uint32_t *)arg_v[1];
       return load_from_e2prom(inici);
     }
  }


  static int8_t load_from_e2prom(uint8_t inici){
    DDS_Active();
    int8_t value = DDS_SetFreq_0(eeprom_read_dword((uint32_t*)(SPAN_M*(inici
        -1)+SFT_FR0)));
    value = value | DDS_SetFreq_1(eeprom_read_dword((uint32_t*)(SPAN_M*(inici
        -1)+SFT_FR1)));
    value = value | DDS_SetPhase_0(eeprom_read_word((uint16_t*)(SPAN_M*(inici
        -1)+SFT_PH0)));
    value = value | DDS_SetPhase_1(eeprom_read_word((uint16_t*)(SPAN_M*(inici
        -1)+SFT_PH1)));
    DDS_SetCtlReg(eeprom_read_word((uint16_t*)(SPAN_M*(inici-1)+SFT_CTRL)));
    DDS_Idle();
    return value;
  }


  /************************
   *  Funcions Públiques *
   ************************/

  void cli_init(const uint8_t inici){
    /* Funció d'inicialització del mòdul cli */
    /* Primer inicia DDS de manera incondicional */
    DDS_Init();
    if (inici){
      /* Si és diferent de 0, carrega un preset des de
         l'E2PROM i el programa en el DDS */
      load_from_e2prom(inici);
    }
  }

  int8_t exec_command(const uint8_t * command) {
    /* Funció que rep un apuntador a una cadena de
```

```
        caracters i explora totes les comandes integrades
        en el CLI. Si existeix una funció associada
        a la comanda entrada, la executa inmmediatament.
        En cas contrari retorna el error (-1) per marcar
        la funció com desconeguda.  */

    uint8_t i;
    if (*command==0) return 0;
    for (i=0; i<NUM_CMNDS; i++){
      if (!CMP_F_CMND(command, i))
        {
    if (commands[i].fnc) {
      return commands[i].fnc();
    }
        }
    }
    return -1;
}
```

Mòdul de missatges del cli en Flash

Aquest mòdul té la funcionalitat de facilitar la programació de noves funcions-comanda i deixar lliure la memòria SRAM del AVR, forçant a mantenir les cadenes de text constants, en memòria de programa Flash.

Dins del mòdul s'implementen les definicions necessàries i les funcions macros, per tal de que el usuari pugui cridar a determinades funcions i adreces de memòria FLASH de forma simple, obviant el fet que es tingui d'utilitzar l'accés específic aquest espai de memòria.

```
#ifndef _FLASHTXT_
#define _FLASHTXT_

#include <stdio.h>
#include <string.h>
#include <avr/pgmspace.h>



/* Funcions Macro   */

#define HELP_FRMT &(pgmstr_lst[0])
#define CTREG_FRMT &(pgmstr_lst[1])
#define DUMMY_MSG &(pgmstr_lst[2])
#define PRESETBOOT_MSG &(pgmstr_lst[3])
#define DEFAULTBOOT_MSG &(pgmstr_lst[4])


#define DSOUT &(cmnd_lst[0])
#define ENFR0 &(cmnd_lst[1])
#define ENFR1 &(cmnd_lst[2])
#define ENPHR0 &(cmnd_lst[3])
#define ENPHR1 &(cmnd_lst[4])
#define SRFR0 &(cmnd_lst[5])
#define SRFR1 &(cmnd_lst[6])
/***********/
#define SFR0 &(cmnd_lst[7])
```

```
#define SFR1 &(cmnd_lst[8])
#define SPHR0 &(cmnd_lst[9])
#define SPHR1 &(cmnd_lst[10])
#define SINE &(cmnd_lst[11])
#define SQR &(cmnd_lst[12])
#define TRI &(cmnd_lst[13])
/*****/
#define EEOC &(cmnd_lst[14])
#define DEOC &(cmnd_lst[15])
#define SFSC &(cmnd_lst[16])
#define SPSC &(cmnd_lst[17])
#define SFSK &(cmnd_lst[18])
#define SPSK &(cmnd_lst[19])
#define SASK &(cmnd_lst[20])
#define SQAM &(cmnd_lst[21])
#define B202 &(cmnd_lst[22])
#define B103 &(cmnd_lst[23])
/**********/
#define HELP &(cmnd_lst[24])
#define STORE &(cmnd_lst[25])
#define LOAD &(cmnd_lst[26])
#define RBOOT &(cmnd_lst[27])
#define PBOOT &(cmnd_lst[28])
#define SBOOT &(cmnd_lst[29])
/******/
#define TEST1 &(cmnd_lst[30])
#define TEST2 &(cmnd_lst[31])
#define ALIEN &(cmnd_lst[32])
#define CTRL &(cmnd_lst[33])
#define RAW &(cmnd_lst[34])


   /* * *   Funcions MACRO per accedir a les diferents estructures de memò
       ria FLASH  * * */

            /* Accés als missatges i formats genèrics */
#define GET_FLASH(punter,flashm)  strcpy_P(punter,(PGM_P)pgm_read_word(
    flashm))

            /* Accés i comparació als nom de comandes */
#define GET_F_CMND(punter,flashm)  strcpy_P(punter,(PGM_P)pgm_read_word(&(
    cmnd_lst[flashm])))
#define CMP_F_CMND(sram, flashm)  strcmp_PF(sram,(PGM_P)pgm_read_word(&(
    cmnd_lst[flashm])))

            /* Accés al text d'ajuda de les comandes */
#define GET_F_HELP(punter,flashm)  strcpy_P(punter,(PGM_P)pgm_read_word(&(
    htxt_lst[flashm])))




/*******************************/
```

```
      /*   Noms de comanda   */
  const char cmnd1[] PROGMEM="DSOUT";
  const char htxt1[] PROGMEM="Disables Output";
  const char cmnd2[] PROGMEM="ENFR0";
  const char htxt2[] PROGMEM="Drive freq.reg. 0 to output";
  const char cmnd3[] PROGMEM="ENFR1";
  const char htxt3[] PROGMEM="Drive freq.reg. 1 to output";
  const char cmnd4[] PROGMEM="ENPHR0";
  const char htxt4[] PROGMEM="Drive phase reg. 0 to output";
  const char cmnd5[] PROGMEM="ENPHR1";
  const char htxt5[] PROGMEM="Drive phase reg. 1 to output";
  const char cmnd6[] PROGMEM="SRFR0";
  const char htxt6[] PROGMEM="Set and run frequency register 0";
  const char cmnd7[] PROGMEM="SRFR1";
  const char htxt7[] PROGMEM="Set and run frequency register 1";
  /************************/
  const char cmnd8[] PROGMEM="SFR0";
  const char htxt8[] PROGMEM="Set frequency register 0";
  const char cmnd9[] PROGMEM="SFR1";
  const char htxt9[] PROGMEM="Set frequency register 1";
  const char cmnd10[] PROGMEM="SPHR0";
  const char htxt10[] PROGMEM="Set phase register 0";
  const char cmnd11[] PROGMEM="SPHR1";
  const char htxt11[] PROGMEM="Set phase register 1";
  const char cmnd12[] PROGMEM="SINE";
  const char htxt12[] PROGMEM="Select sine waveform";
  const char cmnd13[] PROGMEM="SQR";
  const char htxt13[] PROGMEM="Select square waveform (3.3V)";
  const char cmnd14[] PROGMEM="TRI";
  const char htxt14[] PROGMEM="Select triangle waveform";
  /*****************************/
  const char cmnd15[] PROGMEM="EEOC";
  const char htxt15[] PROGMEM="Enable external output control";
  const char cmnd16[] PROGMEM="DEOC";
  const char htxt16[] PROGMEM="Disable external output control";
  const char cmnd17[] PROGMEM="SFSC";
  const char htxt17[] PROGMEM="Set frequency shift control";
  const char cmnd18[] PROGMEM="SPSC";
  const char htxt18[] PROGMEM="Set pahse shift control";
  const char cmnd19[] PROGMEM="SFSK";
  const char htxt19[] PROGMEM="Set FSK mode";
  const char cmnd20[] PROGMEM="SPSK";
  const char htxt20[] PROGMEM="Set PSK mode";
  const char cmnd21[] PROGMEM="SASK";
  const char htxt21[] PROGMEM="Set ASK mode";
  const char cmnd22[] PROGMEM="SQAM";
  const char htxt22[] PROGMEM="";
  const char cmnd23[] PROGMEM="B202";
  const char htxt23[] PROGMEM="Set Bell-202 mode";
  const char cmnd24[] PROGMEM="B103";
  const char htxt24[] PROGMEM="Set Bell-103 mode";
  /********************/
  const char cmnd25[] PROGMEM="HELP";
  const char htxt25[] PROGMEM="Shows help text";
  const char cmnd26[] PROGMEM="STORE";
  const char htxt26[] PROGMEM="Saves actual status";
  const char cmnd27[] PROGMEM="LOAD";
```

```
const char htxt27[] PROGMEM="Load a preset";
const char cmnd28[] PROGMEM="RBOOT";
const char htxt28[] PROGMEM="Reset device on Boot (defaul)";
const char cmnd29[] PROGMEM="PBOOT";
const char htxt29[] PROGMEM="Load preset on Boot";
const char cmnd30[] PROGMEM="SBOOT";
const char htxt30[] PROGMEM="Gets the Boot status";
/*******************/
const char cmnd31[] PROGMEM="TEST1";
const char htxt31[] PROGMEM="";
const char cmnd32[] PROGMEM="TEST2";
const char htxt32[] PROGMEM="";
const char cmnd33[] PROGMEM="ALIEN";
const char htxt33[] PROGMEM="";
const char cmnd34[] PROGMEM="CTRL";
const char htxt34[] PROGMEM="";
const char cmnd35[] PROGMEM="RAW";
const char htxt35[] PROGMEM="";

#define NUM_CMNDS 35

const char frmt_hlp[] PROGMEM="%8s:   %s";
const char frmt_ctrlreg[] PROGMEM="0x%04X";
const char dummy_txt[] PROGMEM="DuMmY FuNcTiOn.";
const char boot_stxt[] PROGMEM="Boot: preset #%d";
const char boot_dtxt[] PROGMEM="Boot: default";




PGM_P const cmnd_lst[] PROGMEM =
{
  cmnd1,  cmnd2,  cmnd3,  cmnd4,
  cmnd5,  cmnd6,  cmnd7,  cmnd8,
  cmnd9,  cmnd10,  cmnd11,  cmnd12,
  cmnd13,  cmnd14,  cmnd15,  cmnd16,
  cmnd17,  cmnd18,  cmnd19,  cmnd20,
  cmnd21,  cmnd22,  cmnd23,  cmnd24,
  cmnd25,  cmnd26,  cmnd27,  cmnd28,
  cmnd29,  cmnd30,  cmnd31,  cmnd32,
  cmnd33,  cmnd34, cmnd35
};

PGM_P const htxt_lst[] PROGMEM =
{
  htxt1,  htxt2,  htxt3,  htxt4,
  htxt5,  htxt6,  htxt7,  htxt8,
  htxt9,  htxt10,  htxt11,  htxt12,
  htxt13,  htxt14,  htxt15,  htxt16,
  htxt17,  htxt18,  htxt19,  htxt20,
  htxt21,  htxt22,  htxt23,  htxt24,
  htxt25,  htxt26,  htxt27,  htxt28,
  htxt29,  htxt30,  htxt31,  htxt32,
  htxt33,  htxt34, htxt35
};
```

```
PGM_P const pgmstr_lst[] PROGMEM =
  {
    frmt_hlp, frmt_ctrlreg, dummy_txt,
    boot_stxt, boot_dtxt
  };


#endif
```

A.1.4. Gestió del circuit DDS

Els mòduls que estan integrats aquí, són els encarregats de parlar amb el oscil·lador DDS. Per una banda tenim el mòdul que defineix les funcions que són cridades per les capes superiors i per altra banda tenim el nivell més baix de comunicació, que comporta el bus SPI, amb el qual es comunicarà el microcontrolador amb el DDS.

Mòdul dds

En aquest mòdul estan definides les comandes especifiques i genèriques amb que treballa el circuit oscil·lador DDS.

```
/*
*    This library is free software; you can redistribute it and/or
*    modify it under the terms of the GNU Lesser General Public
*    License as published by the Free Software Foundation; version
*    2.1 of the License.
*
*    This library is distributed in the hope that it will be useful,
*    but WITHOUT ANY WARRANTY; without even the implied warranty of
*    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
*    Lesser General Public License for more details.
*
*    You should have received a copy of the GNU Lesser General Public
*    License along with this library; if not, write to the Free Software
*    Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301
    USA
*


 ************************************************
 *  Nom:  DDS
 *
 *  Autor:  Joan Martínez
 *
 *  Descripció: Mòdul control DDS AD9837/AD9833
 *
 ************************************************/


#ifndef DDS_AD9837_h
#define DDS_AD9837_h

/* Aquest include es per passar a les capes superiors els
defines de DD_SS, DD_SCK, etc  */
```

```
#include "spibus.h"


#define DDS_Active()  SPI_Active()
#define DDS_Idle()  SPI_Idle()
#define DDS_Master16bTransmit(data) SPI_Master16bTransmit(data)


void DDS_Init(void);
uint16_t DDS_GetCtlReg(void);
uint32_t DDS_GetFreqR0(void);
uint32_t DDS_GetFreqR1(void);
uint16_t DDS_GetPhaseR0(void);
uint16_t DDS_GetPhaseR1(void);

void DDS_Reset(void);
void DDS_Start(void);
void DDS_Start_0(void);
void DDS_Start_1(void);
void DDS_Phase_0(void);
void DDS_Phase_1(void);
void DDS_Freq_0(void);
void DDS_Freq_1(void);
void DDS_Sine(void);
void DDS_Square(void);
void DDS_Triangle(void);

int8_t DDS_SetPhase_0(const uint16_t phase);
int8_t DDS_SetPhase_1(const uint16_t phase);
int8_t DDS_SetFreq_0(uint32_t frequency);
int8_t DDS_SetFreq_1(uint32_t frequency);
void DDS_SetCtlReg(const uint16_t reg);
#endif
```

```
#include <stdint.h>
#include <inttypes.h>
#include "spibus.h"
#include "dds.h"

/*   Límits màxims de freq. i fase
     -----------------------------
     TO-DO: Automatitzar des de compilació
*/
#define DDS_F_MAX 12500000L
#define DDS_PH_MAX 359


/*   Multiplicadors de Freqüència i Fase
     -----------------------------------
     Es trien diferents multiplicadors en funció de la
freqüència de MCLK.

  - FREQ_MUL
    clk= 5MHz 53.6870912
    clk=16Mhz 16.777216f
    clk=25Mhz 10.73741824f
```

```
    Per el multiplicador de fase es tria en funció de les
unitats del valor d'entrada

    11.3777778f per graus
    651.8986469f per radians
*/
#define FREQ_MUL 10.73741824f
#define PHSE_MUL 11.3777778f

/*  Constants de Registres de Control
    --------------------------------
  Aquests són els valors de les constants necessàries per
programar els registres de el DDS.
*/
#define SIN_WF 0x00
#define SQR_WF 0x28
#define TRI_WF 0x02
#define SLEEP1 0x80

#define RESET_WD 0x0100
#define START_WD 0x2000
#define SET_F_WD 0x2100
#define PHSE_MSK 0xC000

#define FSEL0 0x0000
#define FSEL1 0x0800

#define PSEL0 0x0000
#define PSEL1 0x0400


#define FSEL_MSK 0xF7FF
#define PSEL_MSK 0xFBFF
#define RST_MSK 0xFEFF
#define SLEP_MSK 0xFF7F
#define WF_MSK 0xFFD5

#define START_F0 START_WD|FSEL0
#define START_F1 START_WD|FSEL1
#define START_MSK 0xD6FF

/*  Constants dels Registres de Frequencies
    --------------------------------------
  Per seleccionar en quin registre (0 o 1) volem programar les paraules de
14 bits (LSB o MSB).
*/
#define FREQ0 0x4000
#define FREQ1 0x8000

/*  Constants dels Registres de Fase
    --------------------------------
  Per seleccionar en quin registre (0 o 1) volem programar les paraules de
12 bits per la fase.
*/
#define PHASE0 0xC000
#define PHASE1 0xE000
```

```
uint16_t control_reg=START_WD;
uint32_t freq0_reg;
uint32_t freq1_reg;
uint16_t fase0_reg;
uint16_t fase1_reg;



/*   CAPÇALERES FUNCIONS PRIVADES
     ----------------------------          */
static int8_t DDS_SetDirectFrequency(uint32_t frequency, const uint16_t
    freq_reg);
static int8_t DDS_SetFrequency(uint32_t frequency, const uint16_t freq_reg);
static int8_t DDS_SetPhase(const uint16_t phase, const uint16_t phase_reg);
static int8_t DDS_ChgControlReg(const uint16_t reg_mask, const uint16_t
    reg_value);



/*     FUNCIONS PÚBLIQUES
       ------------------                        */
uint32_t DDS_GetFreqR0(void){
  /* Retorna el contingut de la copia del
     registre de freqüència del DDS  */
  return freq0_reg;
}

uint32_t DDS_GetFreqR1(void){
  /* Retorna el contingut de la copia del
     registre de freqüència 1 del DDS  */
  return freq1_reg;
}

uint16_t DDS_GetPhaseR0(void){
  /* Retorna el contingut de la copia del
     registre de fase 0 del DDS  */
  return fase0_reg;
}

uint16_t DDS_GetPhaseR1(void){
  /* Retorna el contingut de la copia del
     registre de fase 1 del DDS  */
  return fase1_reg;
}

uint16_t DDS_GetCtlReg(void){
  /* Retorna el contingut de la copia del
     registre de control del DDS  */
  return control_reg;
}

void DDS_Reset(void){
  /* Resets output (not registers)
     output is held in middle value */
  DDS_ChgControlReg(RST_MSK, RESET_WD);
}
```

```
void DDS_Start(void){
  /* Posa en marxa la sortida mantenint
     els bits PSEL FSEL sense alterar */
  DDS_ChgControlReg(RST_MSK, START_WD);
}


void DDS_SetCtlReg(const uint16_t reg){
  /* Estableix el valor de registre de
     control i de la seva copia */
  DDS_ChgControlReg(0x0000, reg);
}

void DDS_Init(void){
  /* Prepara Master Device
     i atura DDS */
  SPI_MasterInit();
  SPI_Active();
  DDS_Reset();
  SPI_Idle();
}

void DDS_Start_0(void){
  /* Enables DDS Output */
  DDS_ChgControlReg(START_MSK, START_F0);
}

void DDS_Start_1(void){
  /* Enables DDS Output */
  DDS_ChgControlReg(START_MSK, START_F1);
}


void DDS_Freq_0(void){
  /* Select FR0 */
  DDS_ChgControlReg(FSEL_MSK, FSEL0);
}

void DDS_Freq_1(void){
  /* Select FR1 */
  DDS_ChgControlReg(FSEL_MSK, FSEL1);
}

void DDS_Phase_0(void){
  DDS_ChgControlReg(PSEL_MSK, PSEL0);
}

void DDS_Phase_1(void){
  DDS_ChgControlReg(PSEL_MSK, PSEL1);
}

void DDS_Sine(void){
  DDS_ChgControlReg(WF_MSK, SIN_WF);
}

void DDS_Square(void){
```

```
  DDS_ChgControlReg(WF_MSK, SQR_WF);
}

void DDS_Triangle(void){
  DDS_ChgControlReg(WF_MSK, TRI_WF);
}


int8_t DDS_SetPhase_0(const uint16_t phase){
  /* Estableix el valor de fase del registre de fase 0 */
  int8_t value=DDS_SetPhase(phase, PHASE0);
  if (!value) fase0_reg=phase;
  return value;
}

int8_t DDS_SetPhase_1(const uint16_t phase){
  /* Estableix el valor de fase del registre de fase 1 */
  int8_t value=DDS_SetPhase(phase, PHASE1);
  if (!value) fase1_reg=phase;
  return value;
}

int8_t DDS_SetFreq_0(uint32_t frequency){
  /* Estableix el valor de frequencia del registre de frequencia 0 */
  int8_t value=DDS_SetFrequency(frequency, FREQ0);
  if (!value) freq0_reg=frequency;
  return value;
}

int8_t DDS_SetFreq_1(uint32_t frequency){
  /* Estableix el valor de frequencia del registre de frequencia 1 */
  int8_t value=DDS_SetFrequency(frequency, FREQ1);
  if (!value) freq1_reg=frequency;
  return value;
}


/*      FUNCIONS PRIVADES
        -----------------                              */
int8_t DDS_SetDirectFrequency(uint32_t frequency, const uint16_t freq_reg){
  /* Sets a new frequency without
     reseting the output */
  if(frequency<=12500000){
    uint32_t divisor=frequency*FREQ_MUL;
    SPI_Master16bTransmit((divisor & 0x3fff)| freq_reg);
    SPI_Master16bTransmit((divisor >> 14) | freq_reg);
    return 0;
  }
  else{
    return -3;
  }
}

int8_t DDS_SetFrequency(uint32_t frequency, const uint16_t freq_reg){
  /* Sets a new frequency reseting
     previously the output */
    SPI_Master16bTransmit(SET_F_WD);
```

```
    return DDS_SetDirectFrequency(frequency,freq_reg);
}

int8_t  DDS_SetPhase(const uint16_t phase, const uint16_t phase_reg){
  /* Sets a new phase */
  if(phase<=DDS_PH_MAX){
    SPI_Master16bTransmit((uint16_t)(phase*PHSE_MUL) | phase_reg);
    return 0;
  }
  else
    return -3;
}

static int8_t DDS_ChgControlReg(const uint16_t reg_mask, const uint16_t
    reg_value){
  /* Funció que s'encarrega de modificar el registre de control
     del DDS i guardar una copia */
  control_reg = (control_reg & reg_mask) | reg_value;
  SPI_Master16bTransmit(control_reg);
  return 0;
}
```

Mòdul spibus

Aquest mòdul implementa les funcions de gestió i transferència bàsiques necessàries per realitzar la comunicació en un bus SPI.

El mòdul conté funcions de lectura i escriptura, tant en mode Master com en mode Slave, tot i que en aquest dispositiu únicament són usades les instruccions de escriptura en mode Master, ja que la comunicació que es realitza és en tot moment unidireccional.

```
/*
*    This library is free software; you can redistribute it and/or
*    modify it under the terms of the GNU Lesser General Public
*    License as published by the Free Software Foundation; version
*    2.1 of the License.
*
*    This library is distributed in the hope that it will be useful,
*    but WITHOUT ANY WARRANTY; without even the implied warranty of
*    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
*    Lesser General Public License for more details.
*
*    You should have received a copy of the GNU Lesser General Public
*    License along with this library; if not, write to the Free Software
*    Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301
    USA
*

 ************************************************
 *  Nom:  SPI BUS
 *
 *  Autor:  Joan Martínez
 *
 *  Descripcio: Mòdul control BUS SPI
 *
 ************************************************/
```

```
#include <stdint.h>


#ifndef SPI_BUS_h
#define SPI_BUS_h

/* SPI Pin Config.
 ---------------
 These are the pin configuration for Arduino with ATMEGA328,
 ATMEGA168. To be sure about what the pins to use check the
 documentation of your MCU */
#define DD_SS (1<<DDB2)
#define DD_SCK (1<<DDB5)
#define DD_MOSI (1<<DDB3)
#define DD_MISO (1<<DDB4)


//Master
void SPI_MasterInit(void);
uint8_t SPI_MasterTransmit(const uint8_t cData);
void SPI_Master16bTransmit(uint16_t wData);
void SPI_Active(void);
void SPI_Idle(void);

//Slave
void SPI_SlaveInit(void);
uint8_t SPI_SlaveReceive(void);

#endif
```

```
#include <stdint.h>
#include <avr/io.h>

#include "spibus.h"



/* Master */

void SPI_MasterInit(void){
  /* Set SCK, MOSI & SS as outputs, left unchanged others
   (MISO must!!! be set as input as default of SPI master mode) */
  DDRB = DD_MOSI | DD_SCK | DD_SS;
  PORTB |= DD_SS| DD_MOSI | DD_SCK;

  /* Set mode 2, Enable SPI, set clock, master mode and order
     The Speed is selected at F_CPU/4 (SPI2X = SPR1 = SPR0 = 0)

      F_spi    | F_CPU
     ----------+--------
       4 MHz   | 16 MHz
     2.5 MHz   | 10 MHz

  */
  SPSR &= ~(1<<SPI2X);
  SPCR &= ~((1<<SPIE) | (1<<CPHA) | (1<<DORD) | (1<<SPR1) | (1<<SPR0));
  SPCR |=  (1<<SPE) | (1<<MSTR) | (1<<CPOL);
```

```
}


uint8_t SPI_MasterTransmit(const uint8_t cData){
  SPDR=cData;

  while(!(SPSR & (1<<SPIF)))
    ;
  return SPDR;
}

void SPI_Master16bTransmit(uint16_t wData){
  SPI_MasterTransmit(wData>>8);
  SPI_MasterTransmit(wData & 0xff);
}

void SPI_Idle(void){
  PORTB |= DD_SS;
}

void SPI_Active(void){
  PORTB &= ~DD_SS;
}


/* Slave */

void SPI_SlaveInit(void){
  /* Enable SPI, Slave set clock and Mode */
  SPCR &= ~((1<<CPHA) | (1<<MSTR) | (1<<DORD)) ;
  SPCR |= (1<<SPE) | (1<<CPOL);
}

uint8_t SPI_SlaveReceive(void){
  while(!(SPSR & (1<<SPIF)))
    ;
  return SPDR;
}
```

A.1.5. Gestió de les interrupcions externes

Dins del mòdul *gestirq* es porta a terme la inserció de funcions en els vectors d'interrupcions i la habilitació i deshabilitació de les interrupcions.

Les interrupcions externes juguen un paper bàsic en l'ús d'aquest dispositiu e sistemes de comunicació, ja que permet els canvis de registres de forma àgil de manera externa, donant lloc a modulacions per interrupció de portadora, en fase o bé en freqüència.

```
/*
*    This library is free software; you can redistribute it and/or
*    modify it under the terms of the GNU Lesser General Public
*    License as published by the Free Software Foundation; version
*    2.1 of the License.
*
*    This library is distributed in the hope that it will be useful,
*    but WITHOUT ANY WARRANTY; without even the implied warranty of
```

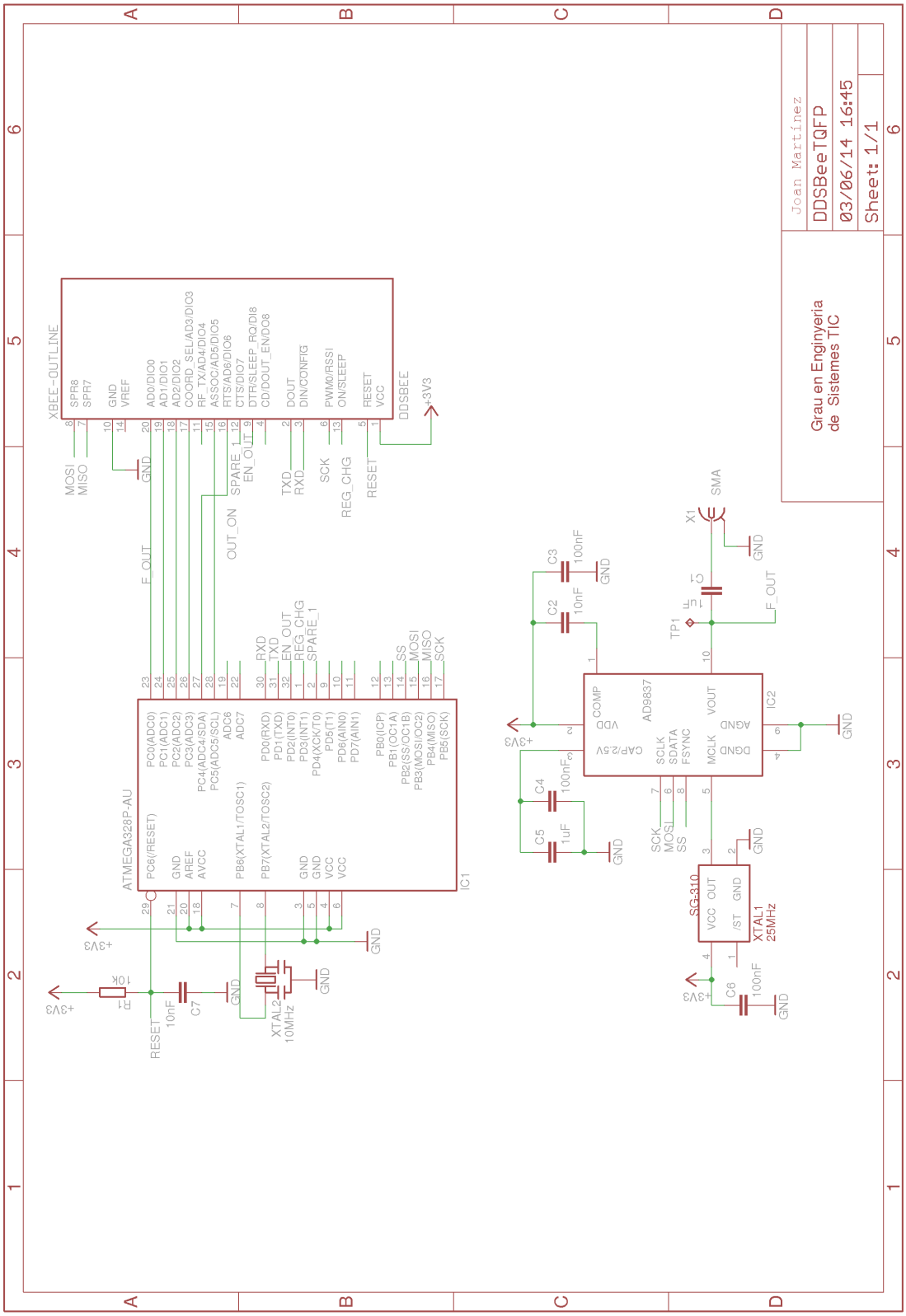
```
 *    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 *    Lesser General Public License for more details.
 *
 *    You should have received a copy of the GNU Lesser General Public
 *    License along with this library; if not, write to the Free Software
 *    Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301
     USA
 *
 ************************************************
 *  Nom:  GESTIRQ
 *
 *  Autor:  Joan Martínez
 *
 *  Descripcio:   Aquest mòdul implementa les funcionalitats
 *             per gestionar les interrupcions produides per
 *             canvis en el mòdul i que han de treballar sobre
 *             el funcionament del DDS.
 ************************************************/

#ifndef _GESTIRQ_
#define _GESTIRQ_

#include <stdbool.h>

typedef void (*nexus_t)(void);

void set_INT0(nexus_t fl, nexus_t fh);
void set_INT1(nexus_t fl, nexus_t fh);
void set_PCINT1(nexus_t fl, nexus_t fh);
void gestirq_init(void);
void INT0_enable(bool active);
void INT1_enable(bool active);

#endif
```

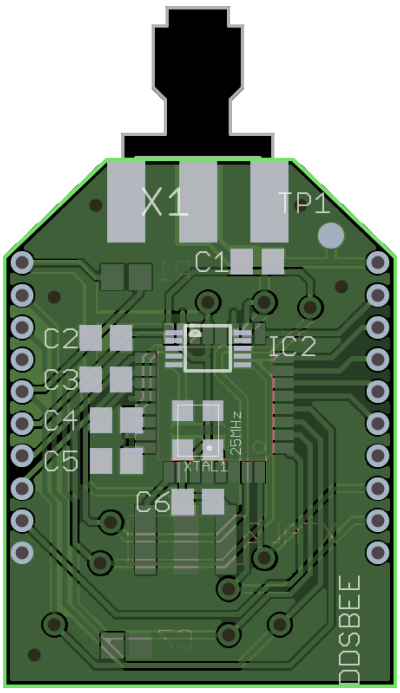
```
#include <stdio.h>
#include <avr/io.h>
#include <avr/interrupt.h>

#include "gestirq.h"

static nexus_t int0_high;
static nexus_t int0_low;
static nexus_t int1_high;
static nexus_t int1_low;
static nexus_t pcint1_high;
static nexus_t pcint1_low;

void gestirq_init(void){
  /*************************
    FUNCIO INICIALITZACIO IRQ
    -------------------------
     Aquesta funció s'ha de
  encarregar de inicialitzar les
  condicions de inerrupció generals
  del microcontrolador
```

```
  ****************************/
  DDRD &= 0xF3;              // INT0 i INT1 configurats com entrades
  EICRA = 0x05;              // Interrupcions en INT0 i INT1 per canvi de
      nivell


  /*
  DDRC &= 0xC0;              // Port C Com entrades
  PCICR = 0x02;              // Habilitar les interrupcions de PCIE1 (port C)
  PCMSK1 = 0x01;             // Habilita la interrupció unicament del pin 1
  */

  sei();                     // Habilita les inerrupcions generals

}

void INT0_enable(bool active){
  /* habilita o deshabilita INT0
     de EIMSK */
  if(active) EIMSK |= 0x01;
  else EIMSK &= 0xFE;
}

void INT1_enable(bool active){
  /* habilita o deshabilita INT1
     de EIMSK */
  if(active) EIMSK |= 0x02;
  else EIMSK &= 0xFD;
}

void set_INT0(nexus_t fl, nexus_t fh){
  int0_high=fh;
  int0_low=fl;
}

void set_INT1(nexus_t fl, nexus_t fh){
  int1_high=fh;
  int1_low=fl;
}

void set_PCINT1(nexus_t fl, nexus_t fh){
  pcint1_high=fh;
  pcint1_low=fl;
}



 /**********************
    RUTINES DE SERVEI
      D'INTERRUPCIO
  **********************/
ISR(PCINT1_vect){
  /* CAL REPLANTEJAR LA FUNCIÓ PER
     TAL D'HABILITAR LA SORTIDA,
     INDEPENDENTMENT DEL REGISTRE ?   */
  /*
  if(PINC & 0x01) pcint1_high();
```

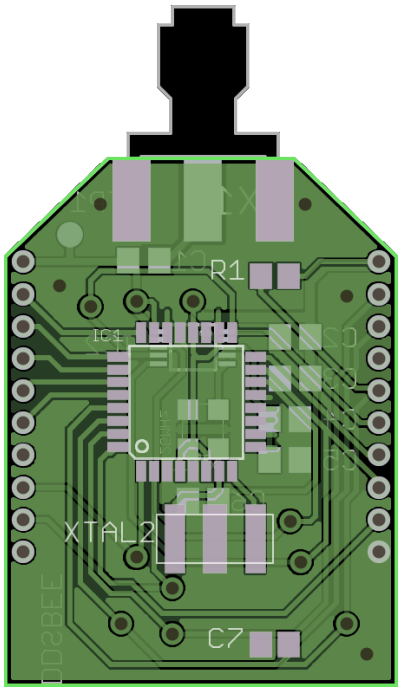
```
    else pcint1_low();
    */
  }

  ISR(INT0_vect){
    /* Utilitza la interrupció INT0
       per seleccionar un dels dos
       registres de freqüència.
       Modulació FSK.              */
    if(PIND & (1<<PIND2)) int0_high();
    else int0_low();
  }

  ISR(INT1_vect){
    /* Utilitza la interrupció INT1
       per seleccionar un dels dos
       registres de fase. Modulació PSK. */
    if(PIND & (1<<PIND3)) int1_high();
    else int1_low();
  }
```

A.2. Esquemes i Disseny de placa





[Top]



[Bottom]

A.3. Llistat de components i cost

item	QTY	Descripció	preu/Unitat	Preu
1	1	ADI - AD9833BRMZ - DDS	7,56 €	7,56 €
2	1	ATMEL - ATMEGA328P-AU - MCU	2,49 €	2,49 €
3	1	EPSON - Q33310FD00003 - OSC., 25.0MHZ	3,81 €	3,81 €
4	1	MURATA - CSTCC10M0G53-R0 - RESONATOR, 10.0MHZ	0,44 €	0,44 €
5	3	C 100NF	0,01 €	0,02 €
6	2	C 10NF	0,01 €	0,01 €
7	2	C 1UF	0,01 €	0,01 €
8	1	R 10K	0,01 €	0,01 €
9	1	MOLEX - 73251-1150 - RF COAXIAL, SMA, 50OHM	3,35 €	3,35 €
10	1	PCB (FABRICACIÓ A SEEEDSTUDIO)	0,93 €	0,93 €
11	1	TIRA DE PINS (2 x 1X10 2mm)	0,23 €	0,23 €
			TOTAL	18,86 €

Taula A.1.: Components i cost d'aquest

A.4. Data Sheet AD9833

Low Power, 12.65 mW, 2.3 V to 5.5 V, Programmable Waveform Generator

Data Sheet **AD9833**

FEATURES

Digitally programmable frequency and phase
12.65 mW power consumption at 3 V
0 MHz to 12.5 MHz output frequency range
28-bit resolution: 0.1 Hz at 25 MHz reference clock
Sinusoidal, triangular, and square wave outputs
2.3 V to 5.5 V power supply
No external components required
3-wire SPI interface
Extended temperature range: −40°C to +105°C
Power-down option
10-lead MSOP package
Qualified for automotive applications

APPLICATIONS

Frequency stimulus/waveform generation
Liquid and gas flow measurement
Sensory applications: proximity, motion, and defect detection
Line loss/attenuation
Test and medical equipment
Sweep/clock generators
Time domain reflectometry (TDR) applications

GENERAL DESCRIPTION

The AD9833 is a low power, programmable waveform generator capable of producing sine, triangular, and square wave outputs. Waveform generation is required in various types of sensing, actuation, and time domain reflectometry (TDR) applications. The output frequency and phase are software programmable, allowing easy tuning. No external components are needed. The frequency registers are 28 bits wide: with a 25 MHz clock rate, resolution of 0.1 Hz can be achieved; with a 1 MHz clock rate, the AD9833 can be tuned to 0.004 Hz resolution.

The AD9833 is written to via a 3-wire serial interface. This serial interface operates at clock rates up to 40 MHz and is compatible with DSP and microcontroller standards. The device operates with a power supply from 2.3 V to 5.5 V.

The AD9833 has a power-down function (SLEEP). This function allows sections of the device that are not being used to be powered down, thus minimizing the current consumption of the part. For example, the DAC can be powered down when a clock output is being generated.

The AD9833 is available in a 10-lead MSOP package.

FUNCTIONAL BLOCK DIAGRAM

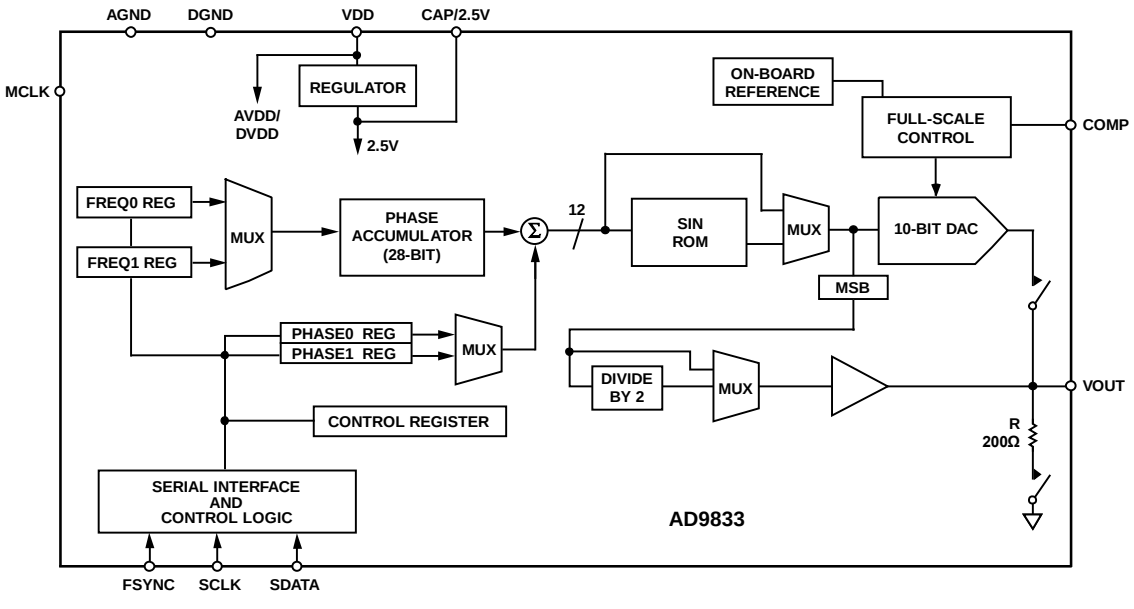


Figure 1.

Rev. E **Document Feedback**

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781.329.4700
Technical Support www.analog.com

TABLE OF CONTENTS

REVISION HISTORY

9/12—Rev. D to Rev. E

Changed Input Current, I_{INH}/I_{INL} from 10 mA to 10 μA ... 3

4/11—Rev. C to Rev. D

Change to Figure 13 ... 8
Changes to Table 9 ... 15
Deleted AD9833 to ADSP-2101/ADSP-2103 Interface Section ... 20
Changes to Evaluation Board Section ... 21
Added System Demonstration Platform Section, AD9833 to SPORT Interface Section, and Evaluation Kit Section ... 21
Changes to Crystal Oscillator vs. External Clock Section and Power Supply Section ... 21
Added Figure 32 and Figure 33; Renumbered Figures Sequentially ... 21
Deleted Prototyping Area Section and Figure 33 ... 22
Added Evaluation Board Schematics Section, Figure 34, and Figure 35 ... 22
Deleted Table 16 ... 23
Added Evaluation Board Layout Section, Figure 36, Figure 37, and Figure 38 ... 23
Changes to Ordering Guide ... 24

9/10—Rev. B to Rev. C

Changed 20 mW to 12.65 mW in Data Sheet Title and Features List ... 1
Changes to Figure 6 Caption and Figure 7 ... 7

6/10—Rev. A to Rev. B

Changes to Features Section ... 1
Changes to Serial Interface Section ... 13
Changes to VOUT Pin Section ... 16
Changes to Grounding and Layout Section ... 17
Updated Outline Dimensions ... 24
Changes to Ordering Guide ... 24
Added Automotive Products Section ... 24

6/03—Rev. 0 to Rev. A

Updated Ordering Guide ... 4

SPECIFICATIONS

VDD = 2.3 V to 5.5 V, AGND = DGND = 0 V, $T_A = T_{MIN}$ to T_{MAX}, R_{SET} = 6.8 kΩ for VOUT, unless otherwise noted.

Table 1.

Parameter[1]	Min	Typ	Max	Unit	Test Conditions/Comments
SIGNAL DAC SPECIFICATIONS					
Resolution		10		Bits	
Update Rate			25	MSPS	
VOUT Maximum		0.65		V	
VOUT Minimum		38		mV	
VOUT Temperature Coefficient		200		ppm/°C	
DC Accuracy					
Integral Nonlinearity		±1.0		LSB	
Differential Nonlinearity		±0.5		LSB	
DDS SPECIFICATIONS (SFDR)					
Dynamic Specifications					
Signal-to-Noise Ratio (SNR)	55	60		dB	f_{MCLK} = 25 MHz, $f_{OUT} = f_{MCLK}/4096$
Total Harmonic Distortion (THD)		−66	−56	dBc	f_{MCLK} = 25 MHz, $f_{OUT} = f_{MCLK}/4096$
Spurious-Free Dynamic Range (SFDR)					
Wideband (0 to Nyquist)		−60		dBc	f_{MCLK} = 25 MHz, $f_{OUT} = f_{MCLK}/50$
Narrow-Band (±200 kHz)		−78		dBc	f_{MCLK} = 25 MHz, $f_{OUT} = f_{MCLK}/50$
Clock Feedthrough		−60		dBc	
Wake-Up Time		1		ms	
LOGIC INPUTS					
Input High Voltage, V_{INH}	1.7			V	2.3 V to 2.7 V power supply
	2.0			V	2.7 V to 3.6 V power supply
	2.8			V	4.5 V to 5.5 V power supply
Input Low Voltage, V_{INL}			0.5	V	2.3 V to 2.7 V power supply
			0.7	V	2.7 V to 3.6 V power supply
			0.8	V	4.5 V to 5.5 V power supply
Input Current, I_{INH}/I_{INL}			10	µA	
Input Capacitance, C_{IN}		3		pF	
POWER SUPPLIES					f_{MCLK} = 25 MHz, $f_{OUT} = f_{MCLK}/4096$
VDD	2.3		5.5	V	
I_{DD}		4.5	5.5	mA	I_{DD} code dependent; see Figure 7
Low Power Sleep Mode		0.5		mA	DAC powered down, MCLK running

[1] Operating temperature range is −40°C to +105°C; typical specifications are at 25°C.

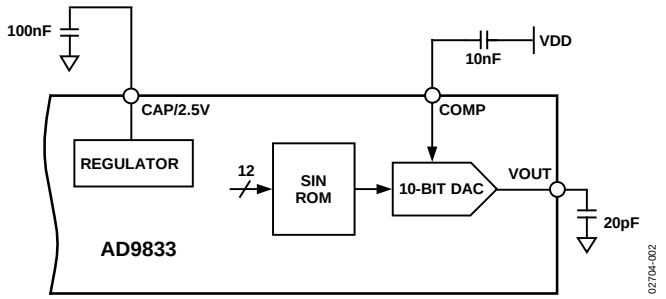


Figure 2. Test Circuit Used to Test Specifications

TIMING CHARACTERISTICS

VDD = 2.3 V to 5.5 V, AGND = DGND = 0 V, unless otherwise noted.[1]

Table 2.

Parameter	Limit at T_{MIN} to T_{MAX}	Unit	Description
t_1	40	ns min	MCLK period
t_2	16	ns min	MCLK high duration
t_3	16	ns min	MCLK low duration
t_4	25	ns min	SCLK period
t_5	10	ns min	SCLK high duration
t_6	10	ns min	SCLK low duration
t_7	5	ns min	FSYNC to SCLK falling edge setup time
$t_{8\ min}$	10	ns min	FSYNC to SCLK hold time
$t_{8\ max}$	$t_4 - 5$	ns max	
t_9	5	ns min	Data setup time
t_{10}	3	ns min	Data hold time
t_{11}	5	ns min	SCLK high to FSYNC falling edge setup time

[1] Guaranteed by design, not production tested.

Timing Diagrams

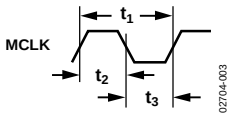


Figure 3. Master Clock

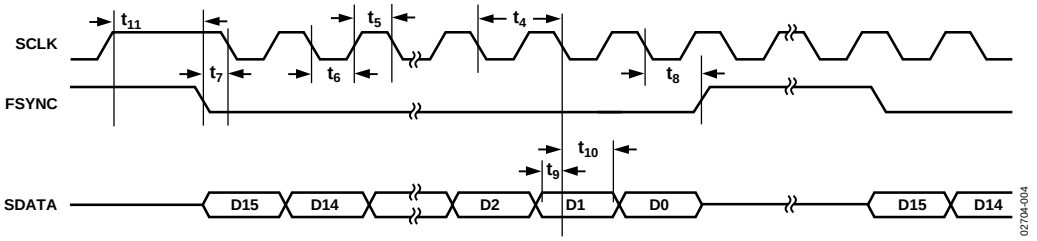


Figure 4. Serial Timing

ABSOLUTE MAXIMUM RATINGS

T_A = 25°C, unless otherwise noted.

Table 3.

Parameter	Rating
VDD to AGND	−0.3 V to +6 V
VDD to DGND	−0.3 V to +6 V
AGND to DGND	−0.3 V to +0.3 V
CAP/2.5V	2.75 V
Digital I/O Voltage to DGND	−0.3 V to VDD + 0.3 V
Analog I/O Voltage to AGND	−0.3 V to VDD + 0.3 V
Operating Temperature Range	
Industrial (B Version)	−40°C to +105°C
Storage Temperature Range	−65°C to +150°C
Maximum Junction Temperature	150°C
MSOP Package	
θ_{JA} Thermal Impedance	206°C/W
θ_{JC} Thermal Impedance	44°C/W
Lead Temperature, Soldering (10 sec)	300°C
IR Reflow, Peak Temperature	220°C

Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; functional operation of the device at these or any other conditions above those indicated in the operational section of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

ESD CAUTION

ESD (electrostatic discharge) sensitive device. Charged devices and circuit boards can discharge without detection. Although this product features patented or proprietary protection circuitry, damage may occur on devices subjected to high energy ESD. Therefore, proper ESD precautions should be taken to avoid performance degradation or loss of functionality.

PIN CONFIGURATION AND FUNCTION DESCRIPTIONS

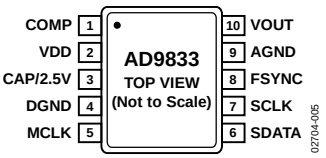


Figure 5. Pin Configuration

Table 4. Pin Function Descriptions

Pin No.	Mnemonic	Description
1	COMP	DAC Bias Pin. This pin is used for decoupling the DAC bias voltage.
2	VDD	Positive Power Supply for the Analog and Digital Interface Sections. The on-board 2.5 V regulator is also supplied from VDD. VDD can have a value from 2.3 V to 5.5 V. A 0.1 µF and a 10 µF decoupling capacitor should be connected between VDD and AGND.
3	CAP/2.5V	The digital circuitry operates from a 2.5 V power supply. This 2.5 V is generated from VDD using an on-board regulator when VDD exceeds 2.7 V. The regulator requires a decoupling capacitor of 100 nF typical, which is connected from CAP/2.5V to DGND. If VDD is less than or equal to 2.7 V, CAP/2.5V should be tied directly to VDD.
4	DGND	Digital Ground.
5	MCLK	Digital Clock Input. DDS output frequencies are expressed as a binary fraction of the frequency of MCLK. The output frequency accuracy and phase noise are determined by this clock.
6	SDATA	Serial Data Input. The 16-bit serial data-word is applied to this input.
7	SCLK	Serial Clock Input. Data is clocked into the AD9833 on each falling edge of SCLK.
8	FSYNC	Active Low Control Input. FSYNC is the frame synchronization signal for the input data. When FSYNC is taken low, the internal logic is informed that a new word is being loaded into the device.
9	AGND	Analog Ground.
10	VOUT	Voltage Output. The analog and digital output from the AD9833 is available at this pin. An external load resistor is not required because the device has a 200 Ω resistor on board.

TYPICAL PERFORMANCE CHARACTERISTICS

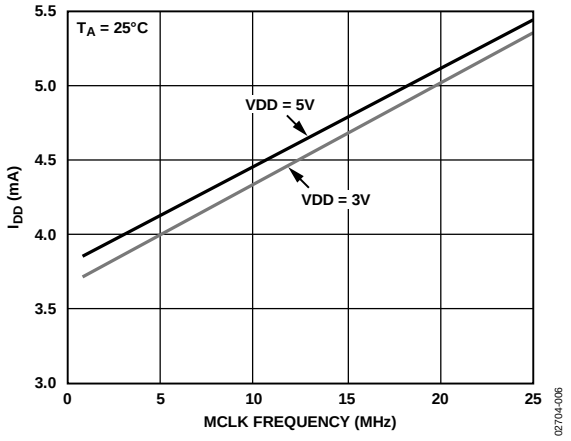


Figure 6. Typical Current Consumption (I_{DD}) vs. MCLK Frequency for f_{OUT} = MCLK/10

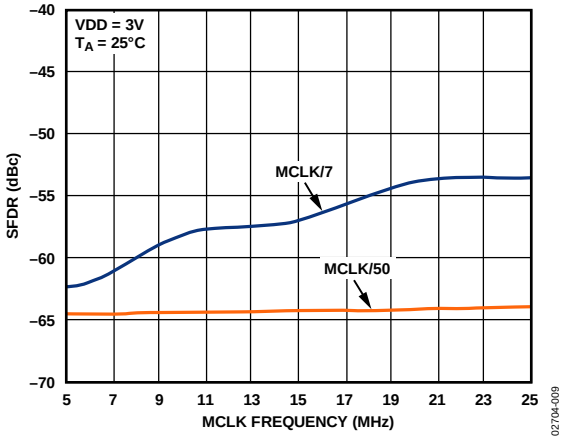


Figure 9. Wideband SFDR vs. MCLK Frequency

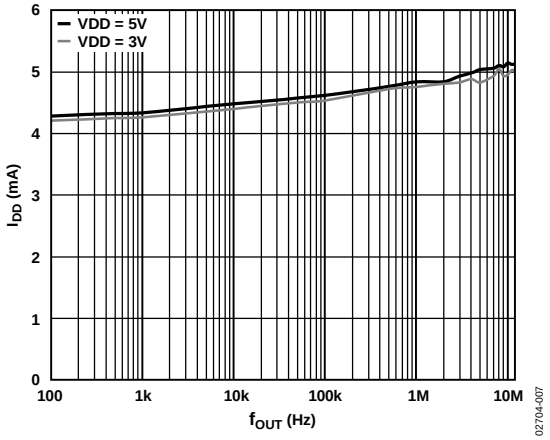


Figure 7. Typical I_{DD} vs. f_{OUT} for f_{MCLK} = 25 MHz

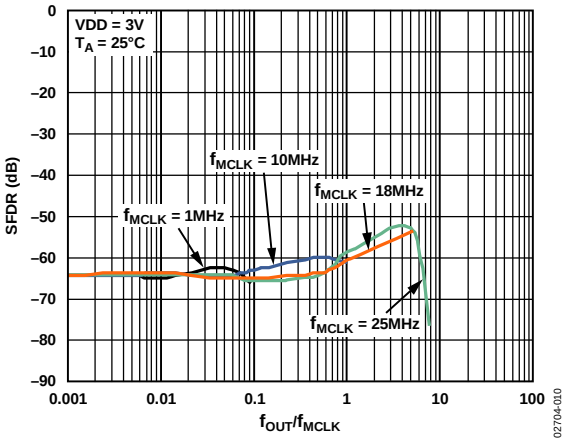


Figure 10. Wideband SFDR vs. f_{OUT}/f_{MCLK} for Various MCLK Frequencies

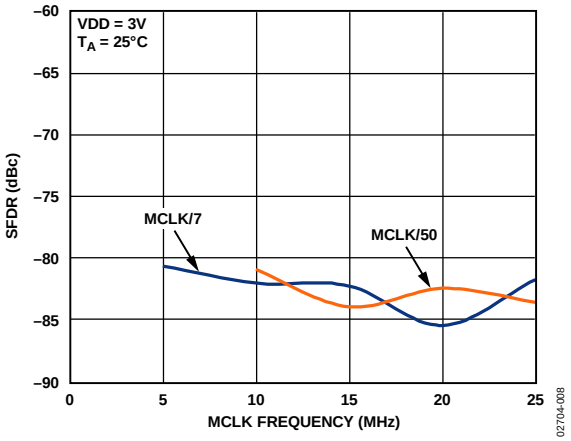


Figure 8. Narrow-Band SFDR vs. MCLK Frequency

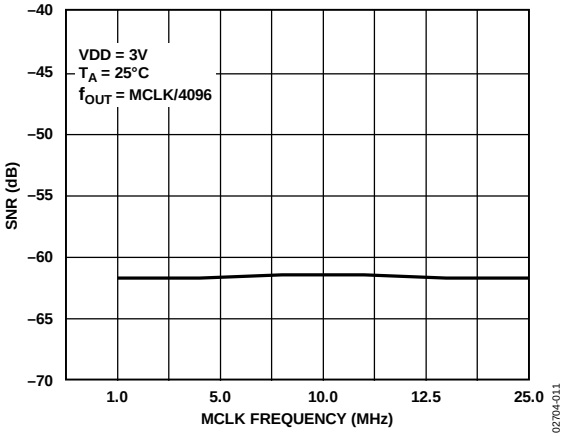


Figure 11. SNR vs. MCLK Frequency

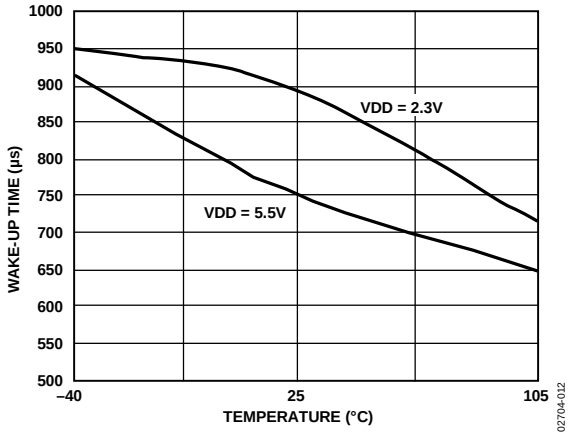


Figure 12. Wake-Up Time vs. Temperature

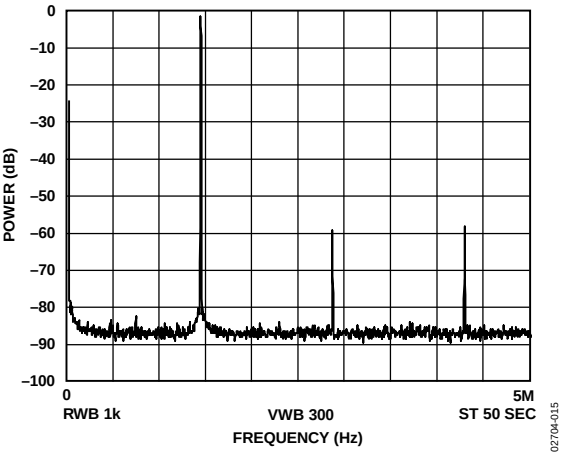


Figure 15. Power vs. Frequency, f_{MCLK} = 10 MHz, f_{OUT} = 1.43 MHz = f_{MCLK}/7, Frequency Word = 0x2492492

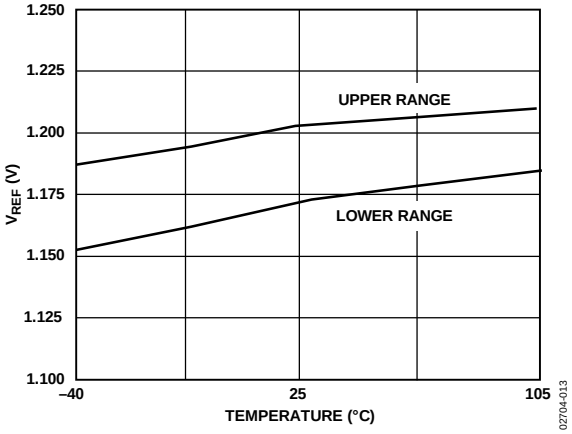


Figure 13. V_{REF} vs. Temperature

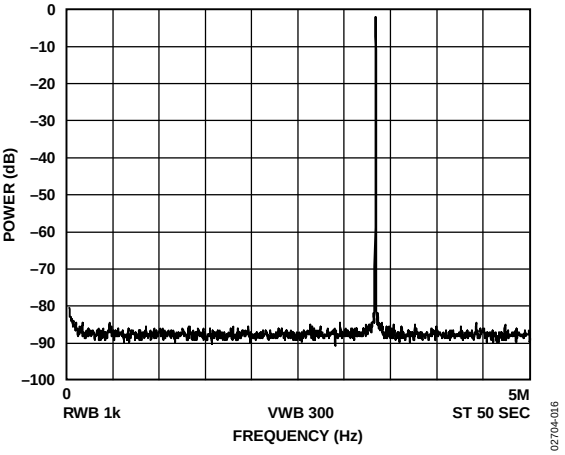


Figure 16. Power vs. Frequency, f_{MCLK} = 10 MHz, f_{OUT} = 3.33 MHz = f_{MCLK}/3, Frequency Word = 0x5555555

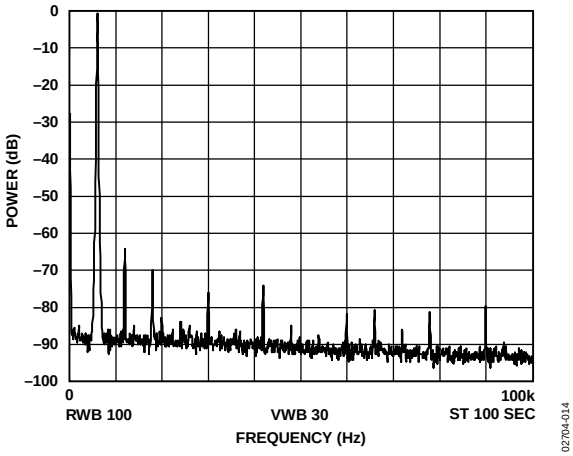


Figure 14. Power vs. Frequency, f_{MCLK} = 10 MHz, f_{OUT} = 2.4 kHz, Frequency Word = 0x000FBA9

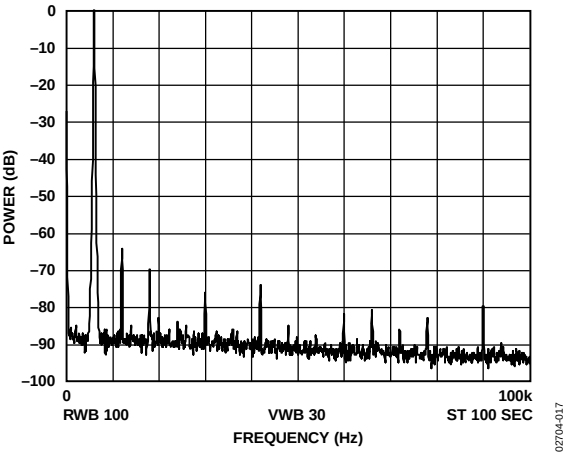


Figure 17. Power vs. Frequency, f_{MCLK} = 25 MHz, f_{OUT} = 6 kHz, Frequency Word = 0x000FBA9

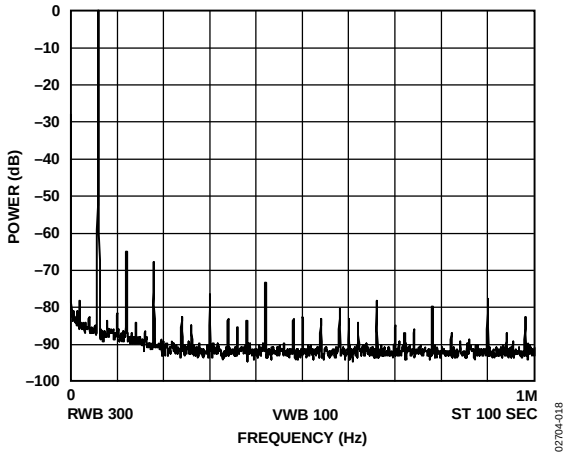


Figure 18. Power vs. Frequency, f_{MCLK} = 25 MHz, f_{OUT} = 60 kHz, Frequency Word = 0x009D495

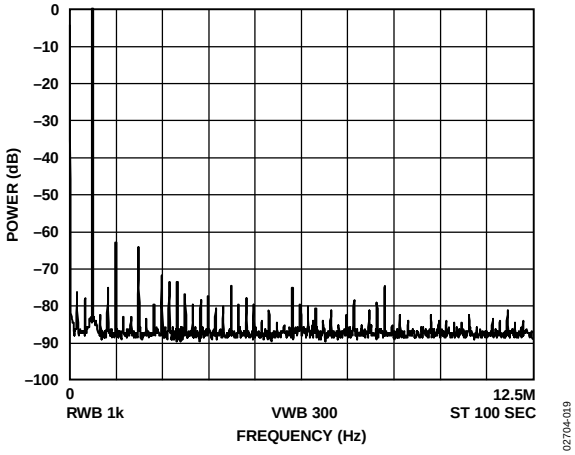


Figure 19. Power vs. Frequency, f_{MCLK} = 25 MHz, f_{OUT} = 600 kHz, Frequency Word = 0x0624DD3

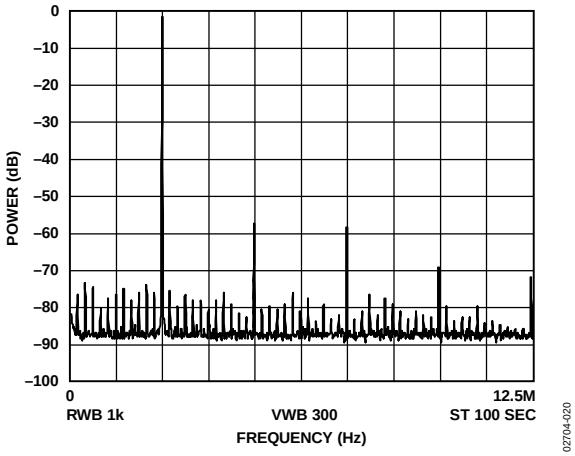


Figure 20. Power vs. Frequency, f_{MCLK} = 25 MHz, f_{OUT} = 2.4 MHz, Frequency Word = 0x189374D

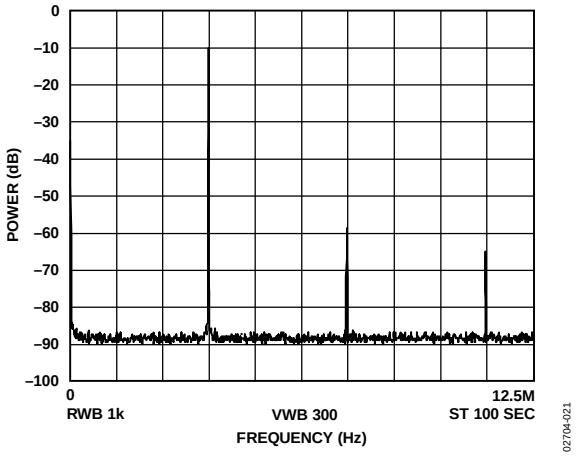


Figure 21. Power vs. Frequency, f_{MCLK} = 25 MHz, f_{OUT} = 3.857 MHz = f_{MCLK}/7, Frequency Word = 0x2492492

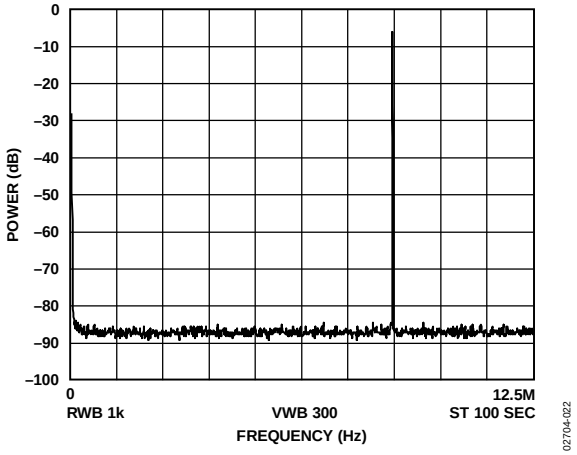


Figure 22. Power vs. Frequency, f_{MCLK} = 25 MHz, f_{OUT} = 8.333 MHz = f_{MCLK}/3, Frequency Word = 0x5555555

TERMINOLOGY

Integral Nonlinearity (INL)
INL is the maximum deviation of any code from a straight line passing through the endpoints of the transfer function. The endpoints of the transfer function are zero scale, a point 0.5 LSB below the first code transition (000 … 00 to 000 … 01), and full scale, a point 0.5 LSB above the last code transition (111 … 10 to 111 … 11). The error is expressed in LSBs.

Differential Nonlinearity (DNL)
DNL is the difference between the measured and ideal 1 LSB change between two adjacent codes in the DAC. A specified DNL of ±1 LSB maximum ensures monotonicity.

Output Compliance
Output compliance refers to the maximum voltage that can be generated at the output of the DAC to meet the specifications. When voltages greater than that specified for the output compliance are generated, the AD9833 may not meet the specifications listed in the data sheet.

Spurious-Free Dynamic Range (SFDR)
Along with the frequency of interest, harmonics of the fundamental frequency and images of these frequencies are present at the output of a DDS device. SFDR refers to the largest spur or harmonic present in the band of interest. The wideband SFDR gives the magnitude of the largest spur or harmonic relative to the magnitude of the fundamental frequency in the zero to Nyquist bandwidth. The narrow-band SFDR gives the attenuation of the largest spur or harmonic in a bandwidth of ±200 kHz about the fundamental frequency.

Total Harmonic Distortion (THD)
THD is the ratio of the rms sum of harmonics to the rms value of the fundamental. For the AD9833, THD is defined as

$$\text{THD} = 20\log\sqrt{\frac{V_2^2 + V_3^2 + V_4^2 + V_5^2 + V_6^2}{V_1}}$$

where:
V_1 is the rms amplitude of the fundamental.
V_2, V_3, V_4, V_5, and V_6 are the rms amplitudes of the second through sixth harmonics.

Signal-to-Noise Ratio (SNR)
SNR is the ratio of the rms value of the measured output signal to the rms sum of all other spectral components below the Nyquist frequency. The value for SNR is expressed in decibels.

Clock Feedthrough
There is feedthrough from the MCLK input to the analog output. Clock feedthrough refers to the magnitude of the MCLK signal relative to the fundamental frequency in the output spectrum of the AD9833.

THEORY OF OPERATION

Sine waves are typically thought of in terms of their magnitude form: a(t) = sin(ωt). However, these sine waves are nonlinear and not easy to generate except through piecewise construction. On the other hand, the angular information is linear in nature. That is, the phase angle rotates through a fixed angle for each unit of time. The angular rate depends on the frequency of the signal by the traditional rate of $\omega = 2\pi f$.

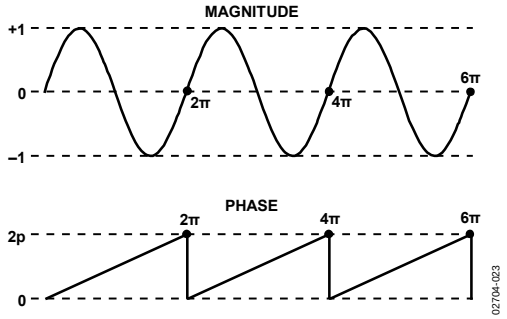


Figure 23. Sine Wave

Knowing that the phase of a sine wave is linear and given a reference interval (clock period), the phase rotation for that period can be determined.

$$\Delta Phase = \omega \Delta t$$

Solving for ω,

$$\omega = \Delta Phase/\Delta t = 2\pi f$$

Solving for f and substituting the reference clock frequency for the reference period ($1/f_{MCLK} = \Delta t$)

$$f = \Delta Phase \times f_{MCLK}/2\pi$$

The AD9833 builds the output based on this simple equation. A simple DDS chip can implement this equation with three major subcircuits: numerically controlled oscillator (NCO) and phase modulator, SIN ROM, and digital-to-analog converter (DAC).

Each subcircuit is described in the Circuit Description section.

CIRCUIT DESCRIPTION

The AD9833 is a fully integrated direct digital synthesis (DDS) chip. The chip requires one reference clock, one low precision resistor, and decoupling capacitors to provide digitally created sine waves up to 12.5 MHz. In addition to the generation of this RF signal, the chip is fully capable of a broad range of simple and complex modulation schemes. These modulation schemes are fully implemented in the digital domain, allowing accurate and simple realization of complex modulation algorithms using DSP techniques.

The internal circuitry of the AD9833 consists of the following main sections: a numerically controlled oscillator (NCO), frequency and phase modulators, SIN ROM, a DAC, and a regulator.

NUMERICALLY CONTROLLED OSCILLATOR PLUS PHASE MODULATOR

This consists of two frequency select registers, a phase accumulator, two phase offset registers, and a phase offset adder. The main component of the NCO is a 28-bit phase accumulator. Continuous time signals have a phase range of 0 to 2π. Outside this range of numbers, the sinusoid functions repeat themselves in a periodic manner. The digital implementation is no different. The accumulator simply scales the range of phase numbers into a multibit digital word. The phase accumulator in the AD9833 is implemented with 28 bits. Therefore, in the AD9833, $2\pi = 2^{28}$. Likewise, the ΔPhase term is scaled into this range of numbers:

$$0 < \Delta\text{Phase} < 2^{28} - 1$$

With these substitutions, the previous equation becomes

$$f = \Delta Phase \times f_{MCLK}/2^{28}$$

where $0 < \Delta Phase < 2^{28} - 1$.

The input to the phase accumulator can be selected from either the FREQ0 register or the FREQ1 register and is controlled by the FSELECT bit. NCOs inherently generate continuous phase signals, thus avoiding any output discontinuity when switching between frequencies.

Following the NCO, a phase offset can be added to perform phase modulation using the 12-bit phase registers. The contents of one of these phase registers are added to the most significant bits of the NCO. The AD9833 has two phase registers; their resolution is 2π/4096.

SIN ROM

To make the output from the NCO useful, it must be converted from phase information into a sinusoidal value. Because phase information maps directly into amplitude, the SIN ROM uses the digital phase information as an address to a lookup table and converts the phase information into amplitude. Although the NCO contains a 28-bit phase accumulator, the output of the NCO is truncated to 12 bits. Using the full resolution of the phase accumulator is impractical and unnecessary, because this would require a lookup table of 2^{28} entries. It is necessary only to have sufficient phase resolution such that the errors due to truncation are smaller than the resolution of the 10-bit DAC. This requires that the SIN ROM have two bits of phase resolution more than the 10-bit DAC.

The SIN ROM is enabled using the mode bit (D1) in the control register (see Table 15).

DIGITAL-TO-ANALOG CONVERTER (DAC)

The AD9833 includes a high impedance, current source 10-bit DAC. The DAC receives the digital words from the SIN ROM and converts them into the corresponding analog voltages.

The DAC is configured for single-ended operation. An external load resistor is not required because the device has a 200 Ω resistor on board. The DAC generates an output voltage of typically 0.6 V p-p.

REGULATOR

VDD provides the power supply required for the analog section and the digital section of the AD9833. This supply can have a value of 2.3 V to 5.5 V.

The internal digital section of the AD9833 is operated at 2.5 V. An on-board regulator steps down the voltage applied at VDD to 2.5 V. When the applied voltage at the VDD pin of the AD9833 is less than or equal to 2.7 V, the CAP/2.5V and VDD pins should be tied together, thus bypassing the on-board regulator.

FUNCTIONAL DESCRIPTION

SERIAL INTERFACE

The AD9833 has a standard 3-wire serial interface that is compatible with the SPI, QSPI™, MICROWIRE®, and DSP interface standards.

Data is loaded into the device as a 16-bit word under the control of a serial clock input, SCLK. The timing diagram for this operation is given in .

The FSYNC input is a level-triggered input that acts as a frame synchronization and chip enable. Data can be transferred into the device only when FSYNC is low. To start the serial data transfer, FSYNC should be taken low, observing the minimum FSYNC-to-SCLK falling edge setup time, t_7. After FSYNC goes low, serial data is shifted into the input shift register of the device on the falling edges of SCLK for 16 clock pulses. FSYNC may be taken high after the 16th falling edge of SCLK, observing the minimum SCLK falling edge to FSYNC rising edge time, t_8. Alternatively, FSYNC can be kept low for a multiple of 16 SCLK pulses and then brought high at the end of the data transfer. In this way, a continuous stream of 16-bit words can be loaded while FSYNC is held low; FSYNC goes high only after the 16th SCLK falling edge of the last word loaded.

The SCLK can be continuous, or it can idle high or low between write operations. In either case, it must be high when FSYNC goes low (t_{11}).

For an example of how to program the AD9833, see the AN-1070 Application Note on the Analog Devices, Inc., website.

POWERING UP THE AD9833

The flowchart in Figure 26 shows the operating routine for the AD9833. When the AD9833 is powered up, the part should be reset. This resets the appropriate internal registers to 0 to provide an analog output of midscale.

To avoid spurious DAC outputs during AD9833 initialization, the reset bit should be set to 1 until the part is ready to begin generating an output. A reset does not reset the phase, frequency, or control registers. These registers will contain invalid data and, therefore, should be set to known values by the user. The reset bit should then be set to 0 to begin generating an output. The data appears on the DAC output seven or eight MCLK cycles after the reset bit is set to 0.

LATENCY PERIOD

A latency period is associated with each asynchronous write operation in the AD9833. If a selected frequency or phase register is loaded with a new word, there is a delay of seven or eight MCLK cycles before the analog output changes. The delay can be seven or eight cycles, depending on the position of the MCLK rising edge when the data is loaded into the destination register.

CONTROL REGISTER

The AD9833 contains a 16-bit control register that allows the user to configure the operation of the AD9833. All control bits other than the mode bit are sampled on the internal falling edge of MCLK.

Table 6 describes the individual bits of the control register. The different functions and the various output options of the AD9833 are described in more detail in the Frequency and Phase Registers section.

To inform the AD9833 that the contents of the control register will be altered, D15 and D14 must be set to 0, as shown in Table 5.

Table 5. Control Register Bits

D15	D14	D13	D0
0	0	Control Bits	

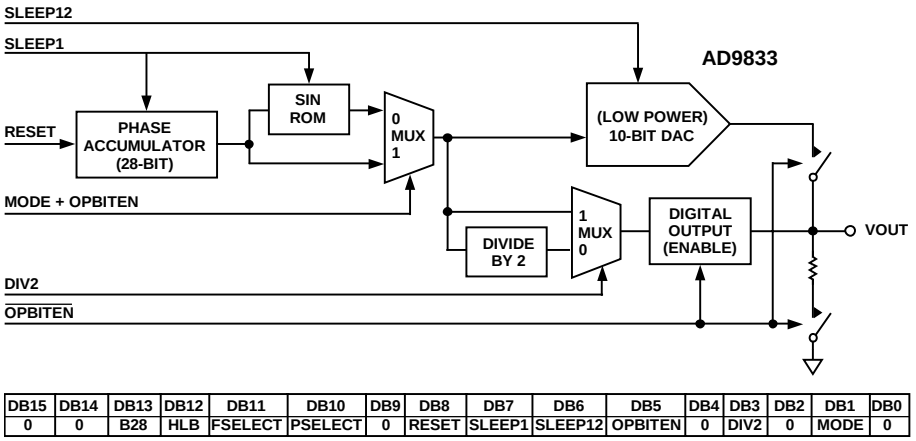


Figure 24. Function of Control Bits

Table 6. Description of Bits in the Control Register

Bit	Name	Function
D13	B28	Two write operations are required to load a complete word into either of the frequency registers. B28 = 1 allows a complete word to be loaded into a frequency register in two consecutive writes. The first write contains the 14 LSBs of the frequency word, and the next write contains the 14 MSBs. The first two bits of each 16-bit word define the frequency register to which the word is loaded and should, therefore, be the same for both of the consecutive writes. See Table 8 for the appropriate addresses. The write to the frequency register occurs after both words have been loaded; therefore, the register never holds an intermediate value. An example of a complete 28-bit write is shown in Table 9. When B28 = 0, the 28-bit frequency register operates as two 14-bit registers, one containing the 14 MSBs and the other containing the 14 LSBs. This means that the 14 MSBs of the frequency word can be altered independent of the 14 LSBs, and vice versa. To alter the 14 MSBs or the 14 LSBs, a single write is made to the appropriate frequency address. The control bit D12 (HLB) informs the AD9833 whether the bits to be altered are the 14 MSBs or 14 LSBs.
D12	HLB	This control bit allows the user to continuously load the MSBs or LSBs of a frequency register while ignoring the remaining 14 bits. This is useful if the complete 28-bit resolution is not required. HLB is used in conjunction with D13 (B28). This control bit indicates whether the 14 bits being loaded are being transferred to the 14 MSBs or 14 LSBs of the addressed frequency register. D13 (B28) must be set to 0 to be able to change the MSBs and LSBs of a frequency word separately. When D13 (B28) = 1, this control bit is ignored. HLB = 1 allows a write to the 14 MSBs of the addressed frequency register. HLB = 0 allows a write to the 14 LSBs of the addressed frequency register.
D11	FSELECT	The FSELECT bit defines whether the FREQ0 register or the FREQ1 register is used in the phase accumulator.
D10	PSELECT	The PSELECT bit defines whether the PHASE0 register or the PHASE1 register data is added to the output of the phase accumulator.
D9	Reserved	This bit should be set to 0.
D8	Reset	Reset = 1 resets internal registers to 0, which corresponds to an analog output of midscale. Reset = 0 disables reset. This function is explained further in Table 13.
D7	SLEEP1	When SLEEP1 = 1, the internal MCLK clock is disabled, and the DAC output remains at its present value because the NCO is no longer accumulating. When SLEEP1 = 0, MCLK is enabled. This function is explained further in Table 14.
D6	SLEEP12	SLEEP12 = 1 powers down the on-chip DAC. This is useful when the AD9833 is used to output the MSB of the DAC data. SLEEP12 = 0 implies that the DAC is active. This function is explained further in Table 14.
D5	OPBITEN	The function of this bit, in association with D1 (mode), is to control what is output at the VOUT pin. This is explained further in Table 15. When OPBITEN = 1, the output of the DAC is no longer available at the VOUT pin. Instead, the MSB (or MSB/2) of the DAC data is connected to the VOUT pin. This is useful as a coarse clock source. The DIV2 bit controls whether it is the MSB or MSB/2 that is output. When OPBITEN = 0, the DAC is connected to VOUT. The mode bit determines whether it is a sinusoidal or a ramp output that is available.
D4	Reserved	This bit must be set to 0.
D3	DIV2	DIV2 is used in association with D5 (OPBITEN). This is explained further in Table 15. When DIV2 = 1, the MSB of the DAC data is passed directly to the VOUT pin. When DIV2 = 0, the MSB/2 of the DAC data is output at the VOUT pin.
D2	Reserved	This bit must be set to 0.
D1	Mode	This bit is used in association with OPBITEN (D5). The function of this bit is to control what is output at the VOUT pin when the on-chip DAC is connected to VOUT. This bit should be set to 0 if the control bit OPBITEN = 1. This is explained further in Table 15. When mode = 1, the SIN ROM is bypassed, resulting in a triangle output from the DAC. When mode = 0, the SIN ROM is used to convert the phase information into amplitude information, which results in a sinusoidal signal at the output.
D0	Reserved	This bit must be set to 0.

FREQUENCY AND PHASE REGISTERS

The AD9833 contains two frequency registers and two phase registers, which are described in Table 7.

Table 7. Frequency and Phase Registers

Register	Size	Description
FREQ0	28 bits	Frequency Register 0. When the FSELECT bit = 0, this register defines the output frequency as a fraction of the MCLK frequency.
FREQ1	28 bits	Frequency Register 1. When the FSELECT bit = 1, this register defines the output frequency as a fraction of the MCLK frequency.
PHASE0	12 bits	Phase Offset Register 0. When the PSELECT bit = 0, the contents of this register are added to the output of the phase accumulator.
PHASE1	12 bits	Phase Offset Register 1. When the PSELECT bit = 1, the contents of this register are added to the output of the phase accumulator.

The analog output from the AD9833 is

$$f_{MCLK}/2^{28} \times FREQREG$$

where *FREQREG* is the value loaded into the selected frequency register. This signal is phase shifted by

$$2\pi/4096 \times PHASEREG$$

where *PHASEREG* is the value contained in the selected phase register. Consideration must be given to the relationship of the selected output frequency and the reference clock frequency to avoid unwanted output anomalies.

The flowchart in Figure 28 shows the routine for writing to the frequency and phase registers of the AD9833.

Writing to a Frequency Register

When writing to a frequency register, Bit D15 and Bit D14 give the address of the frequency register.

Table 8. Frequency Register Bits

D15	D14	D13	D0
0	1	MSB 14 FREQ0 REG bits	LSB
1	0	MSB 14 FREQ1 REG bits	LSB

If the user wants to change the entire contents of a frequency register, two consecutive writes to the same address must be performed because the frequency registers are 28 bits wide. The first write contains the 14 LSBs, and the second write contains the 14 MSBs. For this mode of operation, the B28 (D13) control bit should be set to 1. An example of a 28-bit write is shown in Table 9.

Table 9. Writing 0xFFFC000 to the FREQ0 Register

SDATA Input	Result of Input Word
0010 0000 0000 0000	Control word write (D15, D14 = 00), B28 (D13) = 1, HLB (D12) = X
0100 0000 0000 0000	FREQ0 register write (D15, D14 = 01), 14 LSBs = 0x0000
0111 1111 1111 1111	FREQ0 register write (D15, D14 = 01), 14 MSBs = 0x3FFF

In some applications, the user does not need to alter all 28 bits of the frequency register. With coarse tuning, only the 14 MSBs are altered, while with fine tuning, only the 14 LSBs are altered. By setting the B28 (D13) control bit to 0, the 28-bit frequency register operates as two, 14-bit registers, one containing the 14 MSBs and the other containing the 14 LSBs. This means that the 14 MSBs of the frequency word can be altered independent of the 14 LSBs, and vice versa. Bit HLB (D12) in the control register identifies which 14 bits are being altered. Examples of this are shown in Table 10 and Table 11.

Table 10. Writing 0x3FFF to the 14 LSBs of the FREQ1 Register

SDATA Input	Result of Input Word
0000 0000 0000 0000	Control word write (D15, D14 = 00), B28 (D13) = 0; HLB (D12) = 0, that is, LSBs
1011 1111 1111 1111	FREQ1 REG write (D15, D14 = 10), 14 LSBs = 0x3FFF

Table 11. Writing 0x00FF to the 14 MSBs of the FREQ0 Register

SDATA Input	Result of Input Word
0001 0000 0000 0000	Control word write (D15, D14 = 00), B28 (D13) = 0, HLB (D12) = 1, that is, MSBs
0100 0000 1111 1111	FREQ0 REG write (D15, D14 = 01), 14 MSBs = 0x00FF

Writing to a Phase Register

When writing to a phase register, Bit D15 and Bit D14 are set to 11. Bit D13 identifies which phase register is being loaded.

Table 12. Phase Register Bits

D15	D14	D13	D12	D11	D0
1	1	0	X	MSB 12 PHASE0 bits	LSB
1	1	1	X	MSB 12 PHASE1 bits	LSB

RESET FUNCTION

The reset function resets appropriate internal registers to 0 to provide an analog output of midscale. Reset does not reset the phase, frequency, or control registers. When the AD9833 is powered up, the part should be reset. To reset the AD9833, set the reset bit to 1. To take the part out of reset, set the bit to 0. A signal appears at the DAC to output eight MCLK cycles after reset is set to 0.

Table 13. Applying the Reset Function

Reset Bit	Result
0	No reset applied
1	Internal registers reset

SLEEP FUNCTION

Sections of the AD9833 that are not in use can be powered down to minimize power consumption. This is done using the sleep function. The parts of the chip that can be powered down are the internal clock and the DAC. The bits required for the sleep function are outlined in Table 14.

Table 14. Applying the Sleep Function

SLEEP1 Bit	SLEEP12 Bit	Result
0	0	No power-down
0	1	DAC powered down
1	0	Internal clock disabled
1	1	Both the DAC powered down and the internal clock disabled

DAC Powered Down

This is useful when the AD9833 is used to output the MSB of the DAC data only. In this case, the DAC is not required; therefore, it can be powered down to reduce power consumption.

Internal Clock Disabled

When the internal clock of the AD9833 is disabled, the DAC output remains at its present value because the NCO is no longer accumulating. New frequency, phase, and control words can be written to the part when the SLEEP1 control bit is active. The synchronizing clock is still active, which means that the selected frequency and phase registers can also be changed using the control bits. Setting the SLEEP1 bit to 0 enables the MCLK. Any changes made to the registers while SLEEP1 is active will be seen at the output after a latency period.

VOUT PIN

The AD9833 offers a variety of outputs from the chip, all of which are available from the VOUT pin. The choice of outputs is the MSB of the DAC data, a sinusoidal output, or a triangle output.

The OPBITEN (D5) and mode (D1) bits in the control register are used to decide which output is available from the AD9833.

MSB of the DAC Data

The MSB of the DAC data can be output from the AD9833. By setting the OPBITEN (D5) control bit to 1, the MSB of the DAC data is available at the VOUT pin. This is useful as a coarse clock source. This square wave can also be divided by 2 before being output. The DIV2 (D3) bit in the control register controls the frequency of this output from the VOUT pin.

Sinusoidal Output

The SIN ROM is used to convert the phase information from the frequency and phase registers into amplitude information that results in a sinusoidal signal at the output. To have a sinusoidal output from the VOUT pin, set the mode (D1) bit to 0 and the OPBITEN (D5) bit to 0.

Triangle Output

The SIN ROM can be bypassed so that the truncated digital output from the NCO is sent to the DAC. In this case, the output is no longer sinusoidal. The DAC will produce a 10-bit linear triangular function. To have a triangle output from the VOUT pin, set the mode (D1) bit = 1.

Note that the SLEEP12 bit must be 0 (that is, the DAC is enabled) when using this pin.

Table 15. Outputs from the VOUT Pin

OPBITEN Bit	Mode Bit	DIV2 Bit	VOUT Pin
0	0	X[1]	Sinusoid
0	1	X[1]	Triangle
1	0	0	DAC data MSB/2
1	0	1	DAC data MSB
1	1	X[1]	Reserved

[1] X = don't care.

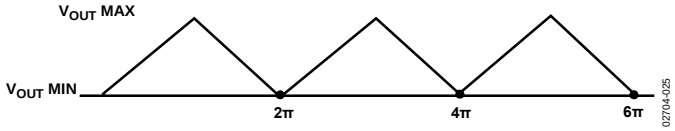


Figure 25. Triangle Output

APPLICATIONS INFORMATION

Because of the various output options available from the part, the AD9833 can be configured to suit a wide variety of applications.

One of the areas where the AD9833 is suitable is in modulation applications. The part can be used to perform simple modulation, such as FSK. More complex modulation schemes, such as GMSK and QPSK, can also be implemented using the AD9833.

In an FSK application, the two frequency registers of the AD9833 are loaded with different values. One frequency represents the space frequency, while the other represents the mark frequency. Using the FSELECT bit in the control register of the AD9833, the user can modulate the carrier frequency between the two values.

The AD9833 has two phase registers, which enables the part to perform PSK. With phase-shift keying, the carrier frequency is phase shifted, the phase being altered by an amount that is related to the bit stream being input to the modulator.

The AD9833 is also suitable for signal generator applications. Because the MSB of the DAC data is available at the VOUT pin, the device can be used to generate a square wave.

With its low current consumption, the part is suitable for applications in which it can be used as a local oscillator.

GROUNDING AND LAYOUT

The printed circuit board (PCB) that houses the AD9833 should be designed so that the analog and digital sections are separated and confined to certain areas of the board. This facilitates the use of ground planes that can be separated easily. A minimum etch technique is generally best for ground planes because it gives the best shielding. Digital and analog ground planes should be joined in one place only. If the AD9833 is the only device requiring an AGND-to-DGND connection, then the ground planes should be connected at the AGND and DGND pins of the AD9833. If the AD9833 is in a system where multiple devices require AGND-to-DGND connections, the connection should be made at one point only, a star ground point that should be established as close as possible to the AD9833.

Avoid running digital lines under the device as these couple noise onto the die. The analog ground plane should be allowed to run under the AD9833 to avoid noise coupling. The power supply lines to the AD9833 should use as large a track as possible to provide low impedance paths and reduce the effects of glitches on the power supply line. Fast switching signals, such as clocks, should be shielded with digital ground to avoid radiating noise to other sections of the board.

Avoid crossover of digital and analog signals. Traces on opposite sides of the board should run at right angles to each other. This reduces the effects of feedthrough through the board. A microstrip technique is by far the best, but it is not always possible with a double-sided board. In this technique, the component side of the board is dedicated to ground planes, and signals are placed on the other side.

Good decoupling is important. The AD9833 should have supply bypassing of 0.1 μF ceramic capacitors in parallel with 10 μF tantalum capacitors. To achieve the best performance from the decoupling capacitors, they should be placed as close as possible to the device, ideally right up against the device.

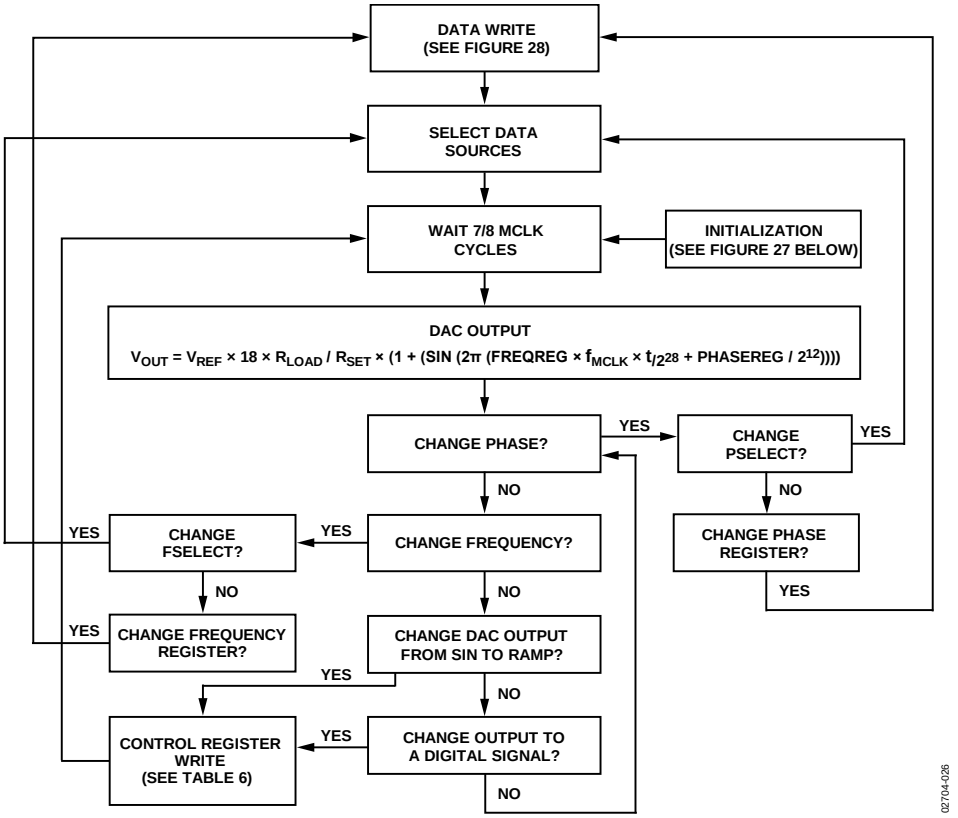


Figure 26. Flowchart for AD9833 Initialization and Operation

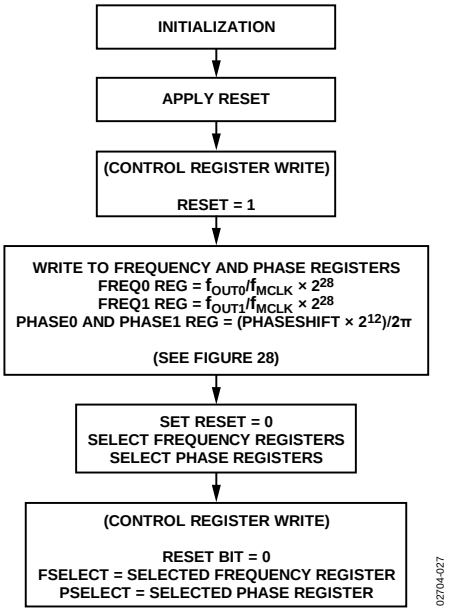


Figure 27. Flowchart for Initialization

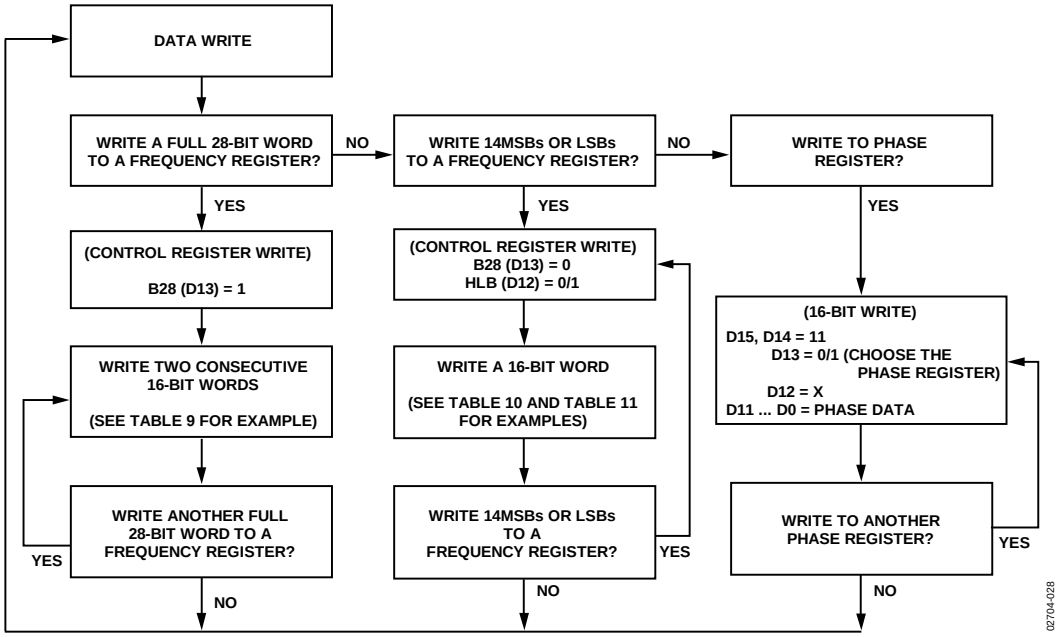


Figure 28. Flowchart for Data Writes

INTERFACING TO MICROPROCESSORS

The AD9833 has a standard serial interface that allows the part to interface directly with several microprocessors. The device uses an external serial clock to write the data or control information into the device. The serial clock can have a frequency of 40 MHz maximum. The serial clock can be continuous, or it can idle high or low between write operations. When data or control information is written to the AD9833, FSYNC is taken low and is held low until the 16 bits of data are written into the AD9833. The FSYNC signal frames the 16 bits of information that are loaded into the AD9833.

AD9833 TO 68HC11/68L11 INTERFACE

Figure 29 shows the serial interface between the AD9833 and the 68HC11/68L11 microcontroller. The microcontroller is configured as the master by setting the MSTR bit in the SPCR to 1. This setting provides a serial clock on SCK; the MOSI output drives the serial data line SDATA. Because the microcontroller does not have a dedicated frame sync pin, the FSYNC signal is derived from a port line (PC7). The setup conditions for correct operation of the interface are as follows:

- SCK idles high between write operations (CPOL = 0)
- Data is valid on the SCK falling edge (CPHA = 1)

When data is being transmitted to the AD9833, the FSYNC line is taken low (PC7). Serial data from the 68HC11/68L11 is transmitted in 8-bit bytes with only eight falling clock edges occurring in the transmit cycle. Data is transmitted MSB first. To load data into the AD9833, PC7 is held low after the first eight bits are transferred, and a second serial write operation is performed to the AD9833. Only after the second eight bits are transferred should FSYNC be taken high again.

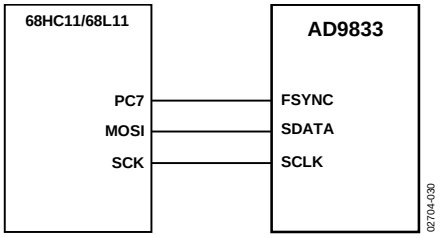


Figure 29. 68HC11/68L11 to AD9833 Interface

AD9833 TO 80C51/80L51 INTERFACE

Figure 30 shows the serial interface between the AD9833 and the 80C51/80L51 microcontroller. The microcontroller is operated in Mode 0 so that TxD of the 80C51/80L51 drives SCLK of the AD9833, and RxD drives the serial data line SDATA. The FSYNC signal is derived from a bit programmable pin on the port (P3.3 is shown in Figure 30).

When data is to be transmitted to the AD9833, P3.3 is taken low. The 80C51/80L51 transmits data in 8-bit bytes, thus only eight falling SCLK edges occur in each cycle. To load the remaining eight bits to the AD9833, P3.3 is held low after the first eight bits are transmitted, and a second write operation is initiated to transmit the second byte of data. P3.3 is taken high following the completion of the second write operation. SCLK should idle high between the two write operations.

The 80C51/80L51 outputs the serial data in a format that has the LSB first. The AD9833 accepts the MSB first (the four MSBs are the control information, the next four bits are the address, and the eight LSBs contain the data when writing to a destination register). Therefore, the transmit routine of the 80C51/80L51 must take this into account and rearrange the bits so that the MSB is output first.

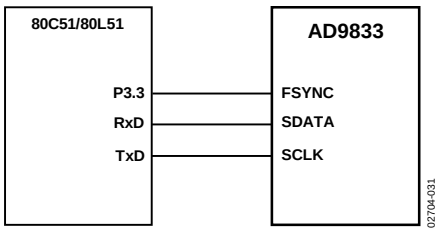


Figure 30. 80C51/80L51 to AD9833 Interface

AD9833 TO DSP56002 INTERFACE

Figure 31 shows the interface between the AD9833 and the DSP56002. The DSP56002 is configured for normal mode asynchronous operation with a gated internal clock (SYN = 0, GCK = 1, SCKD = 1). The frame sync pin is generated internally (SC2 = 1), the transfers are 16 bits wide (WL1 = 1, WL0 = 0), and the frame sync signal frames the 16 bits (FSL = 0). The frame sync signal is available on the SC2 pin, but it must be inverted before it is applied to the AD9833. The interface to the DSP56000/DSP56001 is similar to that of the DSP56002.

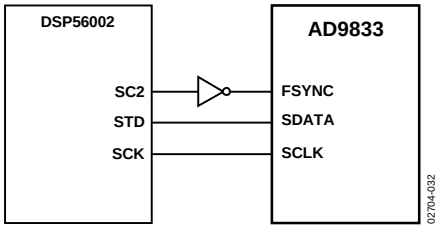


Figure 31. DSP56002 to AD9833 Interface

EVALUATION BOARD

The AD9833 evaluation board allows designers to evaluate the high performance AD9833 DDS modulator with a minimum of effort.

SYSTEM DEMONSTRATION PLATFORM

The system demonstration platform (SDP) is a hardware and software evaluation tool for use in conjunction with product evaluation boards. The SDP board is based on the Blackfin® ADSP-BF527 processor with USB connectivity to the PC through a USB 2.0 high speed port. For more information about the SDP board, see the SDP board product page.

Note that the SDP board is sold separately from the AD9833 evaluation board.

AD9833 TO SPORT INTERFACE

The Analog Devices SDP board has a SPORT serial port that is used to control the serial inputs to the AD9833. The connections are shown in Figure 32.

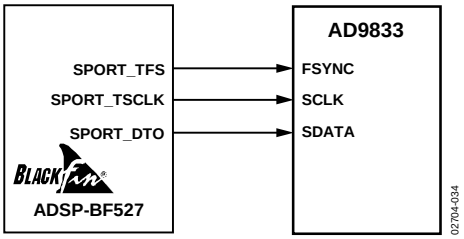


Figure 32. SDP to AD9833 Interface

EVALUATION KIT

The DDS evaluation kit includes a populated, tested AD9833 printed circuit board (PCB). The schematics of the evaluation board are shown in Figure 34 and Figure 35.

The software provided in the evaluation kit allows the user to easily program the AD9833 (see Figure 33). The evaluation software runs on any IBM-compatible PC with Microsoft® Windows® software installed (including Windows 7). The software is compatible with both 32-bit and 64-bit operating systems.

More information about the evaluation software is available on the software CD and on the AD9833 product page.

Figure 33. AD9833 Evaluation Software Interface

CRYSTAL OSCILLATOR VS. EXTERNAL CLOCK

The AD9833 can operate with master clocks up to 25 MHz. A 25 MHz oscillator is included on the evaluation board. This oscillator can be removed and, if required, an external CMOS clock can be connected to the part. Options for the general oscillator include the following:

- AEL 301-Series oscillators, AEL Crystals
- SG-310SCN oscillators, Epson Electronics

POWER SUPPLY

Power to the AD9833 evaluation board can be provided from the USB connector or externally through pin connections. The power leads should be twisted to reduce ground loops.

EVALUATION BOARD SCHEMATICS

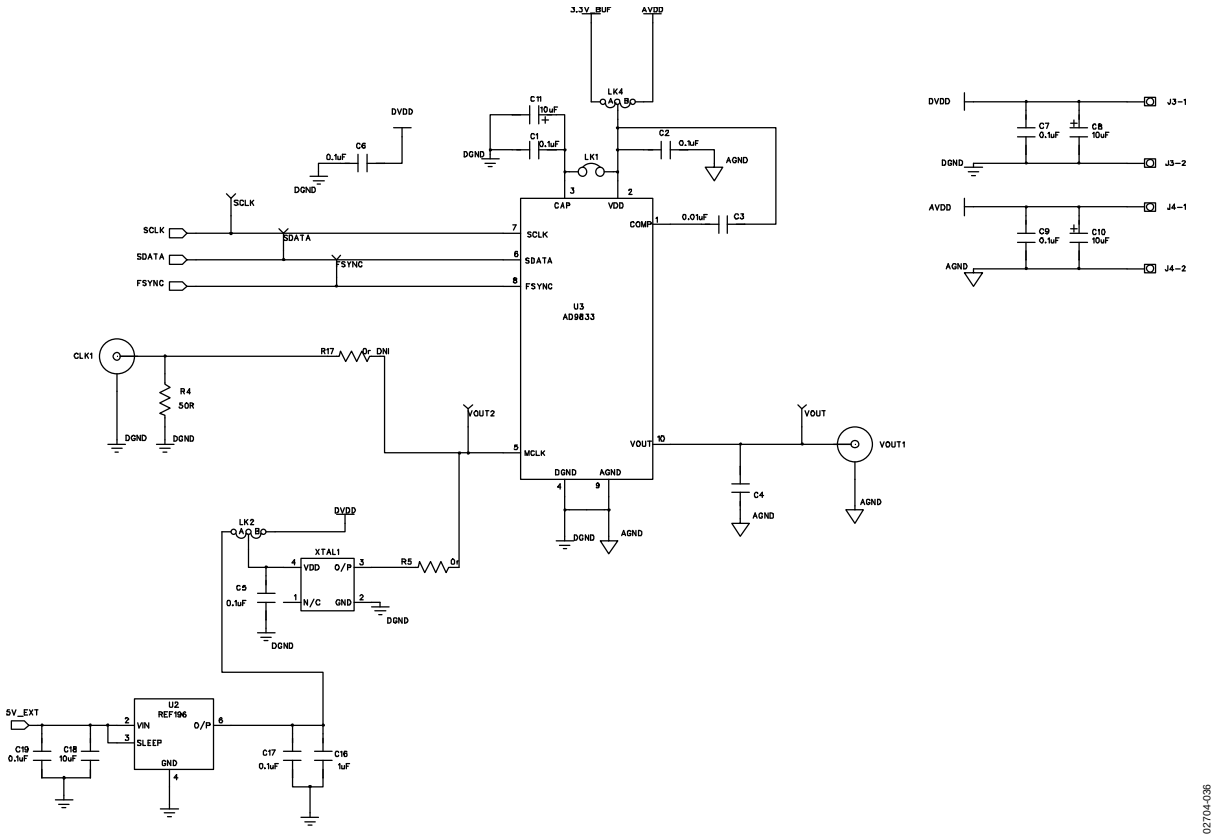


Figure 34. Evaluation Board Schematic

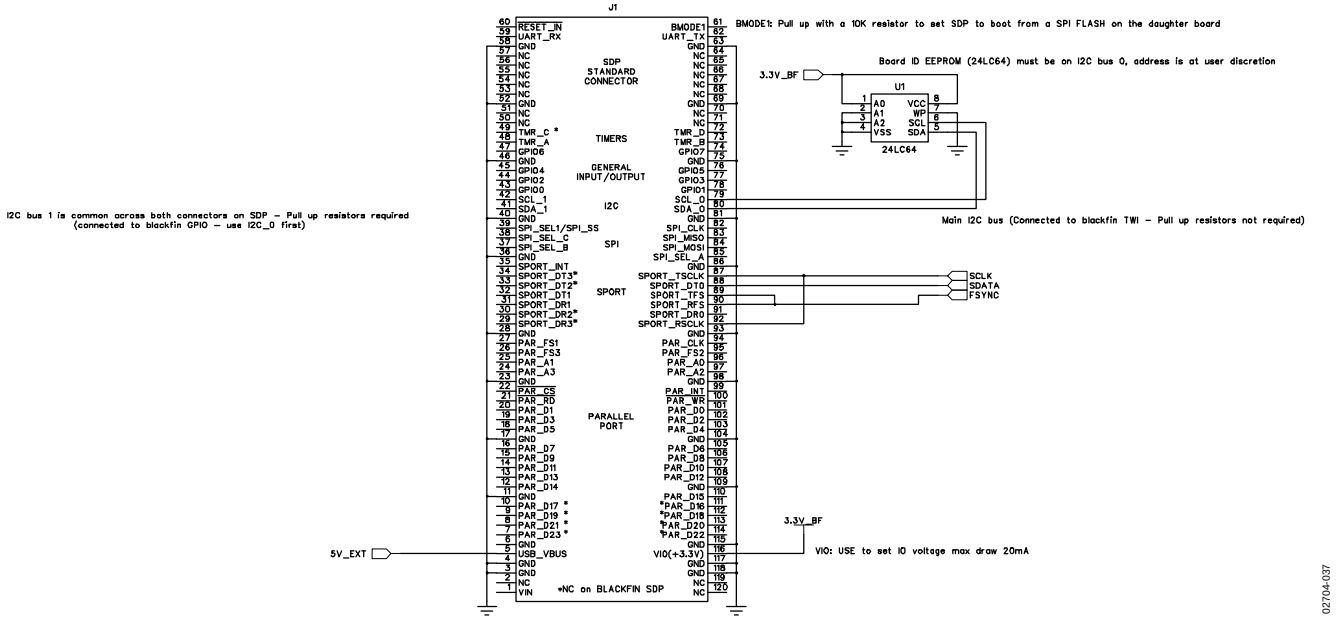


Figure 35. SDP Connector Schematic

EVALUATION BOARD LAYOUT

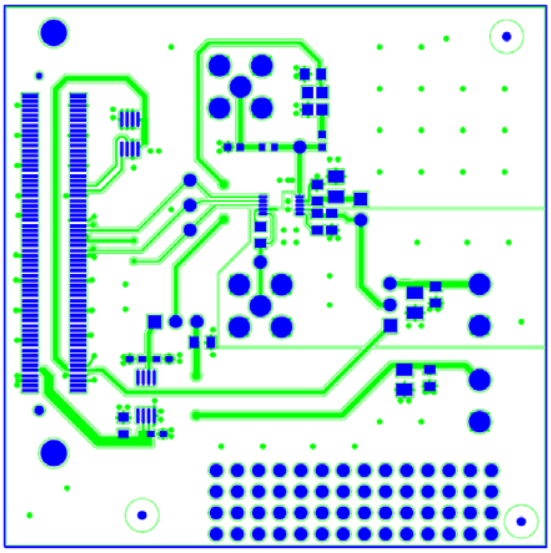


Figure 36. AD9833 Evaluation Board Component Side

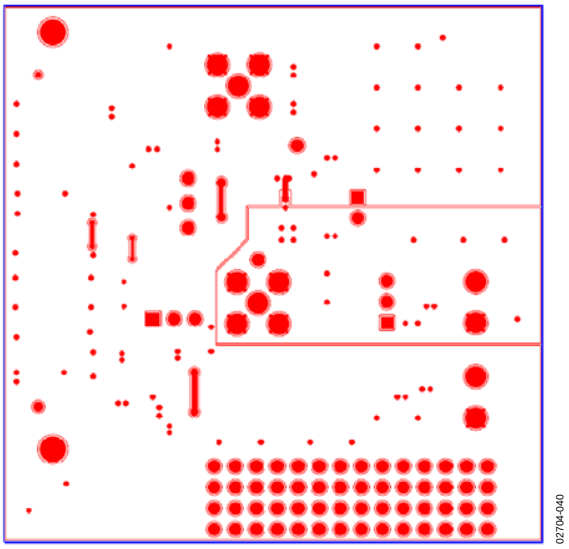


Figure 38. AD9833 Evaluation Board Solder Side

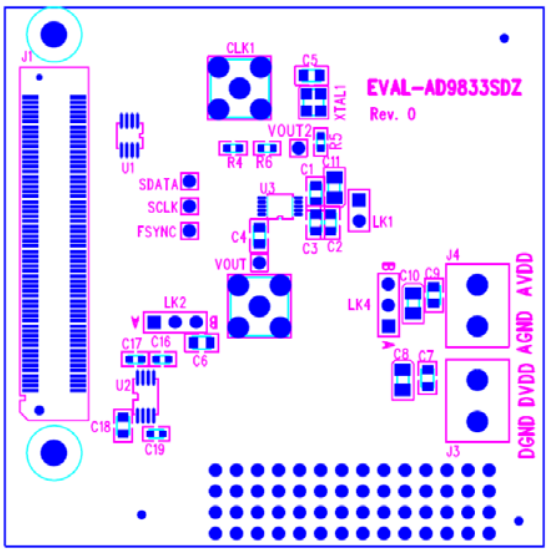


Figure 37. AD9833 Evaluation Board Silkscreen

OUTLINE DIMENSIONS

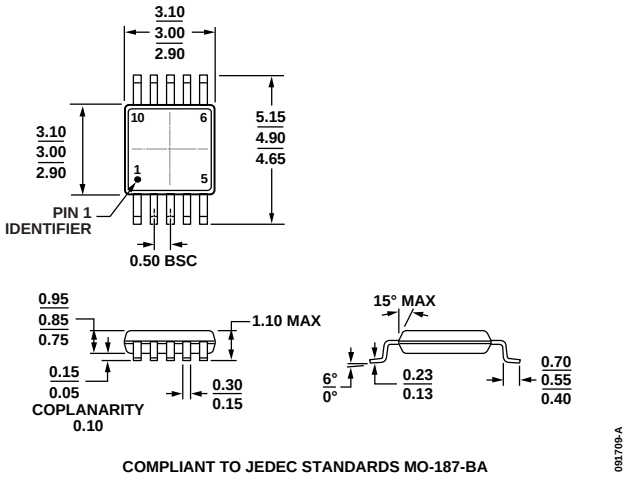


Figure 39. 10-Lead Mini Small Outline Package [MSOP]
(RM-10)
Dimensions shown in millimeters

ORDERING GUIDE

Model[1, 2, 3]	Temperature Range	Package Description	Package Option	Branding
AD9833BRM	−40°C to +105°C	10-Lead MSOP	RM-10	DJB
AD9833BRM-REEL	−40°C to +105°C	10-Lead MSOP	RM-10	DJB
AD9833BRM-REEL7	−40°C to +105°C	10-Lead MSOP	RM-10	DJB
AD9833BRMZ	−40°C to +105°C	10-Lead MSOP	RM-10	D68
AD9833BRMZ-REEL	−40°C to +105°C	10-Lead MSOP	RM-10	D68
AD9833BRMZ-REEL7	−40°C to +105°C	10-Lead MSOP	RM-10	D68
AD9833WBRMZ-REEL	−40°C to +105°C	10-Lead MSOP	RM-10	D68
EVAL-AD9833SDZ		Evaluation Board		

[1] Z = RoHS Compliant Part.
[2] W = Qualified for Automotive Applications.
[3] The evaluation board for the AD9833 requires the system demonstration platform (SDP) board, which is sold separately.

AUTOMOTIVE PRODUCTS

The AD9833WBRMZ-REEL model is available with controlled manufacturing to support the quality and reliability requirements of automotive applications. Note that this automotive model may have specifications that differ from the commercial models; therefore, designers should review the Specifications section of this data sheet carefully. Only the automotive grade product shown is available for use in automotive applications. Contact your local Analog Devices account representative for specific product ordering information and to obtain the specific Automotive Reliability reports for these models.

Bibliografia

[1] Helper macros for baud rate calculations: http://www.nongnu.org/avr-libc/user-manual/group__util__setbaud.html

[2] Viquipèdia - Oscil·lador: http://ca.wikipedia.org/wiki/Oscil%C2%B7lador

[3] Wikipedia - Open Source Hardware Projects: http://en.wikipedia.org/wiki/List_of_open_source_hardware_projects

[4] Wikipedia - Open Sorce Computing Hardware: http://en.wikipedia.org/wiki/Open-source_computing_hardware

[5] OLinuXino - Olimex: https://www.olimex.com/Products/OLinuXino/open-source-hardware

[6] BeagleBoard: http://beagleboard.org/Products/BeagleBoard

[7] Arduino web: http://www.arduino.cc/

[8] Wikipedia -Arduino Compatible Systems: http://en.wikipedia.org/wiki/List_of_Arduino_boards_and_compatible_systems

[9] Arduino Shield List: http://shieldlist.org/

[10] AD9837 Evaluation Board - Analog Devices: http://www.analog.com/en/evaluation/eval-ad9837/eb.html

[11] Viquipèdia - Modulació per desplaçament d'amplitud: http://ca.wikipedia.org/wiki/Modulaci%C3%B3_per_despla%C3%A7ament_d%27amplitud

[12] Wikipedia - Bell 202 modem : http://en.wikipedia.org/wiki/Bell_202_modem

[13] Wikipedia - Bell 103 modem : http://en.wikipedia.org/wiki/Bell_103_modem

[14] Wikipedia - Phase-shift keying : http://en.wikipedia.org/wiki/Phase-shift_keying

[15] ATMEL - "8-bit Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash" www.atmel.com/Images/doc8161.pdf

[16] Murata - "Specification of Piezoelectric Ceramic Resonator" : http://www.farnell.com/datasheets/19280.pdf

[17] Viquipèdia - Serial Peripheral Interface: http://ca.wikipedia.org/wiki/Serial_Peripheral_Interface

[18] Low Power, 8.5 mW, 2.3 V to 5.5 V, Programmable Waveform Generator AD9837 http://www.analog.com/static/imported-files/data_sheets/AD9837.PDF

[19] National Instruments - "Understanding Direct Digital Synthesis (DDS)" : http://www.ni.com/white-paper/5516/en/

[20] "A Technical Tutorial on Digital Signal Synthesis" - 1999 Analog Devices, Inc. : http://www.ieee.li/pdf/essay/dds.pdf

[21] "Low Power, 12.65 mW, 2.3 V to 5.5 V, Programmable Waveform Generator AD9833" - Analog Devices Inc. : http://www.analog.com/static/imported-files/data_sheets/AD9833.pdf

[22] Pallás Areny. R. "Instrumentos electrónicos básicos" Marcombo, 2006. - 317p

[23] Texas Instruments - "MAX232, MAX232I DUAL EIA-232 DRIVERS/RECEIVERS" : http://www.ti.com/lit/ds/symlink/max232.pdf

[24] Future Technology Devices International Ltd. - "FT232R USB UART IC" :http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232R.pdf

[25] LowFER - Wikipedia http://en.wikipedia.org/wiki/LowFER

[26] Bandas Atribuidas - 2200m-137kHz - "URE" : http://www.ure.es/principal/bandas-atribuidas/17-2200-m-137-khz.html