Universitat Oberta de Catalunya

uoc.edu

Determinación de posición y velocidad de satélites mediante algoritmos acelerados por hardware

Juan Jose Garrido Serrato
Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación
Sistemas de Comunicación

Victor Martinez Illamola
Carlos Monzo Sánchez

junio de 2020

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Determinación de posición y velocidad de satélites mediante algoritmos acelerados por hardware*
Nombre del autor:	*Juan Jose Garrido Serrato*
Nombre del consultor/a:	Victor Martinez Illamola
Nombre del PRA:	Raúl Parada Medina
Fecha de entrega (mm/aaaa):	06/2020
Titulación:	*Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación*
Área del Trabajo Final:	*Sistemas de Comunicación*
Idioma del trabajo:	*ES-es*
Palabras clave	*Algoritmos matemáticos. Autocorrelación y Doppler. FPGA. Software empotrado (embedded software). SDR (Software Defined Radio).*

Resumen

El objetivo de este TFG es proponer una serie de procedimientos que puedan ser utilizados para la prueba de transpondedores embarcados en satélites en órbitas bajas y medias. Estos procedimientos se describirán en forma de algoritmos matemáticos y se implementarán sobre un SoC (System On Chip), usando herramientas de generación de código orientadas a la integración en plataformas empotradas, para conseguir un rendimiento óptimo. Más concretamente, se hará uso de las características matemáticas de correlación para determinar el retardo de señal, proporcionando información sobre la distancia a la que se encuentra el satélite. Otra de las características que se medirán serán el efecto Doppler producido por el movimiento relativo entre el satélite y la estación terrena, que revelará la velocidad relativa entre ambos. Estos dos valores, junto con los valores de elevación y azimut de la antena, proporcionaran una estimación precisa de la posición y velocidad del satélite. Finalmente, los algoritmos se implementarán sobre una plataforma de prototipado basada en SDR (Software Defined Radio).

Abstract

The goal of this final degree thesis is to propose a set of procedures that can be used in the context of low and medium earth satellite transponder testing. These procedures will be described in the form of mathematical algorithms and will be implemented in a SoC (System-on-chip) device, using code generation tools oriented to embedded systems, to allow optimum performance. More specifically, we will use the mathematical features of the correlation to determine the group delay, giving information of the satellite distance. Another measurement will be the Doppler effect produced by the relative movement between the satellite and the ground station, from this value we will obtain the relative speed. These 2 values, if we take in consideration also the elevation and azimuth of the antenna, will provide an accurate estimation of the position and speed of the satellite. Finally, the algorithms will be implemented on a SDR (Software Defined Radio) board.

Índice

Lista de figuras

1. Introducción

Este TFG se encuentra orientado al ámbito del procesado digital de señal y en el cambio de ciclo que muchas áreas de la tecnología están sufriendo. Históricamente el campo teórico siempre disfrutó de una posición muy alejada del desarrollo real práctico. Los algoritmos matemáticos simplemente necesitaban una potencia de cálculo que los sistemas que evolucionaban a la par no eran capaces de entregar. Esta situación comenzó a cambiar a medida que se desarrollaron las capacidades de fabricación de circuitos integrados, llegando lentamente a alcanzar el punto en que el procesado matemático era suficiente para poder implementar algoritmos ya desarrollados en un periodo razonablemente reciente, y permitir la introducción de nuevas herramientas para un considerable número de diversas aplicaciones.

1.1 Motivación

Con la introducción de transpondedores en los satélites de órbita baja y media, se consiguen comunicaciones con una latencia considerablemente menor que con los transpondedores embarcados en vehículos en órbitas geoestacionarias. Un inconveniente intrínseco a este tipo de órbitas es que el satélite deja de ser un punto estático en el cielo, desde el sistema de referencia que es la estación terrena. Junto con la expansión de este tipo de constelaciones de satélites, han ido apareciendo nuevos tipos de equipos para facilitar la operación y el mantenimiento de estos sistemas, aunque aún hay aspectos donde existe un vacío considerable.

Este es el caso de la fase IOT (*In-Orbit-Testing*), donde se ejecutan una serie de procedimientos para verificar el estado de los transpondedores a bordo del vehículo. Los procedimientos consisten en pruebas deterministas, donde se envía de una serie de señales con características muy predecibles, y después se analiza su recepción tras haber sido reenviadas por el transponder a diagnosticar. La lista de parámetros que se estudian en este tipo de pruebas no ha sufrido grandes cambios desde los años ochenta, época en que se estandarizan las campañas IOT por los primeros operadores de satélites. Con la proliferación de las constelaciones en otras órbitas distintas a la geoestacionaria, se hace necesario añadir nuevos procesos para poder medir las nuevas dimensiones derivadas del movimiento relativo que existe entre la estación terrena y el satélite. A pesar de que se podrían añadir bastantes nuevas variables para monitorizar, con solamente tres parámetros más podremos determinar con una precisión razonable la posición del transponder. Estos parámetros son el retardo, Doppler y la configuración de apuntado de antena (azimut y elevación). Mientras que la información de apuntado de la antena es intrínseca a la estación terrena y está disponible a través de los propios equipos de apuntado, el resto de los parámetros tienen que ser calculados a partir de las señales transmitidas y recibidas por la estación. Es aquí donde los avances en el procesado digital de

señal pueden ayudar a determinar estos valores, aunque la tecnología ha existido desde hace tiempo, es ahora cuando podemos incorporarla en pequeños dispositivos gracias a los avances en circuitos ASIC (Application Specific Integrated Circuit) y FPGA (*Field Programmable Gate Array*).

Un aspecto relevante en la evolución de los circuitos integrados es la incorporación de procesadores y otros periféricos a la lógica programable. Este es el caso de la serie Zynq de Xilinx, que pone a disposición del usuario un número variable de procesadores ARM A9 (para la serie 7000). De esta manera es posible ejecutar un sistema operativo razonablemente potente (como Linux o FreeRTOS) a la vez que se accede al hardware que se ha programado en el área dedicada puramente a tecnología FPGA. De esta manera, en un único chip, es posible contener la mayoría de los componentes de un sistema, de ahí su denominación de *System-On-Chip.*

1.2 Objetivos del Trabajo

El presente proyecto se propone alcanzar los siguientes objetivos:

- Desarrollar los algoritmos de correlación y Doppler en Matlab y, usando las herramientas proporcionadas por Matlab y Xilinx.
- Usar las herramientas de generación de código para implementar dichos algoritmos en una plataforma de desarrollo basada en el SoC Zynq-7000, fabricado por Xilinx, y transceptores ADALM-PLUTO fabricados por Analog Devices.
- Validar los resultados obtenidos usando una simulación con Mathworks Simulink.
- Validar los resultados usando un simulador de canal construido con dispositivos SDR adicionales y el software GNU Radio con el modelo de simulación GR-LEO [1].

De esta manera completaremos un flujo de trabajo completo que cubrirá desde la idea y formación de un algoritmo desde un punto de vista matemático, hasta la validación de un prototipo compuesto de software y de hardware. Un flujo muy similar al que se seguiría en la ejecución de un proyecto industrial.

1.3 Metodología

En este TFG seguiremos una metodología basada en el flujo de trabajo propuesto por *Mathworks* y *Analog Devices* [2], donde se sigue un proceso evolutivo a partir de unos requisitos y que termina con la elaboración de un prototipo. El orden y la descripción de estas fases son:

1. Definición de requisitos. Estos requisitos servirán de base para la elección de parámetros en el diseño e implementación, y para determinar si la validación es exitosa.

2. Desarrollo de un algoritmo matemático. Usando Matlab se desarrollará el algoritmo de manera puramente matemática (con coma flotante) y se verificarán los requisitos mediante simulación dentro de Matlab. El resultado numérico de este algoritmo servirá para verificar la validez de las sucesivas evoluciones (*Golden Rule*).

3. Desarrollo de un modelo en Simulink para convertir los algoritmos al dominio del tiempo (muestras), que se volverán a verificar contra los requisitos mediante simulación dentro de Matlab.

4. Se realizará una co-simulación usando como transmisores y receptores dos dispositivos SDR ADALM-PLUTO conectados a Matlab.

5. Implementación de los algoritmos, usando las herramientas de Matlab y Xilinx, con despliegue en una placa de desarrollo Zedboard, con SoC Zynq-7000, donde validará la ejecución de los algoritmos.

6. Integración de los algoritmos en un sistema consolidado que permita la operación de manera autónoma a las herramientas de despliegue y ejecución de Matlab.

7. Una importante limitación es no disponer de capacidad de satélite para la validación, que será substituida por una simulación basada en GR-LEO para retrasar y modificar la frecuencia de la señal entre el transmisor y el receptor. Esto nos permitirá realizar pruebas deterministas basadas en el algoritmo matemático para determinar el grado de precisión en los resultados.

1.4 Planificación del Trabajo

Para la ejecución de este proyecto, se propone la planificación presentada en la Ilustración 1, donde se muestran los diferentes hitos asociados a las fases que se seguirán hasta la entrega final del TFG.

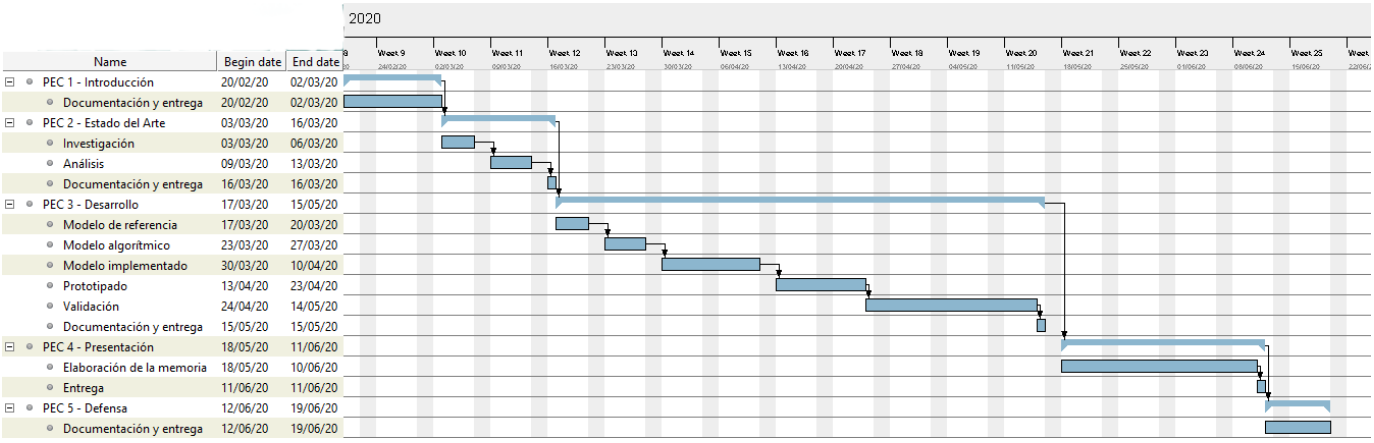


Ilustración 1. Planificación del TFG.

Los hitos están representados por la sucesivas PEC pertenecientes a este semestre, mientras que las tareas serán las asociadas al objetivo de cada PEC y específicas para este TFG en concreto.

El detalle de esta planificación se presenta en la Tabla 1, donde se ha incluido información adicional en las fases de diseño e implementación.

Nombre	Fecha de Inicio	Fecha de Fin	Duración	
PEC 1 - Introducción	20/02/2020	02/03/2020	8	
Documentación y entrega	20/02/2020	02/03/2020	8	
PEC 2 - Estado del Arte	03/03/2020	16/03/2020	10	
Investigación	03/03/2020	06/03/2020	4	
Análisis	09/03/2020	13/03/2020	5	
Documentación y entrega	16/03/2020	16/03/2020	1	
PEC 3 - Desarrollo	17/03/2020	15/05/2020	44	
Modelo de referencia	17/03/2020	20/03/2020	4	Elaboración de un algoritmo en coma flotante en Matlab que sirva de referencia (*Golden rule*) para la verificación y validación de los requisitos.
Modelo algorítmico	23/03/2020	27/03/2020	5	Transformación del algoritmo anterior al dominio del muestreo en Simulink.
Modelo implementado	30/03/2020	10/04/2020	10	Implementación del modelo algorítmico ajustado a los requisitos del sistema de prototipado. Modelo con coma fija.
Prototipado	13/04/2020	23/04/2020	9	Generación de código y despliegue sobre la plataforma de desarrollo.
Validación	24/04/2020	14/05/2020	15	
Documentación y entrega	15/05/2020	15/05/2020	1	
PEC 4 - Presentación	18/05/2020	11/06/2020	19	

Elaboración de la memoria	18/05/2020	10/06/2020	18
Entrega	11/06/2020	11/06/2020	1
PEC 5 - Defensa	12/06/2020	19/06/2020	6
Documentación y entrega	12/06/2020	19/06/2020	6

Tabla 1. Detalle de fases e hitos

1.5 Breve sumario de productos obtenidos

En el desarrollo de este TFG se han conseguido obtener:

- La descripción matemática de tres algoritmos que pueden ser usados para el posicionamiento de satélites en órbita por estaciones terrenas.
- Modelos de Simulink para dos de los algoritmos.
- Modelos de Simulink que soportan la transmisión y recepción de datos a través de SDR para dos de los algoritmos.
- Código VHDL para ser implementado en FPGA y código C para ser ejecutado por procesadores SoC para dos de los algoritmos.

La integración de todos los modelos y binarios generados, junto con la validación del sistema consolidado, han quedado fuera del alcance por requerir más tiempo del disponible para la ejecución.

1.6 Breve descripción de los otros capítulos de la memoria

Este documento se compone de los siguientes capítulos:

- Capítulo 1. Esta introducción, que además incluye una visión general.

- Capítulo 2. Que incluye una descripción del estado actual de las tecnologías y procesos relacionados con este TFG. En este capítulo se presenta el contexto donde poder encuadrar la motivación y objetivos de este trabajo.

- Capítulo 3. Este apartado es el más extenso y el más ambicioso. Aquí se describen los algoritmos y se presentan las diferentes implementaciones en Matlab y Simulink, junto con los resultados en las validaciones unitarias de cada elemento.

- Capítulo 4. Que presenta las conclusiones, los puntos que no han sido cubiertos y los posibles caminos a seguir a partir de la conclusión del TFG.

2. El Estado del Arte

2.1 Sistemas de Comunicación Espaciales

El 4 de octubre de 1957, con el lanzamiento del *Sputnik 1*, se inauguraba la era espacial, y con ella la de vehículos en la órbita terrestre. Este primer satélite ya incluía un sistema de telemetría muy rudimentario, que consistía en la emisión en bucle del famoso "*bip-bip-bip"* que marcaría el inicio de las comunicaciones desde el espacio hacia la tierra. Muy pronto, los satélites que les siguieron incorporaron la otra dirección de la comunicación, esto es, el telecontrol. De esta manera se forma lo que se llamaría TT&C (*Telemetry, Tracking & Control*), que permite la supervisión remota del vehículo en órbita.

Con el lanzamiento del *Early Bird* en 1965 y los INTELSAT a partir de 1966, comenzaría una tendencia que, lejos de romperse, se va acentuando hasta llegar en la actualidad, donde el espacio cercano está plagado de toda clase de objetos en órbita de todo tipo que realizan toda clase de misiones. Uno de los pocos puntos en común de todos ellos es la existencia del subsistema de TT&C, que ha sufrido una constante evolución a medida que los vehículos puesto en órbita eran más complejos y sofisticados.

Independientemente del tipo y de la misión de un vehículo espacial, estos se pueden descomponer genéricamente en:

- *Carga útil (payload)*, que se diseña para una misión específica (como por ejemplo comunicaciones por satélite u observación terrena).
- *Bus (space bus)*, que es el nombre que recibe el vehículo donde se aloja la carga útil. Suele ser de diseño genérico para ser capaz de adaptarse a cualquier tipo de misión.

Cuando la misión es la de proveer servicios de comunicaciones mediante el uso de transpondedores desplegados como carga útil, el vehículo espacial recibe el nombre satélite de comunicaciones.

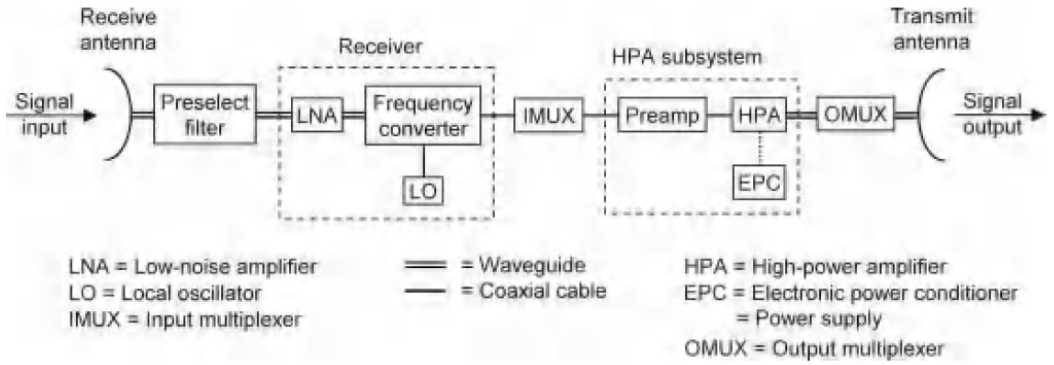


Ilustración 2. Diagrama simplificado de un transpondedor. © Wiley

El subsistema de TT&C pertenece al Bus, mientras que un subsistema de transpondedores de un satélite de comunicaciones pertenece a la Carga Útil. En ambos subsistemas es necesario realizar una estimación del estado del sistema a través de señales enviadas desde la estación terrena que también servirán para determinar las características de la órbita.

De esta manera, un sistema de comunicaciones por satélite queda determinado por segmentos:

- Segmento terrestre: que es aquél que gestiona las comunicaciones a través de los transpondedores, es decir, el envío de comunicaciones hacia el espacio (*uplink*) y la recepción desde el espacio hacia la Tierra (*downlink*).

- Segmento espacial: que es el sistema en órbita que incluye los elementos necesarios para la recepción de la comunicación desde la Tierra, el procesado, si lo hubiera, y el envío de vuelta a los destinatarios.

- Segmento de control: que es el responsable del TT&C del bus. Las comunicaciones en ambos sentidos se hacen mediante canales específicos diferentes a los que se usan en el segmento terreno.

Las definiciones surgidas de esta diferenciación han sido adoptadas por los textos básicos del sector [3]. En este TFG se harán referencia habitualmente.

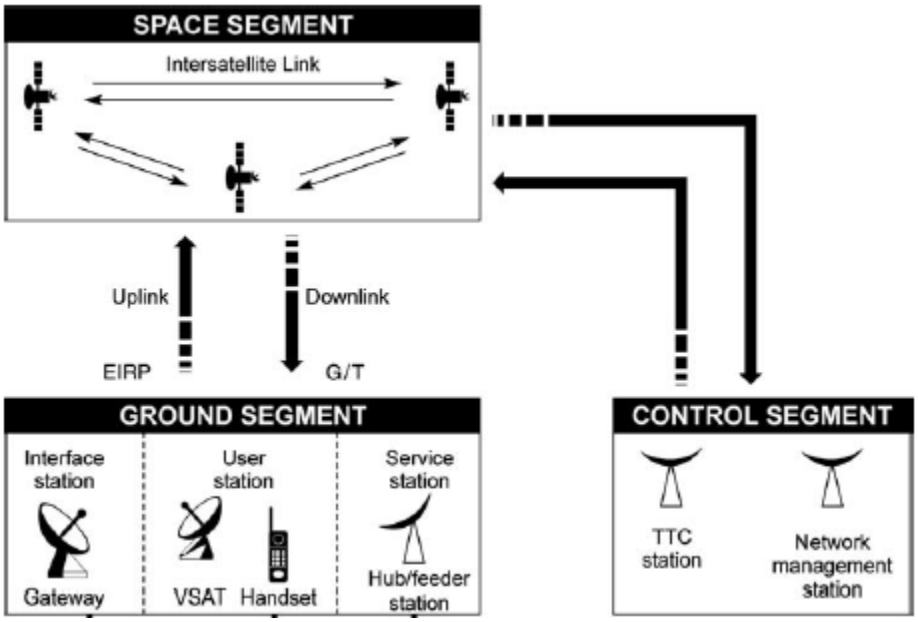


Ilustración 3. Segmentos de un sistema espacial. © Wiley

Redes de comunicaciones por satélite en órbita geoestacionaria

20 años antes de los lanzamientos de los INTELSAT, el escritor y científico Arthur C. Clarke publicaba un artículo [4] en la revista *Wireless World Magazine* donde

sugería el uso de repetidores en el espacio a una órbita a 35.787 km de distancia sobre la superficie de la Tierra, donde el tiempo sidéreo, o sea, el periodo orbital, es de 23 horas y 56 minutos. Esto haría que, en la práctica, un objeto puesto en esa órbita estuviera aparentemente fijo en el cielo, ya que se movería a la misma velocidad a la que se desplaza un punto arbitrario sobre la superficie de la Tierra debido a la rotación que sufre esta sobre su eje.

Hoy en día, la órbita Clarke, más conocida por el nombre común de *órbita geoestacionaria* o GEO, es la órbita más poblada en el espacio cercano a la Tierra. En el año 2011 contaba ya con alrededor de 400 satélites activos operando en esta órbita.

Desde un punto de vista puramente operacional, un enlace de comunicación por satélite no es muy diferente a un enlace de comunicación a través de un repetidor situado en un punto elevado sobre la superficie de la tierra. Entre las pocas distinciones que podemos hacer, está la notable limitación en la frecuencia mínima. Los efectos de la atmósfera en las transmisiones en radiofrecuencia hacen que la banda mínima en operación sea la banda S (2 GHz – 4 GHz), por la excesiva atenuación de la señal.

Las comunicaciones por satélite geoestacionarias gozan un tiempo de enlace permanente. Esto quiere decir que son idóneas para redes de comunicaciones donde el cauce de información sea continuo. Este es el caso de las transmisiones audiovisuales o, en años más recientes, las redes de datos VSAT (*Very Small Aperture Terminal*), que surgen de la necesidad de la interconexión de redes de datos en puntos donde una infraestructura terrestre no le es posible llegar. Un caso muy representativo es el de las terminales montadas en barcos y aeronaves.

Un notable inconveniente, que es especialmente notable en las aplicaciones relacionadas con la transmisión de datos, es el retardo que se produce en la señal por el viaje que realiza desde el segmento terreno, al segmento espacial y vuelta. Esto es debido a la gran distancia a la que se encuentra el satélite y típicamente, el retardo de un *ping* realizado desde una de las terminales VSAT a un dispositivo dentro de la infraestructura en el segmento terreno, es de alrededor 450-500 ms. Este problema motivó el desarrollo de sistemas que pudieran operar en órbitas más bajas, donde la señal tiene que recorrer una distancia considerablemente menor.

Constelaciones de satélite en órbita baja y media

Según la NASA [5], las órbitas que se encuentran por debajo de la órbita geoestacionaria son:

- MEO (*Medium Earth Orbit*): que se expande desde los 2.000 Km hasta una altitud inmediatamente inferior a los 35.786 Km, que es donde se encuentra la órbita GEO.

- LEO (*Low Earth Orbit*): que es cualquier órbita posible por debajo de los 2.000 Km.

Hasta la llegada de las constelaciones GPS (*Global Positioning System*) para navegación y posicionamiento geográfico, e IRIDIUM para telefonía básica global, las órbitas LEO (*Low Earth Orbit*) y MEO (*Medium Earth Orbit*) estaban utilizadas casi exclusivamente por satélites de observación terrena y de misiones de científicas.

Al orbitar más cerca del segmento terreno, es posible encontrar satélites operando en banda L (el servicio IRIDIUM, por ejemplo) e incluso la ITU tiene asignada ciertas frecuencias para la comunicación espacio-tierra en VHF y UHF.

Con la expansión de las redes móviles de datos, y su evolución natural en capacidad de ancho de banda y velocidad, se hace necesario redes de comunicaciones por satélite que no tengan las limitaciones que impone la gran distancia a la que se encuentra el satélite. Diferentes iniciativas se ponen en marcha siendo de las primeras en operar la constelación de satélites de *O3b Networks* en 2013, que ofrecía servicio digital desde una órbita MEO de 8.000 Km con anchos de banda de 500-800 Mb/s y una latencia de 140 ms. Estos valores hacían el sistema válido para su uso en un amplio abanico de aplicaciones que iban desde el *backhauling* de redes de telefonía móvil hasta los juegos online desde cruceros de placer en mitad del océano Atlántico.

En la actualidad los lanzamientos de este tipo de satélites son muy frecuentes. Diversas constelaciones están siendo completadas y ofrecerán banda ancha con baja latencia gracias a una combinación de nuevas tecnologías como son:

- HTS (*High Throughput Satellite*) [6].
- *Beam Steering* o apuntamiento mecánico del haz del satélite.
- *Beamforming* [7] [8].

Las ventajas que ofrece el uso de este tipo de órbitas vienen acompañadas de importantes desventajas que incrementan el esfuerzo necesario de operación y mantenimiento y que están directamente relacionadas con las características físicas del tipo de órbita. Que el satélite se mueva dentro del sistema inercial que forman el segmento terreno/de control y el segmento espacial provoca una variación en frecuencia debido al efecto Doppler, y además existe una necesidad crítica de determinar la órbita para sincronizar los elementos responsables de apuntar las antenas durante el tiempo del enlace. Para poder cubrir estas necesidades, se hace necesario determinar la posición en el espacio del satélite, y para ello son necesarios los siguientes parámetros:

- Elevación sobre el plano del horizonte.
- Azimut sobre el plano perpendicular al plano del horizonte.
- Distancia medida sobre un observador situado en la superficie de la tierra.
- Velocidad del satélite.

Una vez obtenidos estos parámetros, será posible determinar únicamente la órbita de un satélite alrededor de la tierra.

2.2 IOT (In-Orbit-Testing)

IOT (*In-Orbit-Testing*) es una de las operaciones que se realizan tanto por el segmento terreno como por el segmento de control, sobre el segmento espacial. Esto es, son una serie de pruebas que se realizan sobre el satélite para determinar su estado. Estos procedimientos no han cambiado mucho desde las primeras campañas lanzadas en los satélites INTELSAT [9], que sentaron las bases para este tipo de validación en otros operadores de satélites. Varias decenas de años después, estos procedimientos han cambiado poco [10], si bien han incorporado algunos tipos de nuevas pruebas para cubrir ciertas necesidades debido a la incorporación de transpondedores regenerativos y comunicaciones digitales [11].

Este tipo de campañas de pruebas se realizan de manera recurrente y siguiendo el plan de mantenimiento del satélite, o bien ad-hoc, para investigar algún tipo de evento inesperado o durante las maniobras LEOP (*Launch and Early Orbit Phase*) del satélite, donde se encuentra en tránsito a su órbita final.

Parámetros estudiados en campañas de pruebas

En las campañas IOT se estudian una serie de parámetros básicos para comprobar el estado de los subsistemas de comunicaciones y de TT&C del satélite. En [9] se detallan estos parámetros que se listan en la Tabla 2 junto con la condición básica del satélite para realizar la medición.

Nombre del parámetro	Condición del satélite
EIRP	Con tráfico/sin tráfico
Densidad de flujo	Con tráfico/sin tráfico
Ganancia en pequeña señal	Sin tráfico
Ganancia en saturación	Sin tráfico
Respuesta en frecuencia	Sin tráfico
G/T	Sin tráfico
Noise	Sin tráfico
Polarización cruzada	Sin tráfico
Conversión de frecuencia	Con tráfico/sin tráfico
Frecuencia de la baliza	Con tráfico/sin tráfico
Retardo de grupo	Sin tráfico
Sensibilidad del receptor	Sin tráfico
Frecuencia de portadora	Con señal FM
EIRP de portadora	Con señal FM

Tabla 2. Principales parámetros en IOT

Uno de los principales problemas que se presenta a los operadores es el hecho de que es necesario suspender el servicio a los usuarios para realizar la medición de la mayoría de las pruebas para controlar el tráfico. Esto supone un costo para el operador durante las campañas de pruebas.

El caso particular de la distancia, velocidad y posición

El cálculo y monitorización de estos tres parámetros no es novedoso. Incluso en órbitas geoestacionarias, los satélites son cuidadosamente vigilados ya que una desviación de su posición podría poner en peligro a otros satélites cercanos.

La posición espacial de un objeto que se encuentra en órbita queda determinada por los siguientes parámetros:

- El azimut y la elevación sobre el plano ecuatorial y sobre su perpendicular.
- La distancia sobre la superficie de la tierra.
- La velocidad del objeto.

Estos tres parámetros son calculados tradicionalmente desde el segmento de control, y sirven para la preparación de maniobras de orientación y desplazamiento. Según [10], los parámetros eran obtenidos de la siguiente manera.

- La posición venía de la orientación de la antena. Esta antena, al ser especializada y de gran diámetro, debido a su apertura, puede apuntar con precisión y detectar el punto de máxima potencia recibida.

- La velocidad se calcula mediante el efecto Doppler. Se envía una señal CW (*Continuous Wave*) de una determinada frecuencia. A la recepción es fácil calcular la desviación que sufre debido a la velocidad.

- La distancia a la que se encuentra el satélite se determina enviando 4 señales moduladas en 4 frecuencias:

    - 35.4 Hz
    - 283.4 Hz
    - 3.968 KHz
    - 27.778 KHz.

    La distancia entre el satélite y la estación terrena se calcula midiendo la diferencia de fase entre estas frecuencias y convirtiendo esta diferencia de fase en distancia. Estas 4 frecuencias son usadas habitualmente y se eligieron para eliminar la ambigüedad de fase.

En la actualidad, el cálculo de distancia ha sido sustituido por el envío de secuencias de números pseudoaleatorios. De esta manera es más fácil la correlación entre lo transmitido y recibido [12] y [13].

Dispositivos y equipamiento para IOT

Aunque existen ciertos equipos que están orientados a las campañas de IOT, no se ha desarrollado un mercado dedicado especialmente a este tipo de aplicaciones, y los pocos sistemas integrados han sido desarrollados bajo

financiación de organizaciones como la ESA. Este es el caso del modem IOT fabricado por Indra Espacio o GHOST [14], en desarrollo por la compañía SES.

La mayoría de los operadores han desarrollado sus propios procedimientos que incluyen un despliegue de equipos. Estos varían de uno a otro, pero entre los dispositivos que usan se encuentran:

En el lado donde se genera la señal de prueba.

- Un equipo generador de señales en banda L, que a veces es sustituido por un modem dedicado.
- Un conversor en frecuencia que traslada la señal a la frecuencia de operación (de subida) del satélite ya sea S, Ka, Ku, C o X.
- Un HPA (*High Power Amplifier*) conectado a una antena apuntada al satélite.

En el lado donde se recibe la señal:

- Generalmente un duplexor conectado a la misma antena a la que está conectado el transmisor, aunque también puede usarse una antena independiente.
- Un LNA (*Low Noise Amplifier*).
- Un conversor de frecuencia que traslada la señal de la frecuencia de bajada del satélite a banda L.
- Un instrumento de medida, que generalmente es un analizador de espectro o un VNA (*Vector Network Analyzer*).

2.3 Sistemas empotrados y de lógica programable

Los sistemas empotrados (*embedded systems*) han sufrido una tremenda evolución en los últimos años, especialmente en el campo de la integración, culminando con la generalización del SoC (*System-on-chip*) a un precio razonable. Esta fuerte integración, impulsada por la necesidad del mercado de suplir de terminales de telefonía móvil que sean *inteligentes,* ligeros y muy potentes.

Por otro lado, otra tecnología asociada a los sistemas empotrados, la lógica programable FPGA evolucionó de manera paralela al resto del sector y fue incrementando la velocidad y capacidad de sus circuitos integrados, mientras que reducía el coste de su segmento más modesto, que veía como su rendimiento era equivalente al del segmento alto de un par de generaciones anteriores.

Una tercera vía de evolución ha sido el software de diseño y fabricación asociado al diseño con sistemas empotrados o EDA (*Electronic Design Automation*), que han permitido el diseño, simulación e implementación de sistemas muy avanzados de manera eficiente, rápida y con un menor coste en prototipos. Estas herramientas permiten una abstracción considerable de los detalles de la implementación mientras que se pone el foco en la funcionalidad. Poco a poco se ha pasado de la programación casi en exclusiva en un lenguaje HDL a la

implementación gráfica usando bloques IP (*Intellectual Property*) o bloques con funcionalidad autocontenida creado por terceros de los que solamente conocemos las entradas, algunos valores de configuración y las salidas. Otra herramienta que ha ganado mucha popularidad son los lenguajes HLS (High Level Synthesis). Estos lenguajes permiten usar lenguaje de programación comunes como C o Python [15] para generar código HDL. Un notable ejemplo es *OpenCL* [16]*,* un esfuerzo de un consorcio de empresas para implementar una tecnología común que permite ejecutar un fragmento de código (*kernel*) en diferentes dispositivos, como CPUS, FPGA o incluso en GPU sin la necesidad de modificar el kernel.

Inevitablemente ambas tecnologías terminaron fusionándose, y los primeros circuitos integrados SoC, que incorporan lógica programable además de los periféricos típicos de este tipo de sistemas integrados, son lanzados por los principales fabricantes de FPGA.

Este tipo de chips integran dos tipos de procesadores:

- *Soft Processor*. O *procesador blando.* Este tipo de procesador está completamente implementado en el área de lógica programable y programado en un lenguaje HDL. Este tipo de procesadores permiten una gran flexibilidad en la configuración, que incluso permite cambiar entre diferentes arquitecturas, a expensas de velocidad de ejecución.

- *Hard processor*. O *procesador duro.* Este tipo de procesador está implementado usando transistores impresos en el chip. Esto limita las opciones de configuración del procesador, pero incrementa drásticamente el rendimiento obtenido.

Es importante también destacar la importancia que tiene la arquitectura de bus en este tipo de sistemas, ya que tiene que ser capaz de mover información entre los procesadores (ya sean *soft* o *hard*), los periféricos incorporados al circuito integrado, y la lógica programable. Existen varios estándares, entre los que destacan *Wishbone*, AMBA y AXI.

Xilinx Zynq-7000 SoC

Una de las empresas líderes en el mercado de fabricación de circuitos integrados con lógica programable es Xilinx, que junto con Intel (tras la adquisición de Altera), tal como se muestra en la Ilustración 4.

Top 5 Company's Sales of FPGA/PLD, in All Applications, Worldwide, (Millions of U.S. Dollars)

Rank 2017	Rank 2018	Company Name	2017	2018	2018 Change	Share of Market
1	1	Xilinx	2,476	2,904	17.3%	51.1%
2	2	Intel	1,858	2,033	9.4%	35.8%
3	3	Microchip (formerly Microsemi)	321	376	17.1%	6.6%
4	4	Lattice Semiconductor	261	285	9.2%	5.0%
		Others	71	85	19.7%	1.5%
		Total Market	**4,987**	**5,683**	**14.0%**	**100.0%**

Source: Gartner (April 2019)

Ilustración 4. Reparto del mercado de FPGA

Xilinx ofrece [17], en su cartera de productos SoC, dos tipos de soluciones, ambas similares en arquitectura, pero diferentes en potencia y periféricos, para cubrir la totalidad de las necesidades del mercado actual.

Estos productos son:

- Zynq-7000 [18]: que incorpora un procesador dual ARM Cortex-A9 y un conjunto de periféricos empotrados, además de una lógica programable basada en sus FPGA Artix. Este procesador es usado en dispositivos SDR como en el ADALM-PLUTO.

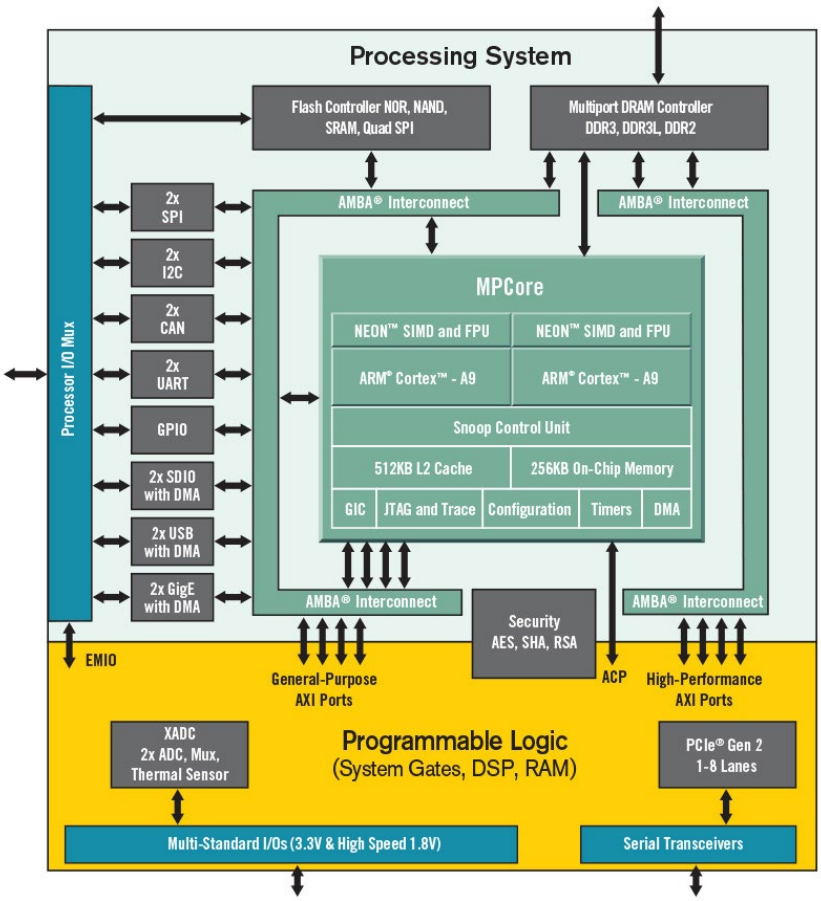


Ilustración 5. Arquitectura Zynq-7000. © Xilinx

- Zynq UltraScale+: que cubre las necesidades más exigentes, ya que añade otros dos procesadores (estos dedicados a tareas en tiempo real) con variantes que incluyen GPU y conversores de datos de alto rendimiento en el mismo chip.

A la hora de ejecutar programas en este tipo de arquitecturas, las herramientas de desarrollo ofrecen la posibilidad de tres entornos de ejecución:

- Nativo en ARM: usando un conjunto de bibliotecas que ofrecen la funcionalidad básica de interactuar con los periféricos en forma de BSP (*Board Support Packages*) formando un pseudo-sistema que Xilinx ha bautizado como *Standalone.*

- En un sistema Linux. Que puede ser generado usando las herramientas que ofrece Xilinx (*Petalinux*) o incluso mediante otras herramientas de código libre, que también están soportadas oficialmente, y que son mantenidas por terceras personas (*Yocto*).

- En un sistema *FreeRTOS,* que es un sistema operativo en tiempo real de código abierto.

Para el prototipado de nuevos sistemas de comunicaciones que no sean especialmente intensivos en proceso, el SoC Zynq-7000 ofrece un rendimiento aceptable, una cantidad de periféricos suficiente y un entorno de desarrollo muy completo, con la posibilidad de integrarse con otros productos como Matlab.

Un uso muy adecuado para la experimentación y diseño de nuevos sistemas con este SoC es la placa de desarrollo Zedboard [19]. Esta placa incorpora interfaces físicos a los periféricos del SoC junto con una cantidad de memoria DDR suficiente para ejecutar programas

2.4 SDR y Conversores de datos

SDR (*Software Defined Radio*) es un concepto que surge en la última década al evolucionar los conversores de datos hasta obtener frecuencias que entran dentro del rango que se usa en comunicaciones inalámbricas. De esta manera, se reemplazan procesos en la cadena de recepción que tradicionalmente estaban implementados como hardware y que ahora pueden ser realizados en el dominio digital mediante técnicas de DSP (*Digital Signal Processing*). Más concretamente, se reemplazan los bloques relacionados con el filtrado, modulación/demodulación y multiplexación/demultiplexación. En algunos sistemas incluso se reemplaza la conversión en frecuencia si entra dentro de los límites del conversor de datos.

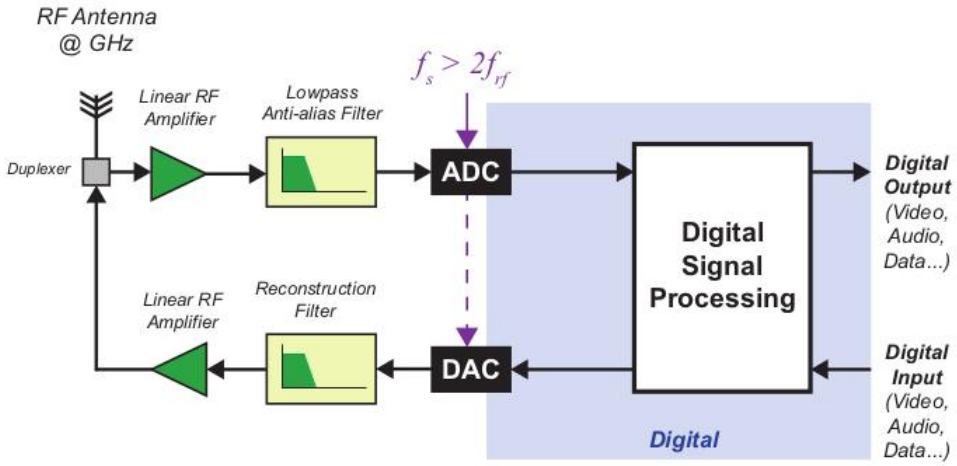


Ilustración 6. Sistema SDR típico. © Universidad de Strathclyde

En la Ilustración 6 se presenta un sistema SDR típico, donde una antena es compartida por la cadena de recepción y transmisión gracias a un duplexor. Las señales recibidas son amplificadas por un LNA (*Low Noise Amplifier*) y filtradas antes de ser transformadas por un conversor de datos ADC (analog-to-digital-converter). A partir de aquí, en el dominio digital, la señal puede ser tratada, según esté modulada o codificada, y redireccionada a la salida. De la misma manera, cualquier información de entrada será codificada y modulada antes de dejar el dominio digital pasando a través de otro conversor de datos, esta vez un DAC (*digital-to-analog-converter*), que transforma de vuelta la señal al dominio analógico para ser filtrada otra vez, amplificada y transmitida.

La tecnología SDR ha incrementado notablemente la flexibilidad en receptores y transmisores de radio, permitiéndoles una reconfiguración mediante actualizaciones de software. Esta característica hace que los equipos SDR sean muy útiles en equipos de comunicaciones remotos o redes muy extensas, como son los sistemas embarcados en satélites, repetidores o instalaciones submarinas. Pero también son extremadamente valorados en las fases de diseño y prototipado, ya que permiten experimentar, diseñar, implementar y validar sistemas de manera muy eficiente, reduciendo el tiempo entre iteraciones.

Analog Devices ADALM-PLUTO SDR

En la actualidad es posible encontrar bastantes tipos diferentes de transceptores SDR, según la aplicación, ya que es un sector que ha madurado considerablemente. Dentro del ámbito de la docencia y la investigación, está ganando bastante popularidad el dispositivo de Analog Devices ADALM-PLUTO [20]. Es completamente de código abierto e incorpora el transceptor en un solo chip AD9363 [21] y un SoC Xilinx Zynq-7000 como el que ya hemos descrito. Esto hace que pueda ejecutar un sistema empotrado Linux. Este pequeño dispositivo está provisto de múltiples formas de conectividad, incluyendo conexiones USB, a través de red e incluso soporta conexión inalámbrica si se conecta un *dongle wifi.*

Ilustración 7. ADALM-PLUTO. © Analog Devices

Analog Devices proporciona los *drivers* y complementos adecuados para que pueda ser usado en entornos como Matlab o GNU Radio entre otros. De esta manera, se hace muy sencillo el prototipado de sistemas de comunicaciones.

GNU Radio

GNU Radio [22] es una plataforma de diseño y entorno de ejecución para dispositivos SDR. Al ser de código abierto, sirve de referencia para la implementación y recibe contribuciones de innumerables desarrolladores, que proporcionan toda clase de herramientas para el tratamiento digital de la señal, incluyendo demoduladores, codificadores, control de errores, mezcladores y generación de señales, entre otros muchos.

GNU Radio proporciona un entorno gráfico muy intuitivo (*GNU Radio Companion*) que ayuda al prototipado de sistemas. El usuario podrá diseñar un flujo por donde circularán las muestras procedentes de una fuente (principalmente un receptor SDR), añadir bloques de procesado digital sobre esas muestras, y volcarlas sobre un destino (que probablemente será un transmisor SDR, aunque también podría ser un archivo o una conexión de red). Este flujo podrá ser exportado a código en lenguaje Python o, en su versión más reciente, en lenguaje C.

2.5 Prototipado de algoritmos con Matlab

Matlab es un paquete de software distribuido por la compañía Mathworks y que es, en la actualidad, el estándar *de facto* en software científico por su potencia en el modelado de algoritmos matemáticos. La arquitectura de Matlab se basa en su modularidad, es decir, se establece un motor matemático, con un leguaje específico asociado, y se complementa con módulos (*toolboxes* y *block sets*) especializados por áreas. De esta manera, existen módulos para gran cantidad de disciplinas.

Matlab está estructurado de forma que se puede trabajar siguiendo diferentes filosofías o metodologías para conseguir el objetivo. Además, estas metodologías se complementan entre sí para permitir el uso del software durante todo el proceso de diseño, implementación y validación de sistemas de comunicaciones.

Para la fase de experimentación y diseño inicial del modelo matemático, se dispone del leguaje matemático integrado junto con paquetes como *DSP Systems toolbox* y *Signal Processing Toolbox,* permiten trabajar con señales contenidas en estructuras de almacenamiento lógicas, sin tener en cuenta el tiempo como variable, de manera que es posible realizar aproximaciones y asumir conceptos para valorar después el impacto en los resultados. De este proceso se obtiene lo que se denomina *Golden Rule* o lo que es lo mismo, el patrón por el que vamos a validar los resultados de las subsiguientes fases.

Al avanzar en el desarrollo, Matlab dispone de *Simulink,* un lenguaje visual pero equivalente al usado en la etapa anterior, que permite estructurar la solución incorporando el tiempo como parte de ella. Desde Simulink las señales se estructuran en muestras con una determinada frecuencia, que equivalen a la salida de un conversor ADC, y que permite orientar la solución hacia un punto de vista más cercano a la implementación.

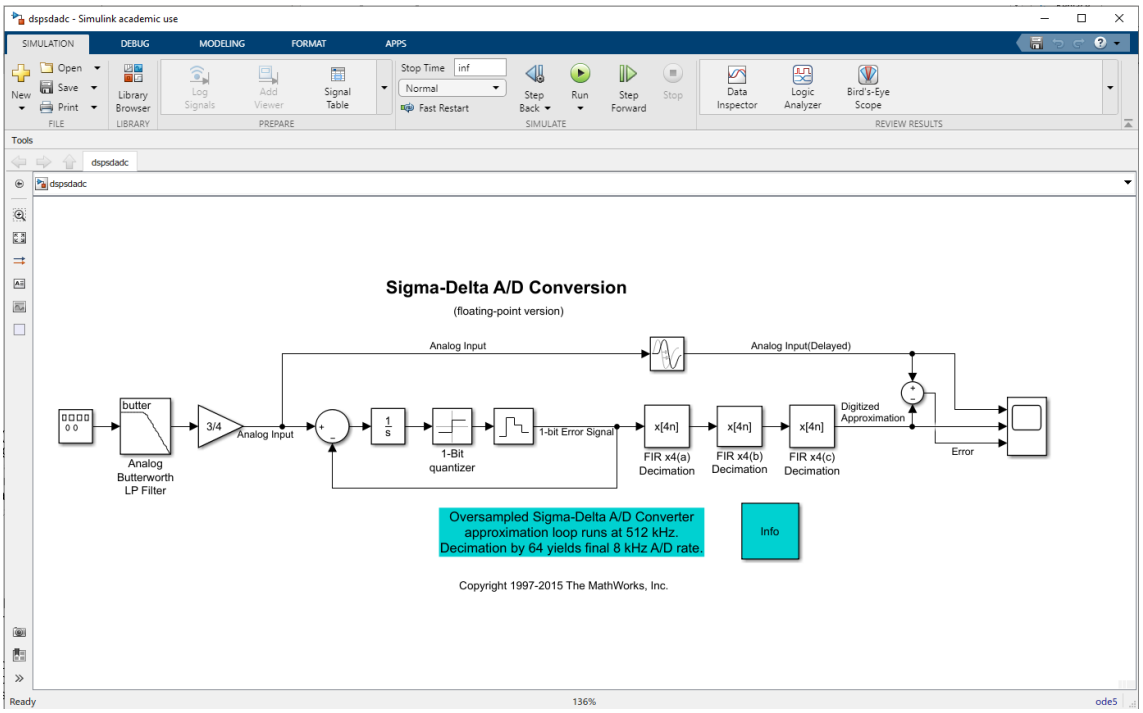


Ilustración 8. Simulink

Tras validar y verificar los resultados, Matlab es capaz de generar código C/C++ para ciertos procesadores, entre los que se encuentran las variantes de ARM en los sistemas Zynq-7000. Alternativamente también es capaz de generar código

HDL para dispositivos de lógica programable. Dispone de generación genérica, es decir, sin dependencias ligadas a ningún tipo de chip en concreto, o generación específica, usando incluso las herramientas del fabricante para generar también el *bitstream* necesario para programar el dispositivo. De especial interés para nuestro caso es que soporte Vivado, el software para dispositivos Xilinx Zynq-7000.

3. Implementación y validación

3.1 Descripción de la plataforma de desarrollo

El objetivo es poder calcular con la máxima precisión que podamos obtener de la frecuencia de muestreo máxima de los conversores de datos, la posición de un satélite en órbita respecto a la posición de la estación terrena que está llevando a cabo el estudio. Para ello, es necesario determinar los siguientes parámetros físicos:

- Localización en un determinado momento dentro del sistema referencial que tiene como centro el punto donde realiza el movimiento orbital el satélite
- La velocidad en ese instante
- La distancia que separa la estación del vehículo espacial

Estos parámetros serán determinados usando los algoritmos que se presentan en las diversas secciones de este capítulo.

Consideraciones técnicas

La implementación de estos algoritmos se ha hecho principalmente con Mathworks Simulink 2020a/2019a y con Xilinx Vivado HLS 2019.3.

Dependiendo del tipo de herramienta, una versión de Matlab o de Vivado era necesaria.

La versión de software	Trabaja con
Matlab 2020a • HDL Coder • Embedded Coder • HDL Verifier	Xilinx Vivado HLS 2019.1
Xilinx Vivado HLS 2019.1 • System Generator • DSP Design	Matlab 2019a

Tabla 3 Relación de versiones de software

Nuestro flujo de trabajo ha sido casi exclusivamente el diseño y la implementación en Matlab usando las herramientas de Mathworks, para posteriormente ejecutar el análisis de RTL, la síntesis y la implementación específica para nuestro dispositivo Xilinx desde Vivado. Este último paso lo

puede realizar Matlab una vez configurado la herramienta de FPGA con el comando ***hdlsetuptoolpath*.**

Limitaciones de software

Tanto Matlab como Simulink tienen las limitaciones de la potencia de computación del hardware donde se está ejecutando. Simular frecuencias de muestreo en banda base del orden de megahercios supone realizar cálculos de millones de muestras para cada segundo de simulación. Algo similar ocurre con otros valores ligados a la visualización de señales de frecuencias muy altas. Por esta razón, para muchas de las simulaciones y como no se dispone de un sistema de computación, vamos a escalar los valores, especialmente los de muestreo.

Limitaciones de hardware

El dispositivo SDR Pluto impone las siguientes limitaciones:

- Ancho de banda en banda base: 200 kHz - 20 MHz
- 12-bit DAC (Tx) y ADC (Rx) integrados
- Tasa de bits de muestreo variable: 61.44 MSPS - 65.1 kSPS

En la práctica, el ancho de banda y la tasa de bits de salida estarán limitados por la capacidad de computación más que por las capacidades del SDR en sí. Es importante tomar nota de la resolución de los conversores de datos, ya que, cuando trabajemos con HDL, tendremos que tomar decisiones sobre la conversión de coma flotante a coma fija. No tiene sentido usar valores de 32 bits, ya que se produce una pérdida importante de precisión. Aunque es posible usar directamente 12 bits, hemos tomado la decisión de usar 16 bits. El error por truncamiento no es significativo, y nos permite una mayor portabilidad. Esto es especialmente notable en los casos donde la lógica programable interactúa con código C ejecutándose en el procesador. Además, esta decisión facilita la portabilidad a otro tipo de plataformas.

La placa de desarrollo Zedboard no impone ninguna limitación importante. Esto es debido a que los algoritmos se irán probando uno a uno. Un escenario considerablemente distinto sería aquel donde todos los algoritmos estuvieran integrados en la lógica programable del FPGA, el código C ejecutándose en los procesadores ARM, y hubiéramos desarrollado un interfaz para el SDR funcionando a tasas de bits altas. En este tipo de situación, se vuelve a imponer una limitación en potencia. Esta vez sin Simulink o Matlab, que hace que los valores sean considerablemente más altos, pero sin llegar a aprovechar el máximo rendimiento del SDR. Consideramos que para ello necesitaríamos un FPGA más potente, como por ejemplo un dispositivo de la familia de Xilinx Ultrascale+.

El sistema operativo de la placa de desarrollo

Matlab provee de un sistema Linux que es adaptado a cada placa a través de BSP (*Board Support Packages*) descargables desde el administrador de *add on* del propio software. Esto paquete incluye el sistema operativo adaptado al procesador que vayamos a usar, los compiladores y demás software de apoyo para compilar y enlazar (t*oolchain*), drivers y *device-tree* para periféricos que pudieran existir en el sistema de desarrollo. De esta manera, podemos abstraer los diseños del hardware donde desplegaremos.

Matlab también incluye utilidades y soporte para el desarrollo de BSP para sistemas creados por el usuario, aunque en nuestro caso no es necesario ya que encontramos que Matlab incluye como descargable adicional los BSP para HDL Coder y Embedded Coder para placas Xilinx, entre las que se incluye Zedboard.

El sistema Linux es accesible mediante puerto serie o conexión SSH y permite monitorizar el estado de la placa, los binarios desplegados o incluso ejecutar los binarios generados.

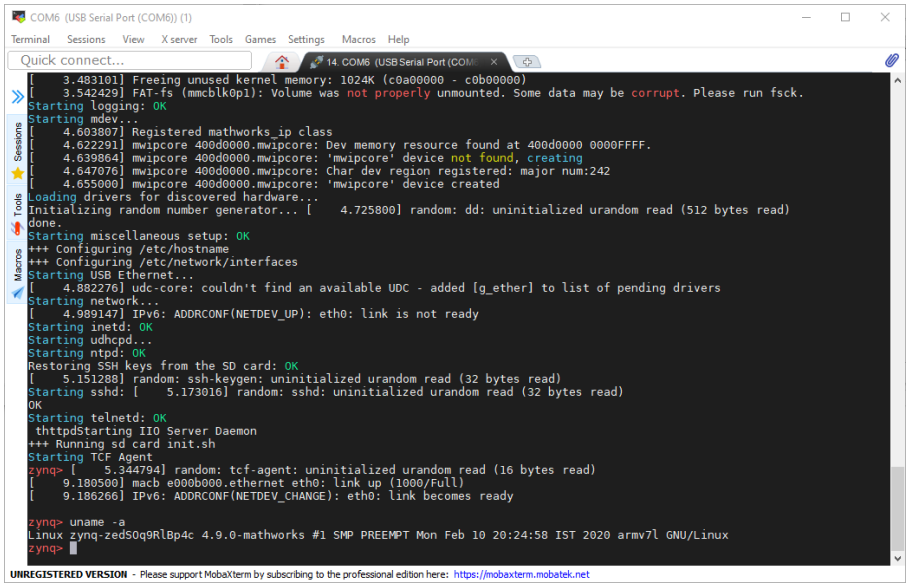


Ilustración 9 Acceso al sistema Linux de Mathworks desplegado en la placa

3.2 Determinación de velocidad relativa mediante el efecto Doppler

El efecto Doppler es el cambio aparente de frecuencia que se produce entre un transmisor y un receptor cuando la velocidad relativa entre ambos es distinta de cero. En el caso de las comunicaciones por satélite, tanto las estaciones terrenas como los satélites están en constante movimiento. Los primeros se desplazan a la misma velocidad que sufre la superficie de la tierra en su giro diario sobre su eje. Los segundos, incluso aquellos que están en órbita geoestacionaria, mantienen un movimiento de *station keeping* además de las maniobras propias del mantenimiento de la órbita.

En general, el efecto Doppler se rige por la siguiente expresión [23]:

$$\frac{f_o}{f_s} = \frac{c}{c + v_s}$$

Ecuación 1. Expresión general del efecto Doppler

Donde f_o es la frecuencia observada, f_s es la frecuencia en la fuente, ***c*** es la velocidad de la luz (por ser una onda electromagnética), y v_s es la velocidad de la fuente.

Hay que tener en cuenta que este es el caso general, donde el emisor y el receptor se alejan o se acercan dentro de un mismo eje. En realidad, y para un satélite orbitando, habría que tener en cuenta el ángulo que forman el vector de velocidad del satélite con el eje que forma con la estación terrena [3]:

$$\Delta f_d = V_r \frac{f}{c} = V \cdot cos\Phi \left(\frac{f}{c}\right)$$

Ecuación 2. Efecto Doppler para un satélite orbitando.

Siendo el ángulo Φ el ángulo entre la dirección del punto de referencia (estación terrena) y el vector velocidad del satélite en el momento de la medición. Aunque vamos a considerar a ambos en el mismo eje (o sea, $cos\ \Phi = 1$), este término será importante más tarde cuando tengamos el valor del ángulo.

Para el caso particular de una estación terrena transmitiendo a un satélite, existen una serie de matices que hay que tener en cuenta según sea el tipo de transponder a donde se dirija la transmisión:

- Transpondedor no regenerativo.
- Transpondedor regenerativo.

El primer caso corresponde a un satélite que simplemente realiza una translación en frecuencia de la señal. Esto quiere decir que recibirá una señal en su frecuencia de subida (*uplink frequency*) y sin ningún tipo de tratamiento digital de señal, trasladará la señal a la frecuencia de bajada (*downlink frequency*) después de ser transmitida. Este tipo de transpondedores son conocidos como *bent pipe* (tubería doblada).

En este caso, cualquier modificación que sufra la señal debido al medio será arrastrada de una frecuencia a otra y retransmitida. Esto significa que el efecto Doppler que es observado por el satélite, de una señal no modulada, se acumulará al observado por la estación terrena una vez que la señal ha sido retransmitida de vuelta. Debido a esto, y partiendo de la Ecuación 1, podemos decir que la velocidad del satélite vendrá dada por:

$$v_s = \frac{c \cdot \left[f_o + \frac{\Delta f_s}{2} \right]}{f_o} - \mathrm{c}$$

Ecuación 3. Doppler en U/L y D/L

Es decir, la variación de frecuencia es compensada para tener en cuenta un solo tramo, asumiendo que no existe variación de velocidad relativa entre los extremos durante el tiempo de transmisión y recepción.

En los transpondedores regenerativos se producen procesamientos de señales digitales para demodular la información, aplicar las correcciones sobre los bits que pudieran haber sido afectados en la transmisión, para volver a ser codificados y modulados nuevamente antes de ser transmitidos de vuelta. Este tipo de transpondedores permiten un tipo de comunicación más resistente para transmisiones digitales, e incluyen además otro tipo de sistemas complementarios que mejoran la calidad del enlace. Entre ellos está la compensación de Doppler. Por eso, aunque estemos lanzando señales sin modular, en este tipo de transpondedores el cálculo debe hacerse de diferente manera. La velocidad del satélite se calculará como:

$$v_s = \frac{c \cdot f_s}{f_o} - c$$

Ecuación 4. Doppler en D/L

Que no es más que la versión de la expresión anterior (Ecuación 3), pero teniendo en cuenta que la variación en frecuencia se producirá solamente durante el tramo de *downlink*.

Implementación en Matlab

Se ha implementado una función ***estimate_doppler.m*** que permite calcular la velocidad de una onda electromagnética. El código de esta función se encuentra en 7.1 de este TFG.

La cabecera de esta función es la siguiente:

Función	Parámetro 1	Parámetro 2
estimate_doppler	Frecuencia observada	Frecuencia emitida

Para validar la función, se usará la información recogida por la doctora Carol F. Milazzo, sobre la magnitud de Doppler en una observación experimental que realizó en septiembre de 2012 sobre el satélite VO-52 [24].

En estas observaciones, determinó los parámetros que se muestran en Tabla 4 Parámetros observados del VO-52.

	LOS	
f	145858100	Hz
f0	145861450	Hz
c	299,792,458	m/s
Relative velocity	6885.49	m/s
True velocity	7554.86	m/s

Tabla 4 Parámetros observados del VO-52

La función estimate_doppler arrojó como resultado un valor de $6.8834 \cdot 10^3$ m/s, que consideramos suficientemente aproximado a la velocidad relativa medida en las observaciones.

Modelo de Simulink

En el modelo de Simulink tenemos que introducir el concepto de muestras a lo largo del tiempo.

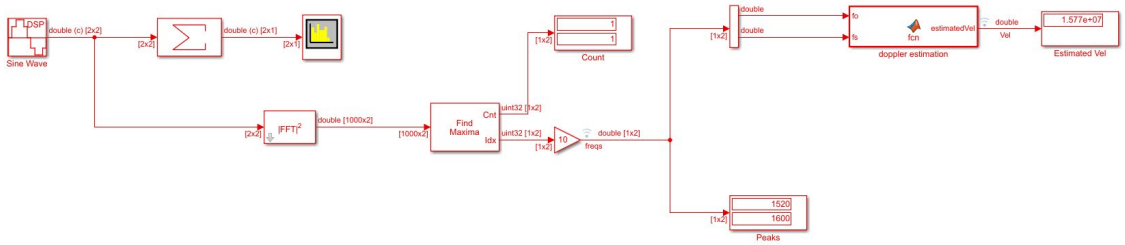


Ilustración 10 Estimación de velocidad por Doppler

Este modelo incluye algunos bloques que permiten obtener información extra durante la ejecución. El color de los bloques y sus caminos nos indican que todo el modelo trabaja en una única frecuencia de muestreo.

El flujo se inicia con la generación de una señal con dos componentes seno tal que:

$$y_1(x,t) = A \cdot sen(k_1 x - \omega_1 t)$$
$$y_2(x,t) = A \cdot sen(k_2 x - \omega_2 t)$$

Que no es más que la definición matemática de la suma de dos ondas sinusoidales de distinta frecuencia.

Particularizamos para $\omega_1 = 9550\, rad/s$ y $\omega_2 = 10053\, rad/s$ a través de los parámetros del generador.

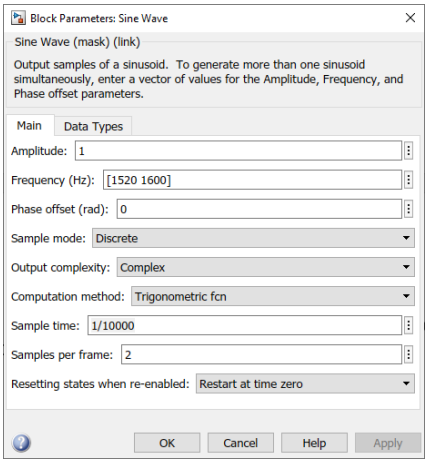


Ilustración 11 Parámetros del generador de señales

Un parámetro particularmente interesante es *samples per frame*, que representa el número de muestras procesadas simultáneamente por unidad de tiempo de Simulink. Una de las limitaciones que nos impondrá los modelos para ser traducidos a HDL, será la de una única muestra por frame. Esto tiene sentido, en el ámbito digital trabajaremos muestra a muestra dirigidos por uno o más relojes que establecerán la tasa de muestreo del sistema. El uso de frames es una buena práctica en modelos comunes de Simulink porque mejora el rendimiento y permite cierta abstracción de los datos como un conjunto, pero su aplicación no es compatible con la generación de código HDL.

Además, estableceremos un periodo de muestreo de 1/10000 (10 KHz). El bloque sumatorio y el analizador de espectro nos permite confirmar que los valores introducidos son correctos.

Para ello, introduciremos un bloque sumatorio que realice la operación:

$$y_1 + y_2 = 2 \cdot A \cdot cos\left(\frac{\Delta k}{2}x - \frac{\Delta \omega}{2}t\right) sen\left(\frac{(k_1 + k_2)}{2} - \frac{\omega_1 + \omega_2}{2}\right)$$

El analizador de espectro tomará esta salida y mostrará los componentes en frecuencia.

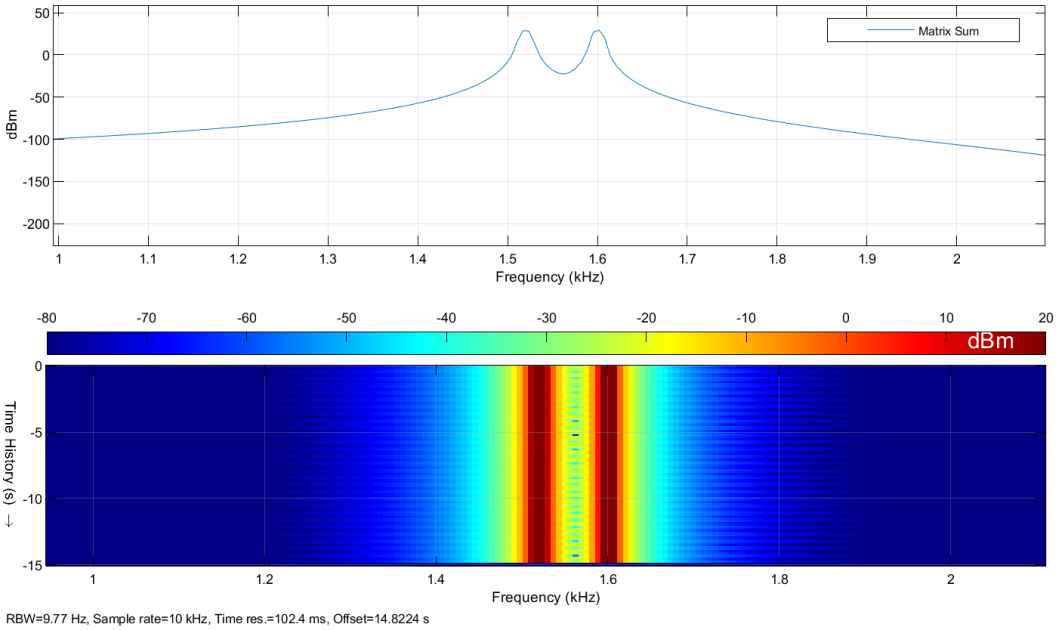


Ilustración 12 Salida del analizador de espectro

Las frecuencias mostradas contienen un margen de error introducido por la implementación del propio analizador de espectro.

Para poder determinar qué frecuencias están dentro de la señal que hemos muestreado, trabajaremos en el dominio de la frecuencia usando un algoritmo de transformada de Fourier (*Fast Fourier Transform*). Este bloque, perteneciente a la biblioteca estándar de Simulink, incluye una capa adicional de proceso que permite extraer el valor magnitud al cuadrado, de manera que será posible observar la potencia de cada frecuencia.

Configuraremos el bloque como se muestra en la Ilustración 13.

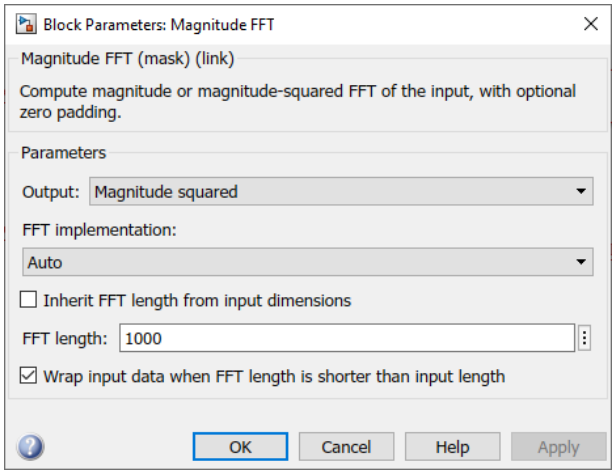


Ilustración 13 Configuración FFT

Se ha elegido un valor de 1000 en la longitud de FFT para obtener un valor coherente de puntos respecto a la frecuencia. Es decir, elegimos los puntos para simplificar más tarde la correlación entre el índice de los máximos del resultado de la FFT y la frecuencia real de cada pico. Este proceso ocurre en el siguiente bloque, donde usamos un bloque *Find Peaks,* configurado como buscador de

máximos, para obtener el valor máximo de cada vector. Conectamos un visualizador de datos a la salida *count* para confirmar que el número de máximos es el correcto para cada vector. Y los índices (alineados con la frecuencia por el número de puntos) son propagados a otro visualizador (para monitorizar el proceso) y a un bloque *demux* que separará los dos parámetros de que alimentarán a la función *estimate_doppler* implementada dentro de un bloque *MATLAB Function*, que permite ejecutar código escrito en lenguaje de Matlab.

La implementación es casi idéntica al algoritmo original, tan solo se cambiaron el orden de los parámetros para evitar cruzar las flechas entre bloques.

```
function estimatedVel = fcn(fo, fs)
%#codegen
c = 2.997e8; % Velocidad de la luz
estimatedVel = (c*fs)/fo - c; % Fórmula del efecto Doppler
```

La simulación arrojó idénticos resultados que el algoritmo original usando los mimos parámetros.

Modelo compatible con Pluto SDR

Para simular este algoritmo con integración de SDR, tenemos varias opciones:

- Simulación en tándem con dos instancias de Simulink.
- Simulación secuencial, almacenando variables en el espacio de trabajo de Matlab.

La primera opción es conveniente cuando el número de muestras, junto con el proceso asociado, no es muy alto. Además de iniciar dos instancias de Matlab y Simulink, es necesario conectar ambos dispositivos SDR y asociar cada uno a una instancia diferente. Nuestra metodología ha sido la de realizar la implementación usando una simulación secuencial, pasando a la simulación en tándem cuando hemos obtenido un modelo muy estable al que sólo modificaremos parcialmente algunos parámetros.

Ilustración 14 Dos Pluto en tándem junto con la placa Zedboard

Antes de comenzar, es necesario la calibración del sistema. Uno de los puntos débiles de los dispositivos ADALM PLUTO es la baja estabilidad del oscilador. Esto hace que sea necesario reemplazarlo en otro tipo de aplicaciones relacionadas con las comunicaciones por satélite, como por ejemplo los transceptores construidos por radio aficionados para las comunicaciones a través del satélite geoestacionario QO-100 [25] [26].

Este problema no es significativo en nuestro caso ya que la duración de los experimentos es bastante corta. A pesar de todo, el uso continuado durante una sesión produce un incremento en la temperatura del oscilador y, por tanto, un desplazamiento en frecuencia. Realizando algunas calibraciones durante la sesión se mitiga completamente el problema.

La calibración comienza conectando directamente el transmisor y el receptor de modo que el medio de transmisión afecte lo mínimo a la señal.

Una instancia de Simulink transmitirá una señal arbitraria en banda base (en nuestro caso una sinusoide de 12 KHz) a una frecuencia concreta (900 MHz). Para ello se necesita un modelo muy simple como el que se muestra en la Ilustración 15.

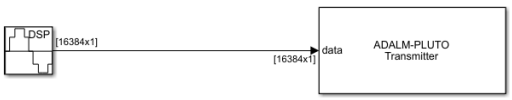


Ilustración 15 Calibración Tx

En la otra instancia de Simulink se usará un modelo similar que contenga un bloque analizador de espectro para poder analizar la señal.

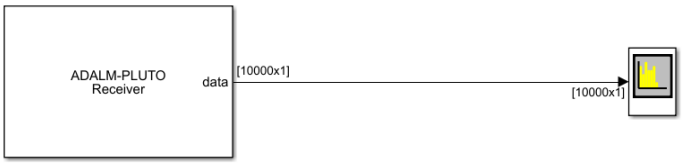


Ilustración 16 Calibración Rx

La salida de este instrumento virtual nos presenta la banda base de 100 KHz con la señal y sus armónicos. Esto es normal ya que estos dispositivos SDR no implementan filtros analógicos ni nosotros hemos incluido ningún filtro digital en el modelo de Simulink.

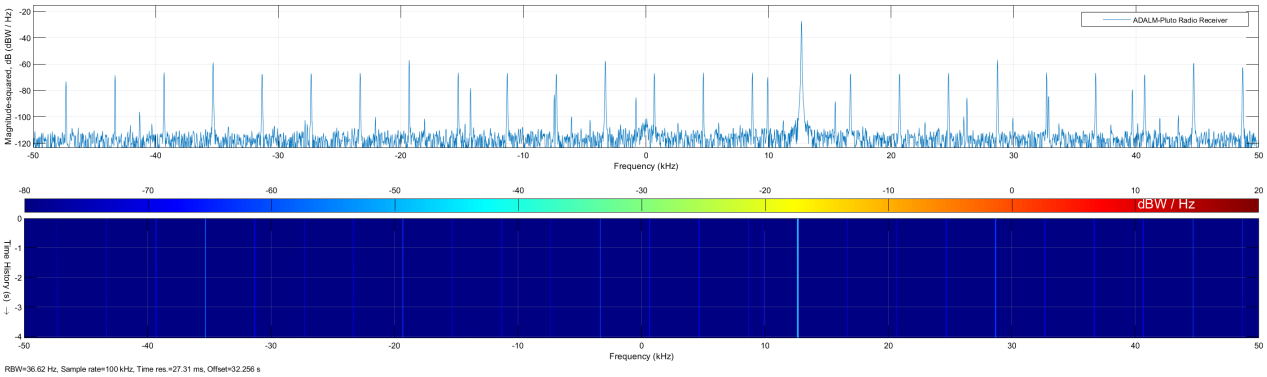


Ilustración 17 Espectro del proceso de calibración

Usando las herramientas del analizador de espectro, medimos que el tono principal se encuentra en 12.650 KHz. Esto quiere decir que la desviación en frecuencia que deberemos considerar como cero es 650 Hz.

Una vez que tenemos la calibración del sistema, se puede proceder a la ejecución de la implementación de la parte transmisora, cuyo diagrama del nivel más alto se presenta en Ilustración 18. Este modelo se ha implementado teniendo en cuenta las condiciones necesarias para ser convertido a HDL. El subsistema *Signal Generation* sirve de contendor para la parte que será ejecutada en la sección de lógica programable del FPGA.

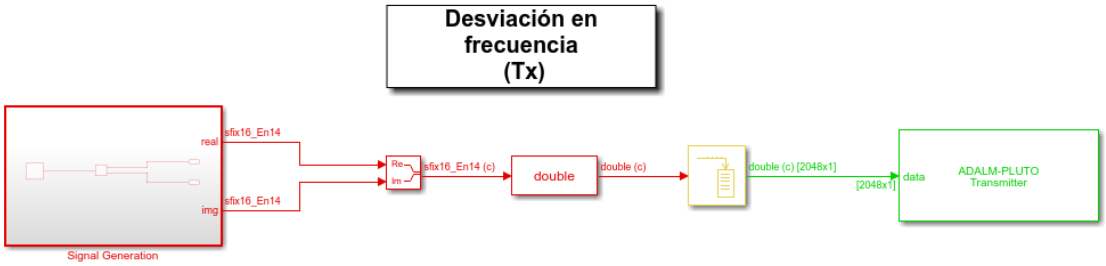


Ilustración 18 Nivel alto de la parte transmisora

Los diferentes colores representan las diferentes tasas de muestreo, siendo la principal 100 KHz (rojo), después convertida a la tasa de entrada necesaria para el bloque transmisor del ADALM PLUTO. Esta conversión es necesaria debido a la forma en que se gestionan las muestras en cada dominio. En el dominio de HDL, en cada paso se propaga una muestra, mientras que en el dominio al que pertenece los bloques de ADALM PLUTO, y que pertenecen al uso más tradicional de Simulink, cada paso propaga un *frame* que contiene un número superior de muestras. De esta manera, para la transmisión se necesita una frecuencia de muestreo inferior, pero trabajando con múltiples muestras en cada ciclo. La conversión entre uno y otro dominio se hace con el bloque *buffer*, que acumula muestras hasta completar un frame de 2048 muestras.

Otro aspecto relevante es el tipo de dato. En el dominio del HDL no disponemos de coma flotante ni de valores complejos. Para los primeros, y como ya se ha mencionado anteriormente, realizamos conversiones de coma fija de 16 bits.

Para los valores complejos, separamos la parte real de la imaginaria y los propagamos paralelamente.

Bajando la jerarquía, el interior del subsistema se presenta en la Ilustración 19.

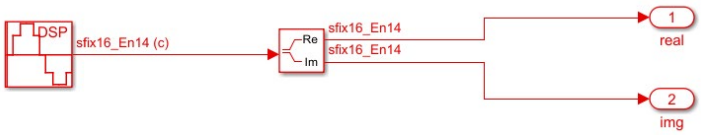


Ilustración 19 Subsistema de generación de señal

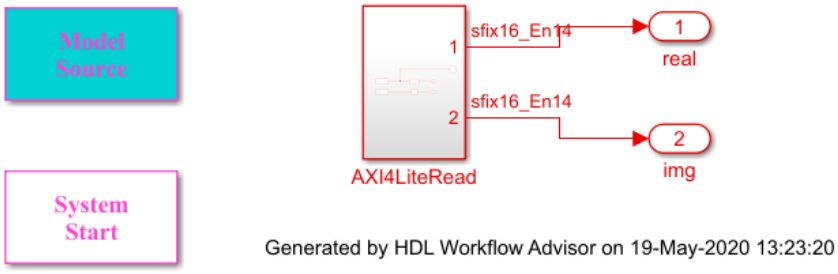
La generación de código HDL se realiza mediante la herramienta *HDL Coder* de Simulink. Para ello es necesario seleccionar el bloque a convertir para realizar la conversión. El proceso no es para nada automático. Es necesario configurar un gran número de opciones que condicionan el resultado final. Esta configuración puede ser guiada usando el *HDL Workflow Advisor.* En el caso concreto que nos ocupa, es importante excluir del modelo los bloques de buffer y de ADALM PLUTO, ya que no son compatibles con HDL Coder.

Los parámetros más importantes que hemos seleccionado son:

- Generación de IP Core: el resultado no será solo código HDL ejecutable, sino también un contenedor listo para ser incluido en un proyecto de Vivado.
- Los puertos *img* y *real* estarán mapeados a un interfaz del tipo AXI-Lite, con direcciones de acceso 0x100 y 0x104.
- Frecuencia del reloj principal es de 10 MHz. Un valor mayor arrojaría un error en la generación al no poder alcanzarse *los objetivos de timing*.

El resultado es un código HDL, que además ha sido sintetizado en un *bitstream* y descargado en el dispositivo.

HDL Coder también es capaz de ayudar a generar interfaces para el IP que acabamos de generar. En este caso, se ha generado un interfaz AXI4-Lite con dos registros (en 0x100 y 0x104) para acceder a los puertos de salida.



Este interfaz AXI4-Lite debe ser generado con *Embedded Coder* y desplegado en el sistema antes de poder ser usado.

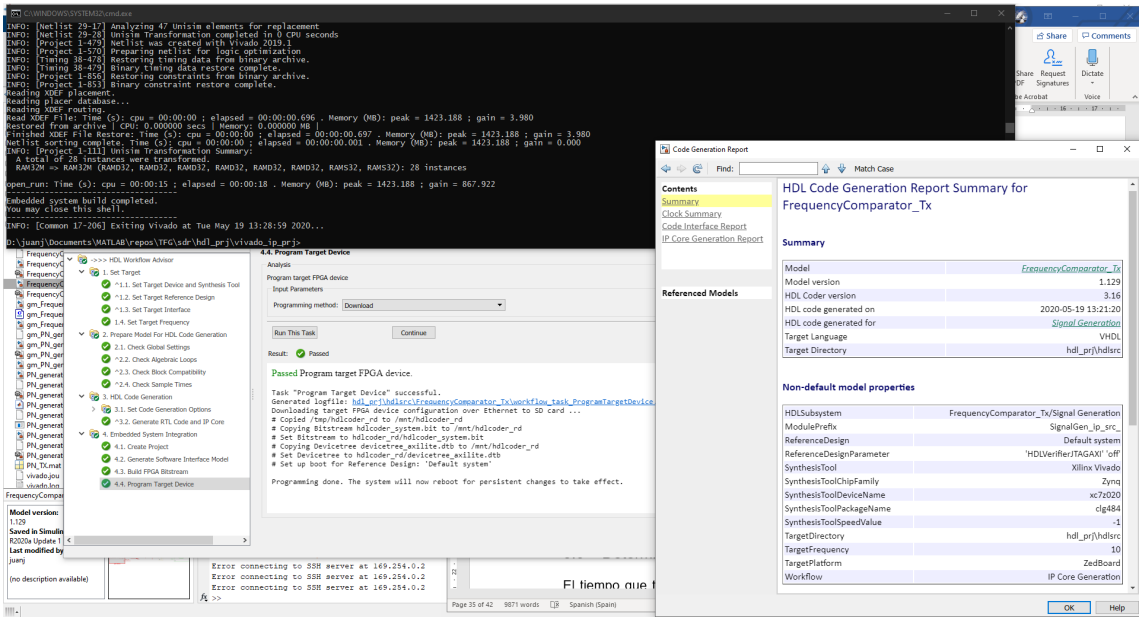


Ilustración 20 Proceso de generación de código, síntesis y descarga al dispositivo completado

La parte receptora se ejecuta en otra instancia de Simulink, y tiene como nivel más alto del modelo el flujo que se muestra en la Ilustración 21.

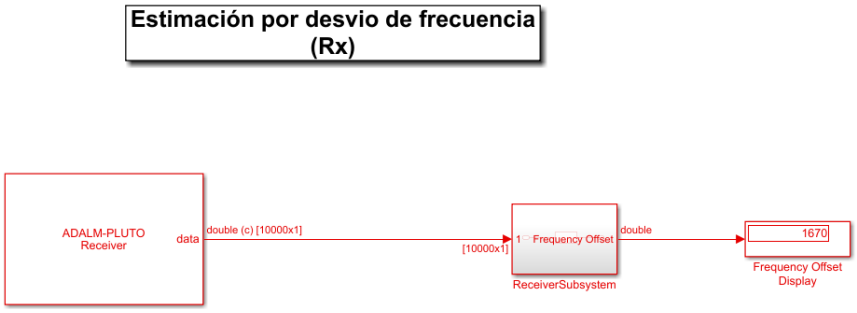


Ilustración 21 Recepción de la señal para la estimación de velocidad

Receiver Subsystem será el bloque del que generaremos código más adelante. Este bloque contiene la lógica que recibe la señal en banda base desde el receptor SDR y realiza la búsqueda de picos en el dominio de la frecuencia, además de calcular el desvío sobre el tono que insertamos en la transmisión.

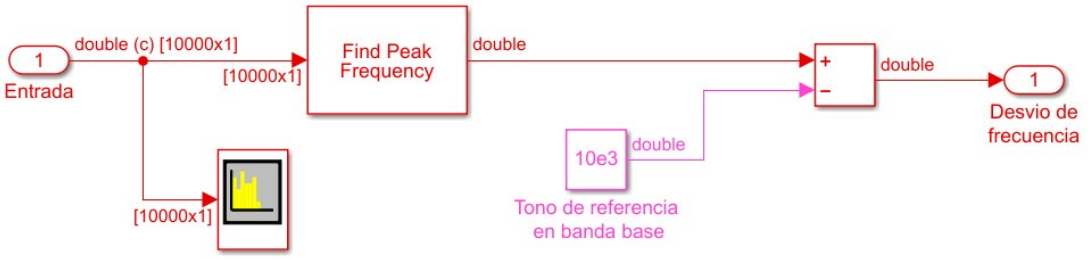


Ilustración 22 Detalle de Receiver Subsystem

Find Peak Frequency es un bloque proporcionado por Mathworks cuya implementación es la misma que la de nuestro modelo inicial de Simulink (basado en la FFT), con la excepción de que incluye un filtrado y una salida más precisa de la frecuencia del pico, que será substraída al valor de tono de referencia para una salida numérica del desvío de frecuencia.

La generación de código C del bloque arrojó los siguientes resultados:

Archivo	Descripción
ert_main.c	Ejemplo de integración en ejecutable
ReceiverSubsystem.c	Biblioteca de funciones específicas del bloque
ReceiverSubsystem.h	Archivo de cabecera con los tipos y los prototipos de llamada de las funciones
ReceiverSubsystem_data.c	Constantes usadas que incluyen los coeficientes de los filtros y otros parámetros necesarios
rtwtyper.h	Archivo genérico de tipos usados por Mathworks. No es generado automáticamente pero sí es enlazado
rtmodel.h	Cabecera interfaz que contiene ciertas funciones necesarias para mantener compatibilidad con versiones anteriores

Una vez ejecutado el modelo de transmisión en una instancia de Simulink, y el modelo de recepción en otra instancia, después de haber realizado una calibración que arrojó un valor de desvío de 740 Hz para un tono transmitido en 10 KHz, tal como se muestra en la Ilustración 23.

Para verificar el funcionamiento, hemos ido variando la frecuencia de sintonización del receptor, de manera que simulamos el desvío debido al Doppler. Nos encontramos con las limitaciones del oscilador cuando intentamos desplazar la señal al valor cero.

Frecuencia Sintonizada	Desvío sobre la frecuencia original	Desvío mostrado en el modelo en el tono de 10 KHz
900000000 Hz	0	740 Hz
900001000 Hz	1 KHz	-332 Hz
900000500 Hz	500 Hz	90 Hz
900000740 H	740 Hz	-50 Hz

Aunque existe cierta exactitud, que el oscilador del transmisor y del receptor varíen ligeramente las frecuencias a medida que varíen otros parámetros, principalmente la temperatura. Esto quiere decir que para obtener valores precisos en satélites que orbiten de manera que la velocidad relativa sea baja (como las órbitas geoestacionarias), será necesario un SDR que sea mucho más estable en frecuencia.

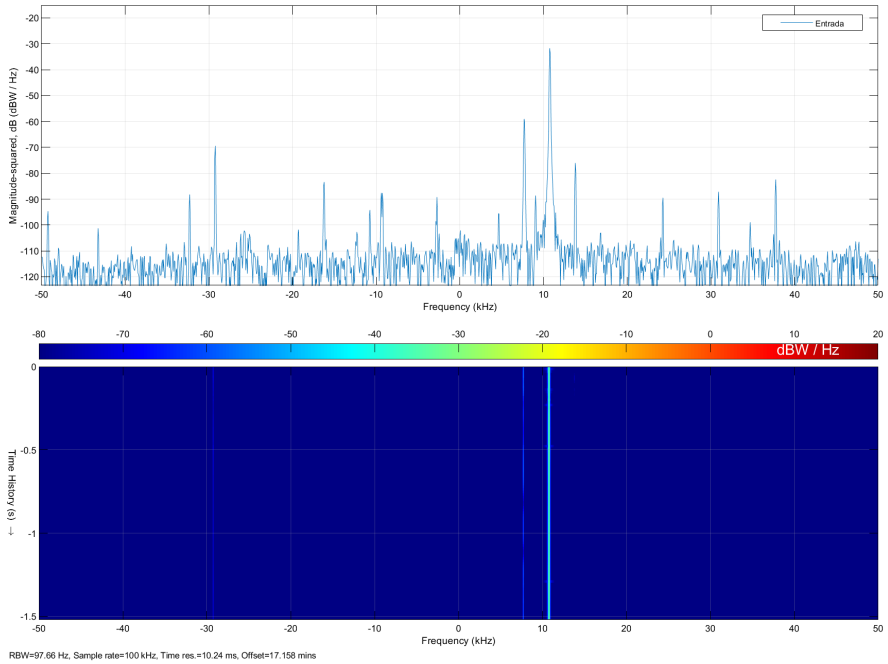


Ilustración 23 Espectro en BB mostrando el tono de 10 KHz con un desvío de 740 Hz

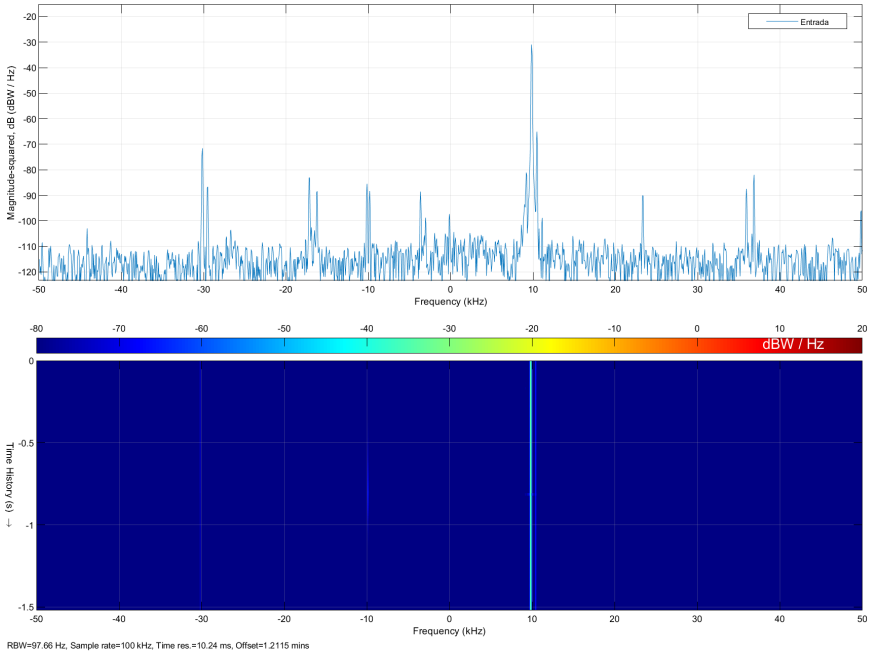


Ilustración 24 Espectro en BB con una frecuencia de sintonización desplazada para un tono de 10 KHz

3.3 Determinación de la distancia mediante el retardo

El tiempo que tarda en llegar la señal al satélite, desde la estación terrena, y vuelta, nos puede indicar con bastante precisión la distancia a la que se encuentra.

Según [3], el tiempo que tarda la señal en recorrer un enlace de satélite es:

$$t_{SL} = \frac{R}{c}$$

Ecuación 5. Retardo en un tramo

Siendo c la velocidad de la luz y R la distancia recorrida. Como el camino para todo el enlace incluye subida y bajada, la expresión se expande para incluir el tramo de U/L y de D/L:

$$t_{enlace} = t_u + t_{gd} + t_d$$

Ecuación 6. Retardo en un enlace por satélite

Donde t_u es el tiempo transcurrido desde que se transmite la señal hasta que llega a la antena del transpondedor, t_{gd} es el retardo de grupo (*group delay*) introducido por los sistemas internos del transpondedor para la frecuencia de la señal, y t_d es el intervalo de tiempo desde que la señal deja el satélite hasta que es captada por la estación terrena.

Para el caso en que la estación terrena receptora es la misma que transmitió la señal original, los valores t_u y t_d serán idénticos, simplificando la expresión.

$$t_{enlace} = 2t_{ud} + t_{gd}$$

Ecuación 7. Retardo para una única estación terrena

El retorno de grupo para la frecuencia a la que estamos emitiendo debería ser conocido. Se asume que, dentro de las pruebas de ensamblado, integración y prueba (AIT) del vehículo, se identificó de manera experimental este parámetro.

Una vez recibida la transmisión, la estación terrena deberá ser capaz de identificar de la manera más precisa posible, el retardo de la señal. Para eso es importante usar usa un tipo de información que sea fácil de ser comparada consigo misma. Para ello usaremos la propiedad de autocorrelación de una señal, cuyo máximo nos dará el retardo de la señal.

Para que la autocorrelación de una señal pueda dar valores máximos claros, es importante que no hay ambigüedades dentro de la señal. En el contexto actual, donde el objetivo no es transmitir información arbitraria sino medir el tiempo de viaje de la señal, es posible definir qué señal vamos a transmitir. Esta señal deberá tener las mejores propiedades posibles para una autocorrelación que no sea ambigua.

Según los estándares actuales para este tipo de sistemas, se recomienda el uso de un generador de números pseudoaleatorios de tipo *Weighted-voting balanced Tausworthe* [12]. Este tipo de procesos genera un código a partir de 6 secuencias binarias periódicas con un algoritmo de lógica combinacional que usa a su vez un reloj externo para seleccionar uno u otro. Esta recomendación se hizo con la idea de poder enviar comandos al mismo tiempo que se realizaba las mediciones de retardo. De esta manera, las secuencias pueden ser utilizadas para el envío de datos usando técnicas de espectro ensanchado (*Spread Spectrum*) con una longitud de secuencia de 1.009.470 chips [12].

Para el desarrollo de este algoritmo vamos a usar una versión más simplificada de generador pseudoaleatorio que no permita el envío de secuencias arbitrarias de información. El mensaje será conocido a priori y semánticamente no tendrá información.

Dentro de las herramientas matemáticas y de la teoría de la comunicación que se encuentran a nuestra disposición, hemos elegido una técnica usada comúnmente en comunicaciones de telefonía móvil.

Las secuencias de Gold son unas secuencias de números pseudo aleatorios que se basan en dos generadores independientes de donde se seleccionan unos elementos denominados *par preferido* que son calculados basados en dos polinomios que son seleccionados y que caracterizan los registros de cada generador. Las secuencias de Gold tienen muy buenas características de autocorrelación y son usadas habitualmente en comunicaciones CDMA [27].

Implementación en Matlab

El algoritmo matemático que se propone es la generación de dos secuencias de Gold (con idénticos parámetros) que representen la generación de bits en el transmisor y en el receptor. Luego realizaremos la correlación entre ambas, introduciremos un retardo y volveremos a realizar la correlación.

Hemos creado un *live script* de Matlab que demuestra este comportamiento y que hemos incluido en los anexos.

El primer paso es crear una secuencia de Gold, cuya creación se basa en los dos polinomios siguientes:

$$g(x) = x^6 + x^5 + x^2 + 1$$
$$z(x) = x^6 + x^5 + x^4 + x^3 + x^2 + 1$$

Con condiciones iniciales de [1 0 1 0 1 0] y [0 1 0 1 0 1] respectivamente. El tipo de datos que hemos usado es *logical,* es decir, la secuencia de números será de ceros y unos, o lo que es lo mismo, una secuencia binaria.

Después de crear ambas secuencias pseudoaleatorias, debemos comprobar que efectivamente son idénticas. De esta manera garantizamos que podremos crear la misma secuencia en el transmisor y en el receptor. Para ello usaremos

la función de Matlab *xcorr,* que realiza una correlación cruzada entre ambas secuencias.

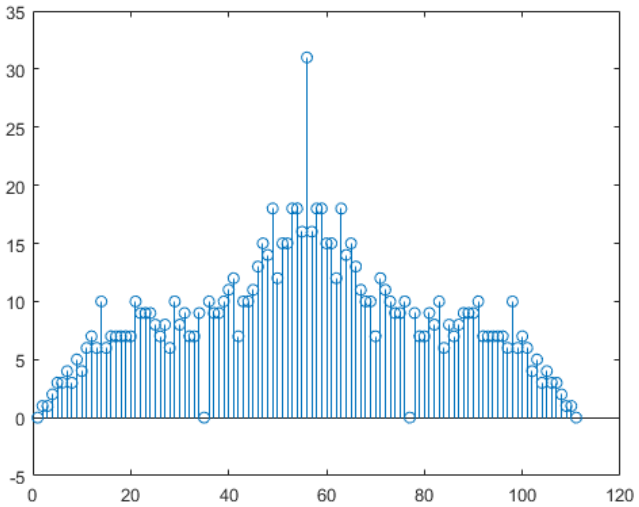


Ilustración 25 Correlación cruzada de las dos secuencias de Gold

La Ilustración 25 indica un máximo claro en la posición 56, que corresponde a la longitud de la secuencia generada (el último elemento). Ambas secuencias están alineadas.

Si introducimos un retardo en la señal, usando el comando *Delay* del *DSP toolbox* de Matlab, de 16 muestras, comprobamos que la nueva correlación nos indica el retardo de 16 elementos con un máximo en 40 (que corresponde a 56 - 16).

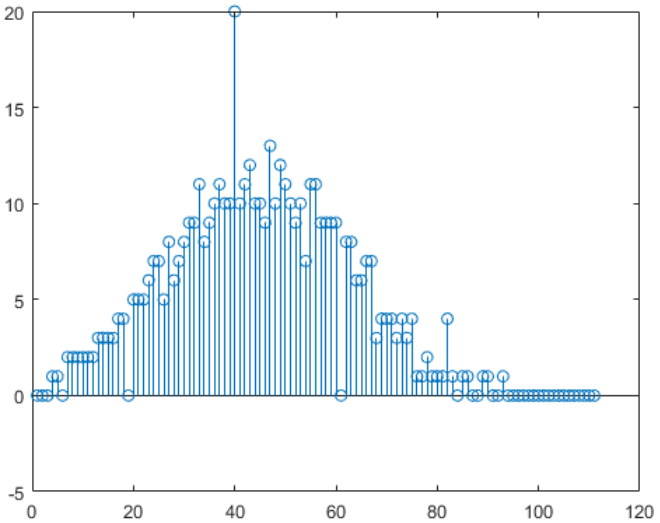


Ilustración 26 Correlación con un retardo de 16 muestras

Hasta ahora se han considerado que las dos secuencias son finitas, es por eso por lo que al generar la gráfica vemos valores aproximándose al cero a medida que progresamos a lo largo del eje x. Este no es el caso real, ya que iremos repitiendo una y otra vez la secuencia de manera que a la primera muestra le precederá la última de la secuencia precedente. Para simular este aspecto, utilizaremos la función *circshift,* que trata la secuencia como si fuera circular, desplazando elementos en una dirección y volviendo a introducir por un extremo los que salen por el otro.

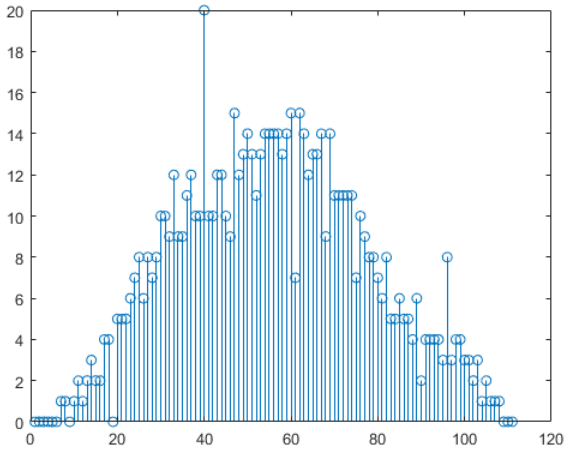


Ilustración 27 Correlación de la secuencia desplazada de manera circular

En la Ilustración 27 se puede observar que el máximo se mantiene en la posición 40 después de realizar este desplazamiento circular.

Implementación en Simulink

La implementación en el dominio del tiempo mediante Simulink se simplifica bastante gracias al bloque *Find Delay* de la biblioteca DSP de Simulink.

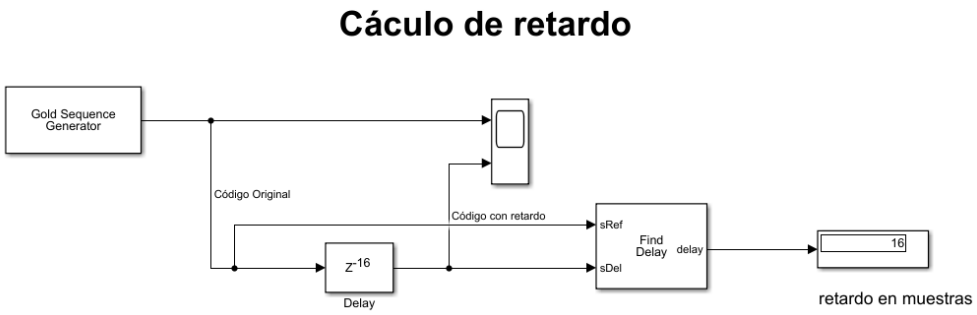


Ilustración 28 Modelo de Simulink para el cálculo de retardo

Como se puede ver en la Ilustración 28, trasladar el algoritmo de Matlab implica el uso de bloques estándar, aunque, para que los resultados sean coherentes, la configuración debe hacerse adecuadamente.

La configuración del generador de secuencias de Gold se configuró tal como se muestra en la Ilustración 29.

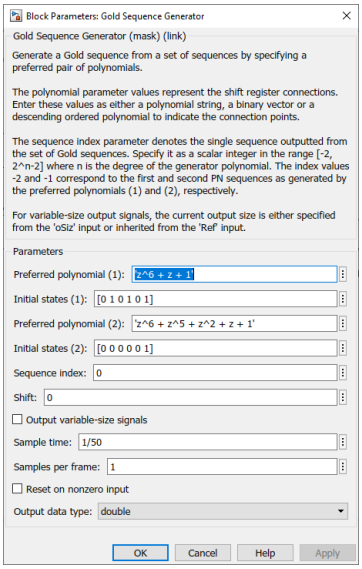


Ilustración 29 Configuración del generador de secuencias de Gold

Esta configuración incluye los mismos polinomios y establece una frecuencia de muestreo de 50 Hz. Más tarde se retarda la señal en 16 muestras. En la Ilustración 30 se puede observar que existe un retardo de 320 ms. Este resultado es compatible con la frecuencia de muestreo de 50 Hz. Si una muestra supone un retardo de 20 ms, 16 muestras suman un total de 320 ms.

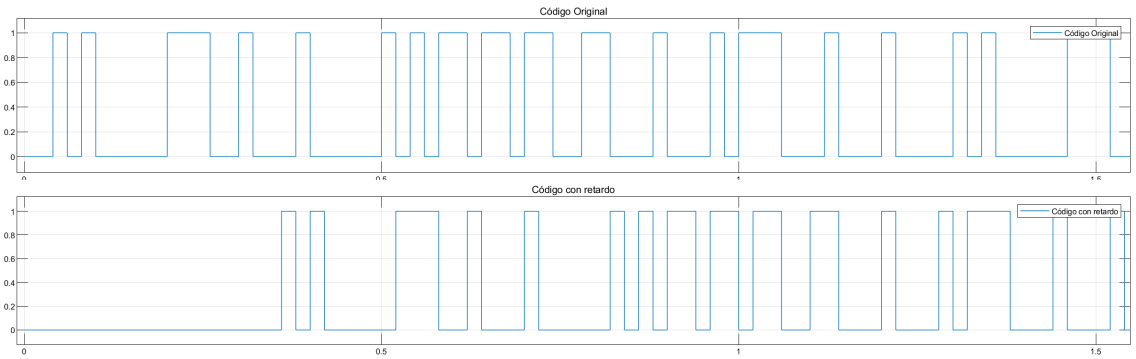


Ilustración 30 Código original y retardado en 16 muestras

La salida del modelo de Simulink es de 16 muestras, luego realiza una estimación correcta.

Modelo compatible con sistemas de radio

Un sistema de radio que incluya todos los subsistemas necesarios para la transmisión de información digital de manera fiable es algo muy complejo. Dentro de los modelos de ejemplo de ADALM PLUTO para Simulink nos encontramos

con un receptor que contiene las mínimas características para recibir un mensaje corto. Parte de este modelo, con los subsistemas correspondientes se muestran en la Ilustración 31.

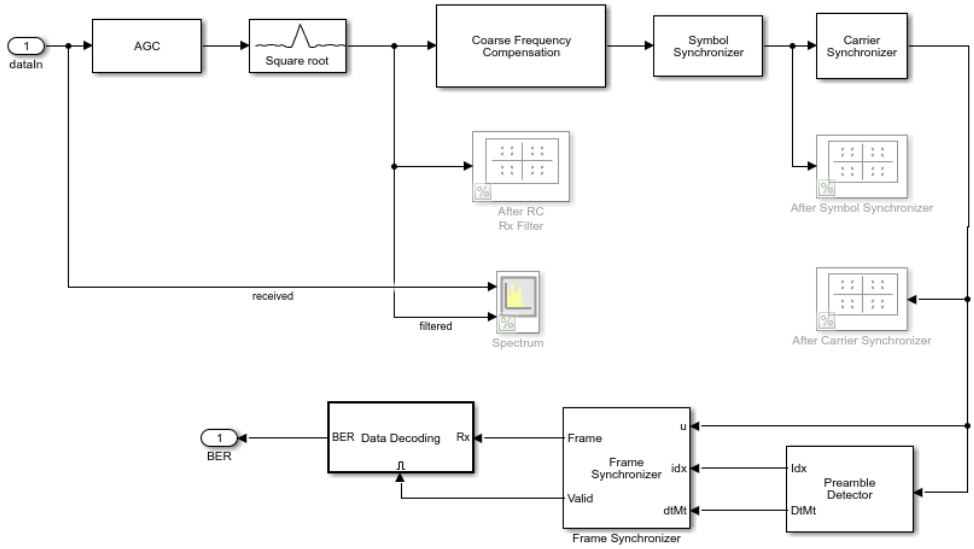


Ilustración 31 Ejemplo de un sistema completo de recepción digital en Simulink

Debido a la limitación en tiempo para la ejecución de este TFG, se queda fuera del alcance del trabajo la implementación completa de un transmisor y un receptor para nuestro algoritmo. Aun así, sí es posible implementar dos subsistemas que bien podrían ser integrados en un sistema completo de jerarquía superior.

Estos subsistemas, uno de transmisión y otro de recepción, van a ser integrados en Simulink y se añadirá un canal gaussiano para simular un canal de comunicaciones.

El modelo propuesto es el mostrado en la Ilustración 32.

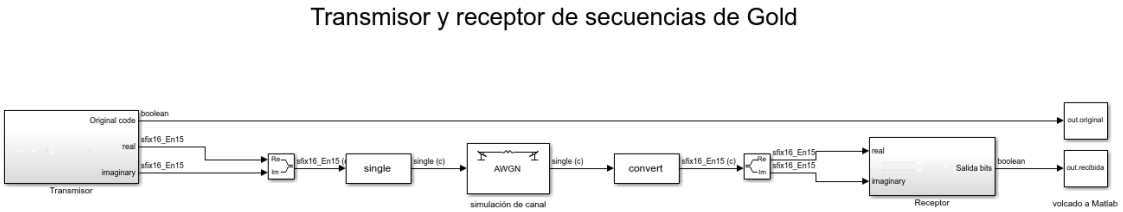


Ilustración 32 Modelo de Simulink con simulación de canal

Los subsistemas de transmisión y recepción están conectados por un canal simulado. Las salidas de ambos subsistemas se vuelcan al espacio de trabajo de Matlab para poder ser comparadas.

El bloque que contiene el sistema de transmisión se presenta en la Ilustración 33.

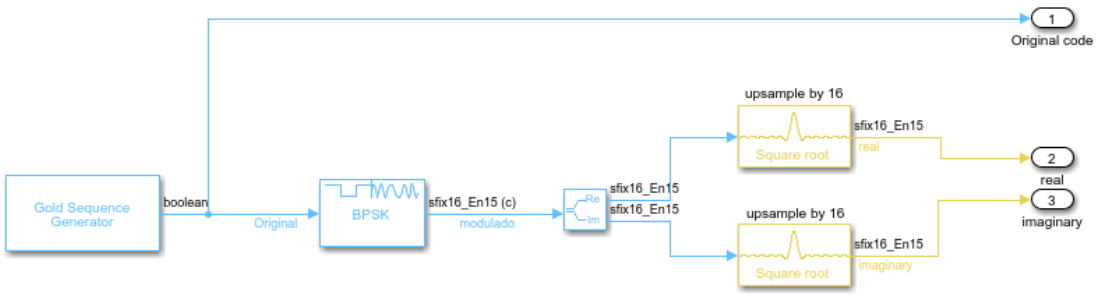


Ilustración 33 Subsistema de Transmisión

La transmisión consta de los siguientes elementos:

- Un modulador BPSK (siguiendo las recomendaciones del CCSDS para telemetría [28]) con una constelación mostrada en la Ilustración 34.
- Un filtro de coseno realzado (dual) que realiza a su vez un cambio de tasa de muestreo de x16 (upsample). El valor de *roll off* elegido es de 0.5, aunque es común encontrar sistemas de comunicaciones por satélite que funcionan con un mejor valor, pero este parámetro aún sigue siendo bastante común.

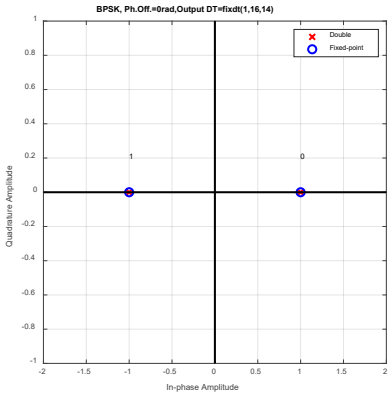


Ilustración 34 Constelación BPSK

Cabe destacar que, al ser una señal digital modulada en BPSK, no esperamos que el camino de la parte imaginaria sea distinto de cero. De hecho, si realizamos una simulación, veremos que no toma ningún valor. Esto queda claro después de inspeccionar el diagrama con la constelación. Los valores se moverán a lo largo del eje real (*in phase*), mientras que el valor de cuadratura será siempre cero.

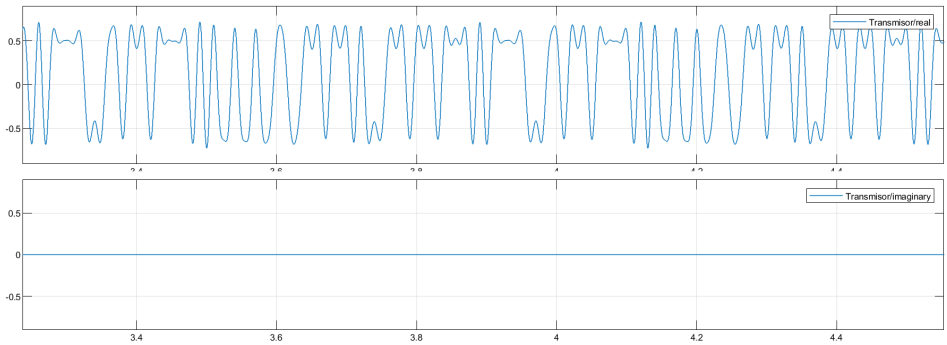


Ilustración 35 Salida del bloque de transmisión. Valores I/Q

Los caminos de la parte imaginaria se han dejado para poder ser verificados y para presentar una versión más completa. Una mejora del sistema que incluyera otro tipo de constelación (como QPSK, por ejemplo), necesitaría muy pocas modificaciones en subsistema original.

El subsistema de recepción (Ilustración 36) es simétrico al de recepción.

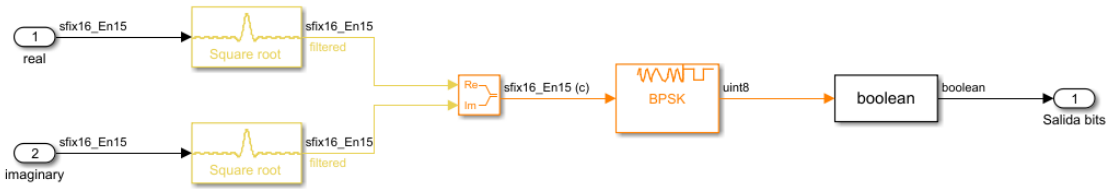


Ilustración 36 Subsistema de recepción de secuencias de Gold

Los filtros de coseno realzado contienen valores espejo a los del transmisor y realizan una conversión en la tasa de muestreo que divide por 16 muestras cada símbolo (*downsample*). El resultado es convertido a bits y enviado al espacio de trabajo de Matlab donde puede ser comparado.

Hay que tener en cuenta que se producen errores de bits. No estamos usando ninguna técnica que previene este problema, y estamos simulando un canal de comunicaciones que añade ruido. Aún así, usar la correlación entre señales es la mejor opción. Puede que las señales no sean idénticas, pero gracias a que se han escogido códigos que permiten una alta correlación, es posible realizar comparaciones más o menos precisas de retardo.

En el caso de nuestro modelo, en su forma actual, no tiene retardo. Para verificarlo realizamos las mismas operaciones que verificaron el algoritmo inicialmente.

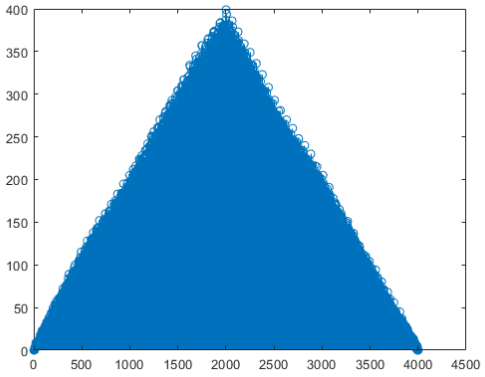


Ilustración 37 Correlación sin retardo

El diagrama de la Ilustración 37 presenta una gran cantidad de muestras debido a que la frecuencia de muestreo es mucho mayor. Pensamos que la forma triangular es a causa de la correlación de tramas periódicas. Y los máximos locales van formando los laterales del triángulo hasta su vértice, donde el máximo global coincide con el máximo local. La posición del máximo es exactamente el punto medio (retardo 0).

Ahora introduciremos un retardo de 200 muestras usando el bloque estándar de Simulink *Delay.* La función correlación resultado se muestra en la Ilustración 38, cuyo máximo se encuentra desplazado 200 muestras, indicando precisamente esta cantidad de retardo.

También notamos que no hay mucho margen de ruido. Un detrimento de la señal dificulta la correlación y el cálculo del retardo comienza a no ser preciso. Después de incrementar el valor de ruido en el canal, el cálculo de retardo empieza a arrojar pequeños valores de varias muestras, que acaban siendo decenas al incrementar todavía más los valores de ruido de canal.

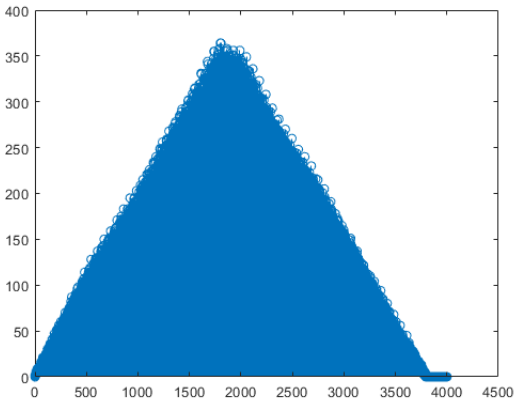


Ilustración 38 Correlación con un retardo de 200 muestras

Es muy importante resaltar que, una vez implementados y sintetizados los subsistemas, los filtros de coseno realzado introducirán un retardo. La herramienta de análisis de filtros de Matlab reporta un retardo de 128 muestras,

según se muestra en la Ilustración 39. Este retardo, como parte inherente del sistema, debería tenerse en cuenta al ejecutar los algoritmos de transmisión y recepción desde el hardware.

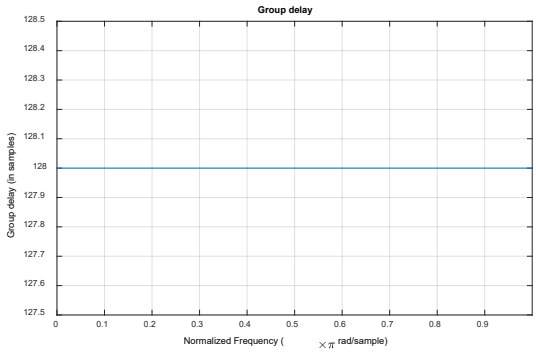


Ilustración 39 Retardo de grupo de los filtros de coseno realzado

La conversión a HDL se realizó para los subsistemas de transmisión y recepción.

Para el transmisor se generaron los siguientes archivos:

Archivo	Descripción
FIR_Interpolation.vhd	Coeficientes del filtro
FIR_Interpolation_block.vhd	Implementación del filtro
PN_Sequence_Generator1.vhd	Generador 1 del generador de Gold
PN_Sequence_Generator2.vhd	Generador 2 del generador de Gold
Gold_Sequence_Generator.vhd	Generador de secuencias de Gold
Raised_Cosine_Transmit_Filter2.vhd	Filtro de coseno realzado (real)
Raised_Cosine_Transmit_Filter3.vhd	Filtro de coseno realzado (imaginario)
BPSK_Modulator_Baseband.vhd	Modulador BPSK
Transmisor_tc.vhd	*Timing controller* de Transmisor.vhd
Transmisor.vhd	Módulo superior

Para el Receptor se generaron los siguientes archivos:

Archivo	Descripción
FIR_Decimation.vhd	Coeficientes del filtro
FIR_Decimation_block.vhd	Implementación del filtro
Raised_Cosine_Receive_Filter.vhd	Filtro de coseno realzado (real)
Raised_Cosine_Receive_Filter1.vhd	Filtro de coseno realzado (imaginario)
BPSK_Demodulator_Baseband.vhd	Modulador BPSK
Receptor_tc.vhd	*Timing controller* de Receptor.vhd
Receptor.vhd	Módulo superior

A su vez, se generaron interfaces AXI-LITE para ambos módulos.

3.4 Determinación de la posición mediante la diferencia de tiempos de llegada

Dentro del contexto de la aplicación de las técnicas descritas en este TFG, donde un transmisor o modem está conectado a otros equipos. Estos son típicamente un conversor en frecuencia, un HPA o LNA, y un sistema de antena que incluye un dispositivo de apuntamiento de antena (*antenna tracker*) que monitoriza la radiobaliza del satélite y que permite tener unos valores más o menos precisos del azimut y la elevación para una máxima potencia, según el diámetro de la antena. Esto se debe a que las antenas parabólicas reducen su apertura cuanto más grande es el diámetro del reflector.

Estudiando la órbita [3] es posible definir un punto P sobre la superficie de la Tierra que será la localización de la estación terrena. Para localizar el satélite desde el punto P son necesarios dos ángulos: elevación y azimut.

El ángulo de elevación es el ángulo entre el horizonte en el punto P y el satélite, medido desde el plano que contiene a P, al satélite y el centro de la tierra. Según [3], este vendrá dado por:

$$senE = \frac{\left[cos\Phi - \frac{R_E}{r}\right]}{\frac{R}{r}}$$

Ecuación 8. Cálculo geométrico de la elevación

Donde E es el ángulo de elevación, Φ es el ángulo que forman el eje del centro de la Tierra con el satélite y el eje que forman el centro de la Tierra con P, R_E es el radio de la Tierra, R es la distancia entre P y el satélite, y finalmente r será la distancia entre el centro de la tierra y el satélite.

El ángulo de azimut es el ángulo medido en el plano horizontal, entre el norte geográfico y la intersección de este plano con otro que contenga al satélite y al centro de la Tierra.

A partir de unos valores medidos de azimut y elevación aceptables, es posible determinar la posición instantánea del satélite dentro del sistema inercial formado con la estación terrena. Aun así, y para dar una visión más completa del proceso de localización dentro de este TFG, se incluye una implementación del algoritmo en Matlab de *diferencia de tiempos de llegada* (TDOA) también conocida en el ámbito del espacio como ΔDOR [29], que permite el cálculo de una posición basándose en el tiempo que tarda en llegar una señal transmitida a una serie de diferentes receptores. Si el sistema es capaz de medir con precisión fracciones de segundo, entonces será capaz de determinar la posición de manera muy exacta. El CCSDS estandariza este tipo de mediciones para lo que ellos denominan *Misiones de categoría B*, es decir, aquellas que operan a una altitud mayor de $2x10^6$ Km de altitud. Esto excluye a la práctica totalidad de satélites que orbitan la Tierra, incluidas las plataformas geoestacionarias, que se encuentran a una órbita de $35x10^3$ Km. Aun así, existen sistemas basados en

este principio para la monitorización de satélites, como PaCoRa [30], que se basa exclusivamente en la medición de la diferencia de tiempos [31].

Según las recomendaciones del CCSDS [28], para frecuencias en la banda de los 2 GHz, se recomienda el uso de un tono sinusoidal que tenga como armónico de la frecuencia fundamental 1 MHz o 4 MHz, a diferencia de otras bandas, donde la recomendación es usar varios tonos con valores de frecuencia fundamental más restrictivos.

Implementación en Matlab

Este algoritmo está basado en un estudio de Ufuk Tamer [32], que presenta una técnica simplificada para calcular TDOA tanto por el método lineal como por el método de Taylor. Nosotros vamos a presentar una implementación en Matlab basada en el método lineal. Esta implementación se encuentra en el anexo.

Al ejecutar el script se eligen aleatoriamente las posiciones de las estaciones terrenas y la del satélite. Después calcula (con un margen artificial de error) los tiempos de llegada, y realiza la estimación a partir de estos.

En cada ejecución los resultados son diferentes, pero siempre deben de ser coherentes. Un ejemplo se muestra en la Ilustración 40.

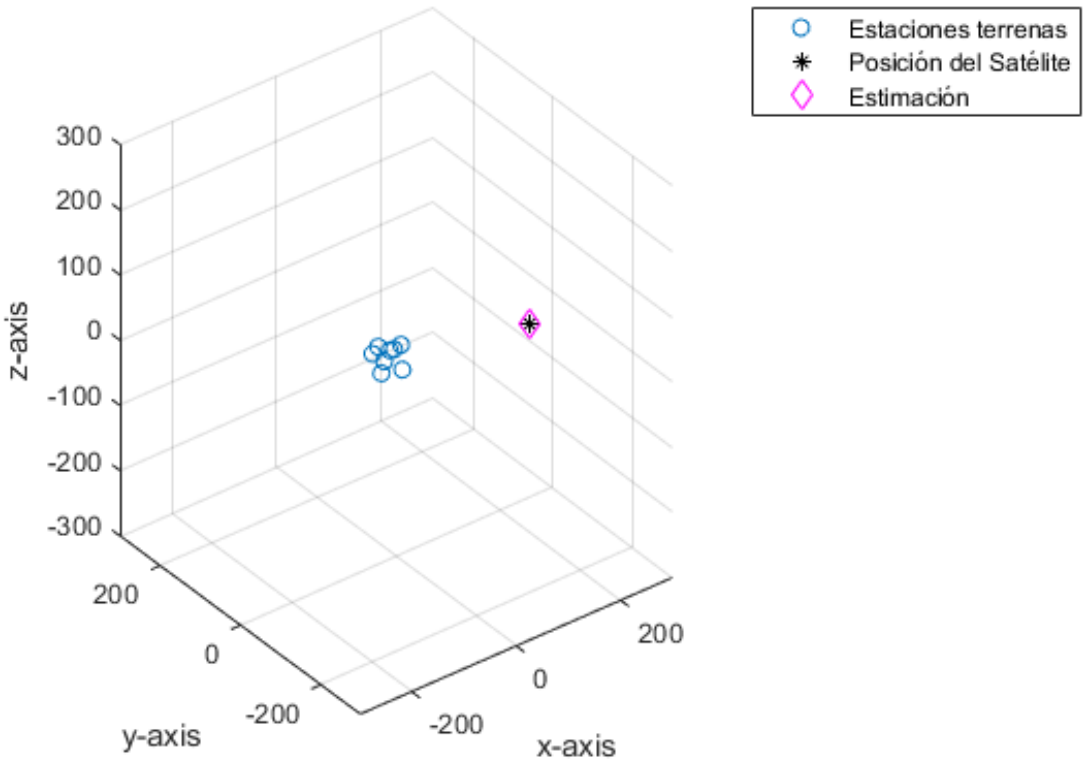


Ilustración 40 Ejemplo de ejecución de la simulación de TDOA

3.5 Integración de los algoritmos

En este apartado se presenta un diseño a muy alto nivel de un modem que integre todos los algoritmos. Si el trabajo de diseño de los algoritmos se realizó desde Matlab, el trabajo de integración se ha de realizar desde Xilinx Vivado. Aunque el esfuerzo necesario para realizar una integración completa lo deja fuera del alcance de este TFG, sí que queremos presentar un diseño inicial del sistema que pudiera servir como punto de partida para un desarrollo posterior. La estructura general de este sistema será la presentada en la Ilustración 41.

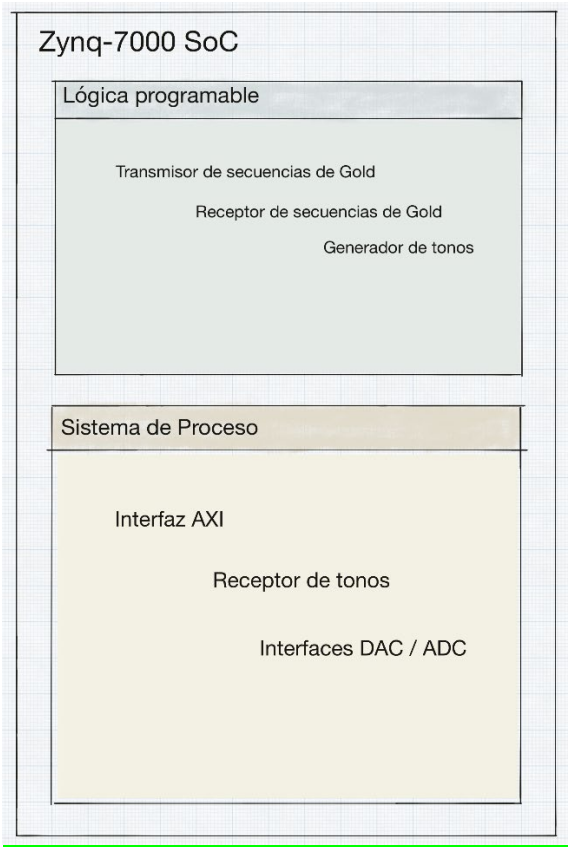


Ilustración 41. Estructura general del Sistema.

El sistema tendrá en el área de la lógica programable:

- El algoritmo transmisor de estimación de velocidad por desviación de frecuencia.
- El algoritmo transmisor de estimación de retardo
- El algoritmo receptor de estimación de retardo

En la zona del procesador:

- El algoritmo receptor de estimación de velocidad por desviación en frecuencia.
- Los interfaces AXI-LITE con la lógica programable
- Los interfaces con los conversores de datos

Diseño de hardware e integración con la lógica programable

Los algoritmos implementados en HDL deberán alojarse en el área de lógica programable del FPGA (PL) y además deberán ser interconectados con el sistema de proceso (PS).

La Ilustración 42 presenta el diseño de un sistema completo basado en la placa de desarrollo Zedboard. Este sistema incorpora los siguientes elementos:

- El sistema de proceso (sys_cpu): que representa los núcleos ARM y los periféricos empotrados en el SoC. En el diagrama se puede apreciar que tan solo hemos conectado la línea de interrupciones (IRQ_F2P), la memoria DDR y las entradas y salidas que incorpora la placa, uno de los relojes (FCLK_CLK0) y reset. Hemos conectado también el puerto maestro AXI a un bloque de interconexión para controlar múltiples periféricos AXI esclavos. Por otro lado, hemos decidido mantener un diseño minimalista y hemos desactivado periféricos y entradas adicionales, como son el USB y el GPIO.

- Sistemas de reset (sys_100m_rstgen, sys_core_rstgen y proc_sys_reset_0). Son sistemas que gestionan el reinicio de cada IP core de manera independiente.

- Un reloj dual (core_clkwiz). Este generador de reloj está configurado para producir dos frecuencias de reloj diferentes. Los IP core de transmisión y recepción de códigos de Gold se generaron para funcionar a una frecuencia de 10 MHz, mientras que el generador de tonos para medir la desviación de frecuencia funciona a 50 MHz.

- Un bloque de interconexión de periféricos AXI (axi_cpu_interconnect), que permite conectar múltiples esclavos AXI a un único maestro AXI, que en nuestro caso es el PS.

- Los subsistemas que hemos generado en Matlab:
  - PN_genera_ip_tx_0: transmisor de códigos de Gold en BPSK.
  - PM_genera_ip_rx_0: receptor de códigos de Gold en BPSK.
  - SignalGen_ip_0: generador de tonos para el cálculo de la desviación de frecuencia.

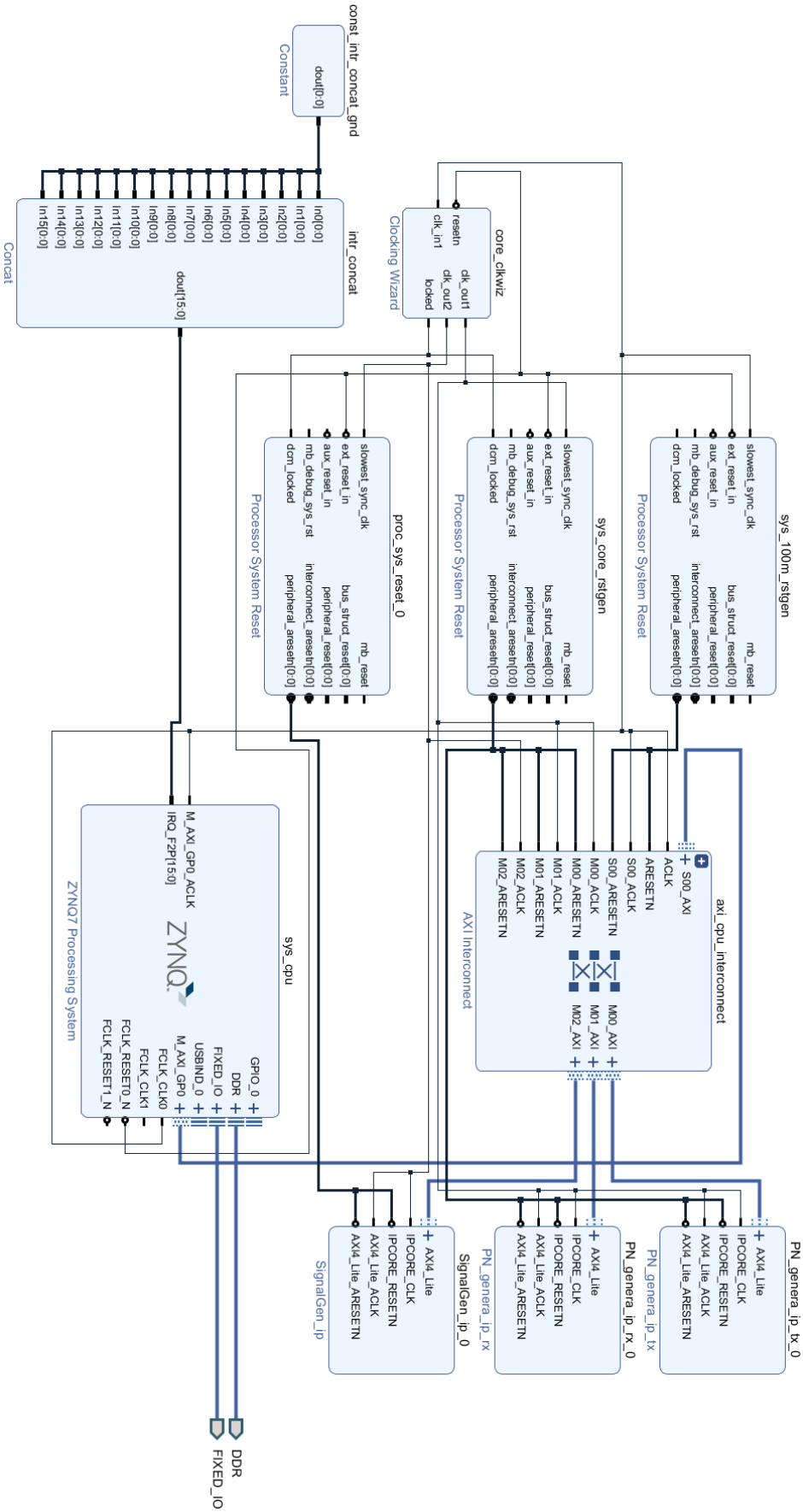


Ilustración 42. Diseño del sistema (HW)

El acceso a los dispositivos será a través de las direcciones de memoria asignadas a entrada y salida, y que en nuestro caso se muestran en la Ilustración 43.

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
sys_cpu					
Data (32 address bits : 0x40000000 [1G])					
PN_genera_ip_rx_0	AXI4_Lite	reg0	0x43C0_0000	64K	0x43C0_FFFF
PN_genera_ip_tx_0	AXI4_Lite	reg0	0x400D_0000	64K	0x400D_FFFF
SignalGen_ip_0	AXI4_Lite	reg0	0x43C1_0000	64K	0x43C1_FFFF

Ilustración 43. Mapa de memoria para el sistema

Estas direcciones de memoria permitirán el acceso a las entradas y salidas que hemos ido definiendo a la hora de generar cada elemento. Esto queda representado en la Ilustración 44.

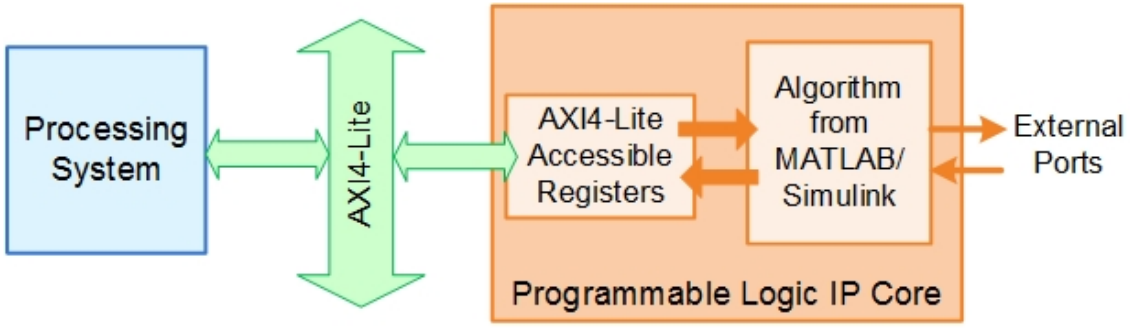


Ilustración 44. Interfaz de IP Core generado por Matlab. ©Mathworks.

El código que se ejecute en el sistema de proceso podrá interactuar con nuestros subsistemas usando el interfaz AXI en las áreas de memoria que hemos asignado en el mapa de memoria de la Ilustración 43, que es 64K para cada uno de ellos. Un valor muy elevado, teniendo en cuenta el número de parámetros de entrada y salida de cada uno de ellos. La intención es dejar espacio en caso de que tuviéramos que añadir parámetros adicionales más tarde. De esta manera se evitaría tener que desplazar a otros elementos dentro del mapa de memoria.

Con el mapa de memoria, y con los parámetros seleccionados de cada subsistema, podemos elaborar la siguiente tabla, que será muy útil para el acceso por software.

PN_genera_ip_tx		
I/O	Dirección	Longitud
Original_code	0x400D_0000+x100	bit
real	0x400D_0000+x104	sfix16_En14
imaginary	0x400D_0000+x108	sfix16_En14

Ilustración 45. I/O para el transmisor de secuencias de Gold

PN_genera_ip_rx		
I/O	Dirección	Longitud
Real	0x43C0_0000+x100	sfix16_En14
Imaginary	0x43C0_0000+x104	sfix16_En14
Salida_bits	0x43C0_0000+x108	bit

Ilustración 46. I/O para el receptor de secuencias de Gold.

SignalGen_ip		
I/O	Dirección	Longitud
Real	0x43C1_0000+x100	sfix16_En14
Imaginary	0x43C1_0000+x104	sfix16_En14

Ilustración 47. I/O para el transmisor de tonos

Tenemos que recordar que hemos generado los IP core en modo *free running*, lo que quiere decir que se ejecutan de manera asíncrona al sistema de proceso. De esta manera, cuando el sistema de proceso acceda al hardware, obtendrá siempre el valor más actualizado sin tener lugar ningún tipo de señalización entre las regiones.

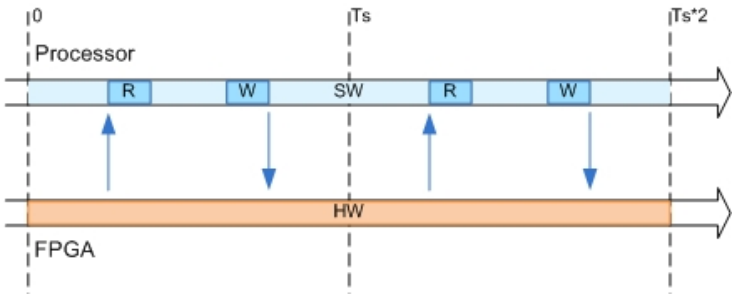


Ilustración 48. Modo free running. ©Mathworks.

Este diseño de hardware fue generado usando Vivado y desplegado como *bitstream* en la placa Zedboard.

Sistema de proceso y la integración de software

Parte de los elementos están generados en código C y compilados para ser ejecutados en los núcleos ARM del SoC Zynq. Las herramientas de desarrollo de software para este tipo de chips nos permiten varias opciones:

- Trabajar directamente sobre uno o ambos núcleos con una serie de bibliotecas de funciones llamada *Standalone*.
- Trabajar con un sistema Linux.
- Trabajar con el sistema en tiempo real FreeRTOS.

Existen otras opciones, pero no están soportadas directamente por Xilinx y esto conlleva el riesgo de usar herramientas o sistemas que no sean completamente compatibles con la versión que estamos usando.

Aunque desde una metodología más tradicional se suele comenzar usando un solo núcleo y sin un sistema operativo (*Bare Metal*) y tan solo usando las bibliotecas de *Standalone*, nosotros apostamos por el uso de un sistema Linux. La motivación fundamental se basa en la forma de generación de código de Matlab, donde tenemos diferentes módulos generados independientemente. Desde nuestro punto de vista, es más fácil ir generando binarios que sean desplegados en el sistema operativo. Además, podemos añadir diferentes herramientas de depuración que permitan monitorizar el comportamiento.

Para implementar un sistema Linux compatible con nuestro diseño de hardware, usaremos *Petalinux* [33], que no es más que la adaptación hecha por Xilinx del proyecto *Yocto* a sus dispositivos, y que permite generar sistemas Linux basándose en *recetas* de los elementos que queremos que formen parte del sistema operativo. Por ejemplo, si queremos que el sistema operativo tenga soporte para TCP/IP, podemos agregar esta receta. En caso contrario liberaremos espacio y tendremos menos procesos ejecutándose en el sistema resultante.

Petalinux nos permite importar un BSP de nuestra placa que contenga los elementos hardware no específicos del SoC (como DDR RAM, botones, LEDs o interruptores), e incluirlos por defecto en el sistema generado.

Un elemento crítico para la generación de un sistema operativo que sea capaz de interactuar con los dispositivos implementados en la lógica programable es el *Device Tree*, que es el que se encarga de transformar los valores del mapa de direcciones del hardware en otros valores, también direcciones de memoria, pero dentro del ámbito del sistema operativo. Además, el device tree permite al sistema operativo tener constancia de qué dispositivos están conectados y pueden controlados por él. A priori, esto podría no tener sentido, ya que las situaciones donde existen dispositivos físicamente conectados, pero no disponibles no son obvias. El caso más habitual es aquél donde usamos en paralelo dos sistemas operativos, uno en cada núcleo, o un sistema operativo en un núcleo y código ejecutándose en *bare metal* en el otro. De esta manera se pueden agrupar los dispositivos en dos sistemas operativos completamente diferentes, que tendrán cada uno su propio device tree donde se delimitará el acceso a la memoria para cada uno, y los dispositivos a los que puede acceder.

Cada uno de los subsistemas implementados en la lógica programable, con su interfaz AXI, tendrá un controlador (*driver*) implementado como módulo en el espacio del kernel de Linux (*kernel space*). Cada controlador accederá al *device tree* para obtener la dirección de memoria que necesita para las operaciones de entrada o salida. De esta manera, cualquier software que se esté ejecutando en el espacio de usuario puede hacer uso de estos subsistemas abstrayéndose del proceso de lectura de la memoria.

La aplicación que se ejecuta en el espacio de usuario puede contener todas las funciones de interfaz con el usuario y de comunicación con los conversores de datos, que en el caso de ADALM PLUTO usan un protocolo conocido como Linux Industrial I/O (*IIO*) [34], cosa que hace que no necesite de drivers específicos en

el espacio del kernel al usar directamente la biblioteca *LibIIO.* Incluso podría contener el código generado por Embedded Coder con los subsistemas generados en C, después de haber sido adaptados convenientemente. La estructura de la parte cubierta por el software del sistema se podría resumir en la Ilustración 49.

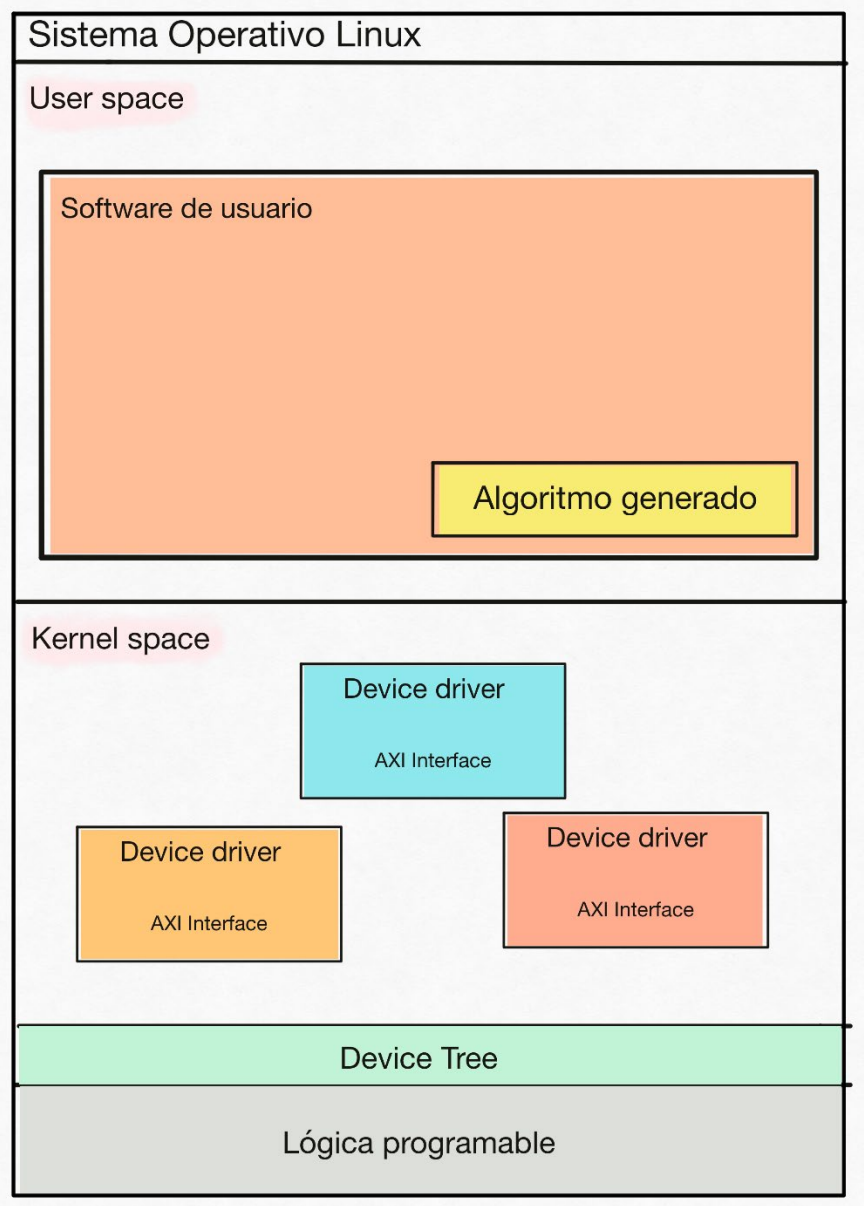


Ilustración 49. Sistema operativo.

3.6 Validación del sistema completo

El paso natural a seguir una vez que el sistema ha sido integrado es la validación. El uso de capacidad satelital para las pruebas de validación y verificación es algo completamente fuera de consideración para sistemas no certificados. En el momento de escribir este TFG, tan solo sería posible el uso del espacio asignado a radioaficionados en el satélite QO-100. Desgraciadamente el acceso no es

trivial, y se necesitaría una considerable inversión de tiempo y recursos para ensamblar los elementos necesarios para permitir a nuestro sistema enviar y recibir en las bandas asignadas. Como alternativa, se presentó un modelo de simulación de tipo *Flat Sat* (satélite plano), donde los subsistemas del satélite se despliegan horizontalmente en una mesa y se realizan pruebas con un simulador de canal. Un ejemplo de este procedimiento se presenta en [35].

Para nuestra validación decidimos usar otro par de dispositivos SDR, esta vez del tipo *hackrf* [36]. Este tipo de SDR permiten ser modificados para realizar barridos en frecuencia de manera muy rápida para tener un comportamiento similar a un analizador de espectro. Además, la señal de reloj puede conectarse a otro dispositivo de forma que es posible sincronizar el muestreo de ambos. Esta característica es importante ya que el diseño incluiría conectar dos hackrf, uno como receptor de la señal de U/L, procesar esa señal con el simulador de canal, y enviar la señal de D/L a través del otro dispositivo.

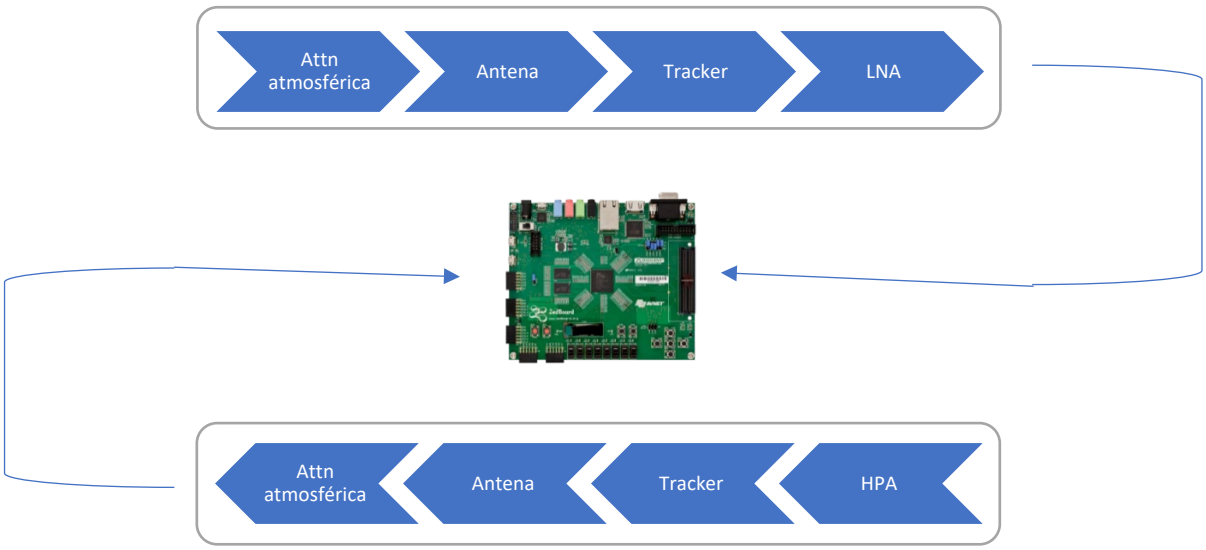


Ilustración 50. Proceso de validación

El procesado de señal se haría usando GNU Radio [22] y el módulo GR-LEO, que incluye la simulación de los siguientes elementos:

- Antenas de U/L y D/L con sus características de ruido.
- Apuntador (*tracker*) incluyendo precisión de apuntamiento configurable.
- Amplificadores LNA y HPA.
- Ruido del sistema.
- Atenuación atmosférica.

GR-LEO permite definir la órbita del satélite mediante sus *TLE* (*Two-Line-Elements*) y parámetros dependientes de la órbita, como la hora de pasada y el efecto Doppler, serán calculados y aplicados automáticamente.

El flujo de trabajo de GNU Radio y GR-LEO propuesto para este escenario es el que se puede ver en la Ilustración 51.

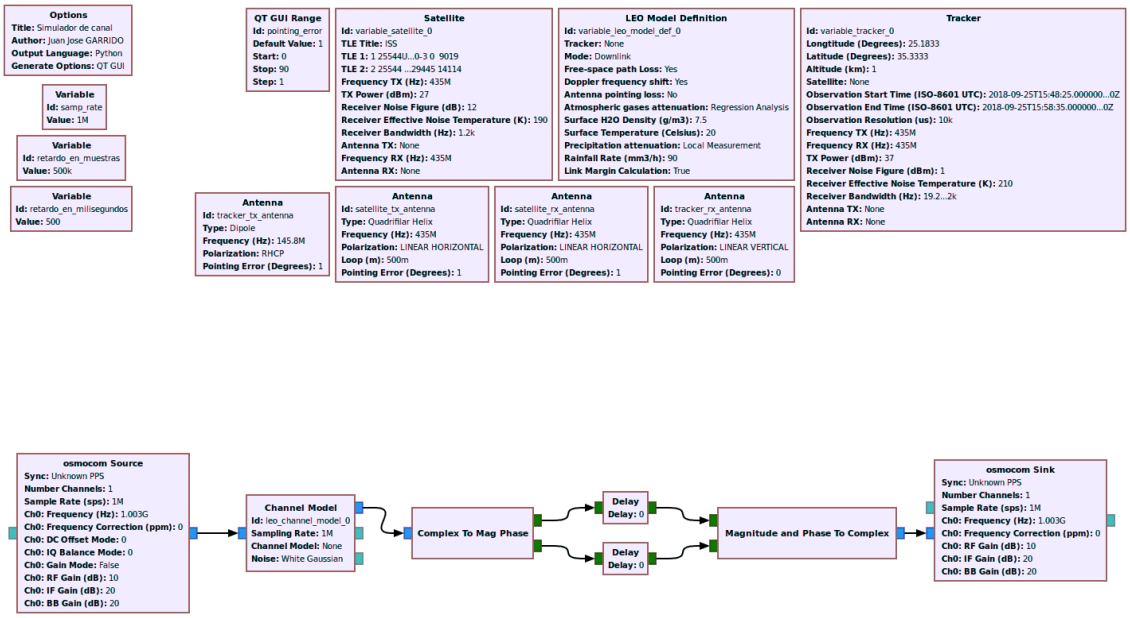


Ilustración 51. Flujo de GNU Radio para validación.

En el flujo se pueden apreciar los siguientes elementos:

- Una fuente de muestras del ADC (*osmocom*).
- Un modelo de canal.
- Un conversor de muestras complejas a magnitud y fase.
- Retardo variable.
- Un conversor de magnitud y fase a muestra compleja.
- Un destino de muestras DAC.

Los bloques osmocom servirán de interfaces a los dos dispositivos hackrf, mientras que el modelo de canal modificará la señal recibida según los parámetros que se presentan en las diferentes ventanas del diagrama. Tenemos que resaltar que GR-LEO no incorpora el retardo entre las características físicas simuladas, así que se ha añadido el retardo por defecto incorporado a GNU Radio. La frecuencia de muestreo de ambos conversores de datos se establece en 1 Msps.

La conexión física de los dispositivos es la que se muestra a continuación en la Ilustración 52.

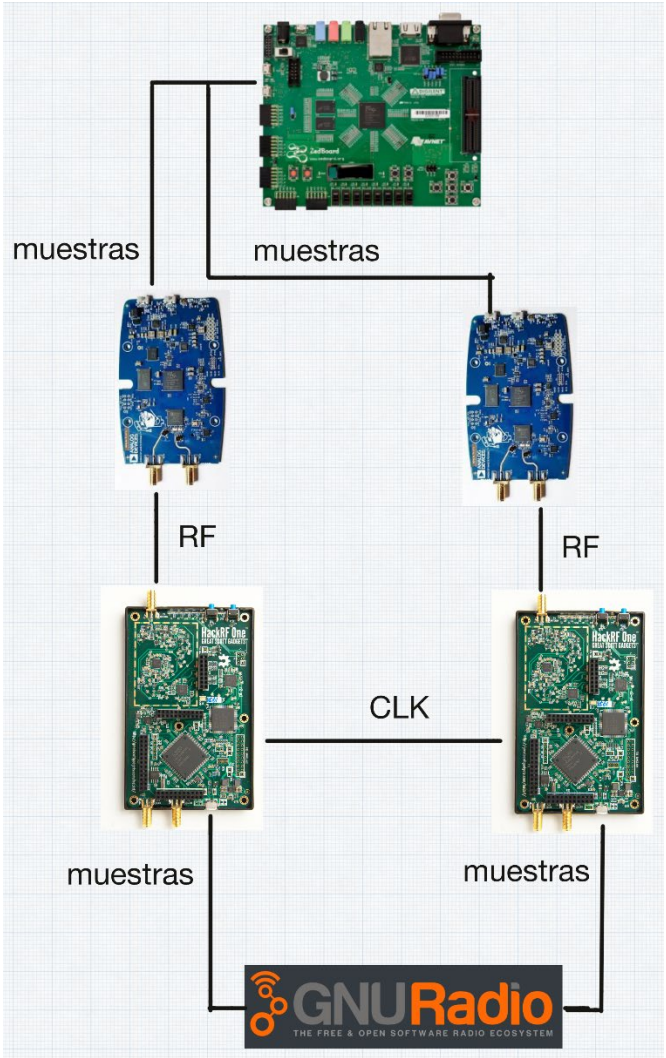


Ilustración 52. Conexiones para el proceso de validación.

4. Conclusiones

4.1 Objetivos y alcance del TFG

El objetivo general era el de poder definir una serie de algoritmos que puedan ser usados para el posicionamiento de satélites en órbita y su implementación. El alcance debía pues ajustarse al tiempo y recursos definidos por la universidad. Es por eso por lo que se optó por el uso de las herramientas de Matlab, incluyendo dispositivos de hardware soportados por el software, para realizar un prototipado rápido que nos permitiera iterar varias veces hasta dar con un resultado convincente.

El detalle de los modelos les permite ser razonablemente precisos, suficiente para demostrar la idea que los motivó, aunque necesitarían de cierto desarrollo adicional para poder ser certificados para su uso en misiones espaciales [37]. Especialmente son relevantes los volúmenes ECSS-E-ST-40C (*Software*) y ECSS-Q-ST-60-02C (*ASIC and FPGA development*).

Las herramientas de diseño y generación ofrecidas por Mathworks y Xilinx permitieron el prototipado de manera rápida y eficiente. Realizar cambios razonables en los modelos no fue problema, ya que generar y desplegar nuevas versiones no resultó complicado. Sí fue necesario invertir tiempo en comprender el funcionamiento de las innumerables opciones de las herramientas y bloques disponibles, ya que el entorno está muy lejos de la automatización real, donde el usuario sólo necesita pulsar un botón para realizar todo el proceso satisfactoriamente. De hecho, es absolutamente crítico desarrollar los modelos de Simulink teniendo presente la naturaleza del funcionamiento de un componente HDL, donde la información, siempre en formato de coma fija, se procesa muestra a muestra y está subordinada a diferentes tasas de muestreo. Menos rígidos, pero también con bastantes reglas, son los subsistemas que fueron diseñados para generar código C que sería ejecutado por los procesadores ARM.

Desgraciadamente, estas herramientas no fueron diseñadas para construir sistemas independientes y consolidados, por lo que una vez que el prototipado termina, es necesario realizar y consolidar todos los componentes de una manera manual.

4.2 Planificación del proyecto

Inicialmente se planificó una integración completa junto con una validación basada en el uso de dispositivos SDR adicionales. Desgraciadamente resultó ser una planificación excesivamente optimista, que no tuvo en cuenta el trabajo de integración de todos los elementos generados. Es quizá este punto el eslabón más débil de todo el proceso, ya que tenemos herramientas que simplifican la generación del código junto con el despliegue y ejecución en placas de desarrollo, pero el código (o los binarios) deben de ser integrados con otros componentes manualmente. Esto llevaría a realizar un trabajo tedioso que

incluiría realizar cambios a nivel del sistema operativo de la placa, en el código (incluyendo la realización de la compilación cruzada de nuevos elementos) y posiblemente el desarrollo de interfaces entre los procesos donde se ejecutan cada algoritmo y los conversores de datos externos (los SDR).

Al no llegar a concluir la integración en un sistema externo completo, la validación usando un canal simulado, mediante SDR adicionales y GNU Radio, no pudo realizarse.

Por otro lado, se pudieron generar código y archivos binarios válidos, de manera individual, siguiendo una metodología que nos permitía centrarnos en un aspecto concreto del proceso. Si bien para algoritmos no demasiado complejos pudiera parecer excesivamente largo, en un contexto donde los algoritmos implementados en Matlab son de un tamaño considerable, se unen los *bugs* de cualquier producto de software con los potenciales errores de diseño. En esta situación, sería posible añadir batería de pruebas en cada fase que asegurara la calidad del producto.

Dejando a un lado a los algoritmos y modelos de Simulink, el código y los binarios ya compilados (por compilación cruzada desde las herramientas de Matlab), incluyendo el *bitstream* con la implementación de la lógica programable, hubiera supuesto una inversión considerable del tiempo de programadores de C y VHDL. Esto no quiere decir que no sean necesarios, porque ya se ha mencionado que existe un considerable trabajo de integración, pero los desarrolladores partirán de un código existente generado a partir de ideas, requisitos y parámetros comunes definidos desde la abstracción que proporcionan las herramientas con las que se han hecho los diseños.

4.3 Dificultades técnicas

Al comenzar el TFG, al evaluar Matlab 2019b, encontramos un bug en el BSP de Xilinx que impactó en el funcionamiento de nuestra placa y en el ADALM PLUTO (que también está basado en Xilinx Zynq-7000) y que es reportado a Mathworks para ser corregido más tarde en la versión Matlab 2020a. Afortunadamente, esto no impidió trabajar en los algoritmos matemáticos ni en los primeros modelos de Simulink. La validez de la versión pudo ser comprobada con la versión beta, lo que permitió establecer Matlab 2020a como la versión de desarrollo del proyecto frente a Matlab 2019a, que era también válida pero que carecía de importantes funcionalidades relacionadas con Simulink, HDL Coder y otras herramientas que iban a ser usadas.

Durante la ejecución de la validación de los elementos relacionados con la variación de frecuencia, experimentamos uno de los problemas más relevantes del ADALM PLUTO: la estabilidad de su oscilador local. La solución era simplemente reemplazar el componente por un TCXO de calidad directamente en la placa.

Durante los meses que transcurrieron mientras se ejecutaba este TFG, una pandemia global del coronavirus COVID-19 afectó a todos los aspectos del funcionamiento normal de la sociedad, entre ellos la posibilidad de adquirir a tiempo los osciladores para los SDR. Además, obligó a replantearse la planificación ya que impactó de manera general al desarrollo de cualquier actividad durante este tiempo.

4.4 Posibles caminos para seguir a partir de los resultados obtenidos

A partir del punto donde dejamos este TFG, se abren dos caminos:

- Una integración completa en un sistema coherente.
- Un desarrollo más específico de los algoritmos.

El desarrollo de la implementación y validación demostró que la planificación fue excesivamente optimista al incluir la integración completa de los resultados de la generación en un sistema autónomo, cuando el trabajo de integración por si solo es un considerable esfuerzo.

Existe una tercera vía, quizá más exótica, pero que para sistemas complejos que son producidos de forma industrial a veces se realiza. Esta vía es la implementación de sistemas simulados. Los algoritmos de Matlab o los modelos de Simulink que no están orientados a la generación de código, pueden servir para crear simuladores, aunque la simulación de los algoritmos presentados en este TFG no presenta una especial utilidad.

De los dos caminos originales, quizá el más interesante sería el desarrollo específico, entendiendo como específico la implementación para un objetivo concreto. En nuestro caso sería la aplicación en un contexto y para un tipo de satélite concreto. Por ejemplo, se podría especializar el sistema en un tipo de satélite de comunicaciones gubernamentales en banda MIL-Ka, en una órbita geoestacionaria. De esta manera, el desvío de frecuencia de Doppler no sería nada crítico mientras que la monitorización del retardo o de su posición sería más relevante. Al trabajar en esa banda tan concreta, habría que determinar qué conversores en frecuencia harían falta y cómo afectaría al retardo, por poner un ejemplo.

La integración es un proyecto ambicioso, porque no sólo contemplaría lo descrito en 3.5, si no debería incluir una implementación completa de la Ilustración 31, junto con el equivalente en transmisión. Aun así, demostraría un sistema completo y autónomo de IOT.

5. Glosario

AIT/AIV: *Assembly, Integration and Test/Assembly, Integration and Verification*. Ensamblado, integración y prueba/validación. Es el proceso de integración de todos los módulos y subsistemas construidos por las diferentes empresas en el satélite, junto con los subsiguientes procesos de pruebas.

ASIC: *Application Specific Integrated Circuit* o circuito integrado para una aplicación específica. Al contrario que otro tipo de circuitos como procesadores o de lógica combinacional, que son de propósito general, estos dispositivos están diseñados para un tipo determinado de aplicación.

Bare Metal: Código que se ejecuta directamente en uno de los núcleos del procesador sin ningún tipo de sistema operativo.

D/L: *Downlink* o tramo entre el satélite y el receptor en un enlace.

FPGA: *Field Programmable Gate Array* es un tipo de circuito integrado que permite cierta configuración interna, de ahí que suela ser referido como un dispositivo de hardware programable.

IOT: *In-Orbit-Testing*. Proceso de pruebas en el satélite cuando está en órbita y está operativo.

SDR: *Software Defined Radio* o Radio Definida por Software. Es un tipo de radio muy reconfigurable, donde la mayoría del proceso es realizado mediante técnicas de DSP.

TCXO: *Temperature Compensated Crystal Oscillator*. Oscilador de cristal con compensación de temperatura. Este tipo de cristales son muy estables y sus características cambian muy poco frente a la temperatura.

TFG: Trabajo de fin de grado.

TT&C: *Telemetry, Tracking & Control*. Es el subsistema que existe tanto en la estación terrena como el vehículo espacial que es responsable de la supervisión y control remota del objeto en órbita.

TLE: *Two-Line-Elements* o elementos de dos líneas. Una forma de definir de manera inequívoca una órbita y el objeto que se mueve en ella.

U/L: *Uplink* o tramo entre el transmisor y el satélite en un enlace.

6. Referencias

[1] Librespace Foundation, «GR-LEO,» [En línea]. Available: https://gitlab.com/librespacefoundation/gr-leo/-/wikis/home. [Último acceso: 16 3 2020].

[2] Matworks, Analog Devices, «Radio Deployment on SoC Platform,» [En línea]. Available: https://www.youtube.com/watch?v=kL_SyVODxgw.

[3] G. Maral y M. Bousquet, Satellite Communications Systems (5th ed), Wiley, 2016.

[4] A. C. Clarke, «Extra Terrestrial Relays,» *Wireless World Magazine,* nº October, pp. 305-308, 1945.

[5] National Aeronautics and Space Administration (NASA), «Orbit Definition. Ancillary Description Writer's Guide.,» [En línea]. Available: https://web.archive.org/web/20131231000203/http://gcmd.nasa.gov/add/ancillaryguide/platforms/orbit.html. [Último acceso: 2013].

[6] S. B. (Intelsat), «Understanding High Throughput Satellite (HTS) Technology,» [En línea]. Available: http://www.intelsat.com/wp-content/uploads/2013/06/HTStechnology_bhartia.pdf. [Último acceso: 3 2020].

[7] P. . Angeletti, N. . Alagha y S. . D'Addio, «Space/ground beamforming techniques for satellite communications,» , 2010. [En línea]. Available: https://ieeexplore.ieee.org/document/5561170. [Último acceso: 15 3 2020].

[8] A. I. Perez-Neira, M. R. Campalans y N. Z. Barah, «Beamforming techniques for satellite communications bandwidth.,» , 2007. [En línea]. Available: https://patents.google.com/patent/es2332077b1/en. [Último acceso: 15 3 2020].

[9] A. F. Standing, Measurement Techniques for In-Orbit Testing of Satellites, W H Freeman & Co, 1990.

[10] K. . Rathnakara, D. . Ravindranath y M. Y. S. Prasad, «In-Orbit Testing of Satellite Communication Payloads,» *Iete Technical Review,* vol. 20, nº 5, pp. 447-462, 2003.

[11] M. T. Braun, Satellite Communications Payload and System, Wiley IEEE Press, 2012.

[12] CCSDS Standards, CCSDS 414.1-B-2 PSEUDO-NOISE (PN) Ranging Systems, CCSDS, 2014.

[13] J. E. Col Keese, «Satellite Telemetry, Tracking and Control Subsystems,» Boston, 2003.

[14] «ESA ARTES GHOST Program,» ESA, [En línea]. Available: https://artes.esa.int/projects/ghost.

[15] «MyHDL,» [En línea]. Available: http://www.myhdl.org/. [Último acceso: 2020].

[16] «OpenCL,» Kronos Group, [En línea]. Available: https://www.khronos.org/opencl/. [Último acceso: 16 03 2020].

[17] Xilinx Devices, «Xilinx Product Portfolio,» Xilinx, [En línea]. Available: https://www.xilinx.com/products/silicon-devices/soc.html. [Último acceso: 16 3 2020].

[18] Xilinx, «Zynq-7000 Product Page,» Xilinx, [En línea]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html. [Último acceso: 16 3 2020].

[19] AVNET, «Zedboard product page,» [En línea]. Available: http://zedboard.org/product/zedboard. [Último acceso: 16 3 2020].

[20] Analog Devices, «ADAML-PLUTO product page,» Analog Devices, [En línea]. Available: https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/adalm-pluto.html. [Último acceso: 16 3 2020].

[21] Analog Devices, «AD9363 Product Page,» Analog Devices, [En línea]. Available: https://www.analog.com/en/products/AD9363.html#product-overview. [Último acceso: 16 3 2020].

[22] GNU Radio, «GNU Radio,» [En línea]. Available: https://www.gnuradio.org/. [Último acceso: 16 3 2020].

[23] M. F. Atienza, «Movimiento relativo de emisor y observador. El efecto Doppler.,» de *Ondas. Introducción a los fenómenos ondulatorios*, Barcelona, Universitat Oberta de Catalunya, pp. 55-61.

[24] K. Dr. Carol F. Milazzo, «Doppler Frequency Shift Observation of the VO-52 Orbiting Satellite,» [En línea]. Available: https://www.qsl.net/kp4md/doppler.htm.

[25] F1TE, «Station QO-100 tous modes. Station mixte déportée à base de SDR ADALM-PLUTO.,» [En línea]. Available: https://www.f1te.org/index.php/satellite-qo-100/43-sation-sdr-pour-qo-100.

[26] L. F1TE, «Amélioration de la stabilité du SDR ADALM-PLUTO,» [En línea]. Available: https://www.f1te.org/index.php/realisations/sdr/adalm-pluto.

[27] Z. Xinyu, «Analysis of M-sequence and Gold-sequence in CDMA system,» de *DOI: 10.1109/ICCSN.2011.6014765*, Xi'an, 2011.

[28] The Consultative Committee for Space Data Systems (CCSDS), «RADIO FREQUENCY AND MODULATION SYSTEMS,» de *CCSDS 401.0-B-30*, 2020.

[29] D. B. J. B. P. B. E. G. M. J. M. T. M. B. .. P. P.G. Antreasian, «2001 MARS ODYSSEY ORBIT DETERMINATION DURING INTERPLANETARY CRUISE,» Jet Propulsion Laboratory, California Institute of Technology .

[30] European Space Agency, «Passive Correlation Ranging (PaCoRa),» ESA, 2013. [En línea]. Available: https://artes.esa.int/projects/passive-correlation-ranging-pacora.

[31] G. K. M. T. J. d. V. L. Rodriguez, «Passive Ranging for Geostationary Satellites: On a Novel System and Operational Benefits,» de *SpaceOps 2014 Conference*, Pasadena, 2014.

[32] U. Tamer, *Localization Using TDOA and FDOA,* Middle East Technical University, 2015.

[33] Xilinx, «Xilinx Petalinux tools,» [En línea]. Available: https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html.

[34] Analog Devices, «Linux Industrial I/O Subsystem,» [En línea]. Available: https://wiki.analog.com/software/linux/docs/iio/iio.

[35] L. A. Ziegler, «Configuration, manufacture, assembly, and integration of a university microsatellite,» Missouri S&T Library, Missouri, 2007.

[36] Great Scott Gadgets, «Hack RF One,» [En línea]. Available: https://greatscottgadgets.com/hackrf/one/.

[37] European Cooperation for Space Standarization, «ECCS Standards».

[38] The Consultative Committee for Space Data Systems (CCSDS), «CCSDS RECOMMENDED STANDARD FOR PSEUDO-NOISE RANGING SYSTEMS,» de *CCSDS 414.1-B-2* , 2014.

7. Anexos

7.1 Matlab: estimate_doppler.m

```
function [estimatedVel] = estimate_doppler(fs,fo)
%ESTIMATE_DOPPLER Cálculo de la velocidad usando el efecto Doppler
%   Cálculo de la velocidad de un transponder en órbita usando el efecto
%   Doppler.
c = 2.997e8; % Velocidad de la luz
estimatedVel = (c*fs)/fo - c; % Fórmula del efecto Doppler
end
```

7.2 Matlab: live script PN_Generator

Generamos la secuencia de Gold inicial (Tx).

```
goldseqA = comm.GoldSequence('FirstPolynomial','x^6+x^5+x^2+1',...
    'SecondPolynomial','x^6+x^5+x^4+x^3+x^2+1',...
    'FirstInitialConditions',[1 0 1 0 1 0],...
    'SecondInitialConditions',[0 1 0 1 0 1],...
    'Index',4,'SamplesPerFrame',56,...
    'OutputDataType', 'logical');
```

Generamos la segunda secuencia (Rx). Esta secuencia tiene los mismos parámetros que la primera.

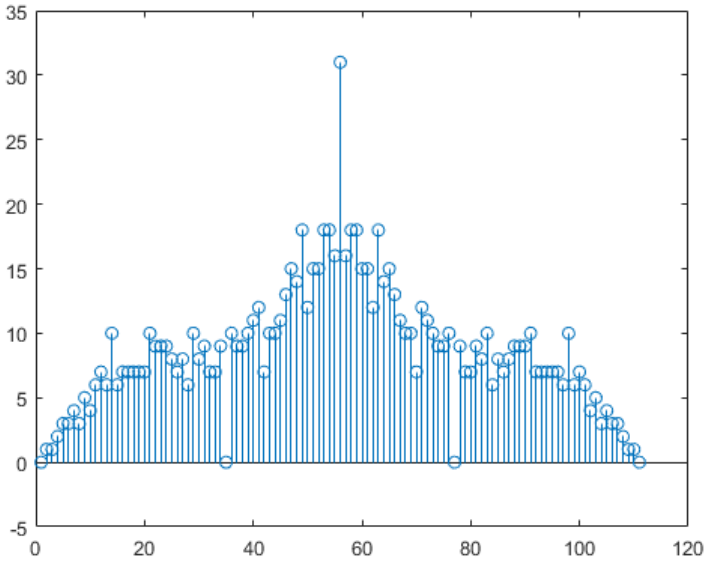
```
goldseqB = comm.GoldSequence('FirstPolynomial','x^6+x^5+x^2+1',...
    'SecondPolynomial','x^6+x^5+x^4+x^3+x^2+1',...
    'FirstInitialConditions',[1 0 1 0 1 0],...
    'SecondInitialConditions',[0 1 0 1 0 1],...
    'Index',4,'SamplesPerFrame',56,...
       'OutputDataType', 'logical');
```

Calculamos la correlación y generamos la gráfica

```
y = xcorr(goldseqA(),goldseqB())
```

```
y = 111×1
     0.0000
     1.0000
     1.0000
     2.0000
     3.0000
     3.0000
     4.0000
     3.0000
     5.0000
     4.0000
       ⋮
```

```
stem(y)
```

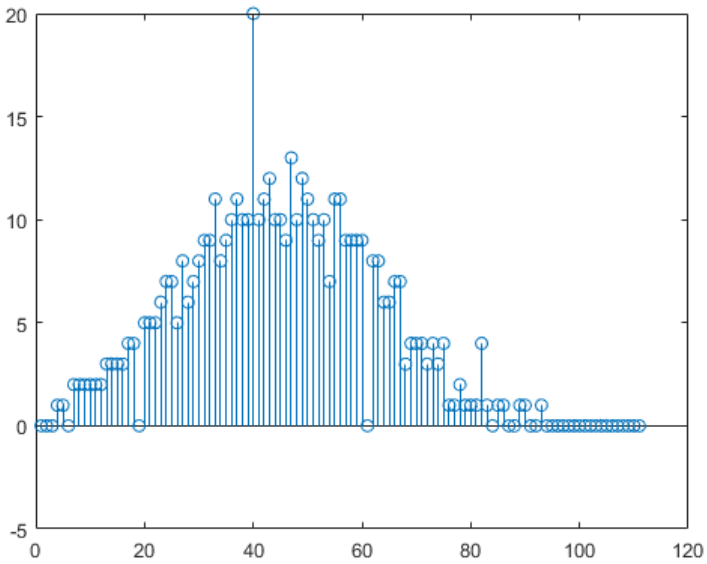



```
[maxVal,maxPos] = max(y)
```

```
maxVal = 31
maxPos = 56
```

Introducimos retardo en la señal.

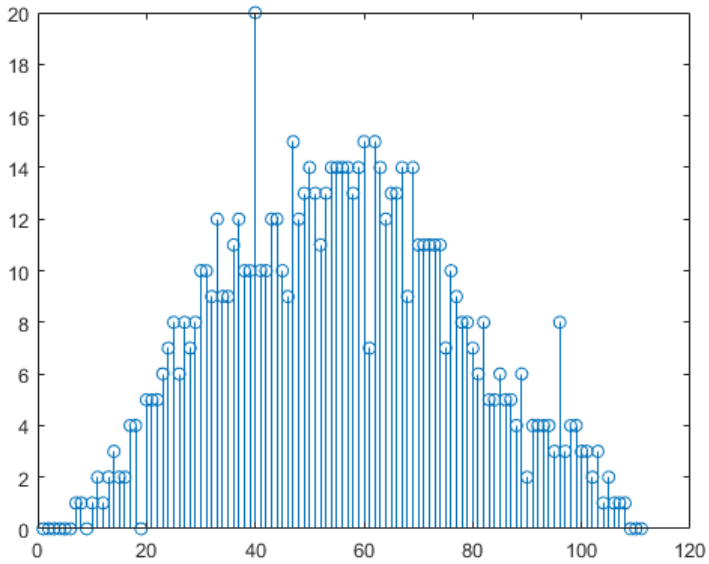
```
delay = dsp.Delay(16);
delayedGoldSeqB = delay(goldseqB());

y = xcorr(goldseqA(),delayedGoldSeqB);
stem(y)
```



```
[maxVal,maxPos] = max(y)
```

```
maxVal = 20
maxPos = 40
```

```
shiftedGoldSeqB = circshift(goldseqB(),16);
y = xcorr(goldseqA(),shiftedGoldSeqB);
stem(y)
```




```
[maxVal,maxPos] = max(y)
```

```
maxVal = 20
maxPos = 40
```

7.3 Matlab: TDOA simulación

```
% Parámetros de la simulación
range_s = 30; %Alcance máximo de las estaciones terrenas
range_T = 300; %Alcance máximo del satélite
c = 3e8;     %Constante que indica la velocidad de la luz
M = 8;       %Número de estaciones
E = linspace(30,0.1,20);
in_est_error = 20;  % Error inical estimado en metros
trials = 3;  % Número de experimentos
fail_thr = 1e3;    % Umbral que marca el éxito de la estimación
RMSE = zeros(length(E),1);
for m=1:length(E)
    rmse = zeros(trials,1);
    err_std = E(m)*1e-10; % Error de la medición de TDOA en segundos
    for k=1:trials % Bucle con el número de experimentos
        % Posición de las estaciones terrenas
        P = zeros(3,M);
        for ii=1:M
            P(:,ii)=range_s*2*(rand(3,1)-0.5);
        end

        p_T = range_T*2*(rand(3,1)-0.5);    % Vector de posción del satélite

        %Cálculo de TDOA (simulación de los tiempos de llegada a cada estación)
        dummy = repmat(p_T,1,M)-P;
        toa = zeros(M,1);
```

```
        for ii = 1:M % Cada estación recibe a un tiempo diferente
            toa(ii) = norm(dummy(:,ii))/c;
        end

        tdoa = toa-toa(1); tdoa(1)=[];
        tdoa = tdoa + err_std*randn(M-1,1);
        p_1 = P(:,1);
        dummy = P(:,2:M)';
        A = 2*[(p_1(1)-dummy(:,1)), (p_1(2)-dummy(:,2)), (p_1(3)-dummy(:,3)), -c*tdoa];
        b = (c*tdoa).^2 + norm(p_1)^2 - sum((dummy.^2),2);
        x_lin = pinv(A)*b;
        rmse(k) = norm(p_T-x_lin(1:3))^2;

    end
    fails = sum(rmse > fail_thr^2);
    RMSE(m) = sqrt(mean(rmse(rmse < fail_thr^2)));
end

figure
plot(E*1e-10,RMSE);
ylabel('RMSE (m)');
xlabel('\sigma_e (sec)')
figure
plot3(P(1,:), P(2,:),P(3,:),'o');

hold on;

plot3(p_T(1), p_T(2),p_T(3),'k*');
xlim([-range_T range_T]);ylim([-range_T range_T]);zlim([-range_T range_T]);
xlabel('x-axis'); ylabel('y-axis'); zlabel('z-axis');

plot3(x_lin(1), x_lin(2),x_lin(3),'md','MarkerSize',7.75);

legend('Estaciones terrenas', 'Posición del Satélite', 'Estimación')
grid on;
hold off;
```