TESIS DE GRADO EN INGENIERÍA ELECTRÓNICA

MÓDULO DE COMUNICACIONES DE UN NANOSATÉLITE

Integrantes:
Fernández Aguirre, Juan Pablo
Villa Fernández, Emanuel Ignacio

2017

Índice

Índice de figuras

Índice de cuadros

Nomenclatura

ADC	Analog to Digital Converter
AMSAT	Asociación Mundial de Satélites de Radioaficionados
Antipodales	Dos señales son antipodales cuando poseen la misma magnitud pero fase opuesta en todo momento.
BER	Tasa de error de bit.
CAN	Controller Area Network
Comisión de venta	Comisión pagada a la empresa electrónica responsable de la logística del pago.
DAC	Digital to Analog Converter
Downlink	En telecomunicaciones, es el enlace utilizado para transmitir señales desde el satélite hacia una estación terrestre.
GEO	Órbita de un satélite cuya orientación se mantiene fija con respecto a la superficie de la Tierra.
GPIO	General Purpose Input Output
HEO	High Earth Orbit. Órbita de un satélite que se encuentra a una distancia mayor de 36.000 Km de la Tierra.
IARU	Unión internacional de Radio Amateur
ISGEN	In-Situ Genetics Experiments on Nanosatellites. Experimentos Genéticos In-Situ en Nanosatélites.
ISIS	Innovative Solutions In Space
ISS	Estación Espacial Internacional (por su sigla en inglés International Space Station).
IVA	Impuesto al Valor Agregado
Latch-Up	El efecto de Latch-Up es un cortocircuito en un transistor producido generalmente por iones pesados o radiación cósmica.
LEO	Low Earth Orbit. Órbita de un satélite que se encuentra a una distancia entre 160 Km y 2.000 Km de la Tierra.
M-ASK	Modulación en amplitud. Utiliza diferentes amplitudes para representar los diferentess símbolos de la comunicación.
M-FSK	Modulación en frecuencia (Frequency Shift Keying por su sigla en inglés). Utiliza varias frecuencias para representar los diferentes símbolos de la comunicación.
MEO	Medium Earth Orbit. Órbita de un satélite que se encuentra a una distancia entre 2.000 Km y 36.000 Km de la Tierra.
MPSK	Modulación en fase (Phase Shift Keying por su sigla en inglés). Utiliza varias fases de la misma frecuencia para representar los diferentes símbolos de la comunicación.
NASA:	National Aeronautics and Space Administration. Administración Nacional de la Aeronaútica y del Espacio.
ONU	Organizaciones de Naciones Unidas
OOK	Modulación en amplitud que utiliza dos niveles para representar los símbolos de la comunicación.

Ortogonales	Dos señales son ortogonales cuando el producto interno entre las mismas es nulo.
PCB	Printed Circuit Boards
PWM	Pulse Width Modulation
QPSK	Modulación en fase utilizando cuatro fases ortogonales para maximizar la distancia entre los símbolos. Es un caso particular de M-PSK.
SMA	SubMiniature version A
SPI	Por su sigla en inglés Serial Peripheral Interface bus, es una interfaz de comunicación sincrónica serie utilizada principalmente en comunicaciones de corta distancia.
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
TIR	Tasa Interna de Retorno
TLV	Protocolo de comunicación que se compone de tres partes: tipo, longitud y valor.
UART	Universal Asynchronous Receiver Transmitter
UHF	Banda de frecuencias electromagnéticas comprendida entre las frecuencias de 300 MHz a 1000 MHz según norma IEEE.
Uplink	En telecomunicaciones, es el enlace utilizado para transmitir señales desde una estación terrestre hacia el satélite.
USB	Universal Serial Bus
VAN	Valor Actual Neto

Parte I
Introducción

1. Historia y antecedentes

Un satélite es cualquier objeto que órbita alrededor de otro objecto, denominado principal. Un satélite artificial es un vehículo espacial, tripulado o no, que se coloca en órbita alrededor de la Tierra u otro astro, y puede llevar elementos apropiados para recoger información y transmitirla[1].

Los satélites artificiales nacieron durante la guerra fría entre Estados Unidos y la Unión Soviética. Ambos países buscaban lanzar satélites artificiales y el viaje espacial a la Luna. El 29 de Julio de 1955, Estados Unidos anunció intentaría lanzar satélites a partir de la primavera de 1958. Esto se convirtió en el Proyecto Vanguard. El 31 de Julio, los soviéticos anunciaron que tenían intención de lanzar un satélite en el otoño de 1957. La Unión Soviética, desde el Cosmódromo de Baikonur, lanzó el primer satélite artificial de la humanidad el día 4 de Octubre de 1957. Este suceso marcó un antes y después en la carrera espacial, logrando que la Unión Soviética, se adelantara a Estados Unidos. Dicho programa fue llamado Sputnik[1][2].

Un hecho que no se debe pasar por alto es el año de fabricación del Sputnik. El mismo se lanzó en Octubre de 1957, mientras que Jack Kilby diseñó el primer circuito integrado en 1959. El mismo consistía en un oscilador formado por seis transistores. Es decir, la industria aeroespacial y la industria de la microelectrónica nacieron prácticamente en simultáneo. Sin embargo, si se establece una comparación entre el crecimiento de ambas industrias, es evidente que la industria aeroespacial no creció a la par de la industria de la microelectrónica. Solo cabe pensar cuánto se avanzó desde ese oscilador al nivel de integración actual. Muchos de los celulares que se utilizan actualmente tienen mayor capacidad de procesamiento que los satélites antiguos. Una de las barreras para el desarrollo es que dado el elevado costo de un satélite geoestacionario, se buscan utilizar componentes confiables y que hayan sidos utilizados previamente reduciendo así el grado de innovación en la industria y generando un freno en su desarrollo. Sin embargo, los nanosatélites buscan romper con dicha idea. El concepto detrás de este nuevo paradigma es producir satélites de pequeño tamaño, menor costo y que se puedan fabricar con componentes comerciales[3].

Ahora bien, teniendo en mente la necesidad que generó la creación de los nanosatélites a continuación se desarrollará la forma en las que se pueden clasificar los satélites con el fin de entender un poco más lo que es en sí un nanosatélite.

Los satélites se pueden clasificar de diversas formas, una de ellas es el uso esperado, otra de acuerdo a su órbita, y una tercera de acuerdo a su peso. Con respecto al uso, los satélites pueden destinarse a telecomunicaciones, meteorología, uso militar, observaciones a la Tierra, de uso para radio-aficionados, entre otros. Cada una de las clasificaciones anteriores requieren diversa complejidad a la hora del diseño y del validación del satélite.

En cuanto a la clasificación de acuerdo a su órbita, se clasifican en órbita baja terrestre (LEO, por sus siglas en ingles), órbita media terrestre (MEO), órbita alta terrestre (HEO), y finalmente los geoestacionario (GEO). Los satélites LEO orbitan a una distancia de 160 a 2.000 Km de la Tierra, y su velocidad les permite dar una vuelta al planeta en 90 minutos. Las condiciones a las que se exponen los satélites de este tipo son menos agresivas que la del resto de las clasificaciones dado que se encuentran protegidos por la magnetosfera terrestre. La desventaja que presentan es que su órbita no es constante. Por otra parte los satélites MEO orbitan a una distancia entre 2.000 y 36.000 Km. Su uso se destina a telefonía y televisión. En el caso de los satélites HEO orbitan a más de 36.000 Km, describen órbitas elípticas y se utilizan para cartografía. Finalmente los satélites GEO poseen una velocidad de traslación igual a la de la Tierra, por lo que se encuentran suspendidos en un mismo lugar con relación a la Tierra. Dicha velocidad es alcanzada a una altura de 35.800 Km sobre el Ecuador. La desventaja que poseen es que se encuentran en un ambiente más hostil dado la distancia con respecto de la Tierra[4].

Finalmente, la clasificación por peso divide a los satélites en satélites grandes, satélites medianos, satélites pequeños, minisatélites, microsatélites, nanosatélites, picosatélites y femtosatélites.

El término nanosatélite refiere a satélites cuyo peso se encuentra entra 1 y 10 kg. Los nanosatélites son el resultado de la evolución tecnológica actual, ya que aprovechan la ventaja de los notables avances en microelectrónica y además se fabrican empleando técnicas de bajo costo y producción masiva. El principio en el que se fundamentan es la miniaturización de componentes y sistemas, resignando la calidad en las prestaciones comparados contra los

[1] Definición de la Real Academia Española.

satélites geoestacionarios.

Debido a su reducido tamaño y potencia, las prestaciones de un nanosatélite aislado son modestas, siendo las más usuales las comunicaciones en diferido, la medición de parámetros ionosféricos o magnetoesféricos y la experimentación y demostración en órbita de nuevas tecnologías, componentes y dispositivos. Los nanosatélites son atractivos debido a su pequeño tamaño y su baja complejidad que hace que sean asequibles. Será en las constelaciones o enjambres con multitud de nanosatélites donde desarrollarán todo su potencial en el futuro, dando lugar a los sistemas distribuidos con posibilidades superiores, en algunos casos, a las grandes plataformas aisladas. Otra ventaja que poseen los nanosatélites es que se pueden aprovechar lanzamientos a la órbita baja que poseen otros propósitos como reabastecimiento de la ISS, evitando la necesidad de un lanzamiento exclusivo lo cual resulta más costoso. Desde una perspectiva militar, un nanosatélite puede ser útil ya que su pequeño tamaño también podría ayudar a evitar su detección[5].

Teniendo en cuenta lo explicado anteriormente, es importante entender que el fin de los nanosatélites es que sean de bajo costo. Por esto se busca que orbiten en la zona LEO, dado que presenta condiciones de ambiente menos dañinas, y se debe utilizar un número elevado de satélites para garantizar cobertura. De esta manera las condiciones ambientales externas no demandan un diseño exhaustivo en cuanto a los componentes utilizados.

Finalmente, y a modo de ejemplificar el alcance que pueden tener los nanosatélites, cabe mencionar el proyecto ISGEN que la NASA anunció en 2008. El proyecto busca investigar el crecimiento y las capacidades necesarias de cultivo para estudiar la expresión de genes y proteínas en microorganismos pequeños. El sistema es totalmente autónomo y autocontenido, su función es enviar los resultados telemétricos a la Tierra sin necesidad de que las muestras retornen. Los principales componentes del proyecto son los subsistemas de demostración tecnológica, que incluyen imágenes fluorescentes cuantitativas, redes microfluídicas, matrices de líquidos para el estudio repetido de construcciones genéticas múltiples y sistemas de control ambiental y de gestión de energía en miniatura. [6]

2. Justificación del proyecto

Este proyecto tiene como interés analizar la posibilidad de incorporarse a la industria aeroespacial ya que está sufriendo un cambio radical en su forma de desarrollo.

Si bien normalmente se suele pensar en satélites de gran tamaño que poseen grandes costos y extensos tiempos de desarrollo, actualmente se encuentra en auge la implementación de satélites miniatura de bajo costo. La causa de este fenómeno es que los grande satélites son una gran barrera de entrada a la industria espacial. En un futuro se espera poder implementar nanosatélites compuestos por componentes electrónicos comerciales por lo que es necesario simplificar, o encontrar formas alternativas y más económicas de poder llevar a cabo las funciones de un satélite convencional.

A diferencia de la Argentina, a nivel mundial el desarrollo en la industria espacial se encuentra en crecimiento en el ámbito privado. En los últimos años grandes empresas como Google empezaron a interesarse por esta nueva tecnología por su versatilidad y bajo costo. Sin embargo los precios resultan prohibitivos para los radioaficionados y pequeñas empresas que desean incursionar en este nuevo mundo.

En la figura 1 se muestra cómo varía año tras año el uso de nanosatélites. Mientras que en la figura 2 se muestran el tipo de usuarios de dichos nanosatélites. De ambas figuras se puede decir que año a año aumentan los nanosatélites y que si bien el mayor porcentaje se relaciona a empresas privadas, las universidades ocupan un rol fundamental en este mercado[2]. Agrupando a las universidades, institutos, organizaciones sin fines de lucro escuelas, independientes y usuarios individuales se obtiene un 43,5 % del mercado. Los gráficos hacen referencia a la actualización de Marzo de 2017[7].

[2]Las imágenes conservan el formato de origen.

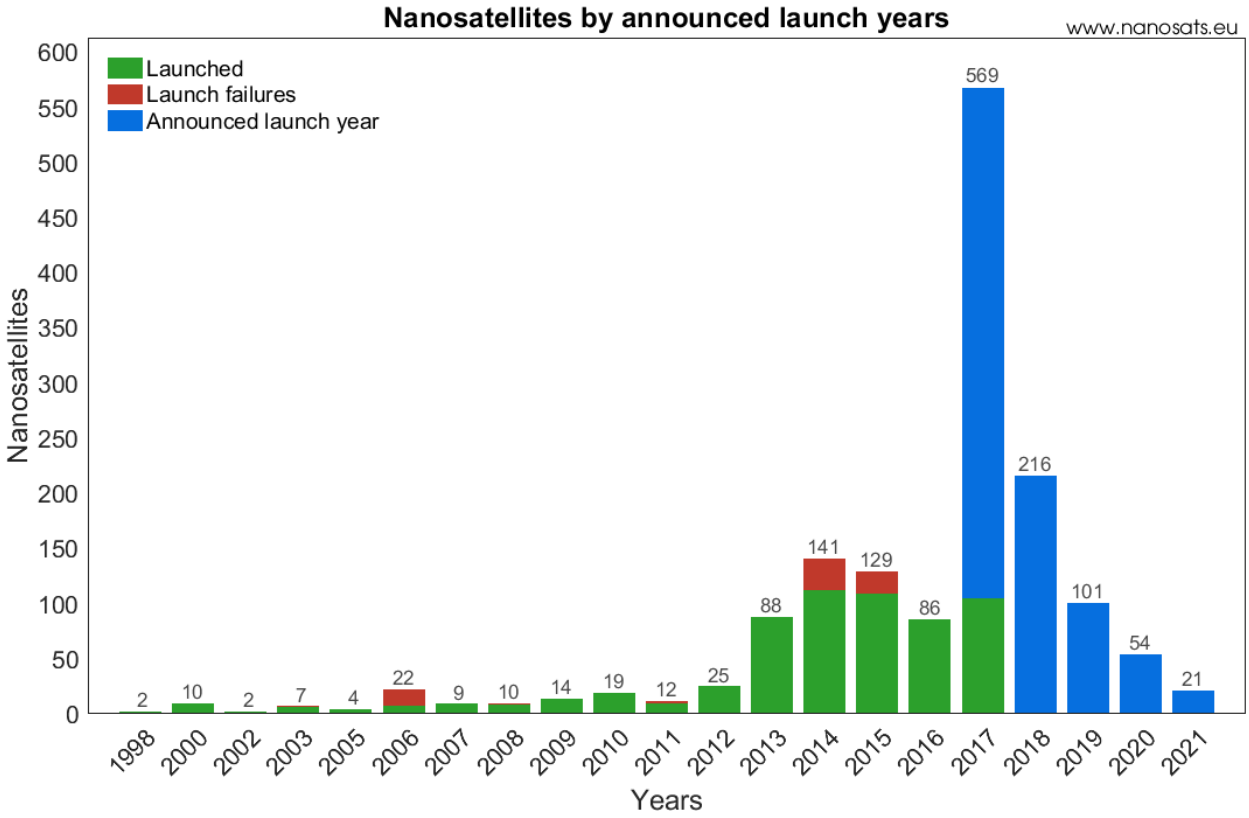


Figura 1: Nanosatélites puestos en órbita por año.

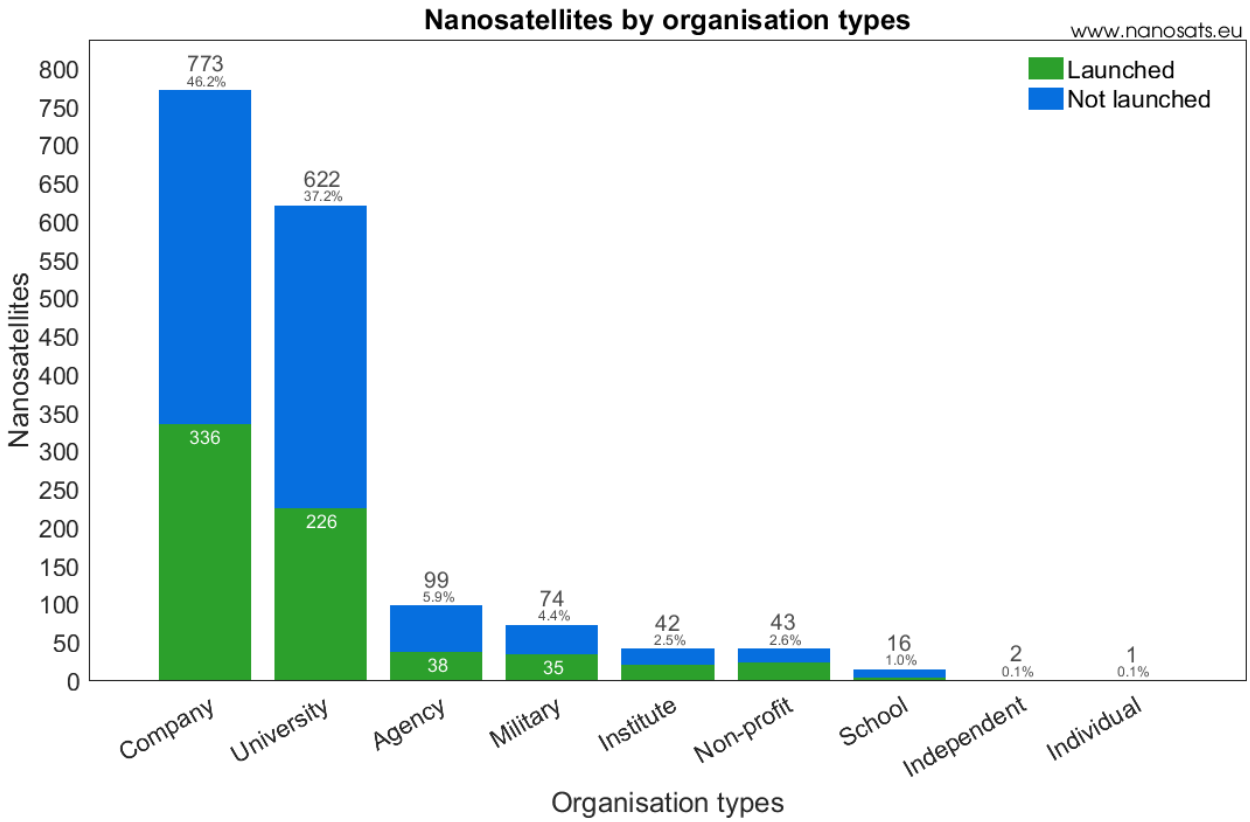


Figura 2: Usuarios de los nanosatélites.

Para poder mostrar la importancia que están ganando los nanosatélites a nivel mundial basta con retomar el caso de Google, que ya participa en la compañía SpaceX de Elon Musk, con una inversión de 1.000 millones de dólares. Uno de sus objetivos es proporcionar Internet a 3.000 millones de personas en países en vías de desarrollo y lugares remotos con una constelación de nanosatélites. Por su lado, Elon Musk se atreve a pronosticar que el coste de envío de nanosatélites será de 200 dólares por kilogramo enviado a la órbita baja. Actualmente el costo de envío por kilogramo a la órbita baja es de aproximadamente 22.000 dólares. Persiguiendo este objetivo, el cohete Falcon 9, también de SpaceX, demostró ser capaz de realizar un descenso exitoso tras haber llegado al espacio, lo que permitiría su reutilización y, por lo tanto, el abaratamiento de los futuros envíos.

Si nos centramos exclusivamente en la construcción de satélites, entonces se está produciendo un descenso de costes aún más marcado. En el 2004, los CubeSats podían ser construidos por un precio estimado de entre 65.000 y 80.000 dólares. Un CubeSat es un tipo de satélite en miniatura, utilizado para investigación espacial, que frecuentemente tiene un volumen de 1 litro y masa inferior a 1,33 kg[8].

Como conclusión se puede decir que cada vez se desarrollan más nanosatélites, siendo su desarrollo motivado por empresas privadas y universidades. A su vez, se busca disminuir su costo, tanto de diseño del prototipo como en el transporte del mismo. Si el objetivo es generar constelaciones entonces su puesta en órbita también debe abaratarse. Sin embargo, el costo sigue siendo prohibitivo para aficionados. Finalmente, para analizar el futuro de ésta tecnología se puede citar la estimación realizada por la consultora “Markets and Markets" en la cual advierte que el mercado de microsatélites y nanosatélites tendrá un valor de 3,49 miles de millones de dólares para 2022, siendo de 1,21 miles de millones de dólares en 2017[9].

Parte II
Objetivo

2.1. Finalidad del Proyecto

La finalidad de este proyecto es lograr introducirse en el mercado aeroespacial, acercándose a los pequeños desarrolladores y radioaficionados. El proyecto no busca un rendimiento económico en sí mismo sino que es la puerta de entrada la cual permitirá el desarrollo de nuevos productos y oportunidades económicas.

Para alcanzar dicha finalidad, se busca proveer a los usuarios no profesionales un sistema de comunicaciones en las bandas amateur y así poder introducirse en la industria aeroespacial. El sistema de comunicaciones se destina a nanosatélites que orbiten a una distancia máxima de 300km. El sistema de comunicación será full-duplex con un "Uplink" en la banda amateur UHF y "Downlink" en la banda amateur UHF. Este producto poseerá un precio accesible para los radioaficionados ya que busca generar el interés y permitir el desarrollo de una parte del mercado ignorada actualmente por la industria aeroespacial.

2.2. Planteamiento del Problema a Resolver

La industria aeroespacial está cambiando su metodología de trabajo migrando de grandes satélites a pequeños satélites que son más económicos en su desarrollo y en su puesta en marcha. Sin embargo, los precios de los nanosatélites actuales siguen siendo prohibitivos para los radioaficionados y pequeños emprendedores que buscan introducirse a la industria. La dificultad del proyecto radica en realizar un prototipo funcional de un sistema de comunicación a un costo reducido. En el proceso de desarrollo del prototipo funcional se incluirá la validación del prototipo y simulación del funcionamiento en condiciones normales para un nanosatélite.

Parte III
Definición de producto

3. Requerimientos

La implementación deberá cumplir con los requerimientos específicos sobre las condiciones bajo las cuales es sometido un nanosatélite que orbita en LEO. Algunas de dichas condiciones son la temperatura a la cual será expuesto y la radiación electromagnética; mientras que otros requerimientos surgen de la compatibilidad con algunas normas, como por ejemplo el tamaño y peso del sistema, ya que estos influyen en la complejidad y costo de la misión. Por último, el resto de los requerimientos provienen del cliente, los radio-aficionados que desean un producto de bajo costo para sus proyectos, que se comunique en bandas amateur y sea compatible con otros estándares, y de la comparación con otros sistemas de comunicaciones disponibles en el mercado.

ID	Descripción	Origen
REQ-01	Se deben utilizar componentes comerciales con el fin de garantizar un bajo costo.	Cliente
REQ-02	Debe tener comunicación full-duplex.	Cliente
REQ-03	El dispositivo debe ser capaz de recibir datos de otros módulos del nanosatélite y enviarlo a un receptor en la Tierra por medio de un enlace UHF.	Cliente
REQ-04	El módulo de comunicación del nanosatélite debe ser alimentado por el sistema que le provee energía a todo el satélite. Se debe de tratar de minimizar el consumo dado que se sólo se obtiene energía en los tramos iluminados de la órbita.	Cliente
REQ-05	El sistema de comunicación debe utilizar frecuencias de comunicación dentro de los rangos amateur.	Cliente
REQ-06	El sistema debe ser diseñado para una órbita LEO de hasta 300 km.	Cliente
REQ-07	Debe poseer dimensiones compatibles con la norma cubesat	Cliente
REQ-08	El módulo se debe apagar en caso de emergencia en cualquier momento en menos de 10 minutos de forma remota.	Requerimiento de la IARU
REQ-09	La comunicación debe ser abierta, no encriptada.	Requerimiento de la IARU
REQ-10	El peso máximo del nanosatélite debe ser menor que 1,33kg. El peso del módulo debe ser menor que la quinta parte.	Norma Cubesat
REQ-11	El producto debe operar en las condiciones normales de presión y temperatura dentro de un nanosatélite.	Tácito
REQ-12	Se debe garantizar ser inmune a la radiación electromagnética propia y ajena, como así también no influir en el comportamiento de otros módulos.	Tácito/Norma Cubesat
REQ-13	La comunicación debe funcionar sin importar la orientación del satélite.	Cliente
REQ-14	El sistema de comunicación debe monitorearla tensión de entrada y su temperatura, pudiendo identificar una falla y comunicársela al nanosatélite.	Cliente
REQ-15	El sistema de comunicación debe operar con una tensión de entrada entre 4.5V y 6V, con un consumo pico menor a 330 mA, y una corriente de inrush menor a 680 mA.	Mercado
REQ-16	El sistema de comunicaciones debe ser alimentado con los sistemas de alimentación típicos para nanosatélites.	Mercado
REQ-17	El sistema de comunicaciones debe poseer una interfaz de comunicación para enviar señales de control al satélite y permitir el intercambio de datos entre el satélite y el sistema de comunicación.	Mercado

Cuadro 1: Requerimientos.

A continuación se aclararán algunos requerimientos listados en la tabla anterior. En el requerimiento REQ-06 se va a tratar de ahora en mas como una órbita a 300 km. La órbita descripta por el nanosatélite es una órbita elíptica con diámetro máximo de 300km. Por este motivo, se considera una órbita de 300 km ya que contempla la peor condición para el enlace satelital. En el caso del REQ-10 el mismo se encuentra establecido en el estándar Cubesat, pero a su vez es especificado por la NASA[12].

En cuanto a los requerimientos REQ-11 y REQ-12 se tomarán como referencia el estándar general de verificación ambiental para proyectos y programas de vuelo propuesto por la NASA, y un estudio realizado por la universidad de Tokio[10].

Una aclaración a realizar en relación a los requerimientos surgidos de la Unión Internacional de Radio Amateur (IARU) es la diferencia entre una comunicación encriptada con una codificada (requerimiento REQ-09). La encriptación junto con la codificación se pueden pensar como técnicas parecidas, pero su objetivo es muy diferente. En el primer caso se busca ocultar información, haciendo entendible la misma para algunos receptores. En cambio, la codificación busca mejorar la comunicación de alguna forma, aprovechando alguna característica del canal o de la forma de comunicarse (probabilidad de error, ancho de banda, potencia de transmisión,etcétera).

La interfaz de comunicación mencionada en el requerimiento REQ-17 establece un medio de comunicación entre el sistema de comunicación y el resto del nanosatélite. Esta interfaz de comunicación debe permitir la transmisión hacia el resto del nanosatélite de los datos recibidos por el sistema de comunicación desde la Tierra y permitir la transmisión de datos desde el resto del nanosatélite hacia la Tierra. Debe ser una interfaz física de hardware que permita el intercambio de información entre el resto del nanosatélite y el sistema de comunicación.

4. Diagrama Funcional de Interfaces

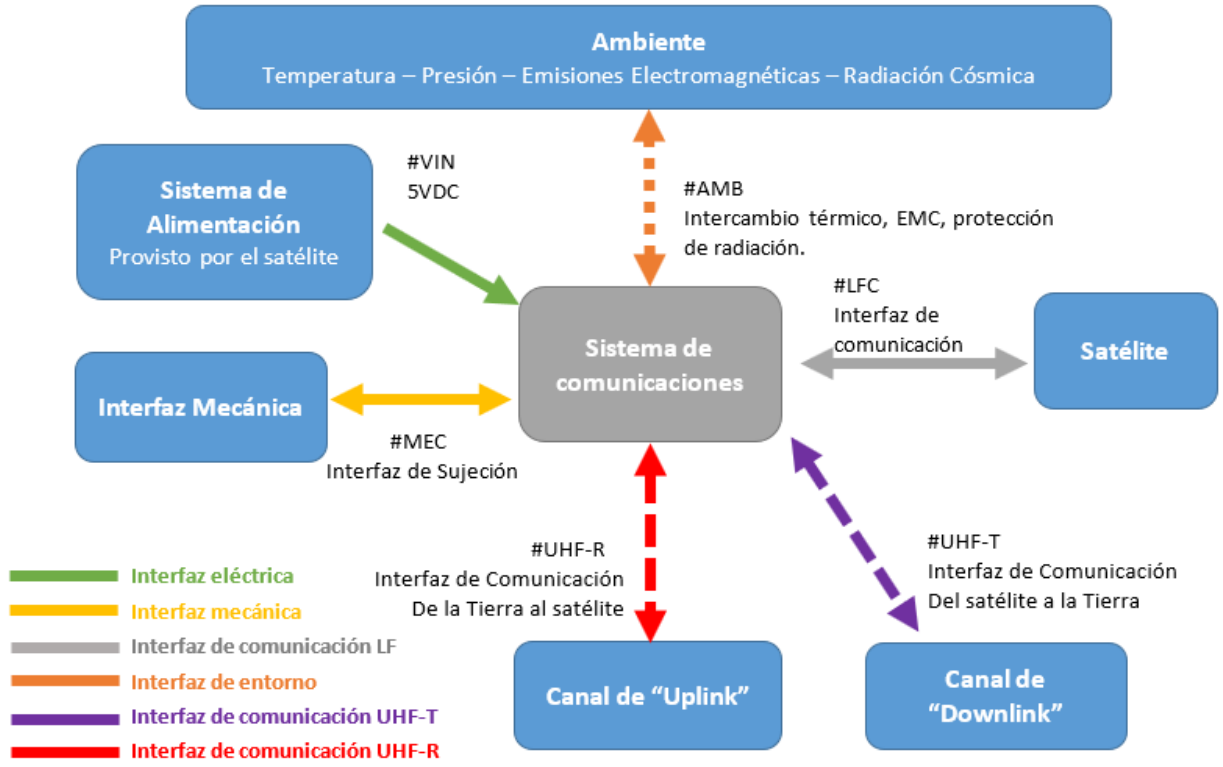


Figura 3: Diagrama funcional de las interfaces.

5. Especificaciones

Tomando en cuenta los requerimientos mencionados anteriormente, se establecieron las especificaciones técnicas que debe tener el producto.

ID	Descripción	Origen
FUN-01	Comunicación full-duplex.	REQ-02
FUN-02	Frecuencia del transmisor desde 430 Mhz a 450 Mhz.	REQ-05
FUN-03	Frecuencia del receptor desde 902 Mhz a 928 Mhz.	REQ-05
FUN-04	Potencia mínima del transmisor de 14 dBm.	REQ-06 y REQ-04
FUN-05	Velocidad de transferencia mayor a 110bps.	-
FUN-06	Sensibilidad del receptor menor a -122 dBm.	REQ-06
FUN-07	Apagado del transmisor de forma remota.	REQ-08
FUN-08	Comunicación no encriptada.	REQ-09
FUN-09	Potencia máxima disipada debe ser menor a 2W.	-
FUN-10	Re encendido del sistema de comunicación por medio de la comunicación con la Tierra dado que el receptor no se apaga remotamente.	-
FUN-11	Ganancia de antenas mayor a 2.5 dBi.	REQ-06
FUN-12	Antenas con patrón de radiación omnidireccional.	REQ -13
FUN-13	Potencia máxima de entrada de las antenas mayor a 30dBm.	REQ-06
FUN-14	Consumo en estado de recepción menor a 50 mA.	REQ-16
FUN-15	Consumo en estado de transmisión menor a 330 mA.	REQ-16
PER-01	BER menor a $10^{-4.}$	-

Cuadro 2: Especificaciones funcionales y de performance

Las especificaciones FUN-04, y FUN-06 se fijaron a partir de presuponer que se utilizará un sistema de comunicaciones similar al del nanosatélite para poder comunicarse con él. Para poder determinarlas se tomaron algunas consideraciones. Como punto de partida se tomó el peor caso de la órbita, es decir, 300 km y dado que los módulos comerciales poseen valores de sensibilidad dentro del rango propuesto se fijó como peor caso -122 dBm. Teniendo en cuenta dichas condiciones y realizando un cálculo básico de enlace se deduce que la potencia necesaria para asegurar un enlace debe ser de 14dBm. El mismo se puede observar en la sección 24. Lo que se debe tener en cuenta es que siempre se debe trabajar con la mínima potencia posible mientras que el enlace pueda ser establecido. Del mismo cálculo surge la especificación FUN-11.

Para la especificación FUN-05 se debe tener en cuenta dos factores. Por un lado el tiempo de enlace en cada transmisión, y por otro lado la cantidad de datos que se desea transmitir. Considerando la altura máxima de su órbita, el satélite se demora 90 minutos en completar una vuelta a la Tierra. Por cada contacto, el satélite

se mantendrá en contacto por un tiempo promedio de 5 minutos[18].En cuanto a la cantidad de datos que se transmitirán, en la figura 4 se muestran los sensores más comunes utilizados en nanosatélites[30]. Como consecuencia de dicha figura se puede decir que los sensores más utilizados son los sensores solares, magnetómetros y giroscopios. En cuanto a los sensores solares se utilizan para realizar un seguimiento al sol y determinar la altitud. Generalmente determinan el ángulo de incidencia del Sol. Hay de distintos precios, y el mismo depende de la precisión del sensor y el tipo de información que entregue el sensor. Los más completos pueden tener datos de 8 bytes, mientras que los menos completos pueden poseer datos de un byte. En cuanto a los magnetómetros también hay de diversos costos, precisión y cantidad de ejes. Los más complejos pueden tener un tamaño de dato de 10 bytes, mientras que otros poseen datos de 2 bytes. En algunos casos se pueden agregar lecturas de temperatura, radiación, hora y fecha entre otros.

Ahora bien, asumiendo que el diseño apunta a un mercado amateur se asume que se utilizarán dos sensores solares, dos magnetómetros, un sensor de temperatura, y para cada medición se almacenará la fecha y hora. Estimando 16 bytes para los sensores solares, 20 bytes para los magnetómetros, 1 byte para el sensor de temperatura y 18 bytes para la fecha se obtiene un paquetes de datos de 55 bytes [31].

Asumiendo que se toman muestras de datos cada 1,5 minutos, en 90 minutos se obtienen 3.300 bytes. Adicionando 50 bytes para iniciar la comunicación se obtiene un total de 3350 bytes, lo que implica una tasa de transferencia de 90 bps. Generando un margen de seguridad se decide poner un valor mínimo de 110 bps.

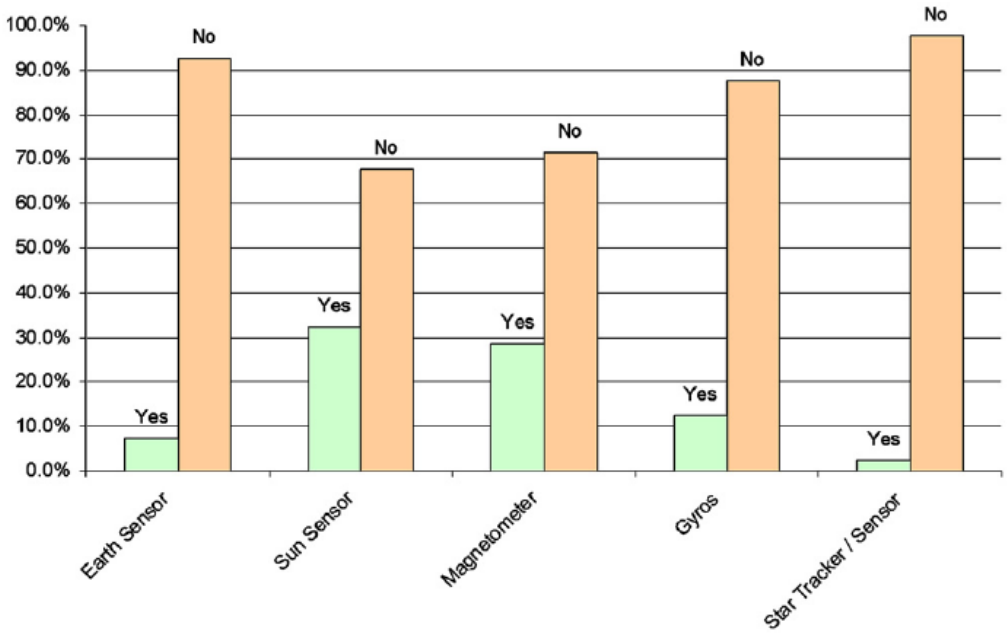


Figura 4: Tipos de sensores incorporados en los nanosatélites.

Por otro lado la especificación FUN-12 surge del desconocimiento de la posición final del nanosatélite en órbita, por lo que con el fin de no condicionar el diseño del resto del nanosatélite se fija dicha especificación.

La especificación FUN-13 se determina en base a dejar un margen de seguridad en relación a la potencia mínima necesaria.

Las especificaciones FUN-14 y FUN-15 surgen de analizar los sistemas de comunicación de la competencia y de la especificación de disipación de potencia[14].

ID	Descripción	Origen
INT-LFC-01	Interface de comunicación UART a una velocidad mayor a 120 kbps	REQ-03
INT-LFC-02	La conexión con el resto del nanosatélite debe realizarse con pines rectos simples de 0.5mm de diámetro y un separación de 1 mm. La alimentación debe encontrarse en un conector separado conformado por dos pines, mientras que en un segundo conector de 2x8 pines se utilizará para el resto de las señales de interacción necesarias.	REQ-17
INT-LFC-03	La tensión máxima de entrada debe ser menor a 3.6V y la tensión mínima mayor a -0.3V.	REQ-17

Cuadro 3: Especificaciones con la interfaz de baja frecuencia.

ID	Descripción	Origen
CONF-01	MTTF mayor a dos años.	REQ-06
CONF-02	Vida útil mayor a un año.	REQ-06

Cuadro 4: Especificaciones de confiabilidad.

Ambas especificaciones de confiabilidad surgen de la comparación de otros diseños. A modo de ejemplo se toma la hoja de datos de la estructura del CubesatKit, en la que se indica que la vida útil mínima es de 1 año[11].Sin embargo, en las figuras de la sección 25 se observa que la vida útil de los nanosatélites de carácter amateur suele ser menor de 1 año, por lo que se establece como especificación una vida útil mayor a un año. Se deberá cumplir con un MTTF mayor a dos años para reducir en un 23 % la probabilidad de falla en el primer año. Además, de acuerdo una investigación llevada acabo en la universidad de Florida, la vida orbital máxima de un nanosatélite a 350km es siempre menor a 400 días debido al fenómeno físico de arrastre.[37]

ID	Descripción	Origen
INT-AMB-01	Temperatura durante el despegue entre -40C a 80C	REQ-11
INT-AMB-02	Temperatura durante la órbita entre -40C a 50C	REQ-11
INT-AMB-03	El sistema debe ser capaz de funcionar en condiciones de vacío.	REQ-11
INT-AMB-04	El diseño deberá soportar los test propuestos por el estandar de verificación ambientalpara proyectos y programas de vuelo (GSFC-STD-7000A)	REQ-11 y REQ-12
INT-EMC-01	El diseño debe utilizar una implementación de Stackup para reducir las emisiones electromagnéticas	REQ-12

Cuadro 5: Especificaciones de la interfaz ambiental.

Las especificaciones INT-AMB-01, INT-AMB-02 y INT-AMB-03 surgen de un estudio realizado por la universidad de Tokio[10].

ID	Descripción	Origen
INT-MEC-01	Agarre garantizado por 4 perforaciones cuyo diámetro se encuentre acotado entre 3.7mm y 3.2mm, ubicados en las esquinas de la placa. El centro de las perforaciones se deben encontrar a una separación del borde de 5mm.	REQ-07
INT-MEC-02	Se ensamblará al resto del nanosatélite utilizando los tornillos citados en la especificación PC/104-Plus	REQ-07
INT-MEC-03	Diseño mecánico tolerante a vibraciones del despegue de un cohete espacial.	REQ-07

Cuadro 6: Especificaciones de la interfaz mecánica.

ID	Descripción	Origen
DYP-01	El sistema de comunicación debe pesar menos de 250g.	REQ-10
DYP-02	El sistema debe poseer las siguientes medidas: Eje X < 100 mm. Eje Y< 100 mm Eje Z < 30mm.	REQ-07

Cuadro 7: Especificaciones de dimensión y peso.

ID	Descripción	Origen
MAN-01	El sistema de comunicación debe monitorear la tensión de entrada.	REQ-14
MAN-02	El sistema de comunicación debe monitorear su temperatura.	REQ-14
MAN-03	El sistema de comunicación debe clasificar una falla en falla de alimentación, temperatura, transmisor o receptor.	REQ-14
MAN-04	El sistema de comunicación debe poseer un sistema de reinicio para intentar corregir una falla.	REQ-14
MAN-05	El sistema de comunicación debe comunicarle al satélite si posee una falla.	REQ-14

Cuadro 8: Especificaciones de mantenibilidad.

Las especificaciones de mantenibilidad surgen de garantizar un comportamiento adecuado del sistema de comunicación ante la detección de una posible falla.

ID	Descripción	Origen
ECO-01	El precio del módulo debe ser menor a 500 dólares	REQ-01

Cuadro 9: Especificaciones económicas.

ID	Descripción	Origen
INT-VIN-01	El módulo deberá operar con una tensión entrada entre 4.5V y 6 V con un rechazo al ripple típico de 60dB.	REQ-15
INT-VIN-02	El consumo pico del sistema de comunicaciones debe ser menor a 330 mA.	REQ-15
INT-VIN-03	El sistema debe poseer protecciones de ESD de al menos 2kV.	REQ-01
INT-VIN-04	El sistema de comunicaciones debe poseer protecciones de cortos o sobrecalientamientos.	REQ-14
INT-VIN-05	El sistema de comunicaciones debe poseer un valor pico de corriente de inrush de 680 mA.	REQ-15

Cuadro 10: Especificaciones de alimentación.

En la especificación INT-VIN-01 se fija esa tensión dado que se considera conveniente desde el punto de vista de compatibilidad con componentes comerciales. La especificación INT-VIN-02 surge de la limitación de potencia, mientras que la especificación INT-VIN-03 surge de analizar sistemas de comunicación de la competencia. En cuanto a la especificación INT-VIN-05 surge de analizar la corriente inrush típica de sensores del mercado. En este caso se toma como referencia el sensor solar nanoSSOC-D60 de SolarMems.

6. Despliegue de la Función de Calidad

Se realizó el despliegue de la función de calidad para encontrar los parámetros centrales sobre los cuales diseñar el producto y cuales deberían ser optimizados para tener una mejor apreciación en el mercado. Como el mercado al que se apunta es un mercado amateur, el precio resulta un factor determinante en cuanto a la aceptación del producto, como así también la capacidad de poder soportar las condiciones ambientales a las que será expuesto el sistema y la fácil integración con el resto del satélite. La velocidad de transmisión y la vida útil en cambio no son parámetros decisivos a la hora del desarrollo dado que los nanosatélites amateur suelen tener pocos sensores y no orbitan mucho tiempo. Para el despliegue de la función de calidad se tomó información de los posibles competidores como *ISIS [20]* y *Clyde-Space*[21].

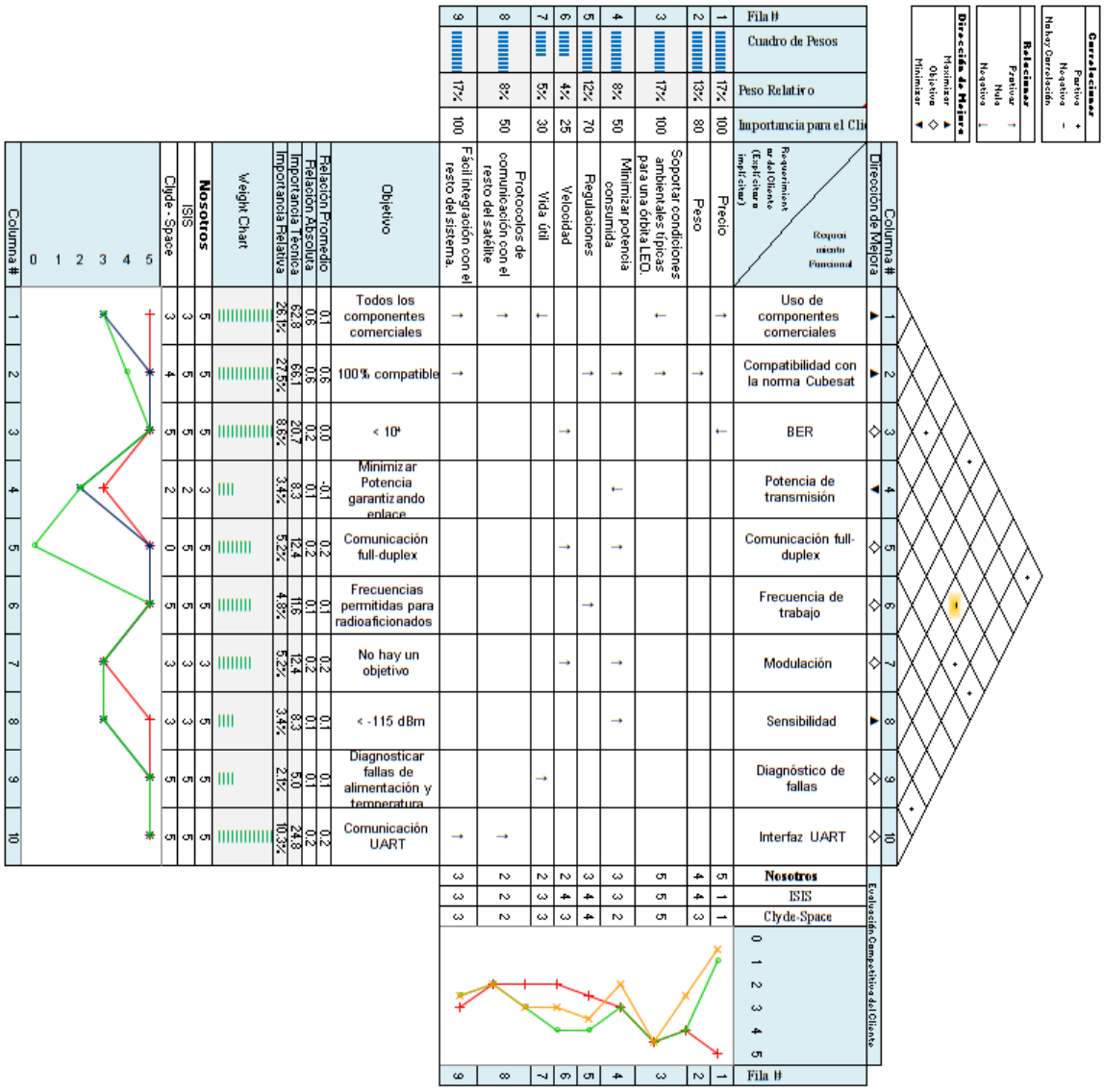


Figura 5: Despliegue de la función de calidad.

Parte IV
Análisis de Factibilidad

7. Factibilidad Temporal

En las siguientes secciones se desarrollará la factibilidad temporal del proyecto, realizando un análisis de planificación (diagrama de PERT) y un análisis de Programación (diagrama de Gantt).

7.1. Planificación

En la tabla 11 se presentan las actividades previstas del proyecto. La duración se indica en días. Se considera una jornada de trabajo de 8hs de Lunes a Viernes.

	Actividad	Tiempo Optimista	Tiempo Más Probable	Tiempo Pesimista	Inmediata Predecesora
A	Investigación Inicial y Análisis de Mercado	26	31	36	-
B	Elección del Sistema de Modulación	12	20	22	A
C	Diseño e Implementación del Modelo de Condiciones	18	20	22	A
D	Simulación de las Condiciones	10	15	20	C
E	Diseño del Modelo de Comunicación	18	25	26	D,B
F	Simulación de Comunicación	12	26	28	E
G	Investigación sobre la Codificación	7	13	19	E
H	Diseño global Terminado	20	22	24	F,G
I	Diseño de Hardware	22	25	34	H
J	Diseño de Software	18	20	22	H
K	Diseño de Pruebas de Validación	15	20	25	I,J
L	Elección del Módulo Transreceptor	13	15	17	I
M	Diseño de la Codificación	14	13	15	H
N	Ensamblado de Hardware	20	25	30	L
O	Testeo de Hardware	16	19	22	N
P	Confección de Software	16	17	24	M
Q	Testeo de Software	21	23	25	P
R	Prototipo del Sistema Transreceptor	30	45	60	O,Q
S	Medición del Prototipo	22	33	44	R
T	Medición del Prototipo con Codificación	6	8	10	S
U	Pruebas de Validación sobre el Diseño	20	25	30	K,T
V	Recopilación Final de Documentación	13	15	17	U

Cuadro 11: Actividades con indicación de precedencia.

7.1.1. Descripción de las Actividades

A continuación se describen las tareas con más detalle:

- Investigación Inicial y Análisis de Mercado: Investigación sobre las diferentes topologías e implementaciones de nanosatélites. Tiene por objetivo delimitar los requerimientos del proyecto así como definir parámetros de trabajo y convenciones usualmente utilizadas. Además se analizan la existencia de productos similares, observando la tendencia de los compradores y abriendo el diseño a todo aquello que pueda resultar interesante para el mercado.
- Elección del Sistema de Modulación: Se define la técnica de modulación con la cual se transmitirá información .

- Diseño e Implementación del Modelo de Condiciones: Se diseña el modelo de condiciones físicas del medio que utilizará el nanosatélite con el fin de utilizarlo en la simulación.
- Simulación de Condiciones: Se simulan las condiciones físicas bajo las cuales trabajará el sistema, utilizando el modelo anterior, con el fin de obtener los parámetros característicos del mismo para poder validar el futuro diseño.
- Diseño del Modelo de Comunicación: Se genera un modelo a utilizar en la simulación de la transmisión y recepción con el fin de condensar todas las características físicas del enlace.
- Simulación de Comunicación: Se simula el sistema de transmisión teórico, tanto receptor como transmisor, considerando ruido en el canal, pérdida de desacople, probabilidad de error de bit, y otros parámetros de la comunicación que resultan relevantes.
- Investigación sobre la Codificación: En esta etapa se investiga sobre diferentes métodos de conversión de la información para proveer mayor robustez a la comunicación detectando y corrigiendo errores producidos en el canal.
- Diseño Global Terminado: En dicha instancia, se espera tener un diseño funcional y global concreto.
- Diseño de Hardware : Se busca pasar del diseño funcional a un diseño más particular en donde se empieza a definir los componentes de Hardware.
- Diseño de Software: Al igual al caso anterior, se busca pasar de un diseño funcional y global a un diseño específico de Software.
- Diseño de Pruebas de Validación: Se diseñan pruebas equivalentes a las condiciones bajo las cuales trabajará el sistema.
- Elección del Módulo Transreceptor: A partir de las simulaciones, se establecen los parámetros relevantes y se busca un módulo de transmisión comercial que permita cumplir con los requerimientos, o en caso de no encontrarlo se establece el diseño de uno que cumpla con lo requerido.
- Diseño de la Codificación: En base a lo investigado, se desarrolla la codificación óptima para el sistema.
- Ensamblado de Hardware: En esta actividad prepara el prototipo a nivel Hardware. La actividad abarca desde la adquisición del Hardware hasta su ensamblado.
- Testeo de Hardware: Verificación del correcto funcionamiento del Hardware revisando que no haya malos contactos, que los módulos estén correctamente alimentados y no se produzca ninguna falla de conexión.
- Confección de Software: Se busca conseguir la implementación final del Software utilizando el diseño realizado previamente.
- Testeo de Software: Testeo individual del código.
- Prototipo del Sistema Transreceptor: Se desarrolla un prototipo del sistema de transmisión para probar su desempeño, observando su rango dinámico, sensibilidad y potencia máxima de salida.
- Medición del Prototipo: En base al prototipo se miden todos los parámetros de interés, entre los cuales se encuentran su sensibilidad, potencia máxima, desvío de la frecuencia de trabajo, ancho de banda ocupado, velocidad de transmisión.
- Medición del Prototipo con Codificación: En base al prototipo, se le agrega la codificación y se vuelven a analizar todos los parámetros de interés observando su variación y mejora.
- Pruebas de Validación sobre el Diseño: Se utilizan las pruebas diseñadas en la etapa anterior para analizar la performance del sistema bajo condiciones que representen un situación real de trabajo.
- Recopilación Final de Documentación : En éste período se le da un cierre a la documentación, revisando que esté todo en orden y cerrando algo que haya quedado pendiente. Es por eso que se realiza una vez finalizado el prototipo definitivo. Esto no quiere decir que en éste período se debe comenzar la documentación, la documentación de un proyecto es una actividad constante y simultánea con desarrollo del mismo; es por eso que no se le asigna un comienzo como cualquier otra actividad.

7.1.2. Diagrama de Camino Crítico

A partir de los valores de la tabla 11, se pueden estimar los tiempos esperados y la varianza para cada actividad. Para realizar dicha estimación se utilizan las ecuaciones 1 y 2, mostrando los resultados en la tabla 12.

$$Tiempo\ Estimado = \frac{Tiempo\ Optimista + 4.Tiempo\ Más\ Probable + Tiempo\ Pesimista}{6} \tag{1}$$

$$\sigma^2 = (\frac{Tiempo\ Pesimista - Tiempo\ Optimista}{6})^2 \tag{2}$$

	Tiempo Esperado	Varianza
A	31	3
B	19	3
C	20	1
D	15	3
E	24	2
F	24	8
G	13	4
H	22	1
I	26	4
J	20	1
K	20	3
L	15	1
M	13	1
N	25	3
O	19	1
P	18	2
Q	23	1
R	45	25
S	33	14
T	8	1
U	25	3
V	15	1

Cuadro 12: Actividades con Tiempos Esperados y Varianza.

Dada toda la información anterior se puede elaborar el siguiente esquema de camino crítico. El mismo se muestra en la figura 6.

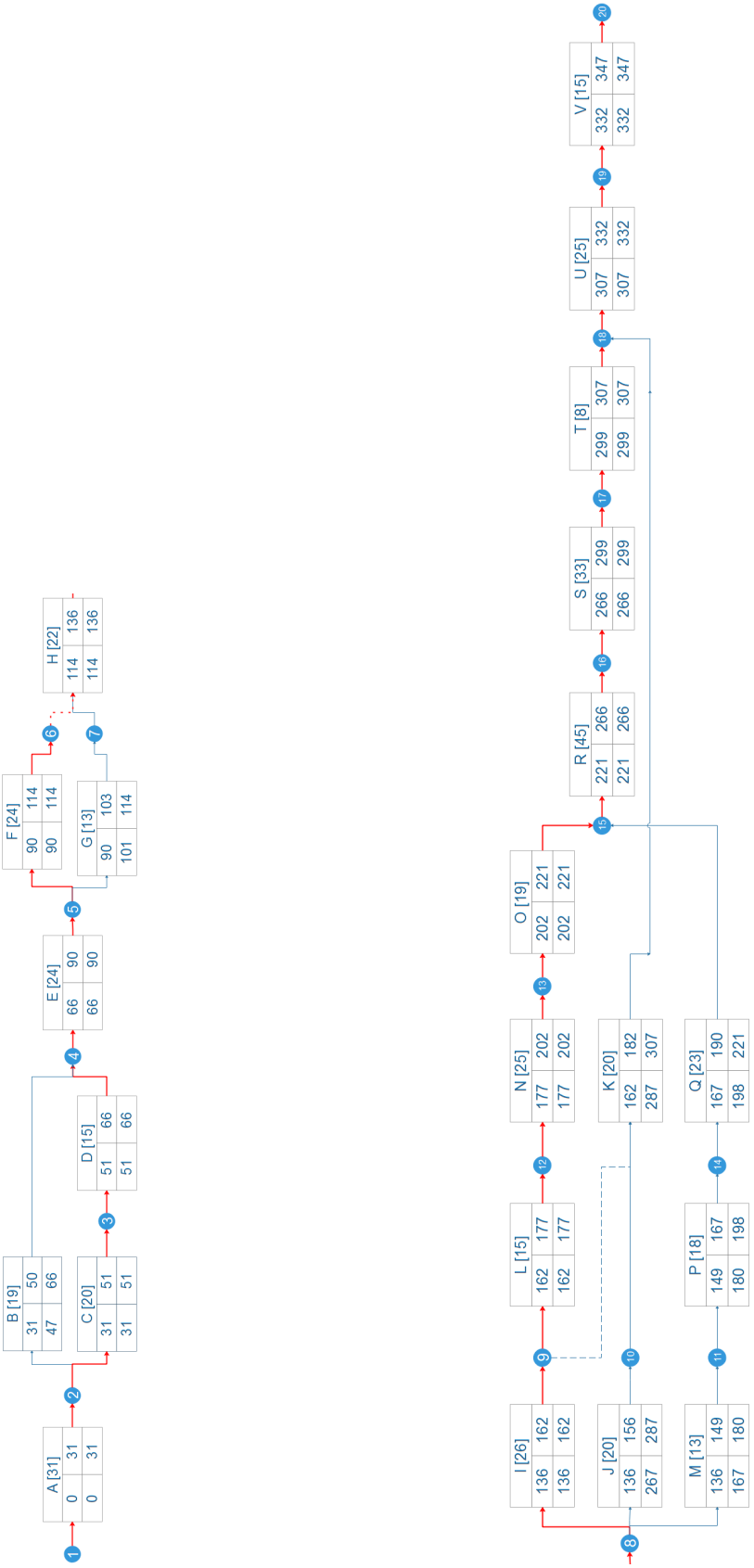


Figura 6: Diagrama de camino crítico.

Ahora bien, considerando que la suma de los tiempos de las actividades independientes sigue una distribución de probabilidades normal, se puede establecer que el tiempo total del proyecto será de 347 días con un desvío estándar de 8.42. Con estos resultados se puede decir que la probabilidad de completar el proyecto antes de 365 días, es de 98.4 % .

7.2. Programación

A continuación se presenta el diagrama de Gantt del proyecto basado en las descripciones de la tareas previamente realizadas. El objetivos de este tipo de diagramas es evidenciar la asignación de recursos a cada tarea con el fin de obtener el resultado esperado.

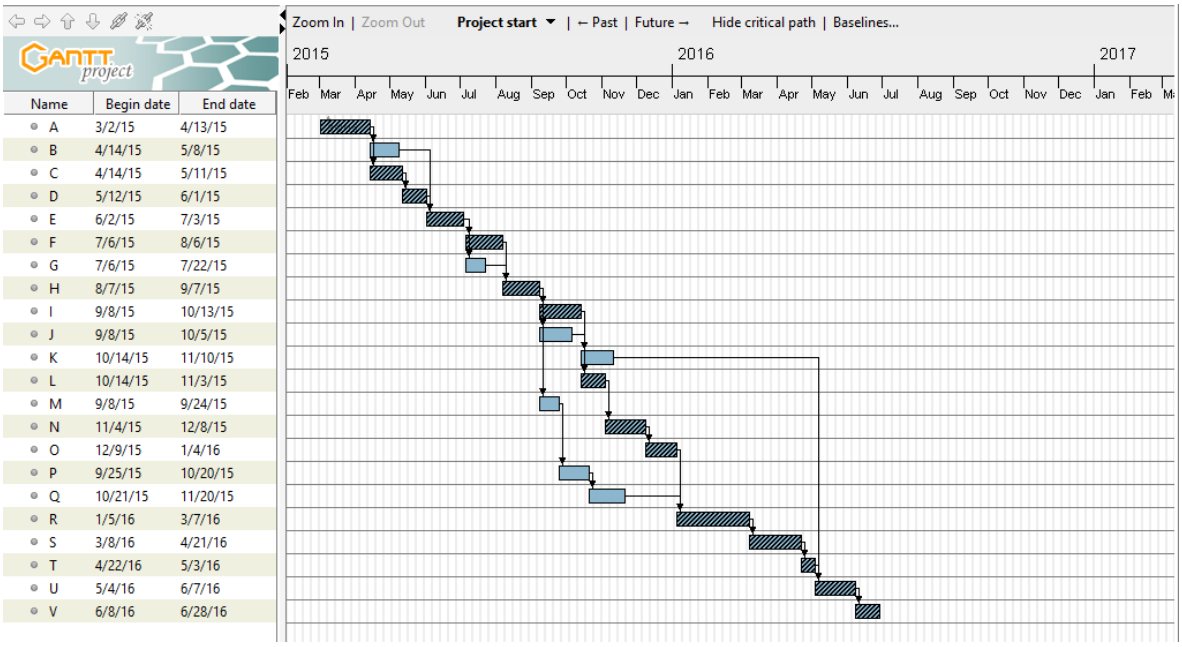


Figura 7: Diagrama de Gantt.

Sin embargo, el mayor provecho de este tipo de diagramas es entender cómo se asignan los recursos para cada actividad. A continuación se muestra el grado de actividad de cada integrante del grupo.

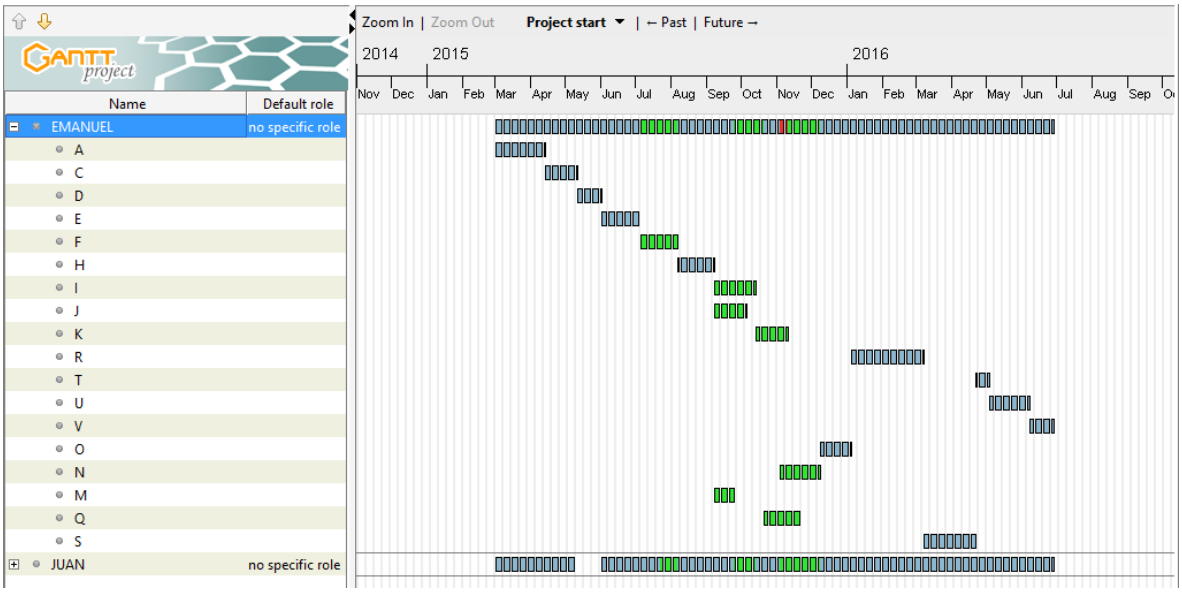


Figura 8: Asignación de Emanuel

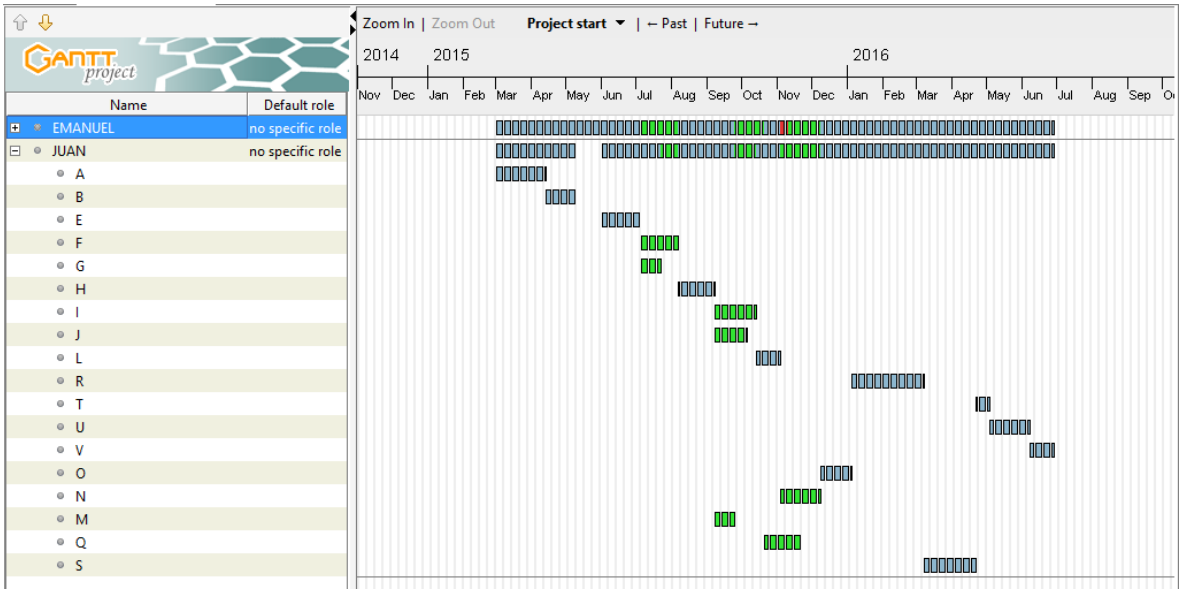


Figura 9: Asignación de Juan.

De las figuras anteriores se puede extraer la dedicación de cada integrante al proyecto y entre cuántas actividades debe alternarse para conseguir la programación deseada. En la práctica, estos valores no se mantuvieron dado que ante la presencia de un problema se generan delays que afectan a todas las actividades. Como una herramienta de mejora se debería generar un registro del desarrollo del producto con el fin de ir optimizando esta programación, identificando demoras, bajas en el rendimiento, entre otros parámetros.

7.3. Comparación con la realidad

A continuación se muestra la comparación de los valores estimados para cada actividad con los valores reales empleados.

	Actividad	Tiempo Optimista	Tiempo Más Probable	Tiempo Pesimista	Inmediata Predecesora	Tiempo Real
A	Investigación Inicial y Análisis de Mercado	26	31	36	-	35
B	Elección del Sistema de Modulación	12	20	22	A	28
C	Diseño e Implementación del Modelo de Condiciones	18	20	22	A	18
D	Simulación de las Condiciones	10	15	20	C	30
E	Diseño del Modelo de Transmisión	18	25	26	D,B	25
F	Simulación de Transmisión	12	26	28	E	30
G	Investigación sobre la Codificación	7	13	19	E	10
H	Diseño global Terminado	20	22	24	F,G	30
I	Diseño de Hardware	22	25	34	H	40
J	Diseño de Software	18	20	22	H	45
K	Diseño de Pruebas de Validación	15	20	25	I,J	20
L	Elección del Módulo de Transmisión	13	15	17	I	15
M	Diseño de la Codificación	14	13	15	H	12
N	Ensamblado de Hardware	20	25	30	L	50
O	Testeo de Hardware	16	19	22	N	20
P	Confección de Software	16	17	24	M	35
Q	Testeo de Software	21	23	25	P	25
R	Prototipo del Sistema de Transmisión	30	45	60	O,Q	45
S	Medición del Prototipo	22	33	44	R	30
T	Medición del Prototipo con Codificación	6	8	10	S	15
U	Pruebas de Validación sobre el Diseño	20	25	30	K,T	45
V	Recopilación Final de Documentación	13	15	17	U	20

Cuadro 13: Actividades con indicación de predecesión.

Como conclusión se puede decir que se produjeron demoras por falta de experiencia. Generalmente se asume una merma en el rendimiento propio al dedicarse a más de una tarea en simultáneo, pero no todo surge de eso. Por ejemplo, si bien se pensó que la implementación del software sería más rápida que la real, los problemas con el proveedor a la hora del conseguir el hardware no fueron contemplados. Lo retrasos en cada actividad se pueden observar en la figura 10, en el cual se comparan los valores esperados para cada actividad con los tiempos reales. Al realizar el cálculo del error acumulado para todas las actividades resulta un total de 151 días. Este valor sirve comparar cualitativamente la demora con la duración del proyecto dado que hay actividades que se realizan en paralelo, por lo que no necesariamente se traslada dicha demora directamente a la demora del proyecto.

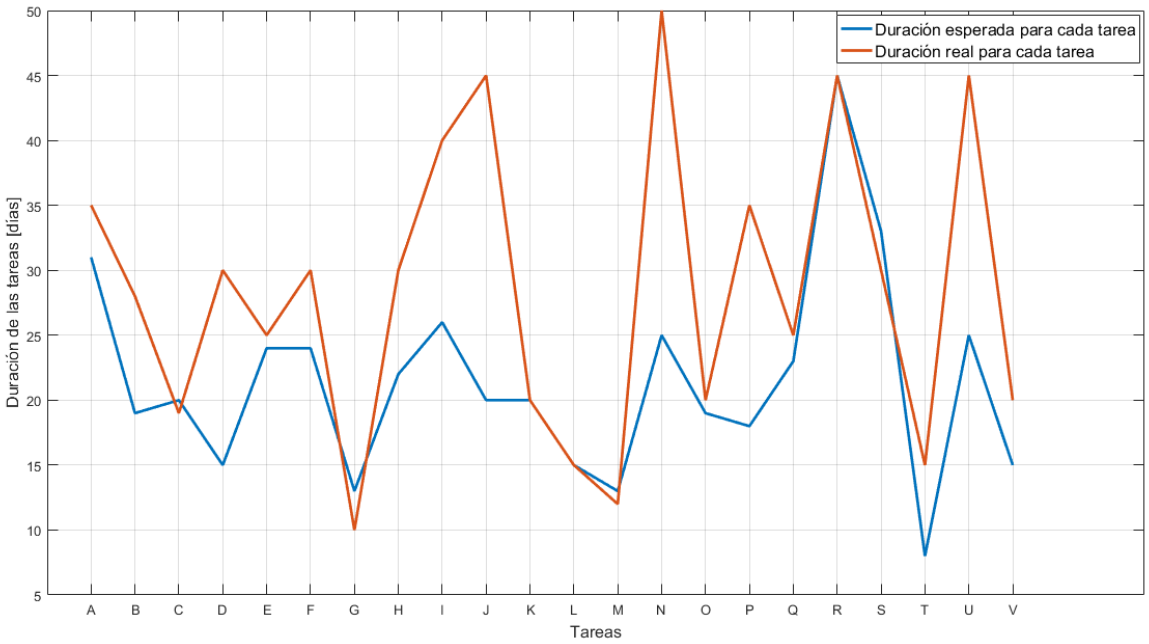


Figura 10: Comparación valores esperados y valores reales empleados.

Por otro lado, otro factor importante en la demora del proyecto se produjo porque este plan estaba pensado para una dedicación full time y eso nunca se llevó a cabo. Los autores tuvieron una dedicación part time, lo que generó más retrasos por el simple hecho de una menor dedicación y que el rendimiento de una persona disminuye al intercambiar entre tareas.

Algo importante a destacar es que los mayores problemas de estimación se encontraron en tareas relacionadas al Software. El error de estimación en estas tareas se debieron a una subestimación de la complejidad del proyecto en éste área y a lo complicado que puede ser desarrollar un código en pequeños bloques de tiempo. En algunos casos es necesario releer lo que se hizo en otro momento para rememorar el avance, que consideraciones se tuvieron en cuenta a la hora de hacer una función, entre otros ejemplos. Estos problemas se podrían haber evitado realizando la documentación correspondiente en el momento adecuado y manteniendo una mejor gestión de control de versiones del software.

Finalmente, la demora real parece escasa teniendo en cuenta los 151 días de error de estimación y el factor de la dedicación part time, pero hay que tener en cuenta que dichos errores se pueden superponer generando un efecto de menor impacto en la duración total del proyecto. Por ejemplo, mientras se esperaba que el proveedor de hardware cumpliera con sus plazos se realizaba una tarea programada con la mitad del rendimiento. A modo de mejora de la estimación se plantearía una condición de trabajo part-time, dando más margen a aquellas tareas en las que se depende de terceros, o rediseñar el proceso para poder maximizar el número de tareas en paralelo con dichas tareas.

8. Factibilidad Legal

En esta sección se realizará un análisis de la factibilidad legal del proyecto. Antes de comenzar se recordará el uso esperado del mismo. El objetivo de este proyecto es desarrollar un prototipo de un sistema de comunicación que sea comercializable para personas de carácter amateur. Si bien el desarrollo se realiza en Argentina, primero se realizará un análisis global para cubrir los requerimientos legales de todo el mercado mundial.

A nivel internacional, la Argentina firmó un tratado denominado "*Tratado sobre los principios que deben regir las actividades de los Estados en la exploración y utilización del espacio ultraterrestre*"[15] generado por la ONU. El mismo representa el marco jurídico básico del derecho internacional del espacio, prohibiendo a los estados partes del tratado, la colocación de armas nucleares u otras armas de destrucción masiva en la órbita de la Tierra, su instalación en la Luna o cualquier otro cuerpo celeste, o de otra estación en el espacio exterior. Además limita exclusivamente la utilización de la Luna y otros cuerpos celestes con fines pacíficos y prohíbe expresamente su uso para pruebas de armas de cualquier tipo, la realización de maniobras militares o el establecimiento de bases militares, instalaciones y fortificaciones. Finalmente el tratado establece que las actividades de las entidades no gubernamentales en el espacio ultraterrestre deberán ser autorizadas y fiscalizadas constantemente por el pertinente Estado Parte en el Tratado, y que los Estados Partes serán responsables internacionalmente de las actividades nacionales que realicen en el espacio ultraterrestre los organismos gubernamentales o las entidades no gubernamentales.

Ahora bien, el tratado anterior establece el uso esperado del espacio ultraterrestre. Sin embargo, no regula el uso de los mismos. Es por esto que otras organizaciones secundarias organizan la comunicación satelital para cada región del planeta. Algunos elementos cuya regulación es necesarias son la posición de los satélites, las frecuencias de transmisión en las que deben trabajar, etc. Para poder determinar dichas características se debe recurrir a legislaciones que se encuentran por debajo del tratado mencionado anteriormente y que agrupan a los países de acuerdo a su ubicación geográfica.

Dado que el objetivo que persigue este trabajo es desarrollar un prototipo que sea comercializable para personas de carácter amateur, y que a nivel mundial las características de espectro no son constantes, se debe definir un potencial mercado. Considerando que en este caso el desarrollo del prototipo se realiza en Argentina, se analizarán las características de dicho país para algunas limitaciones técnicas y legales. Sin embargo este análisis, al involucrar entes y uniones de telecomunicaciones aplica al mercado contemplado para este proyecto.

A la hora de diseñar un sistema de comunicaciones, una de las características más importante es la frecuencia a la que se comunicará el dispositivo. En el caso de la Argentina, no hay un único organismo regulador que concentre todas las regulaciones del espectro a nivel nacional. De todas formas, los distintos organismos argentinos recurren a las regulaciones de uniones internacionales, en este caso la IARU. La IARU coloca a la Argentina en la Región II junto a la mayoría de los países americanos. Dicha organización establece que tanto la banda de 420-450 MHz y 902–928 MHz se utilizan con fines de servicio amateur.[16]Dicha región incluye también a los Estados Unidos, país que representa un mercado atractivo para el proyecto.

FRECUENCIAS PARA RADIOAFICIONADOS		
Banda	DESDE [MHz]	HASTA [MHz]
MF	1.8	2
HF	3.5	3.8
	7	7.3
	10.1	10.15
	14	14.35
	18	18.168
	21	21.45
	24.89	24.99
	28	29.7
VHF	50	54
	144	148
	219	225
UHF	420	450
	902	928
	2300	2450
SHF	3300	3500

Figura 11: Distribución de frecuencias amateur para cada banda.

Dado que las organizaciones argentinas se basan en las reglamentaciones internacionales, a continuación se citan algunas limitaciones destacadas.

Teniendo en cuenta las postulaciones de la AMSAT (Asociación Mundial de Satélites de Radioaficionados), una de las consideraciones más importantes de los satélites amateurs es que no interfieran con señales existentes. La AMSAT especifica el uso amateur como aquel que consiste en un servicio cuyo propósito es el de autoformación, investigación técnica a cargo de amateurs, y que no posean fines de lucro. Además se establece que las comunicaciones no deben ser encriptadas, y deben estar disponibles para cualquier posible usuario. A su vez, como el dispositivo podría generar alguna interferencia, la asociación especifica que se debe poder controlar la actividad del satélite desde la superficie, para poder apagarlo en caso de que fuese necesario.[19]

Es importante mencionar que la Argentina es un país en desarrollo en dicha área, por lo que se encuentra desarrollando nuevas leyes que acompañan a un desarrollo satelital. Como ejemplo se puede citar la ley 27.208 que, si bien es una ley sancionada con el fin de generar un plan satelital geoestacionario para los próximos 20 años, pone en evidencia el interés del país en el área, como así también la necesidad de regulaciones. Sin embargo, es posible que en el caso de los nanosatelites serán necesarias leyes internacionales dado que los mismo orbitan a la Tierra, recorriendo sobre su trayectoria distintos países. La última idea queda en evidencia cuando se analiza el concepto de espacio aéreo. Cáda país es soberano de su espacio aéreo, el cual abarca hasta 22 kilómetros de altura, mientras que los nanosatelites orbitan a partir de los 200 km atravesando distintos países en su paso.[17]

9. Factibilidad Tecnológica

9.1. Análisis cualitativo de la modulación

El objetivo de este proyecto es el de realizar un sistema de comunicaciones satelitales. Deben tenerse en cuenta los problemas intrínsecos del medio en el que se trabaja y las bandas disponibles. Por estos motivos se debe buscar optimizar en la comunicación tanto el ancho de banda utilizado como la potencia de transmisión.

Las modulaciones disponibles que se van a analizar son M-FSK, M-PSK y M-ASK.

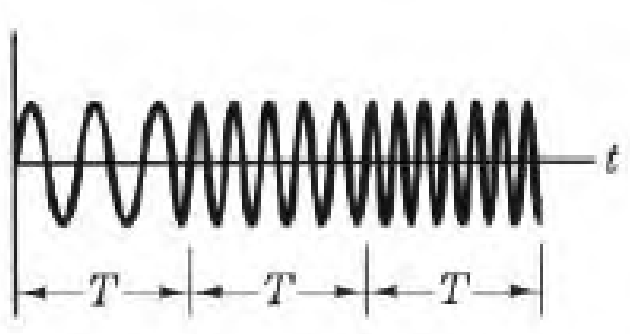


Figura 12: Diagrama temporal de la modulación FSK

La primera opción M-FSK, por su sigla en inglés *multiple frequency shift keying*, consiste en enviar cada símbolo de la comunicación como una frecuencia diferente, todas ortogonales entre ellas. Este método resulta altamente eficiente en potencia, utilizando un gran ancho de banda para lograr esto.

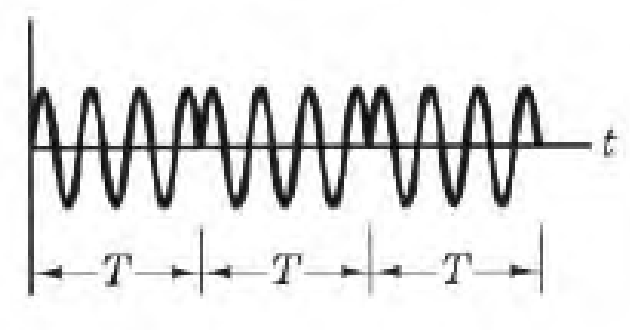


Figura 13: Diagrama temporal de la modulación PSK

La modulación MPSK, por su sigla en inglés *multiple phase shift keying*, consiste en que cada símbolo del sistema es representado por una fase en particular de la comunicación. Esto propone una gran ventaja ya que el aumento de la tasa de símbolo no afecta el ancho de banda de la comunicación aunque sí la probabilidad de error por lo que se afecta el rendimiento de la misma. Esto hace que un alto número de símbolos diferentes termine empeorando la comunicación. La modulación frecuentemente utilizada en comunicaciones satelitales suele ser QPSK debido a ser el punto medio de la relación de compromiso entre ancho de banda y probabilidad de error producida.

Otro inconveniente que posee este método es que los receptores deben ser más complejos debido a que se requiere un perfecto sincronismo entre el receptor y el transmisor para detectar correctamente los cambios de fase.

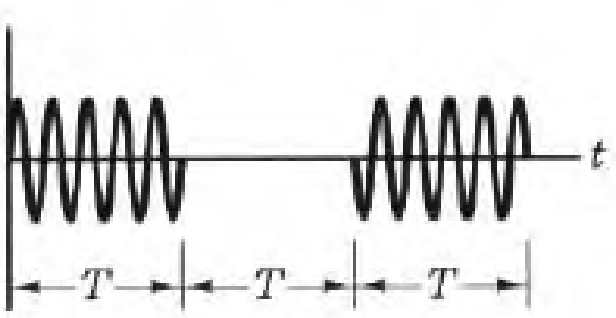


Figura 14: Diagrama temporal de la modulación ASK

Otra opción a considerar es M-ASK, por su sigla en inglés *Multiple Amplitude Shift Keying*, consiste en enviar diferentes amplitudes las cuales representan los diferentes símbolos. El caso particular de esta modulación que será analizado es OOK (*On-Off Keying*) el cual consiste en que un símbolo se produce enviando señal y el otro se representa como la ausencia de la misma. Una ventaja que propone esto comparado con cualquier otro tipo de modulación M-ASK es que permite el uso de amplificadores alineales, los cuales resultan mucho más eficientes. La desventaja que conlleva es una pérdida de 3dB al utilizar símbolos ortogonales en lugar de antipodales.

9.2. Análisis Cuantitativo de las Modulaciones

	2-FSK	QPSK	M-FSK	OOK
Eficiencia de potencia [0.1]	Buena [1]	Media [0.5]	Buena [1]	Media [0.4]
Eficiencia espectral [0.1]	Mala [0.3]	Buena [0.7]	Mala [0.2]	Buena [0.8]
Facilidad de implementación [0.25]	Media [0.6]	Baja [0.4]	Media [0.6]	Alta [0.9]
Costo de los módulos comerciales [0.35]	Medio [0.6]	Alto [0.4]	Baja [1]	Bajo [1]
Tasa de errores [0.2]	Baja [0.9]	Media [0.6]	Baja [1]	Alta [0.4]
	0.67	0.48	0.82	0.77

Cuadro 14: Comparación entre las tres modulaciones.

De acuerdo al análisis, se utilizará M-FSK como modulación debido al bajo costo, su baja tasa de errores y su facilidad de implementación frente a las otras modulaciones analizadas.

9.3. Análisis de las Codificaciones

Las comunicaciones en la banda amateur prohíben el uso de comunicaciones encriptadas. Sin embargo está permitido el uso de codificaciones para mejorar la comunicación. Por este motivo se puede agregar al análisis para mejorar el rendimiento de la comunicación. El uso de una codificación permite mejorar la tasa de errores sin necesidad de incrementar la potencia de transmisión o aumentar la sensibilidad del receptor.

Existen varios tipos de codificaciones en la actualidad, las cuales poseen diferentes requerimientos para su correcto funcionamiento. Las dos bases principales utilizadas en comunicaciones espaciales son codificación Trellis[23] con decodificación utilizando el algoritmo de Viterbi[23] y los Turbo Códigos[22], los cuales son más recientes.

Los turbo códigos poseen la ventaja de ser superiores en varios aspectos a otras codificaciones y son los que se encuentran más próximos al límite teórico de Shannon para las comunicaciones, alejándose del mismo solamente 0.5dB. La desventaja que poseen sin embargo es que requieren *soft input*, es decir que necesitan una entrada cuantizada en varios bits para poder funcionar correctamente. Esto es una desventaja ya que aumenta el costo del receptor debido a que hacen falta varios niveles de comparación para convertir la señal de entrada.

Por otro lado los códigos Trellis son capaces de funcionar tanto con *soft input* como con *hard input*. *Hard input* a diferencia de *soft input* consiste en una entrada binaria. Esto hace que el costo se reduzca considerablemente. Además los mismos proponen una gran mejora en la comunicación, alrededor de 3dB, y se encuentran ampliamente implementados y requieren mejor costo computacional que los turbo códigos.

Factor [importancia]	Sin Codificación	Turbo Códigos	Trellis/Viterbi
Ganancia [0.25]	Nula [0]	Alta [1]	Media-Alta [0.6]
Complejidad [0.25]	Nula [1]	Media [0.5]	Baja [0.8]
Entrada [0.5]	Nula [1]	Soft [0.5]	Soft/Hard [1]
	0.75	0.625	0.85

Cuadro 15: Comparación entre las dos codificaciones.

De acuerdo al análisis, se utilizará Trellis/Viterbi como codificación debido a la ganancia obtenida, su baja complejidad y a que posee la capacidad de utilizar una entrada *hard input*.

9.4. Análisis de la implementación

Se analizaron tres posibles implementaciones para el diseño del producto: diseño analógico del receptor y transmisor implementando *hard input* al cual denominaremos HardIn; diseño analógico del receptor y transmisor implementando *soft input* al cual denominaremos SoftIn; y utilización de módulos comerciales para transmisión y recepción coordinando el funcionamiento con un microprocesador y realizando en el mismo la codificación al cual denominaremos ModIn.

Factor [importancia]	HardIn	SoftIn	ModIn
Performance [0.2]	Alta [0.7]	Muy Alta [1]	Media [0.5]
Complejidad [0.1]	Media/Alta [0.5]	Alta [0]	Baja [1]
Costo [0.5]	Medio [0.5]	Alto [0]	Bajo [1]
Tiempo de desarrollo [0.2]	Medio [0.5]	Alto [0]	Bajo [1]
	0.54	0.2	0.9

Cuadro 16: Comparación entre las tres implementaciones.

Se decide utilizar la opción ModIn porque implica una mejora en la performance del sistema frente a no utilizar codificación, sin agregar mayor complejidad ni costo al módulo.

9.5. Elección de la implementación

Teniendo en cuenta los requerimientos del producto y el objetivo del mismo se decidió optar por la opción más económica que logre satisfacer las necesidades del mercado. Esto es un sistema de comunicaciones que se base en el uso de módulos de transmisor y receptor comerciales, utilizando modulación M-FSK con *hard input* y la implementación de una codificación Trellis utilizando el algoritmo de decodificación de Viterbi.

10. Factibilidad de uso de componentes comerciales

La motivación principal del proyecto radica en el uso de materiales comerciales denominados "*Commercial Off-The-Shelf*" conocidos también como COTS. En la industria aeroespacial se suelen utilizar componentes con antecedentes de uso y gran cantidad de pruebas debido a la confiabilidad necesaria en un satélite convencional. Sin embargo varios estudios se han realizado recientemente por diversas agencias con el fin de analizar el uso de componentes COTS para saber si pueden ser utilizados en diferentes misiones aeroespaciales.

En el trabajo "*Thoughts on Commercial Off the Shelf*"[24] científicos de la NASA analizaron los riesgos involucrados en el uso de componentes COTS y el banco de pruebas necesario para asegurar la confiabilidad. El estudio concluye que es posible el uso de esta clase de componentes pero debe tomarse en cuenta las pruebas necesarias para asegurar el correcto funcionamiento de los componentes durante toda la misión. Es importante destacar que este análisis fue realizado para una misión espacial genérica y no se refiere específicamente al caso de uso de los nanosatélites de órbita baja.

Un trabajo desarrollado por la Universidad Tecnológica de Dinamarca[27] estableció que es posible el desarrollo de instrumentos espaciales con componentes COTS. La conclusión a la que llega el trabajo es que el factor más

importante a analizar en estos componentes es la respuesta a la radiación para asegurar inmunidad al efecto de *Latch-Up* en el espacio exterior.

Una investigación llevada a cabo por la empresa Airbus[25] puso a prueba el uso de capacitores en el espacio exterior. La empresa investigó como se comportaban los capacitores COTS en un ambiente de vacío y bajo el efecto de radiación. La investigación dio como resultado que los capacitores COTS pueden ser utilizados en el espacio exterior.

Una publicación[26] llevada a cabo por la NASA analizó el efecto de la radiación en PCB COTS . La conclusión del trabajo determinó que existe una relación de compromiso entre el riesgo y el costo de un PCB diseñado para soportar la radiación. El trabajo explica que como regla general se puede considerar la órbita baja como un ambiente seguro de radiación por lo que se pueden utilizar PCB COTS.

Teniendo en consideración los antecedentes y trabajos mencionados se considera factible el uso de componentes COTS para el desarrollo de todo el nanosatélite, en particular el sistema de comunicación.

11. FMEA

A continuación se muestra el análisis de FMEA realizado.

Severidad (S)	Ocurrencia (O)	Detección (D)
1: Menores	1: Muy Poco Probables	1: Muy Detectable
3: Significativos	3: Poco Probable	3: Detectable
6: Críticas	6: Probable	6: Poco Detectable
10: Catastróficas	10: Altamente Probable	10: Indetectable

Figura 15: Tabla de definiciones de códigos a utilizar.

Descripción	FMEA del sistema de transmision.	**Responsabilidad de diseño:**	Fernández Aguirre, Juan Pablo Villa Fernández, Emanuel Ignacio	**Numero de FMEA:**	1
Años de aplicación:	2015- actualidad	**Fecha límite:**	-	**Realizado por:**	Fernández Aguirre, Juan Pablo Villa Fernández, Emanuel Ignacio
Motivos:	Analisis del módulo			**Fecha de Inicio**	24/05/2015
Equipo de trabajo:	Fernández Aguirre, Juan Pablo Villa Fernández, Emanuel Ignacio			**Fecha de Revisión**	-

Figura 16: Tabla de información sobre la realización de la FMEA

Nivel de IC	
Óptimo	IC ≤ 15
Aceptable	15 < IC ≤ 39
Bajar hasta razonablemente práctico	39 < IC ≤ 60
No Aceptable	60 < IC

Figura 17: Tabla sobre los niveles de aceptación de la FMEA

Design FMEA

#	Módulo	Modo de Falla Potencial	Efectos Potenciales de Falla	Causas Potenciales de Falla	Severidad	Detección	Ocurrencia	RPN	Control de Prevención Actual	Control de Detección Actual	Acción Recomendada	Responsable	Fecha Objetivo de Resolución	Acción Tomada	Severidad	Detección	Ocurrencia	RPN
1	Transmisor	Aislación rota entre pistas de comunicación.	Problemas de comunicación.	Error en la fabricación del módulo.	9	2	3	54	Distancia entre pistas	Validación de prototipo	Implementar validación de la comunicación.	EVF	3/3/2017	Validación de los módulos antes de la entrega	7	1	1	7
2	Transmisor	Aislación rota con alimentación.	Se genera interferencia no deseada.	Sobretensión en la alimentación.	5	4	2	40	Distancia entre pistas	Validación de prototipo	Protección en la entrada. Manejo unificado de la tensión de entrada.	JPF	3/3/2017	Protección en la entrada. Manejo unificado de la tensión de entrada.	1	1	1	1
3	Transmisor	Módulo de transmisión dañado.	No se pueden enviar datos.	Sobretensión en la alimentación.	10	8	1	80	Distancia entre pistas.	Control de alimentación.	Protección a la entrada de alimentación.	JPF	3/3/2017	Protección a la entrada de alimentación.	7	3	1	21
4	Transmisor	Módulo de transmisión dañado.	No se pueden enviar datos.	Exceso de radiación.	10	8	1	80	Stack Up resistente a EMC.	-	Diseño multicapa como protección de radiación	JPF	3/3/2017	Diseño multicapa como protección de radiación	8	4	1	32
5	Transmisor	Módulo de transmisión dañado.	No se pueden enviar datos.	Golpes de basura espacial.	10	8	1	80	-	-	-	JPF	3/3/2017	-	10	8	1	80
6	Transmisor	Módulo de transmisión dañado.	No se pueden enviar datos.	Exceso de temperatura.	10	8	2	160	-	Medición de temperatura.	Control de temperatura.	JPF	3/3/2017	Control de temperatura.	7	3	1	21
7	Transmisor	Módulo de transmisión dañado.	No se pueden enviar datos.	Falta de temperatura.	10	8	2	160	-	Medición de temperatura.	Control de temperatura.	JPF	3/3/2017	Control de temperatura.	7	3	1	21
8	Transmisor	Módulo desconfigurado.	No se pueden enviar datos.	Sobretensión en la alimentación.	5	4	2	40	-	-	Protección a la entrada de alimentación.	EVF	3/3/2017	Protección a la entrada de alimentación.	3	3	1	9
9	Transmisor	Módulo desconfigurado.	No se pueden enviar datos.	Exceso de radiación.	7	6	1	42	-	-	Diseño multicapa con protección para EMC.	EVF	3/3/2017	Diseño multicapa con protección para EMC.	3	3	1	9
10	Transmisor	Módulo desconfigurado.	No se pueden enviar datos.	Crosstalk.	5	3	3	45	-	-	Diseño multicapa con protección para el crosstalk.	EVF	3/3/2017	Diseño multicapa con protección para el crosstalk	3	3	1	9
11	Transmisor	Módulo apagado.	No se pueden enviar datos.	Abierto en la alimentación.	10	3	2	60	-	-	Implementar validación de la comunicación.	EVF	3/3/2017	Implementar validación de la comunicación.	7	1	1	7
12	Transmisor	Potencia de salida menor a la deseada.	Aumento del BER.	Baja tensión de alimentación.	6	10	3	180	-	-	Control de alimentación.	JPF	3/3/2017	Control de alimentación.	6	3	2	36
13	Transmisor	Módulo atrapado en un ciclo infinito.	No se pueden enviar datos, problemas de comunicación, funcionamiento errático.	Radiación afecto el orden de ejecución del programa.	7	7	2	98	-	-	Diseño multicapa como protección de radiación	JPF	3/3/2017	Diseño multicapa como protección de radiación	5	5	1	25
14	Transmisor	Tensión intermitente en los pines.	El módulo posee un funcionamiento errático e impredecible.	Vibraciones generan falsos contactos	7	9	2	126	-	-	Mejora de las interfaces mecánicas	EVF	3/3/2017	Mejora de las interfaces mecánicas	3	6	2	36
15	Transmisor	Tensión baja en los pines.	Errores de comunicación con el satélite.	Baja tensión de alimentación.	4	8	3	96	-	-	Protección a la entrada de alimentación.	JPF	3/3/2017	Protección a la entrada de alimentación.	4	3	3	36
16	Transmisor	Tensión alta en los pines.	Errores de comunicación con el satélite, consumo excesivo.	Alta tensión de alimentación.	5	8	3	120	-	-	Protección a la entrada de alimentación.	JPF	3/3/2017	Protección a la entrada de alimentación.	4	3	3	36
17	Transmisor	Aislación rota entre pistas de comunicación.	Problemas de comunicación.	Problemas de configuración	4	3	4	48	-	-	Validación del módulo	EVF	3/3/2017	Validación del módulo	2	1	2	4
18	Receptor	Aislación rota con alimentación.	Se genera interferencia no deseada.	Sobretensión en la alimentación.	10	4	2	80	Distancia entre pistas	Validación de prototipo	Protección en la entrada. Manejo unificado de la tensión de entrada.	JPF	3/3/2017	Protección en la entrada. Manejo unificado de la tensión de entrada.	5	2	1	10
19	Receptor	Módulo de recepción dañado.	No se pueden recibir datos.	Sobretensión en la alimentación.	10	10	1	100	Distancia entre pistas.	Control de alimentación.	Protección a la entrada de alimentación.	JPF	3/3/2017	Protección a la entrada de alimentación.	7	5	1	35
20	Receptor	Módulo de recepción dañado.	No se pueden recibir datos.	Exceso de radiación.	10	10	1	100	-	-	Diseño multicapa como protección de radiación	JPF	3/3/2017	Diseño multicapa como protección de radiación	8	4	1	32
21	Receptor	Módulo de recepción dañado.	No se pueden recibir datos.	Golpes de basura espacial.	10	10	1	100	-	-	-	JPF	3/3/2017	-	10	10	1	100
22	Receptor	Módulo de recepción dañado.	No se pueden recibir datos.	Exceso de temperatura.	10	10	2	200	-	Medición de temperatura.	Control de temperatura.	JPF	3/3/2017	Control de temperatura.	7	5	1	35
23	Receptor	Módulo de recepción dañado.	No se pueden recibir datos.	Falta de temperatura.	10	10	2	200	-	Medición de temperatura.	Control de temperatura.	JPF	3/3/2017	Control de temperatura.	7	5	1	35
24	Receptor	Módulo desconfigurado.	No se pueden recibir datos.	Sobretensión en la alimentación.	10	3	3	90		Control de alimentación.	Protección a la entrada de alimentación.	JPF	3/3/2017	Protección a la entrada de alimentación.	7	2	1	14
25	Receptor	Módulo desconfigurado.	No se pueden recibir datos.	Crosstalk.	10	3	3	90	Distancia entre pistas.	-	Diseño multicapa como protección de radiación	JPF	3/3/2017	Diseño multicapa como protección de radiación	7	2	1	14
26	Receptor	Módulo apagado.	No se pueden recibir datos.	Abierto en la alimentación.	10	2	2	40	-	-	Implementar validación de la comunicación.	EVF	3/3/2017	Implementar validación de la comunicación.	7	1	1	7
27	Receptor	Potencia de salida menor a la deseada.	Aumento del BER.	Baja tensión de alimentación.	6	10	3	180	-	-	Control de alimentación.	JPF	3/3/2017	Control de alimentación.	6	3	2	36
28	Receptor	Módulo atrapado en un ciclo infinito.	No se pueden recibir datos, problemas de comunicación, funcionamiento errático.	Radiación afecto el orden de ejecución del programa.	10	7	2	140	-	-	Diseño multicapa como protección de radiación	JPF	3/3/2017	Diseño multicapa como protección de radiación	5	5	1	25
29	Receptor	Tensión intermitente en los pines.	El módulo posee un funcionamiento errático e impredecible.	Vibraciones generan falsos contactos	7	9	2	126	-	-	Mejora de las interfaces mecánicas	EVF	3/3/2017	Mejora de las interfaces mecánicas	4	6	1	24
30	Receptor	Tensión baja en los pines.	Errores de comunicación con el satélite.	Baja tensión de alimentación.	4	8	3	96	-	-	Protección a la entrada de alimentación.	JPF	3/3/2017	Protección a la entrada de alimentación.	4	3	3	36
31	Receptor	Tensión alta en los pines.	Errores de comunicación con el satélite, consumo excesivo.	Alta tensión de alimentación.	5	8	3	120	-	-	Protección a la entrada de alimentación.	JPF	3/3/2017	Protección a la entrada de alimentación.	4	3	3	36
32	Procesamiento	Alta temperatura.	El funcionamiento puede resultar errático. Se pueden dañar los módulos.	Funcionamiento erróneo del sensor de temperatura.	8	4	2	64	-	-	Calibración	JPF	3/3/2017	Calibración	5	2	1	10
33	Procesamiento	Baja temperatura.	El funcionamiento puede resultar errático. Se pueden dañar los módulos.	Funcionamiento erróneo del sensor de temperatura.	4	6	2	48	-	-	Calibración	JPF	3/3/2017	Calibración	3	2	1	6
34	Procesamiento	Módulo atascado en una rutina.	Sistema de comunicaciones fuera de funcionamiento.	Program Counter afectado por radiación. Rutina mal programada causa atascamiento en el sistema.	10	7	7	490	-	-	Diseño multicapa como protección de radiación. Implementación de Watchdog	EVF	3/3/2017	Diseño multicapa como protección de radiación. Implementación de Watchdog	6	3	2	36

12. Factibilidad Económica

Se analizó la viabilidad económica del proyecto y la rentabilidad que puede llegar a ofrecerle a los inversores.

12.1. Modelo de Negocios

La finalidad del proyecto consiste en generar y captar un nuevo mercado por lo que no se busca que el proyecto sea económicamente viable.Sin embargo es conveniente optimizar el proceso de fabricación y determinar la mejor duración del proyecto para minimizar el impacto económico a la empresa y atraer posibles inversores.

Se planea vender el producto vía Internet, haciendo especial hincapié en el mercado estadounidense ya que posee una gran cantidad de desarrolladores amateur y emprendedores interesados en el espacio. El proyecto busca poder ofrecer a los desarrolladores amateur un módulo de comunicaciones que permita un enlace satelital de órbita baja a un costo significativamente menor que los actuales del mercado. Por este motivo se busca establecer un precio de venta bajo y para lograr esto un costo de fabricación que permita esto. Es importante destacar que este proyecto no busca la rentabilidad económica en sí mismo sino que representa un punto de entrada a una tecnología en pleno auge y un mercado virgen que ofrece un gran nicho para ser explotado.

12.2. Benchmarking

Se realizó un relevamiento de la competencia existente en el mercado para analizar las prestaciones ofrecidas por las diferentes empresas, y los precios que manejan. Los precios fueron tomados de la página de servicio de los mismos, por lo que no poseen cargos de envío ni de impuestos de importación.

Fabricante	Rango de Frecuencias	Modulación	BER	Power Consumption	Max Downlink	Uplink	Precio
ISIS Space	VHF/UHF	BPSK/ASK	1.00E-05	< 4.5 W	9600 bps	1200 bps	9200USD
ClydeSpace	S-Band	OQPSK-QPSK	-	< 5 W	-	1Mbps	12500USD

Cuadro 17: Listado de productos de la competencia.

Debido a que el producto busca explotar un sector del mercado ignorado, el objetivo es atraer clientes con una gran ventaja en precio utilizando módulos de venta masiva. Si bien las velocidades de transmisión de datos de la competencia son superiores, nuestro precio apunta a ser inferior. En el caso de Isis Space su velocidad es 10 veces superior mientras que su precio es 18 veces mayor. Por el lado de ClydeSpace, el cual consiste en un transmisor únicamente, su velocidad es 8.000 veces mayor pero su precio es 25 veces mayor. Para obtener un transreceptor full-dúplex de ClydeSpace es necesario comprar un transmisor y un receptor por separado, lo que tiene un costo de 25.000 USD, e implementar la integración de ambos módulos. En este caso, si bien parece conveniente la diferencia de velocidades, hay que recordar que el usuario esperado del proyecto es una persona amateur que no posee un gran capital de inversión.

Proponiendo un precio objetivo para un nanosatélite de 2.500 USD, resultado proveniente de la encuesta de la sección 26, y asumiendo que el costo del módulo de transmisión representa un 20 % del mismo, se establecio un precio objetivo de 500 USD[28]. La encuesta fue desarrollada en diferentes foros de radioaficionados y especializados en nanosatélites relacionados con el desarrollo de nanosatélites CubeSat.

El producto busca penetrar el mercado utilizando ventajas económicas sobre algunas prestaciones y proveyendo una posible versatilidad a futuro. El módulo de comunicación a ser desarrollado posee la capacidad de ser actualizado remotamente a futuro permitiendo mejoras de BER, y velocidades para poder modificar las capacidades del módulo sobre la marcha de forma remota.

La ventaja económica busca también permitir una mayor penetración del mercado, permitiendo desarrollar un contacto con las PyMES y desarrolladores amateur.

12.3. Análisis de costos

12.3.1. Costos hundidos

Este proyecto será desarrollado utilizando los recursos de una sociedad responsabilidad limitada existente radicada en la Argentina. Por este motivo algunos costos serán obviados ya que son costos que ya fueron incurridos

o deberán seguir siendo incurridos por el funcionamiento propio de la sociedad ya existente. Los costos hundidos que no serán tomados en cuenta serán los siguientes:

- Instrumentación básica: soldador, multímetro, analizador de espectro, entre otros.
- Costos de inscripción de una sociedad.
- Costos de apertura y manutención de una cuenta bancaria.

12.3.2. Costo de desarrollo

Se realizó una estimación del costo de desarrollo del prototipo. Para esto se utilizó la estimación de factibilidad de tiempos la cual concluyó que serán requeridos 347 días hábiles con dos ingenieros full-time lo que equivale a 5.552 horas hombre. Esta cantidad de horas implica un costo de 86.400 USD. En la realidad el proyecto de demoró 151 días hábiles pero se utilizaron dos ingenieros part-time (4 horas diarias) lo que equivale a 3.984 horas hombre. Esto significa que el costo de desarrollo efectivo fue menor a pesar del hecho que se haya tenido que pagar más tiempo de alquiler.

Respecto a los materiales se gastaron 200 USD para la compra de componentes para el desarrollo de dos prototipos.

		Estimado [USD]	Real [USD]
Mano de obra	Horas hombre requeridos	5552	3984
	Costo por hora hombre	15	15
	Subtotal costo de mano de obra	83280	59760
Locación	Días requeridos	347	498
	Costo Locación por mes	400	400
	Subtotal locación	11600	16800
Materiales	Costo de Materiales	200	200
Total		**95080**	**76760**

Cuadro 18: Análisis de costos de desarrollo del prototipo.

12.3.3. Detalle de los costos variables

La construcción del sistema de comunicaciones requiere los materiales especificados en la tabla 19. La suma de los costos representa el costo total de fabricación de un módulo de comunicación transreceptor lo cual equivale a un producto. Debido a la pequeña escala de la fabricación los precios fueron considerados al valor final minorista disponible en Argentina.

	Componente	Descripción	Costo por unidad [USD]	Cantidad	Sub-total [USD]
Componentes	Resistencia	SMD 0603	0.03	10	0.3
	Capacitor	SMD 0603	0.03	10	0.3
	Capacitor	Film	0.35	3	1.05
	HopeRF96W	Modulo transmisor	2.4	1	2.4
	HopeRF98W	Modulo receptor	2.4	1	2.4
	Antena 433Mhz	Antena para el transmisor	3	1	3
	Antena 915Mhz	Antena para el transmisor	3	1	3
	Regulador	NPC 1117	2	1	2
	Conector SMA	Conector para las antenas	1	2	2
	Conectores	Pines machos rectos	0.01	20	0.2
	MK64FN1M0VDC12	Microcontrolador	11.09	1	11.09
Fabricación	Fabricación y soldado de placa	Fabricación terciarizada	8	1	8
	Testeo de funcionamiento	Validación de cada módulo unitario previo al envío	15	1	15
Empaquetado	Empaquetado	Papel Pluribol	10	0.002	0.02
	Empaquetado	Caja	1	1	1
	Etiqueta	Empaquetado	0.1	1	0.1
	Armado del empaquetado	Armar el paquete para el envío	15	0.08	1.2
Soporte	Soporte	Soporte adicional al cliente	15	1	15
	Total				**68.06**

Cuadro 19: Listado de costos para la fabricación.

Los componentes utilizados serán componentes de consumo masivo [Sección 10]. Para el cálculo del costo de la mano de obra se realizó benchmarking para determinar el sueldo promedio de un ingeniero Junior en el mes de abril del 2017 en Buenos Aires. Se realizó un análisis de mercado para determinar el costo de un ingeniero Junior en el que se analizaron los sueldos de 10 empresas. El sueldo bruto mínimo encontrado en el mercado fue de 1168.75 dólares, un sueldo bruto máximo de 2000 dólares, una media de 1600.

Empresa	Sueldo
1	1200
2	1587.5
3	2000
4	1500
5	1740
6	1372
7	1168.75
8	1325
9	1625
10	1587.5
Media	1510.575
Mediana	1543.75
Varianza	254.7667

Cuadro 20: Análisis de sueldos brutos en el mercado para un ingeniero Junior

Se estableció un sueldo bruto de 1940 USD, próximo al máximo del mercado obtenido del análisis de mercado. En la tabla 21 se encuentran las cargas sociales que debe pagar un empleador en Argentina.

Tabla de Aportes	
Contribuciones	**Porcentaje**
Jubilación	16%
PAMI	2%
Obra Social	5%
Asignaciones familiares	7.50%
Fondo Nacional de Empleo	1.50%
Seguro de Vida Obligatorio	0.03%
ART	3.70%
Total	36%

Cuadro 21: Aportes a cargo del empleador

El costo de un ingeniero Junior para una empresa se estima que es 2640 dólares.

Se considera que no es necesario la dedicación full-time del ingeniero para este proyecto, pudiendo prorratear sus costos con otros proyectos. Por este motivo para el análisis de los costos se consideraron las horas necesarias del recurso por unidad del producto.

La fabricación y soldado de la placa será tercerizado a una empresa de fabricación de placas argentina, quien utiliza pasta de soldado sin plomo minimizando el riesgo de outgassing.

Se consideró un tiempo para la validación del correcto funcionamiento de cada placa así como un tiempo de soporte al cliente en caso que tengan alguna duda respecto al producto.

El envío de los módulos será efectuado por medio de un Courier y el costo de envío quedará a responsabilidad del cliente siguiendo el modelo de mercado de la competencia.

Los pagos de los clientes serán procesados utilizando los recursos provistos por dicha empresa. Los clientes podrán realizar pagos a través de tarjetas de crédito y PayPal. Los medios de pagos escogidos poseen un costo de comisión dependiente del medio:

- PayPal: 5.4 % del monto transferido más 0.30 USD. Para el producto de 500 dólares el costo de comisión de PayPal es 27.3 USD.
- Tarjeta de crédito: 2.5 % del monto pagado. Para un producto implican 12.5 USD.

El monto pagado a Paypal o a la entidad que provea el servicio de tarjeta de crédito será llamado comisión de venta.

Se asume que se los clientes utilizaran ambos medios de pago por igual.

El costo de fabricación de un producto es de 68,06 USD.

12.3.4. Detalle de costos impositivos

Para el proyecto serán considerados los siguientes impuestos:

- Ingresos Brutos (3 %): Es el impuesto que corresponde a las actividades autónomas, actos u operaciones que consiste en la aplicación de un porcentaje sobre la facturación de un negocio independientemente de su ganancia. Se considera la categoría de ingresos brutos I del régimen simplificado de ingresos brutos establecida por la Ciudad Autónoma de Buenos Aires en el Código Fiscal 2017.
- Impuesto a las ganancias (35 %): es un tributo que se aplica sobre los ingresos percibidos por personas, empresas o cualquier entidad legal como medio de recaudación estatal. Se consideró el tope de la tercera categoría.
- Impuesto al Valor Agregado (21 %): La ley 23.349 establece que el impuesto se aplicará sobre cosas muebles vendidas en el país. Los exportadores que efectúen ventas en el extranjero conjuntamente con el mercado local podrá cancelar el débito fiscal local con el crédito fiscal generado por el pago de IVA de las exportaciones. El excedente de crédito fiscal puede ser recuperado de acuerdo a lo dispuesto por el artículo 43 de dicha ley.

Si bien se estima que no se llegará al máximo de ganancias se utiliza el tope del impuesto para considerar el peor caso. Esto significa que se asume que el costo impositivo será del 40 % del valor del producto.

3.5. Exportación del producto

A partir del año 2015 el presidente Mauricio Macri quitó el impuesto a la exportación a "Aeronaves, vehículos espaciales y sus partes" por medio del decreto nacional 160/2015. Por este motivo la exportación del producto no posee aranceles aduaneros ni impuestos a la exportación. Además la exportación de bienes en la República Argentina, con excepción de la provincia de Misiones, se encuentra exenta se tributar el impuesto a Ingresos Brutos. En la Ciudad Autónoma de Buenos Aires se encuentran exentos del impuesto sobre débitos y créditos aquellas acreditaciones del exterior que cancele la exportación de bienes, no alcanzando la exportación de servicios.

Se utilizará como empresa de transporte a la empresa Fedex, la cuál se encargará de la gestión de exportación del producto. Los costos de importación y envío estarán a cargo del cliente como se suele realizar en los mercados digitales internacionales, de la misma manera que lo realiza hoy en día la competencia.

Es importante destacar que como no se tiene un impuesto a la exportación y que el envío es responsabilidad del cliente entonces el costo de venta al exterior o dentro del país es el mismo para la empresa.

Se considera que la cantidad de productos vendidos en el exterior será mayor que la cantidad vendida en el mercado local, por lo que no se considerará el costo del IVA.

12.3.6. Costos adicionales del producto

Es necesario realizar una inversión inicial para difusión del producto en foros y medios relacionados con la industria aeroespacial amateur, y para el desarrollo de un sitio web que permita hacer compras de todos lados del mundo. Esto último también requerirá un mantenimiento del sitio web mensual. Además se debe considerar el costo de alquiler de una pequeña oficina. Los costos a tener en cuenta son:

- Costo de desarrollo de una página Web: 200 USD mensual.
- Costo de mantenimiento de la página Web: 20 USD mensual.
- Costo de alquiler: 400 USD mensuales. Este costo incluye el costo de stock dado que la máxima producción es de 8 unidades por trimestre.
- Costo de Marketing: 500 USD semestral.

12.4. Flujo de fondos

Antes de comenzar con el flujo de fondos, se debe recordar que la finalidad del proyecto no es generar dinero por sí mismo sino que funciona como un pie de entrada a la industria aeroespacial. Es importante esta mención ya que un retorno negativo no implica que el proyecto no cumpla su finalidad. Igualmente es deseable que el proyecto genere la menor cantidad de pérdidas posible.

Se realizó un análisis de como serán los flujos de fondo mensual para los primeros dos años para analizar en que momento el proyecto se encontrará en pérdida y el tiempo necesario para recuperar las inversiones. En esta sección todos los montos monetarios analizados serán en dólares americanos.

De acuerdo a las estimaciones de nanosatélites existentes y encuestas realizadas se estima que la demanda del producto será variable como se indica en la figura 18.

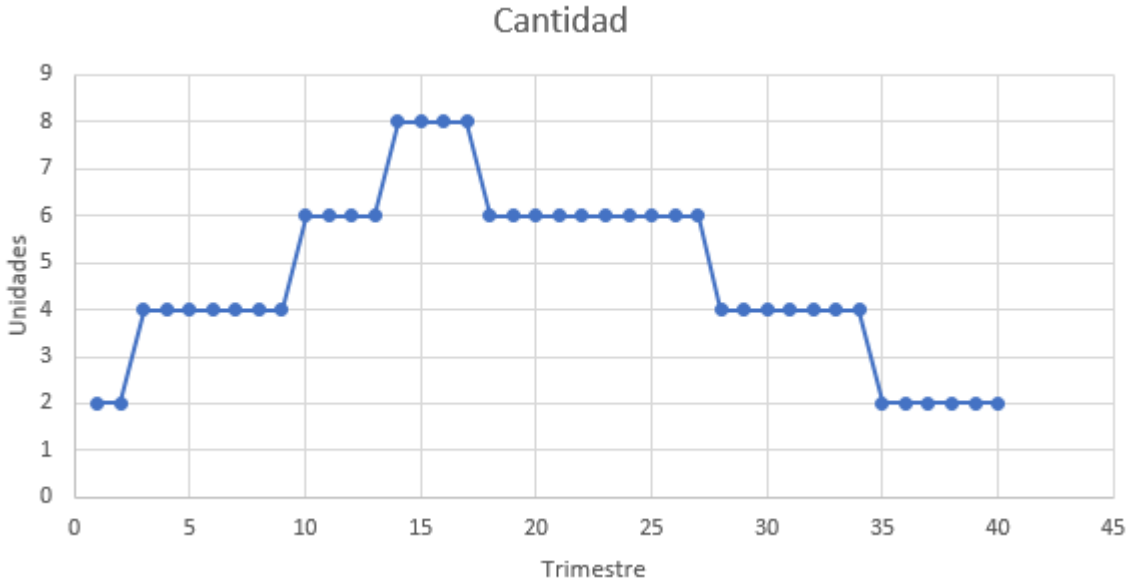


Figura 18: Demanda estimada para el producto.

Se observa que la demanda es creciente hasta el trimestre 17 y luego comienza a decrecer. Se estima que en ese momento el mercado va a empezar a requerir mayores tasas de velocidad de envío a menor costo. Sin embargo, debido al diseño modular del aparato se pueden analizar nuevas implementaciones de los módulos transmisores y receptores sin tener que realizar un rediseño masivo del producto. Esto implica un costo de desarrollo menor a futuro de las próximas iteraciones del producto.

A continuación se puede encontrar el flujo de fondos del proyecto para los primeros diez años. Se realiza un análisis del flujo de fondo a diez años para analizar la duración óptima del proyecto.

Actividades	Etapa 1				
	Año 1	Año 2	Año 3	Año 4	Año 5
Ingresos	$ 6,000.00	$ 8,000.00	$11,000.00	$ 15,000.00	$ 13,000.00
Costos Variables	$ (816.72)	$ (1,088.96)	$ (1,497.32)	$ (2,041.80)	$ (1,769.56)
Ingresos Brutos	$ (5,183.28)	$ (6,911.04)	$ (9,502.68)	$(12,958.20)	$(11,230.44)
Impuesto a las ganancias	$ (1,528.30)	$ (2,037.73)	$ (2,801.88)	$ (3,820.74)	$ (3,311.31)
Impuesto a ingresos brutos	$ (155.50)	$ (207.33)	$ (285.08)	$ (388.75)	$ (336.91)
Comisión de venta	$ (237.00)	$ (316.00)	$ (434.50)	$ (592.50)	$ (513.50)
Flujo de caja	$ 3,417.98	$ 4,557.31	$ 6,266.30	$ 8,544.96	$ 7,405.63
Desarrollo Pagina web	$ (200.00)	$ -	$ -	$ -	$ -
Mantenimiento Pagina Web	$ (80.00)	$ (80.00)	$ (80.00)	$ (80.00)	$ (80.00)
Marketing	$ (1,000.00)	$ (1,000.00)	$ (1,000.00)	$ (1,000.00)	$ (1,000.00)
Alquiler	$ (4,800.00)	$ (4,800.00)	$ (4,800.00)	$ (4,800.00)	$ (4,800.00)
Flujo de fondos	$ (2,662.02)	$ (1,322.69)	$ 386.30	$ 2,664.96	$ 1,525.63
Flujo de fondos acumulados	$ (2,662.02)	$ (3,984.70)	$ (3,598.40)	$ (933.44)	$ 592.19

Figura 19: Flujo de fondos del primer período.

Actividades	Etapa 2				
	Año 6	Año 7	Año 8	Año 9	Año 10
Ingresos	$ 12,000.00	$11,000.00	$ 8,000.00	$ 6,000.00	$ 4,000.00
Costos Variables	$ (1,633.44)	$ (1,497.32)	$ (1,088.96)	$ (816.72)	$ (544.48)
Ingresos Brutos	$ (10,366.56)	$ (9,502.68)	$ (6,911.04)	$ (5,183.28)	$ (3,455.52)
Impuesto a las ganancias	$ (3,056.59)	$ (2,801.88)	$ (2,037.73)	$ (1,528.30)	$ (1,018.86)
Impuesto a ingresos brutos	$ (311.00)	$ (285.08)	$ (207.33)	$ (155.50)	$ (103.67)
Comisión de venta	$ (474.00)	$ (434.50)	$ (316.00)	$ (237.00)	$ (158.00)
Flujo de caja	$ 6,835.97	$ 6,266.30	$ 4,557.31	$ (3,417.98)	$ 2,278.66
Desarrollo Pagina web	$ -	$ -	$ -	$ -	$ -
Mantenimiento Pagina Web	$ (80.00)	$ (80.00)	$ (80.00)	$ (80.00)	$ (80.00)
Marketing	$ (1,000.00)	$ (1,000.00)	$ (1,000.00)	$ (1,000.00)	$ (1,000.00)
Alquiler	$ (4,800.00)	$ (4,800.00)	$ (4,800.00)	$ (4,800.00)	$ (4,800.00)
Flujo de fondos	$ 955.97	$ 386.30	$ (1,322.69)	$ (2,462.02)	$ (3,601.34)
Flujo de fondos acumulados	$ 1,548.16	$ 1,934.46	$ 611.78	$ (1,850.24)	$ (5,451.58)

Figura 20: Flujo de fondos del segundo período.

Analizando el flujo de fondos acumulados en primera instancia se observa que el rendimiento del proyecto no es muy alto. Sin embargo existe un período de tiempo en el que el proyecto es rentable. Se puede utilizar esta información para determinar la duración óptima del proyecto y el mejor momento para la introducción de una nueva tecnología. Esto se puede analizar en la figura 21 donde se puede ver el flujo de fondos del proyecto a 10 años y el VAN del proyecto a diez años. Para el cálculo del VAN se utilizó la tasa de plazo fijo del banco Santander Río en dólares al día 8 de agosto del 2017.

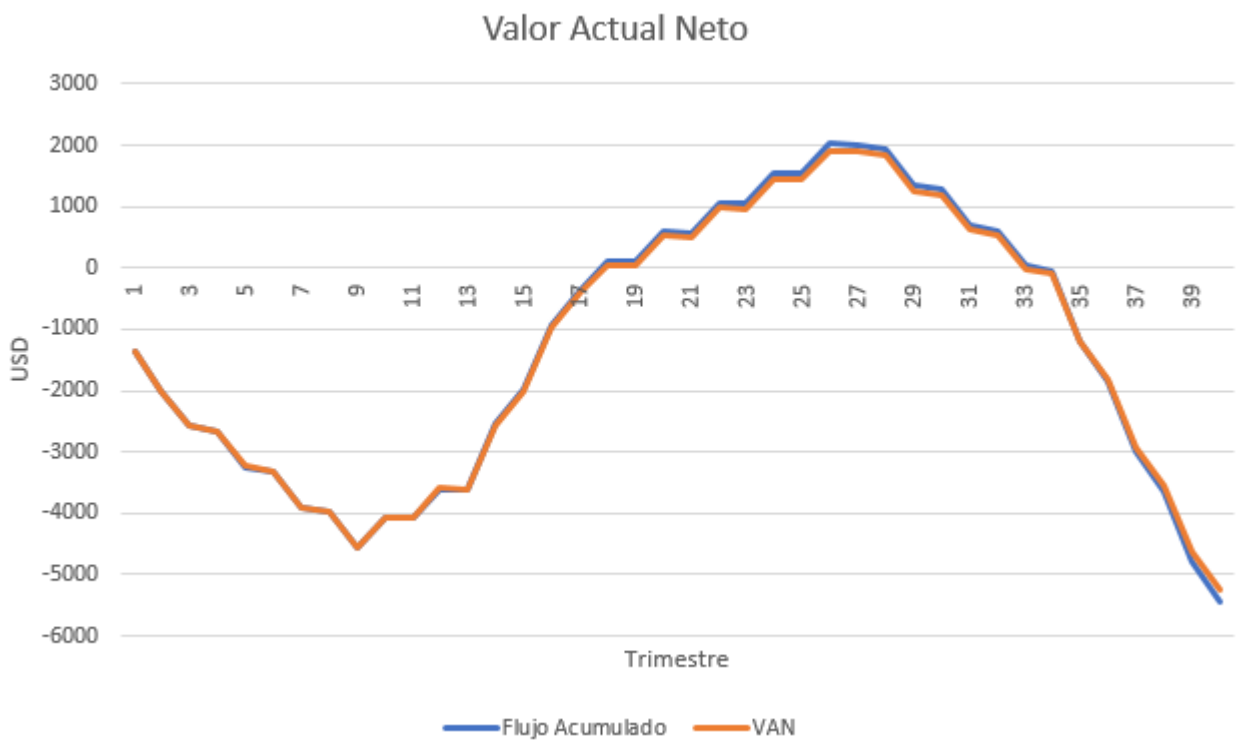


Figura 21: Proyección del proyecto a diez años.

Como la tasa de interés tomada es extremadamente baja (0.05 % nominal anual) el VAN es prácticamente igual al flujo de fondos acumulado. El análisis del proyecto fue realizado en dólares para evitar la incertidumbre inflacionaria del país argentino.

Si se analiza el flujo de fondos del proyecto a largo plazo (10 años) se observa que los primeros trimestres los costos fijos son mayores que las ganancias debido a la escasa demanda. Es por este motivo que el proyecto tiene una variación negativa hasta el trimestre 10. Luego se estima que la demanda va a ir incrementando como lo indica la figura 18 hasta tener un tope en el trimestre 26. Finalmente se asume que la demanda del producto va a ir decrementando con los años debido a que nuevas tecnologías y costos más baratos van a incentivar la migración del producto a otras tecnologías.

Se observa que la duración del proyecto óptima se encuentra entre los 6 años y los 8 años, aunque nunca se logran ganancias con el proyecto. Para contrarrestar la caída de la demanda y ajustarse a las necesidades futuras del mercado, se debería comenzar el desarrollo de un nuevo producto al año 4 del proyecto para ser introducido al mercado entre el año 8 y el año 9.

Si se analiza la TIR se puede observar que durante la duración del proyecto hay momentos en los cuales la TIR es positiva y algunos en los cuales la TIR es negativa. Esto significa que el proyecto tiene momentos en los cuales el rendimiento es mayor a cero y momentos en los cuales el proyecto produce pérdidas.

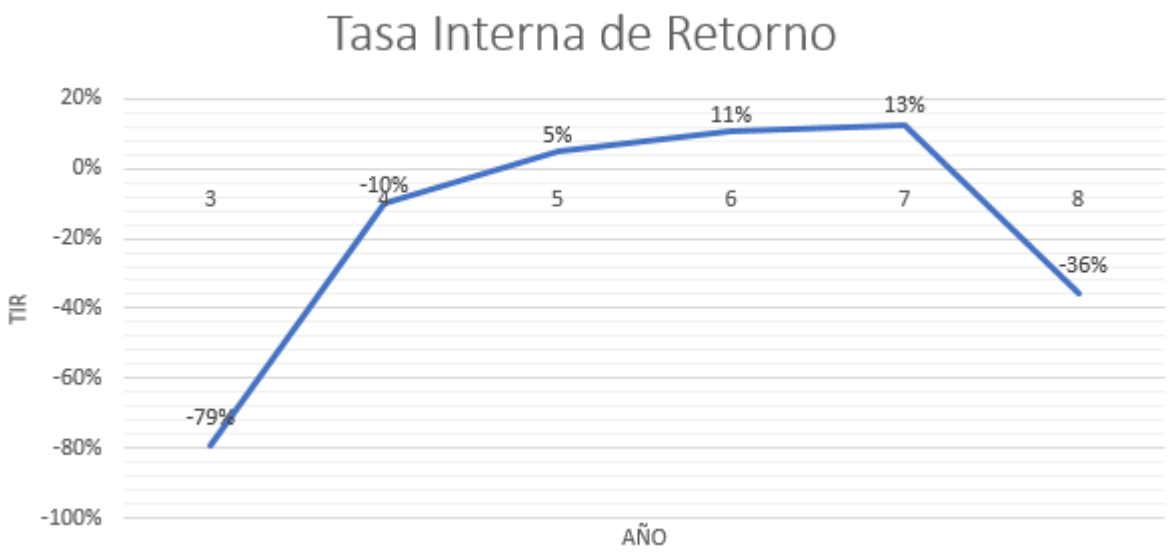


Figura 22: Tasa Interna de Retorno Anual.

Se puede observar que la TIR del proyecto es mayor a la tasa de interés ofrecida por el Banco Santander Río (0.05 %) al momento de la comparación. Sin embargo se puede ver que la TIR decrece rápidamente debido a la caída de la demanda. Para hacer un análisis con mayor detalle en la figura se puede encontrar la TIR trimestral del proyecto.

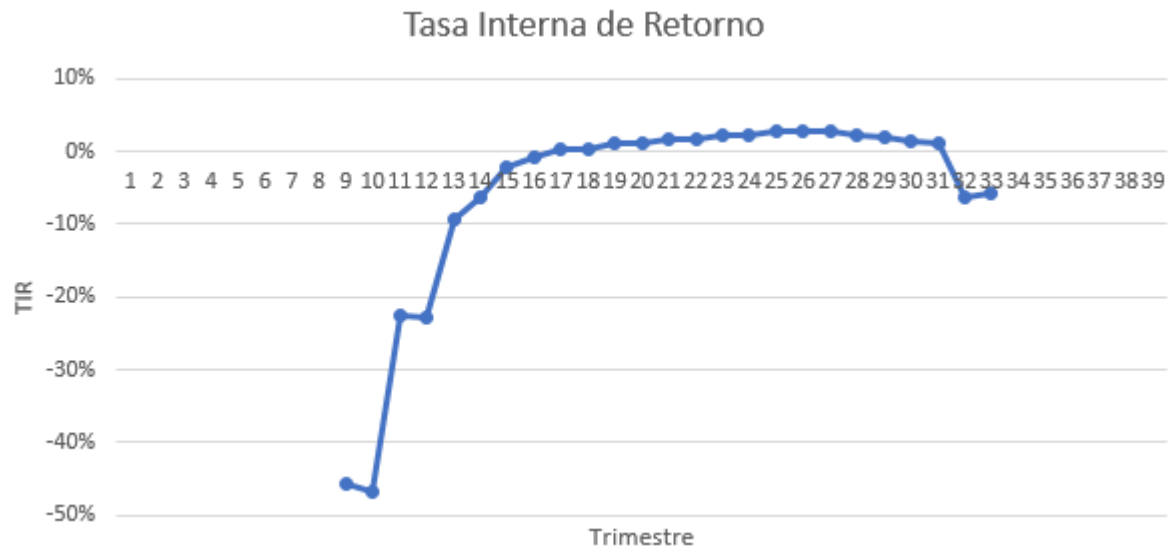


Figura 23: Tasa Interna de Retorno Trimestral.

Si se analiza en mayor detalle la TIR en los meses que el flujo de fondos es positivo, se puede ver que la TIR va aumentando a medida que aumenta la demanda y el flujo de fondos es cada vez mayor. Esto sirve como parámetro cuán sensible es el retorno en base a la demanda y si se puede capturar y fidelizar clientes desde el comienzo del proyecto, entonces la TIR será mayor.

Si bien el rendimiento del proyecto no siempre es positivo, se debe considerar al producto no solamente como un fin en sí mismo sino como la entrada y posicionamiento en un mercado en pleno crecimiento y que va a presentar una mayor capacidad y explotación a futuro.

Se debe destacar que el consumo del producto se estima que decrezca por inclusiones de nuevas tecnologías y desarrollos de la industria. Sin embargo también esto implica que se abaratarán los costos de fabricación y que al haber penetrado el mercado en el momento de crecimiento, se obtendrá una ventaja competitiva al momento de que el mercado llegue a un nivel de maduración mayor.

Además se debe recordar que el producto es considerado como un pie de entrada a la industria permitiendo diferentes oportunidades de desarrollo a futuro.

12.5. Oportunidades de desarrollo

La vida de un nanosatélite en órbita baja es de un año promedio. La finalidad de este producto no es solamente desarrollar un sistema de comunicaciones para radioaficionados y emprendedores amateur sino que también busca involucrarse directamente con los futuros jugadores clave en el mercado, proveyendo desde el principio una relación sólida y productos que ayuden a impulsar emprendimientos.

De esta forma no se debe considerar el flujo de fondos como un resultado en sí mismo. El crecimiento de los clientes será lo que de un puntapié al desarrollo de nuevos productos y módulos nanosatelitales de acuerdo a la evolución generada por esta revolución en el mercado.

12.5.1. Desarrollo de nanosatétites

Se puede analizar como oportunidad de desarrollo el desarrollo del nanosatélite completo en lugar solo una de sus partes.

De acuerdo a los análisis anteriores, se estima que el precio que un usuario final compraría un nanosatélite es de 2.500 USD. Además se estima que el costo del sistema de comunicaciones es la quinta parte del costo total del nanosatélite.

Como el costo de desarrollo del sistema de comunicaciones es de 76.760 USD, se estima que el desarrollo del nanosatélite completo será de 383.800 USD. Adicionalmente el costo variable de cada producto será de 340,3 USD. Se estima que los costos fijos no se verán modificados.

Utilizando estas cifras se puede estimar el costo de desarrollo del nanosatélite completo así como las ganancias en base a la cantidad de nanosatélites vendidos, asumiendo que los costos fijos se mantienen para un proyecto de duración de 10 años.

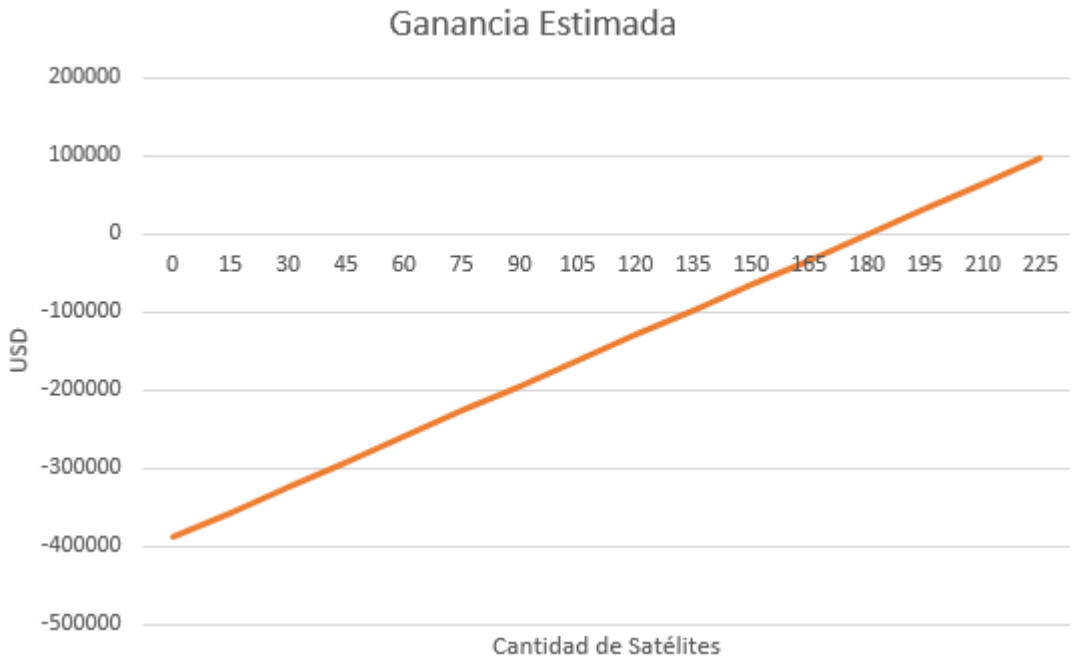


Figura 24: Ganancia estimada de nanosatélites.

En la figura 24 se puede observar que la cantidad de nanosatélites vendidos en 10 años para que el proyecto sea rentable debe ser mayor a 180 nanosatélites. Si se analiza la suma de la cantidad de la figura 18 se estima que la cantidad de sistemas de comunicación vendidos sea 188 en 10 años, lo que hace considerar que es económicamente más viable el desarrollo del nanosatélite completo.

Además el desarrollo del nanosatélite completo puede proponer mayores oportunidades a largo plazo a medida que los clientes crezcan y requieran modificaciones del sistema y grandes cantidades de nanosatélites para crear constelaciones. Otra oportunidad que puede surgir del desarrollo del nanosatélite completo es que la empresa envíe al espacio una constelación propia y venda servicios a otras empresas más pequeñas que no poseen el conocimiento ni el capital para hacerlo.

Parte V
Ingeniería de detalle

13. Diagrama modular

En la figura 25 se muestra un diagrama básico de la composición del sistema de comunicaciones. El fin de dicho diagrama es identificar rápidamente algunos de los módulos que componen el sistema de comunicación, como así también su tipo de implementación. De dicha figura se entiende que habrá tres grandes bloques implementados en hardware, siendo los mismos el receptor, el transmisor y el microprocesador. El último, dentro de sí alberga tres módulos de software denominados módulo de procesamiento, módulo decodificador y módulo codificador. La interfaz de comunicación entre el microprocesador y el receptor, al igual que la del transmisor, será SPI. A su vez el microprocesador puede recibir información del resto del satélite por comunicación UART. El sistema de comunicación se encenderá si el interior del satélite lo determina, aunque puede ser apagado del exterior en caso de ser necesario un apagado externo. Otra posible comunicación del sistema de comunicación con el resto del satélite es una interfaz GPIO, mediante el cual se comunica si posee un error o si se recibió un paquete de datos. Ambas interfaces consisten de un pin cada una cuyo funcionamiento consiste en una bandera , si está encendido hay paquetes disponibles para leer o hay un error, y si está apagado no hay paquetes por leer o no hay error.

Figura 25: Diagrama básico de la composición del sistema de comunicaciones.

Como se mencionó previamente, los módulos de la figura 25 son los más importantes. El receptor es el encargado de recibir los mensajes, derivarlos al decodificador, luego se procesan los datos, y si es necesaria una respuesta, se codificará la respuesta para luego ser transmitida por el transmisor.

13.1. Elección de Hardware

13.1.1. Microprocesador

El microprocesador elegido para el desarrollo del sistema de comunicaciones es el MK64FN1M0VLL12 de NXP. El factor que más pesó a la hora de su elección fue la experiencia previa de los autores en otros proyectos. El objetivo es acelerar la implementación del proyecto y se evitar errores sistemáticos que no estén relacionado con la implementación del proyecto en sí. Si bien en principio parece un microprocesador demasiado equipado para la aplicación, es importante que para diseños futuros tampoco será una limitación o algo que se deba modificar. Algunas de sus características son: frecuencia de operación 120Mhz, memoria interna SRAM de 256kB, memoria FLASH de 1024kB, 6 canales de UART , 3 canales de SPI , 3 canales de I2C, 1 canal de I2S, 1 canal USB , 1 canal CAN , 66 GPIOs , 2 ADCs (16 bits),1 DACs (12 bits), 20 canales de PWM , 4 timers de 32 bits, tensión de operación máxima de 3.6V, tensión de operación mínima de 1.71V, rango de temperatura ambiente de operación de -40 a 105C. La cualidad a destacar en un principio es el rango térmico de temperatura ambiente de funcionamiento, el cual se encuentra dentro del necesario. Otra cualidad a destacar es el número de interfaces disponibles, el cual permitiría en un futuro asociar algún sensor directamente al sistema de comunicaciones o incrementar la complejidad del diseño.

13.1.2. Receptor y Transmisor

A la hora de seleccionar el transmisor y el receptor lo que se decidió utilizar fue un transreceptor para cada uno de ellos por dos motivos. El primero, una oportunidad para que en un futuro se pueda vender un producto con otras frecuencias de *uplink* y *downlink* o un producto con frecuencias programables por el usuario. El segundo motivo es que al utilizar dos módulos del mismo fabricante se garantizaba una alta compatibilidad del software para ambos módulos pudiendo reutilizar el software desarrollado.

Los módulos utilizados son un módulo RFM95W y un RFM96W, ambos de Hoperf. El primero posee un rango de frecuencias de 868 a 915 Mhz, mientras que el segundo de 433 a 470 Mhz. Ambos poseen una sensibilidad programable entre -11 a -148 dBm, una velocidad efectiva de 0.018 a 37.5 kbps, una potencia programable de salida de hasta 17 dBm y un rango de temperatura ambiente de funcionamiento de -55C a 115C.

Finalmente las antenas seleccionadas fueron DAA043SA0100N y GSM3DB027. Ambas poseen conector SMA , su potencia máxima de entrada es 40 dBm, tienen una impedancia de 50 Ohm y una ganancia de 2.5 dBi. Lo más importante es que ambas son omnidireccionales, factor que es importante dado que se desconoce la posición final del sistema de comunicaciones, y su rango de temperatura ambiente de operación es de -40C a 105C.

13.2. Diagrama de interfaces

A continuación se presenta un análisis modular más detallado, analizando todos los módulos, explicando las interfaces entre cada uno y la verificación del funcionamiento del sistema.

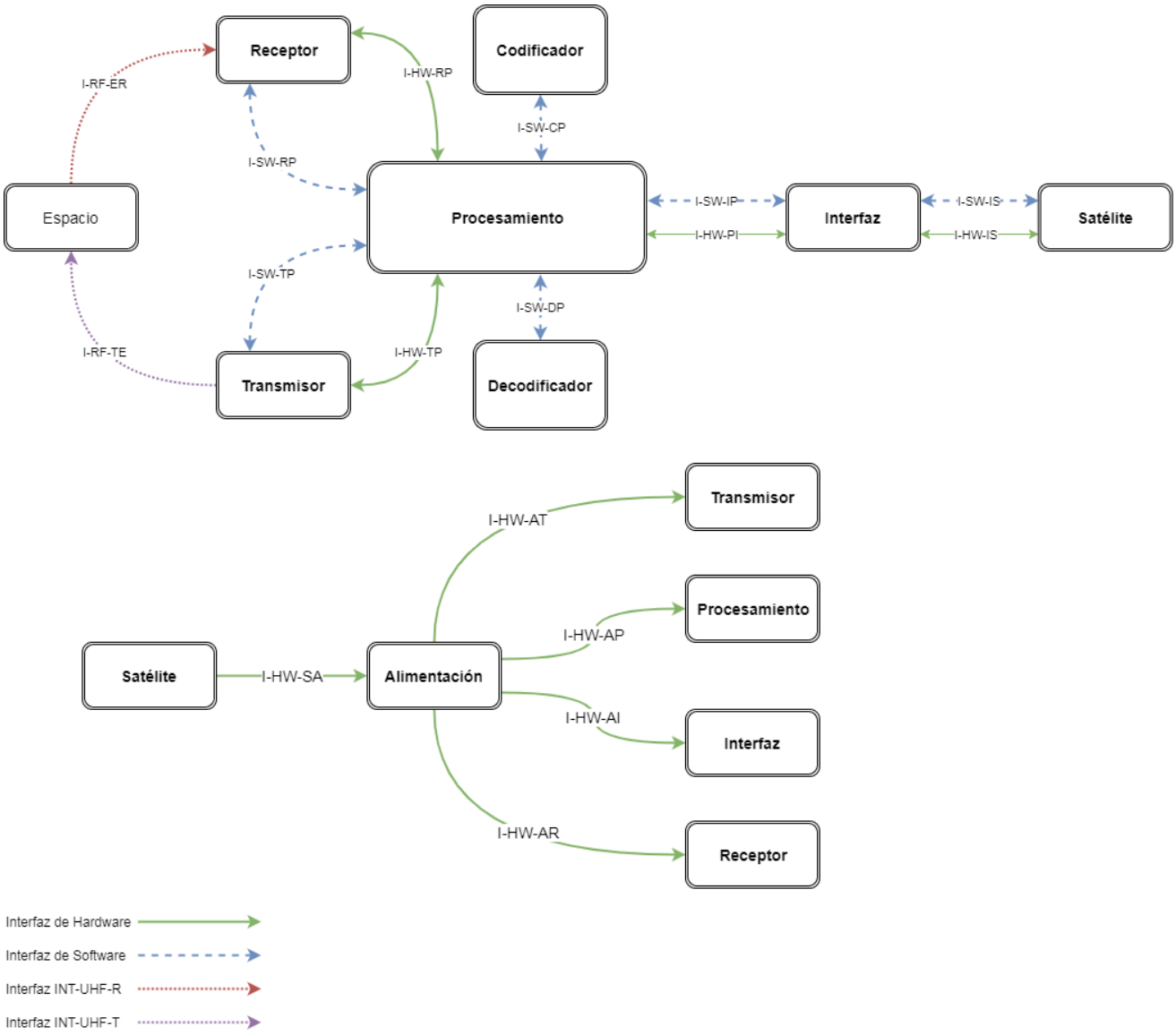


Figura 26: Diagrama modular del proyecto.

El sistema de comunicación se encuentra compuesto por siete módulos: Alimentación, Procesamiento, Transmisor, Receptor, Interfaz, Codificador y Decodificador.

El módulo de Procesamiento se encarga del funcionamiento, control y manejo de las interfaces del sistema de comunicaciones. El módulo de Procesamiento se compone de una implementación en Hardware, contando con un microprocesador, una interfaz SPI para comunicación con los módulos Receptor y Transmisor, un regulador de tensión, un conversor analógico digital para medición de la tensión de alimentación del sistema, y conectores I/O para comunicación con el satélite y con los módulos Transmisor y Receptor. Además, el módulo de Procesamiento posee una máquina de estados montada sobre el microprocesador que se encarga del manejo de la comunicación con el satélite, manejo de los datos recibidos desde la Tierra por el receptor, manejo del envío de datos desde el satélite a la Tierra y el diagnóstico de errores. El módulo de Procesamiento utiliza la interfaz de Hardware UART I-HW-IS y la interfaz de Software I-SW-IS para la comunicación con el satélite. La interfaz de Software I-SW-IS representa un intérprete del protocolo de comunicación TLV definido entre el satélite y el sistema de comunicación, y la interfaz I-HW-IS es el medio físico para el intercambio de señales eléctricas.

Cuando el satélite desea enviar datos hacia el espacio, le envía al sistema de comunicaciones mediante la interfaz de Hardware I-HW-IS la información utilizando el protocolo apropiado. La interfaz de Software I-SW-IS

es la encargada de la interpretación del mensaje, la cual va a ser tratada por una máquina de estados del módulo de Procesamiento. El módulo de Procesamiento debe enviar los datos al módulo Codificador por la interfaz de Software I-SW-CP y luego debe utilizar la interfaz de Software I-SW-TP para el envío de los datos al módulo Transmisor. Este envío debe utilizar la interfaz física de Hardware I-HW-TP. Esta interfaz física se encuentra implementada utilizando una interfaz de comunicación serie SPI.

El módulo Transmisor se encarga de la conversión de los datos de una interfaz de comunicación de Hardware de baja frecuencia (I-HW-TP) a una interfaz de comunicación de radiofrecuencia (I-RF-TE). Para realizar esto, el módulo posee un modulador M-FSK, un amplificador de señal y una antena omnidireccional.

Cuando el sistema de comunicación recibe datos del espacio, el módulo Receptor se encarga de la transformación de la señal de datos de radiofrecuencia de la interfaz I-RF-ER a una señal eléctrica que se guarda en una memoria interna del Receptor. El Receptor informa al módulo de Procesamiento que posee datos recibidos mediante la interfaz de Hardware I-HW-RP. El módulo de Procesamiento debe enviar los datos recibidos del Receptor al módulo Decodificador a través de la interfaz de Software I-SW-DP. El módulo de Procesamiento debe enviar los datos al módulo Interfaz mediante la interfaz I-SW-IP. El módulo Interfaz se encarga de transformar los datos al protocolo TLV apropiado para la comunicación con el satélite utilizando la interfaz de Software I-SW-IS y el protocolo de Hardware UART mediante la interfaz I-HW-IS.

El módulo de Procesamiento además se encarga de detectar algunos errores en el funcionamiento común del sistema de comunicaciones. Los errores que busca son los siguientes: exceso de temperatura, error de conexionado con los módulos Transmisor y Receptor, error de funcionamiento de los módulos Transmisor y Receptor, exceso de alimentación y nivel bajo de alimentación. El exceso de temperatura se mide utilizando un sensor de temperatura. El nivel de alimentación debe ser monitoreado utilizando un conversor analógico digital. El módulo de Procesamiento debe intentar comunicarse con los chips de transmisión y recepción para verificar su correcto funcionamiento y asegurar que no haya ocurrido alguna ruptura en la línea física de comunicación.

El módulo de Alimentación se encarga de unificar las líneas de alimentación de todo el sistema de comunicación. De esta manera, se logra reducir el nivel de ruido en el sistema ya que los diferentes circuitos integrados del sistema afectan en menor medida la alimentación del resto. Además el módulo de Alimentación se encarga de regular la tensión de alimentación y de ajustar los niveles de tensión de acuerdo al requerimiento de cada uno de los circuitos integrados del sistema.

Las interfaces internas que utiliza el módulo para Hardware, RF y Software se pueden ver detalladas en las tablas 22, 23 y 24 respectivamente. En el caso de las interfaces de hardware de alimentación los valores de tensiones máximas y mínimas no hacen referencia a los valores de operación sino que a los valores que permiten que los circuitos integrados modifiquen su funcionamiento en condiciones normales de operación.

Interfaz	Impedancia (Ohm)	Tensión máxima (V)	Tensión mínima (V)	Corriente máxima (mA)	Corriente mínima (mA)
I-HW-RP	50 ± 5	3.6	-0.3	25	-25
I-HW-PT	50 ± 5	3.6	-0.3	25	-25
I-HW-PI	50 ± 5	3.6	-0.3	25	-25
I-HW-IP	50 ± 5	3.6	-0.3	25	-25
I-HW-SI	50 ± 5	3.6	-0.3	25	-25
I-HW-IS	50 ± 5	3.6	-0.3	25	-25
I-HW-AR	50 ± 5	3.9	-0.5	120	50
I-HW-AT	50 ± 5	3.9	-0.5	120	100
I-HW-AP	50 ± 5	3.8	-0.3	400	100
I-HW-AI	50 ± 5	5	-0.3	185	-25

Cuadro 22: Interfaces de Hardware

Interfaz	Velocidad de transmisión	BER	Frecuencia Máxima (MHz)	Frecuencia Mínima (MHz)	Modulación
I-RF-ER	110 bps	< 1e-4	928	902	M-FSK
I-RF-TE	110 bps	< 1e-4	450	430	M-FSK

Cuadro 23: Interfaces de RF

Interfaz	Velocidad de transmisión	BER	Protocolo	Resolución de bits
I-SW-DP	10 Mbps	-	-	16
I-SW-PD	10 Mbps	-	-	16
I-SW-CP	10 Mbps	-	-	16
I-SW-PC	10 Mbps	-	-	16
I-SW-RP	250 kbps	<1e-8	SPI	8
I-SW-TP	250 kbps	<1e-8	SPI	8
I-SW-IP	9600 bps	<1e-8	UART	8
I-SW-IS	9600 bps	<1e-8	UART	8

Cuadro 24: Interfaces de Software

Los detalles de cada módulo será detallado en el desarrollo de la presente sección.

14. Ingeniería de detalle del Hardware

14.1. Módulo Transmisor

14.1.1. Descripción detallada

El módulo de transmisor es el encargado de la transformación de la señal codificada en ondas electromagnéticas las cuales serán enviadas a la Tierra. El mismo deberá ser capaz de transformar el mensaje a transmitir que se encuentra de forma digital a una forma analógica la cual cumpla con los requerimientos establecidos anteriormente. Los componentes físicos que componen a este módulo son la antena de transmisión y el sistema modulador.

14.1.2. Requerimientos y especificaciones

El módulo deberá poseer las siguientes interacciones con el resto del sistema.

	In/Out	Medio	Interfaz
Procesamiento	In	Pista de cobre	I-HW-TP
Procesamiento	In/Out	Software	I-SW-RP
Espacio	Out	Ondas EM	I-RF-TE

Cuadro 25: Tabla de entradas/salidas.

Las especificaciones que se relacionan con el módulo se encuentran detalladas a continuación.

Especificación	Parámetro	Detalle
FUN-02	Frecuencia Transmisor	430 a 450 MHz
FUN-04	Potencia Mínima Transmisor	14dBm
FUN-05	Velocidad de Transferencia	>110 bps

Cuadro 26: Tabla de especificaciones.

Para corroborar el correcto funcionamiento de las especificaciones mencionadas previamente, se realizarán las siguientes verificaciones.

Especificación	Verificación	Detalle	Valor esperado
FUN-02	Se verificará la frecuencia de transmisión.	Se utilizará un analizador de espectro para determinar bajo condiciones normales de trabajo el desvío de la frecuencia portadora, y la frecuencia máxima y mínima del espectro utilizado.	[430,1- 449.9] MHz
FUN-04	Se verificará la potencia de salida en función de la alimentación.	Se utilizará un analizador de espectro para determinar la potencia de la señal de salida a tensión mínima de entrada, tensión normal de entrada y tensión máxima de entrada.	$> 14,1\ dBm$
FUN-05	Velocidad de Transferencia	Utilizando dos módulos de comunicación idénticos, con las frecuencias de receptor y transmisor invertidas, se establecerá una comunicación para verificar la velocidad de transferencia del sistema.	$> 110 bps$

Cuadro 27: Tabla de verificaciones.

14.2. Módulo Receptor

14.2.1. Descripción detallada

El módulo de recepción es el encargado de la transformación de la señal electromagnética en señales digitales las cuales serán enviadas al sistema de decodificación. El mismo deberá ser capaz de transformar el mensaje teniendo una gran sensibilidad para recibir las señales enviadas desde la tierra. Los componentes físicos que componen a este módulo son la antena del receptor y el sistema demodulador.

14.2.2. Requerimientos y especificaciones

El módulo deberá poseer las siguientes interacciones con el resto del sistema.

	In/Out	Medio	Interfaz
Procesamiento	Out	Pista de cobre	I-HW-RP
Procesamiento	In/Out	Software	I-SW-RP
Espacio	In	Ondas EM	I-RF-ER

Cuadro 28: Tabla de entradas/salidas.

Las especificaciones que se relacionan con el módulo se encuentran detalladas a continuación.

Especificación	Parámetro	Detalle
FUN-03	Frecuencia Receptor	902 a 928 MHz
FUN-06	Sensibilidad	< -120dBm
FUN-05	Velocidad de Transferencia	>110 bps

Cuadro 29: Tabla de especificaciones.

Para corroborar el correcto funcionamiento de las especificaciones mencionadas previamente, se realizarán las siguientes verificaciones.

Especificación	Verificación	Detalle	Valor esperado
FUN-03	Se verificará la frecuencia de recepción.	Se utilizará un analizador de espectro para determinar bajo condiciones normales de trabajo el desvío de la frecuencia portadora, y la frecuencia máxima y mínima del espectro utilizado.	[902,1 - 927,9] MHz
FUN-06	Se verificará la sensibilidad en función de la alimentación.	Se utilizará un módulo transmisor para verificar la sensibilidad del módulo. Se verificará la misma bajo la tensión mínima y la tensión máxima de funcionamiento. Se variará la potencia de salida del transmisor así como el desacople de las antenas para determinar la sensibilidad.	$< -120dBm$[3]
FUN-05	Velocidad de Transferencia	Utilizando dos módulos de comunicación idénticos, con las frecuencias de receptor y transmisor invertidas, se establecerá una comunicación para verificar la velocidad de transferencia del sistema.	$> 110bps$

Cuadro 30: Tabla de verificaciones.

14.3. Módulo Alimentación

14.3.1. Descripción detallada

El módulo de alimentación se encarga de proveer energía al sistema de comunicaciones.

14.3.2. Requerimientos y especificaciones

El módulo deberá poseer las siguientes interacciones con el resto del sistema.

[2]Sensibilidad utilizada para el cálculo del enlace.

	In/Out	Medio	Interfaz
Satélite	In	Pista de cobre	I-HW-SA
Procesamiento	Out	Pista de cobre	I-HW-AP
Receptor	Out	Pista de cobre	I-HW-AR
Transmisor	Out	Pista de cobre	I-HW-AT

Cuadro 31: Tabla de entradas/salidas.

Las especificaciones que se relacionan con el módulo se encuentran detalladas a continuación.

Especificación	Parámetro	Detalle
INT-VIN-01	Alimentación del sistema de comunicación.	[4.5V - 6V]
INT-VIN-02	Consumo pico del sistema de comunicaciones.	$<$ 330 mA
INT-VIN-03	Protección de ESD.	$>$ 2kV
INT-VIN-04	Protecciones contra cortocircuitos o sobrecalentamientos.	-
INT-VIN-05	Valor pico de corriente de inrush.	$<$ 680 mA

Cuadro 32: Tabla de especificaciones.

Se verificará la salida del módulo en base a las entradas máximas y mínimas. Se verificará la salida máxima y mínima ante un ruido en la entrada de $\pm 10\,\% \, V_i$.

Para corroborar el correcto funcionamiento de las especificaciones mencionadas previamente, se realizarán las siguientes verificaciones.

Especificación	Verificación	Detalle	Valor esperado
INT-VIN-01	Alimentación del sistema de comunicación.	Se verificará el correcto funcionamiento del sistema de comunicaciones para diferentes tensiones de entrada. La misma será monitoreada con un multímetro.	[4.51 - 5.9]
INT-VIN-02	Consumo pico del sistema de comunicaciones.	Se utilizará un osciloscopio con una punta de corriente para monitorear el consumo del sistema de comunicaciones. Se forzará al sistema en distintos casos de funcionamiento.	$<$ 329 mA
INT-VIN-05	Valor pico de corriente de inrush.	Se utilizará un osciloscopio con una punta de corriente para monitorear el consumo del sistema de comunicaciones. Con una fuente de tensión se forzará un escalón de tensión con diferentes valores dentro del rango de operación.	$<$ 679 mA

Cuadro 33: Tabla de verificaciones.

14.4. Módulo de Interfaz

14.4.1. Descripción detallada

El módulo de Interfaz se encarga de comunicarse con el resto del satélite, permitiendo el encendido y apagado de los módulos, obtención de la información recibida por el sistema de comunicación y envío de información desde el satélite al espacio.

El módulo deberá poseer las siguientes interacciones con el resto del sistema.

	In/Out	Medio	Interfaz
Satélite	In/Out	Pista de cobre	I-HW-SI
Satélite	In/Out	Software	I-SW-SI
Procesamiento	In/Out	Software	I-SW-PI
Procesamiento	In	Pista de cobre	I-HW-PI
Procesamiento	Out	Pista de cobre	I-HW-IP

Cuadro 34: Tabla de entradas/salidas.

Las especificaciones que se relacionan con el módulo se encuentran detalladas a continuación.

Especificación	Parámetro	Detalle
INT-LFC-01	Interfaces de comunicación	UART

Cuadro 35: Tabla de especificaciones.

Para corroborar el correcto funcionamiento de las especificaciones mencionadas previamente, se realizarán las siguientes verificaciones.

Especificación	Verificación	Detalle	Valor esperado
INT-LFC-01	Se verificarán los protocolos de comunicación con el satélite.	Se realizará el envío de 100 millones de bits y se analizará el BER de la comunicación.	<1e-6

Cuadro 36: Tabla de verificaciones.

La comunicación interna del satélite posee una interfaz más robusta y con menor tasa de error comparado a las interfaces de radiofrecuencia. Por este motivo se busca diseñar la interfaz de comunicación INT-LFC con una tasa de error que resulte despreciable respecto a la tasa de error del sistema completo.

14.5. Módulo Procesamiento

14.5.1. Descripción del módulo

El módulo de procesamiento es el encargado de correr las rutinas de software.

14.5.2. Requerimientos y especificaciones

El módulo deberá poseer las siguientes interacciones con el resto del sistema.

	In/Out	Medio	Interfaz
Alimentación	In	Pista de cobre	I-HW-AP
Interfaz	In/Out	Pista de cobre	I-HW-IP
Transmisor	In/Out	Pista de cobre	I-HW-PT
Receptor	In/Out	Pista de cobre	I-HW-RP

Cuadro 37: Tabla de entradas/salidas.

Para verificar el correcto funcionamiento se evaluará el correcto funcionamiento de la máquina de estados en las condiciones de uso extremas de alimentación.

15. Ingeniería de detalle del Software

15.1. Diagrama de estados del Software

Por motivos de prolijidad y simpleza de la implementación se hizo un paralelismo entre el software y hardware. Por este motivo el módulo se encuentra implementado a través de tres máquinas de estados que corren en paralelo: una máquina de estados que se encarga de la generación e interpretación de eventos recibidos; una máquina de estados dedicada a la funcionalidad del módulo de recepción; y una máquina de estados que se encarga del manejo del módulo transmisor.

Esta implementación permite el administrar de forma simplificada cada uno de los módulos por separado sin incrementar la complejidad del software.

15.1.1. Máquina de estados principal

La máquina de estados principal es la encargada del funcionamiento continuo del sistema de comunicaciones. Posee tres estados: "Low Power", "Error" y "Stand-By". Esta máquina de estados recibe eventos generados por la comunicación con el resto de nanosatélite, por la máquina de estados de recepción y por diferentes condiciones de error.

El primer estado, "Low Power", indica que todo el sistema de comunicaciones se encuentra en modo de ahorro de energía. Este modo puede ser utilizado por el nanosatélite en el momento de despegue de ser necesario, en momentos de penumbra para minimizar el consumo de energía o en momentos que el nanosatélite esté con exceso

de consumo de energía. El nanosatélite puede poner al sistema de comunicación en modo "Low Power" utilizando el módulo "Interfaz".

El estado "Stand-By" es el estado de funcionamiento normal del sistema de comunicaciones. En este estado todos los elementos activos de hardware se encuentran alimentados, haciendo posible la recepción y envío de información al espacio. Es posible apagar alguno de los módulos de transmisión o recepción sin tener que apagar el sistema de comunicaciones en su totalidad.

La máquina de estados principal se encarga además del diagnóstico de algunos errores para que el satélite pueda tener un análisis del sistema de comunicaciones y un posible método de resolución del problema. La máquina de estados principal se encarga de diagnosticar los siguientes tipos de errores: exceso de alimentación en la entrada del sistema; un nivel menor al esperado en la entrada del sistema; exceso de temperatura en el sistema; problema de conexionado entre el microprocesador y el módulo de transmisión; problema de conexionado entre el microprocesador y el módulo de recepción; error de configuración del transmisor; y error de configuración del receptor.

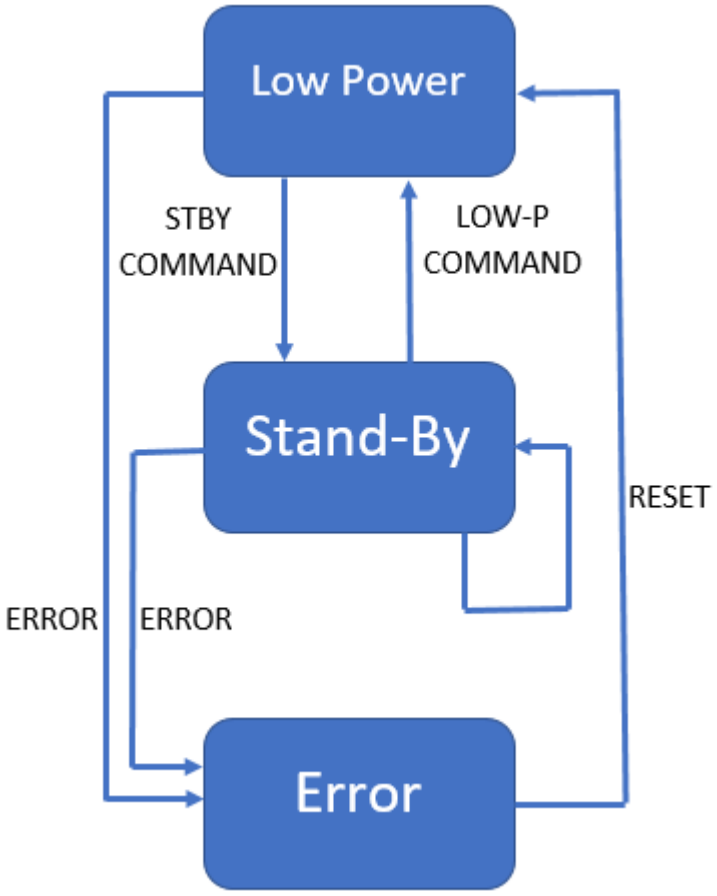


Figura 27: Máquina de estados principal

En la tabla 38 se presenta los eventos y acciones de la tabla de estados principal.

Estado Inicial	Estado Final	Evento	Acción
Low Power	Stand-By	Satélite prende el Receptor	Prende módulo de recepción
Low Power	Stand-By	Satélite prende el Transmisor	Prende módulo TX
Low Power	Stand-By	Satélite prende ambos módulos	Prende ambos módulos
Low Power	Error	Ocurre un error contemplado	Clasifica al error y envia una señal lógica de 1 al satélite utilizando el módulo Interfaz.
Stand-By	Low Power	Satélite apaga el Receptor	Apaga módulo de recepción
Stand-By	Low Power	Satélite apaga el Transmisor	Apaga módulo de transmisión
Stand-By	Low Power	Satélite apaga ambos módulos	Apaga ambos módulo
Stand-By	Stand-By	Satélite pide enviar datos utilizando la interfaz I-SW-IS	Envía datos al satélite por UART
Stand-By	Stand-By	Satélite pide enviar datos utilizando la interfaz I-SW-IS	Envía datos al módulo TX
Stand-By	Error	Ocurre un error contemplado	Clasifica al error y envia una señal lógica de 1 al satélite utilizando el módulo Interfaz.
Error	Low Power	Satélite hace un reset del sistema	Establece todas las señales lógicas del módulo Interfaz a 0 y reinicia los módulos Transmisor y Receptor

Cuadro 38: Tabla de estados principal.

Error	Descripción	Acción
Alta temperatura	La temperatura registrada es mayor a 60 grados Celsius.	El sistema de comunicaciones entra en modo "Low Power" y envía una señal lógica de 1 al satélite utilizando el módulo Interfaz.
Baja temperatura	La temperatura registrada es menor a -20 grados Celsius.	El sistema de comunicaciones entra en modo "Low Power" y envía una señal lógica de 1 al satélite utilizando el módulo Interfaz.
Exceso de alimentación	La alimentación de entrada al sistema es mayor a 6V	El sistema de comunicaciones entra en modo "Low Power" y envía una señal lógica de 1 al satélite utilizando el módulo Interfaz.
Baja alimentación	La alimentación de entrada al sistema es menor a 4V	El sistema de comunicaciones entra en modo "Low Power" y envía una señal lógica de 1 al satélite utilizando el módulo Interfaz.
Error de configuración en un módulo	La respuesta de la comunicación no es la esperada.	Se reconfigura el módulo y se lo apaga. Se envía una señal lógica de 1 al satélite utilizando el módulo Interfaz
Error de conexión con un módulo	La respuesta de la comunicación retorna el mismo valor indicando una interrupción en la conexión.	Se pone en "Low Power" la máquina de estados del módulo sin conexión y se envía una señal lógica de 1 al satélite utilizando el módulo Interfaz

Cuadro 39: Tabla de errores.

La medición de la temperatura será realizada cada 60 segundos. La medición será realizada en tres elementos de hardware del sistema: el microprocesador, el módulo de transmisión y el módulo de recepción. En caso que cualquiera de las tres mediciones se encuentre fuera del rango de trabajo, se considerará que el sistema se encuentra ante una condición de error. Dicha medición se realiza cada 60 segundo ya que de acuerdo al estudio "*Thermal modelling of the PICSAT nanosatellite platform and synergetic prestudies of the CIRCUS nanosatellite*" de la univesidad de Lulea en Suecia, al prender un sensor de 1W dentro de un nanosatelíte en órbita, sin tener en cuenta cualquier efecto de convección, la temperatura final se establece cerca de los 200 segundos posteriores al arranque del sensor. Es por esto que si se chequea la temperatura cada 60 segundos al cabo de tres mediciones se podría identificar algún problema interno, como así también al correr otros chequeos.

La medición del nivel de alimentación será realizado utilizando un conversor analógico digital que compare con un nivel de referencia. Cuando el nivel de alimentación exceda el nivel de referencia por exceso, se considerará que el sistema se encuentra ante una condición de error. De manera análoga, si el nivel de alimentación es menor que el nivel de referencia por defecto, también se considerará que el sistema se encuentra ante una condición de error.

Se revisará el conexionado con los módulos internos de forma periódica cada 60 segundos. Para validar que el estado de la conexión se hará lectura de la configuración de los módulos, conociendo el valor esperado. Dependiendo de la tecnología de los transistores utilizados en los microprocesadores, y conociendo sus salidas típicas ante un circuito abierto, se puede saber cuando una conexión se encuentra rota. En caso que el resultado sea erróneo se considerará que hay un error de conexionado.

Se revisará la configuración de los módulos internos de forma periódica cada 60 segundos. Para validar que el estado de la configuración se hará lectura de la configuración de los módulos, conociendo el valor esperado. En caso que el resultado sea erróneo se considerará que hay un error de configuración.

En el caso que el sistema detecte cualquiera de los errores mencionados en la tabla 39 entonces la máquina de estados entrará en el estado "Error". En este estado se utilizará el módulo "Interfaz" para comunicar al nanosatélite de que ha ocurrido un error. Utilizando la interfaz UART, el nanosatélite puede consultar el error y accionar en consecuencia.

15.1.2. Máquina de estados del transmisor

La máquina de estados del transmisor se encarga del control del módulo Transmisor. Esto involucra las acciones de prendido, apagado y envío de datos por la interfaz I-RF-TE.

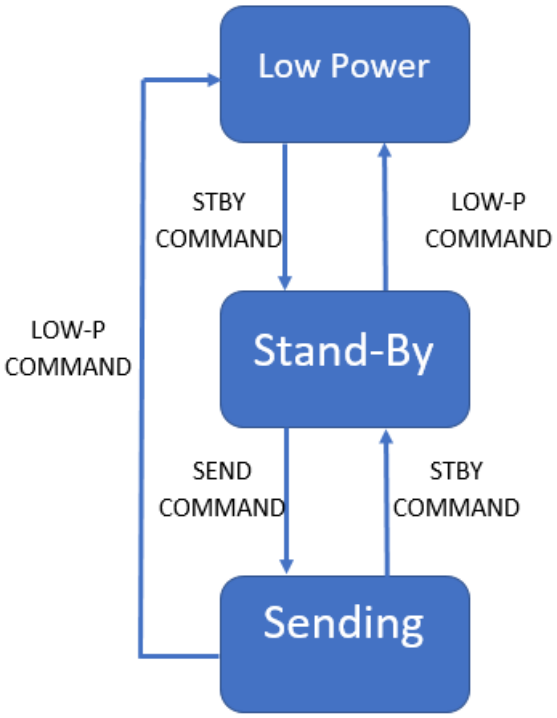


Figura 28: Máquina de estados del transmisor.

En la tabla 41 se presenta los eventos y acciones de la tabla de estados principal.

Estado Inicial	Estado Final	Evento	Acción
Low Power	Stand-By	Satélite prende el Transmisor	Prender módulo de transmisión
Stand-By	Low Power	Satélite apaga el Transmisor	Apagar módulo de transmisión
Stand-By	Sending	Satélite pide enviar datos utilizando la interfaz I-RF-TE	Enviar datos utilizando la interfaz I-RF-TE
Sending	Sending	Satélite pide enviar datos utilizando la interfaz I-RF-TE	Enviar datos utilizando la interfaz I-RF-TE

Cuadro 40: Tabla de estados del transmisor..

15.1.3. Máquina de estados del receptor

La máquina de estados del receptor se encarga del control del módulo Receptor. Esto involucra las acciones de prendido, apagado y la recepción de datos por la interfaz I-RF-ER.

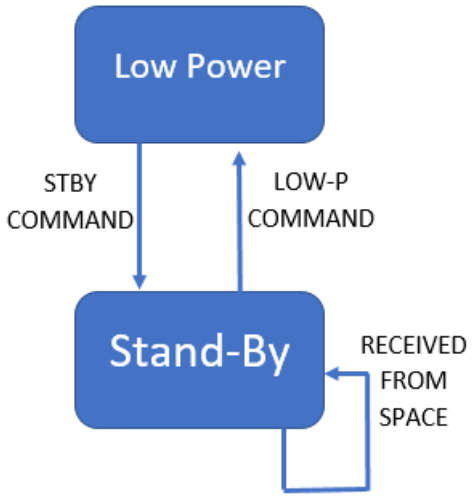


Figura 29: Máquina de estados del receptor.

En la tabla 40 se presenta los eventos y acciones de la tabla de estados principal.

Estado Inicial	Estado Final	Evento	Acción
Low Power	Stand-By	Satélite prende el Receptor	Prender el módulo
Stand-By	Low Power	Satélite apaga el Receptor	Apagar el módulo
Stand-By	Stand-By	Recibe datos por la interfaz I-RF-ER	Guarda la información en un búffer interno de 8 bits con capacidad de 1000 palabras y envía una señal lógica de 1 al satélite utilizando el módulo Interfaz.

Cuadro 41: Tabla de estados del receptor.

15.1.4. Políticas de diseño de Software

A la hora de diseñar el Software se tomaron algunas decisiones sobre la funcionalidad del prototipo y el alcance del mismo.

En lo que respecta al manejo de errores, la política tomada fue avisar mediante una señal digital al resto del satélite, y que el usuario decida qué acción debe tomar. Dicha decisión se tomó pensando en no reducir la libertad en la toma de decisiones del usuario ante cada error, dado que el sistema de comunicaciones forma parte de un sistema más grande . La señal digital se extrae por un pin, y se codifica como 1 si hay error y 0 si no lo hay. El error es clasificado, y el usuario puede consultar, en caso que lo crea necesario, dicha clasificación (INT-LFC).

Los errores pueden ser de alimentación, temperatura, transmisor, receptor o forzado de apagado exterior. El error de alimentación se detecta utilizando un conversor analógico digital monitoreando la alimentación del sistema, el error de temperatura se detecta con los sensores internos de los circuitos integrados utilizados, los errores de transmisor y receptor se detectan mediante errores de comunicación entre el microprocesador interno del sistema y los módulos de comunicación y, finalmente, el forzado de apagado externo se detecta cuando se recibe un paquete específico que indica que se debe apagar el sistema de comunicación. El forzado de apagado externo se relaciona con la especificación FUN-07.

En cuanto a la comunicación, tanto el receptor como el transmisor poseen un buffer circular de 1.000 bytes alojados en el microprocesador del sistema. Si bien el transmisor y el receptor poseen la capacidad de almacenar dicha información en sus memorias internas se decidió alojar los buffers en el microprocesador dado que en caso de que el receptor o el transmisor se deban apagar, o posean algún tipo de error, la información enviada por el usuario al sistema de comunicación no se pierde. A su vez, cuando se recibe un paquete de datos del espacio (INT-COM), se indica con una señal digital la disponibilidad de información. Los datos se leen individualmente desde el resto del satélite, apagando la señal digital en caso de vaciar el buffer de recepción.

A la hora de comunicarse desde el interior del satélite con el sistema de comunicaciones (INT-LFC), se decidió utilizar el protocolo de comunicaciones TLV que consiste en encabezado, longitud y dato. El encabezado clasifica los distintos comandos posibles, la longitud indica la longitud de datos, y luego se envían los datos que coinciden con dicha longitud. El mismo protocolo se utilizó para enviar datos desde el satélite al exterior (INT-COM). Las ventajas de éste protocolo es que es fácil de interpretar, se pueden añadir comando nuevos de forma sencilla, no posee una gran redundancia, y permite correr un primer chequeo de si se recibieron todos los datos deseados.

15.2. Análisis modular del Software

15.2.1. Módulo Codificador

Descripción de la subrutina

El módulo codificador es el encargado de convertir el mensaje digital en bits en un mensaje digital con redundancia y un método corrector de errores.

Requerimientos y especificaciones

El módulo deberá poseer las siguientes interacciones con el resto del sistema.

	In/Out	Medio	Interfaz
Procesamiento	In	Software	I-SW-PC
Procesamiento	Out	Software	I-SW-CP

Cuadro 42: Tabla de entradas/salidas.

15.2.2. Módulo Decodificador

Descripción de la subrutina

El módulo decodificador es el encargado de convertir el mensaje digital con redundancia en el mensaje digital en bits inicial.

Requerimientos y especificaciones

El módulo deberá poseer las siguientes interacciones con el resto del sistema.

	In/Out	Medio	Interfaz
Procesamiento	In	Software	I-SW-PD
Procesamiento	Out	Software	I-SW-DP

Cuadro 43: Tabla de entradas/salidas

15.2.3. Módulo Procesamiento

Descripción de la subrutina

El módulo de procesamiento es el encargado de administrar la información. El mismo se encarga de distribuir los paquetes al codificador y decodificador para que los procesen y luego los envía al transmisor y al receptor según corresponda.

Requerimientos y especificaciones

El módulo deberá poseer las siguientes interacciones con el resto del sistema.

	In/Out	Medio	Interfaz
Codificador	In	Software	I-SW-CP
Codificador	Out	Software	I-SW-PC
Decodificador	In	Software	I-SW-DP
Decodificador	Out	Software	I-SW-PD
Interfaz	In/Out	Software	I-SW-IP
Transmisor	In/Out	Software	I-SW-TP
Receptor	In/Out	Software	I-SW-RP

Cuadro 44: Tabla de entradas/salidas de Software.

16. Verificación del Sistema de comunicaciones

A continuación se detallarán las pruebas que se desarrollarán sobre el sistema de comunicaciones completo.

ID	Parámetro	Descripción	Valor Esperado
FUN-01	Comunicación Full-Dúplex	Utilizando dos módulos de comunicación idénticos, con las frecuencias de receptor y transmisor invertidas, se enviarán datos en simultáneo evaluando la comunicación.	Full dúplex
FUN-02	Frecuencia del Transmisor	Se verificará la frecuencia de transmisión. Se utilizará un analizador de espectro para determinar bajo condiciones normales de trabajo el desvío de la frecuencia portadora, y la frecuencia máxima y mínima del espectro utilizado. El banco de medición debe poseer un error menor a 0,1 MHz.	[430,1 - 449,9] MHz
FUN-03	Frecuencia del Receptor	Se verificará la frecuencia de recepción. Se utilizará un analizador de espectro para determinar bajo condiciones normales de trabajo el desvío de la frecuencia portadora, y la frecuencia máxima y mínima del espectro utilizado.	[902,1 - 927,9] MHz
FUN-04	Potencia de Salida	Se verificará la potencia de salida en función de la alimentación. Se utilizará un analizador de espectro para determinar la potencia de la señal de salida a tensión mínima de entrada, tensión normal de entrada y tensión máxima de entrada. El banco de medición debe poseer un error menor a 0,1 dBm.	> 14,1 dBm
FUN-05	Velocidad de Transferencia	Utilizando dos módulos de comunicación idénticos, con las frecuencias de receptor y transmisor invertidas, se establecerá una comunicación para verificar la velocidad de transferencia del sistema. El banco de medición debe poseer un error menor a 1 bps.	> 111 bps

Cuadro 45: Verificación del sistema de comunicaciones.

ID	Parámetro	Descripción	Valor Esperado
FUN-06	Sensibilidad	Utilizando dos módulos de comunicación idénticos, con las frecuencias de receptor y transmisor invertidas, se establecerá una comunicación para verificar la sensibilidad del sistema de comunicaciones. Se variará la potencia de salida del transmisor así como el desacople de las antenas para determinar la sensibilidad. El banco de medición debe poseer un error menor a 0,1 dBm.	< -122,5 dBm
FUN-07	Apagado del Transmisor de Forma Remota	Utilizando dos módulos de comunicación idénticos, con las frecuencias de receptor y transmisor invertidas, se establecerá una comunicación para verificar el apagado de forma remota.	Apagado de forma remota.
FUN-10	Re encendido del Sistema de Comunicaciones	Utilizando dos módulos de comunicación idénticos, con las frecuencias de receptor y transmisor invertidas, se establecerá una comunicación para verificar el apagado de forma remota y el posterior re encendido.	Re encendido de forma remota.
FUN-14	Consumo en Estado de Recepción	Utilizando un módulo de comunicaciones, se lo forzará al estado de recepción y se medirá el consumo promedio con un amperímetro en serie. El banco de medición debe poseer un error menor a 0,5 mA.	< 49,5 mA
FUN-15	Consumo en Estado de Transmisión	Utilizando un módulo de comunicaciones, se lo forzará al estado de transmisión y se medirá el consumo promedio con un amperímetro en serie. El banco de medición debe poseer un error menor a 0,5 mA.	< 329,5 mA
PER-01	BER	Se utilizará una secuencia conocida y un transmisor conocido para determinar el BER del sistema de comunicaciones bajo condiciones de alimentación máximas y mínimas.	$< 10^{-4}$

Cuadro 46: Verificación del sistema de comunicaciones.

ID	Parámetro	Descripción	Valor Esperado
INT-LFC-01	Protocolo de comunicación con el satélite.	Se realizará el envío de 100 millones de bits y se analizará el BER de la comunicación.	$< 10^{-6}$
INT-MEC-01	Verificación de las dimensiones para acople mecánico	Se utilizará un calibre para medir las dimensiones de interés. El banco de medición debe poseer un error menor a 0,1 mm.	Diámetro de perforaciones: [3,3 - 3,6] mm
DYP-01	Peso	Se utilizará una balanza de cocina con un error de 1 gramo para obtener el peso del sistema.	< 149 g
DYP-02	Tamaño	Se utilizará una regla para medir las dimensiones del sistema. El banco de medición debe poseer un error menor a 0,1 cm.	Largo < 9,9 cm Ancho < 9,9 cm Alto < 2,9 cm
MAN-05	Señalización de Falla	Se emulará una falla sobre un sistema de comunicación y se controlará el aviso al reto del satélite.	Señalización correcta
INT-VIN-01	Rango de Tensión de Operación	Se utilizará un multímetro para monitorear la tensión de entrada al sistema de comunicaciones mientras se verifica que su funcionamiento no se vea alterado frente a variaciones de tensiones de entrada. El banco de medición debe poseer un error menor a 0,1 V.	[4,6 - 5,9] V
INT-VIN-02	Consumo Pico	Utilizando una punta de corriente conectada a un osciloscopio, se medirá el consumo del sistema de comunicaciones mientras se alterna entre los distintos modos de funcionamiento. El banco de medición debe poseer un error menor a 1 mA.	< 329 mA
INT-VIN-05	Corriente de Inrush	Utilizando una punta de corriente conectada a un osciloscopio, se encenderá el módulo de comunicaciones con un escalón de tensión de 0V a 6V en un tiempo de 10 ns. El banco de medición debe poseer un error menor a 1 mA.	Corriente pico < 679 mA

Cuadro 47: Verificación del sistema de comunicaciones.

Parte VI
Construcción del Prototipo

A continuación se realiza un análisis de la construcción de dos prototipos. El primero, posee un enfoque basado en la verificación del funcionamiento del sistema, mientras que el segundo consiste en un prototipo con características de producto final.

17. Diseño de los Circuitos Impresos

17.1. Primer prototipo

A continuación se muestra el esquemático de la placa del primer prototipo. El mismo cuenta con los módulos de transmisión, conectores SMA de las antenas, pines para el interconexionado con la placa de desarrollo y pines para testeo de funcionamiento.

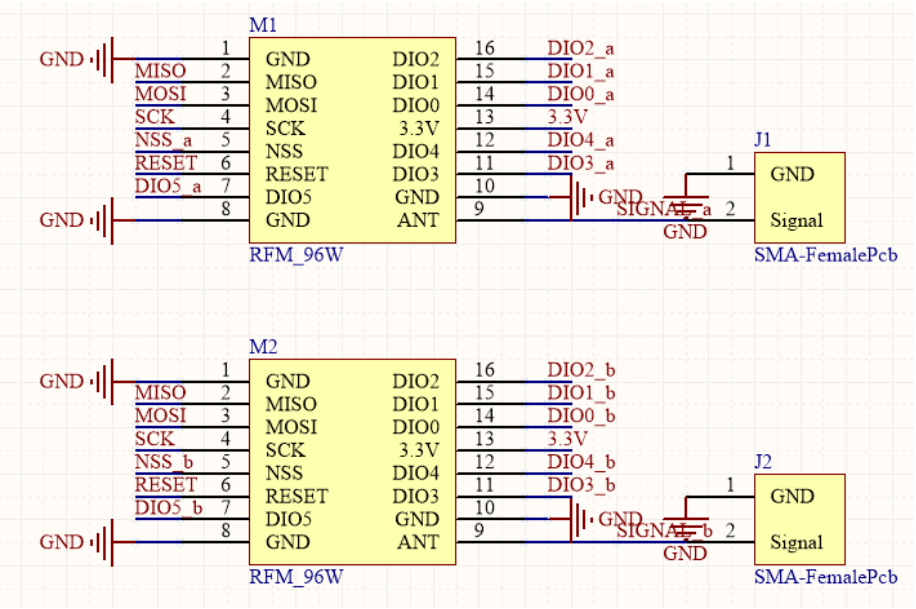


Figura 30: Esquemático de los módulos Rx y Tx.

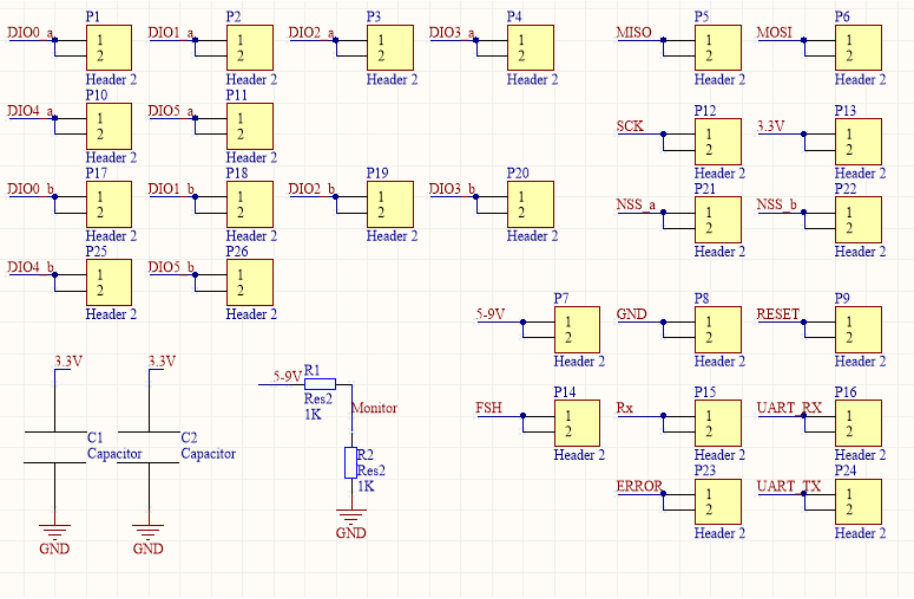


Figura 31: Pines de señalización y testeo.

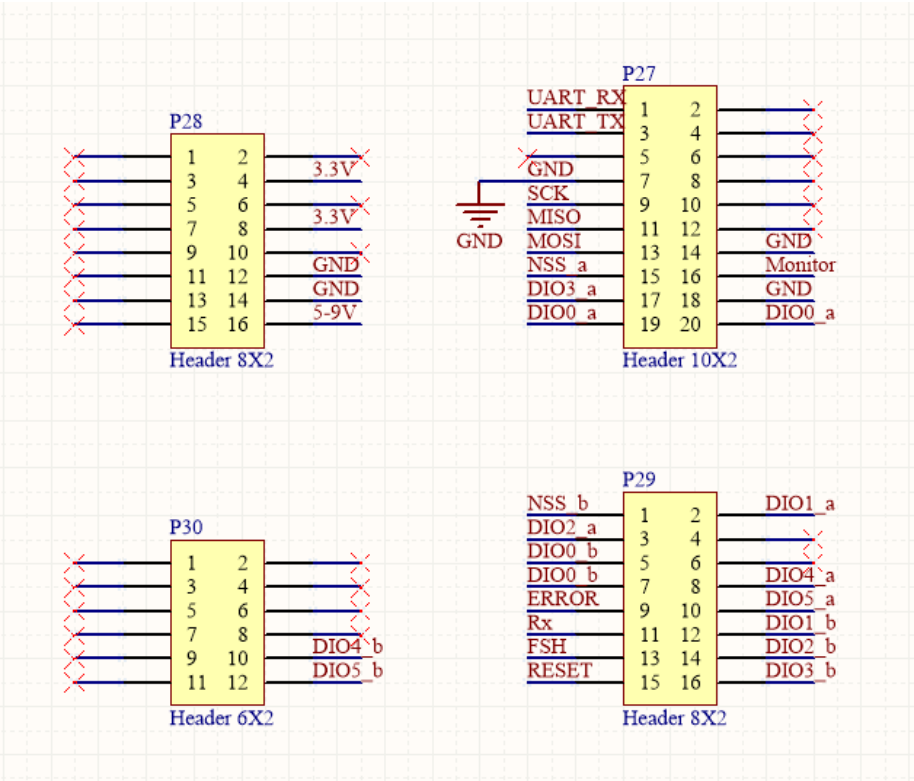


Figura 32: Pines para acoplamiento con placa de desarrollo.

En la siguiente imagen se muestra el layout del esquemático anterior. El mismo consiste en una placa de dos capas, dado que no es el prototipo final y solo posee el objetivo de verificar el funcionamiento del sistema.

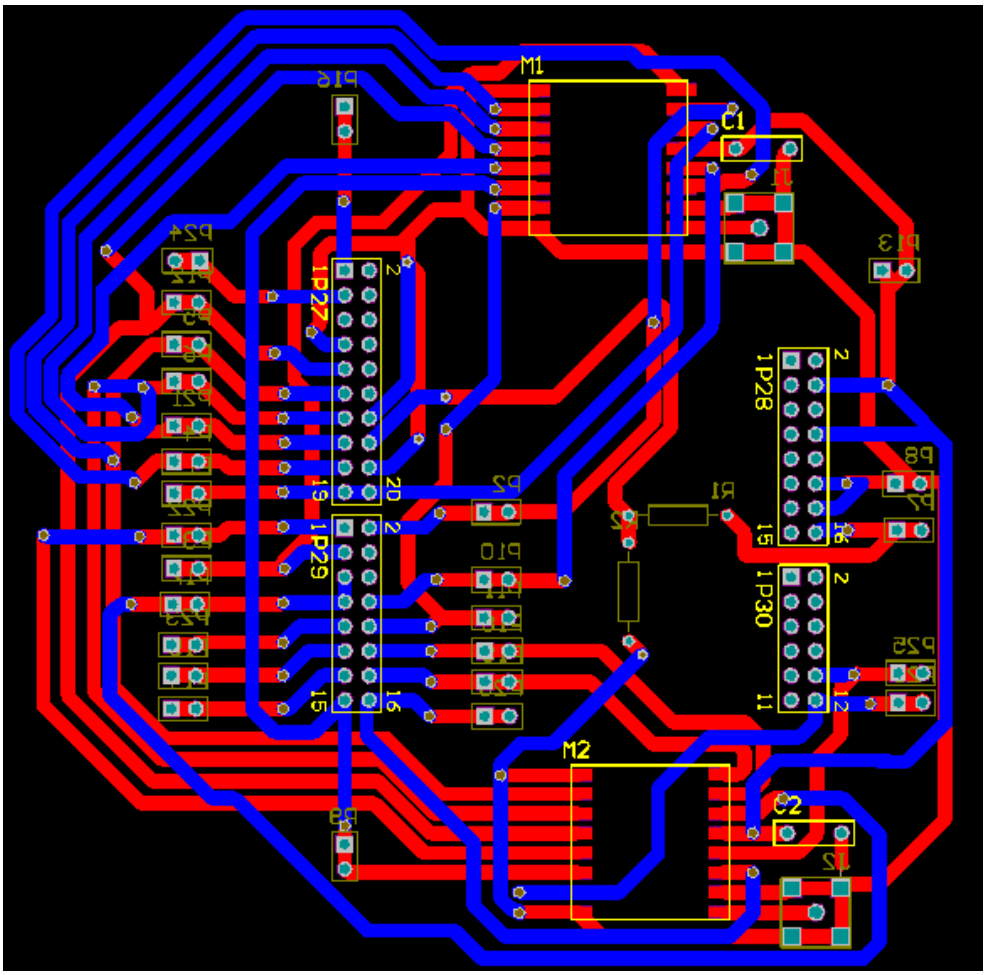


Figura 33: Layout del primer prototipo.

Como conclusiones de este primer prototipo se encontraron dos conexiones erróneas las cuales fueron corregidas para el segundo diseño.

17.2. Segundo prototipo

En la siguiente imagen se muestra el esquemático de la placa final. En la misma se puede apreciar la incorporación del microprocesador y la eliminación de los pines de debugging.

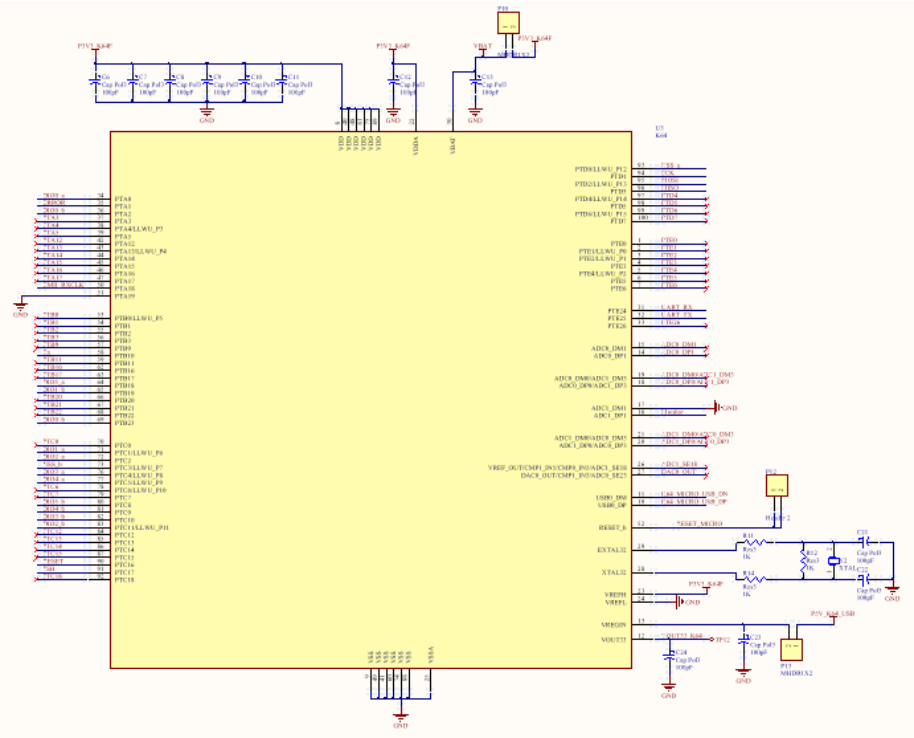


Figura 34: Esquemático del conexionado del microprocesador.

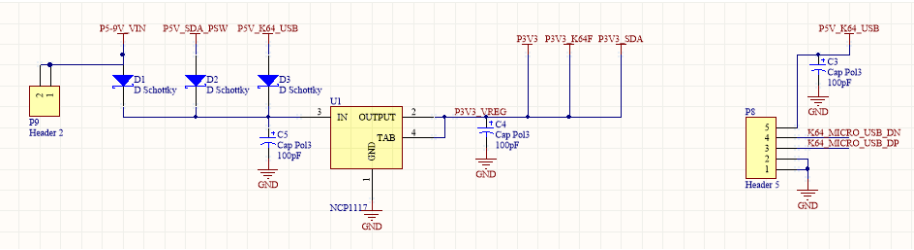


Figura 35: Conexionado de la alimentación de la placa.

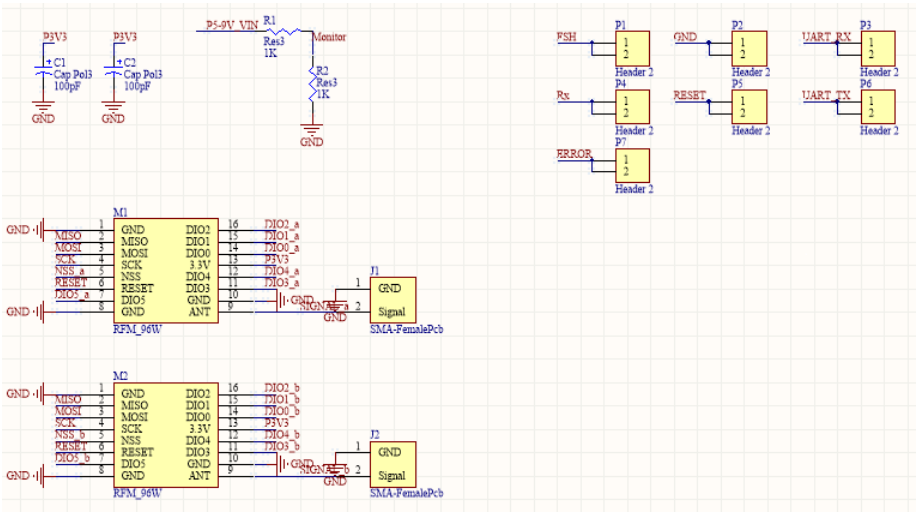


Figura 36: Conexionado de los módulos.

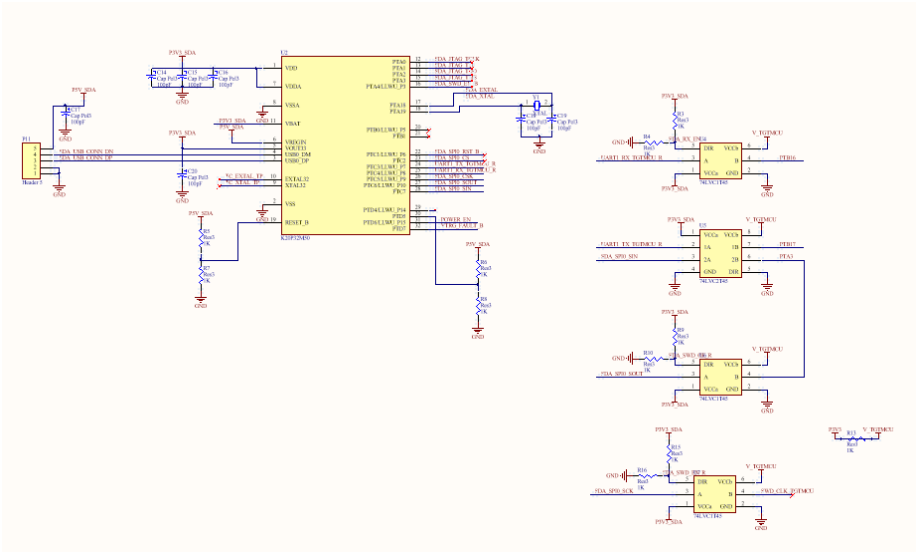


Figura 37: Conexionado para el bloque de SDA.

En cuanto al layout, vale la pena aclarar que se realizó una placa de cuatro capas con el fin de generar un mejor rendimiento en cuanto a interferencias. Es importante destacar que si bien se cuentan con señales de alta frecuencia, no fue necesario realizar una placa de ocho capas dado que la línea de alta frecuencia sólo se debe dirigir hacia la antena.

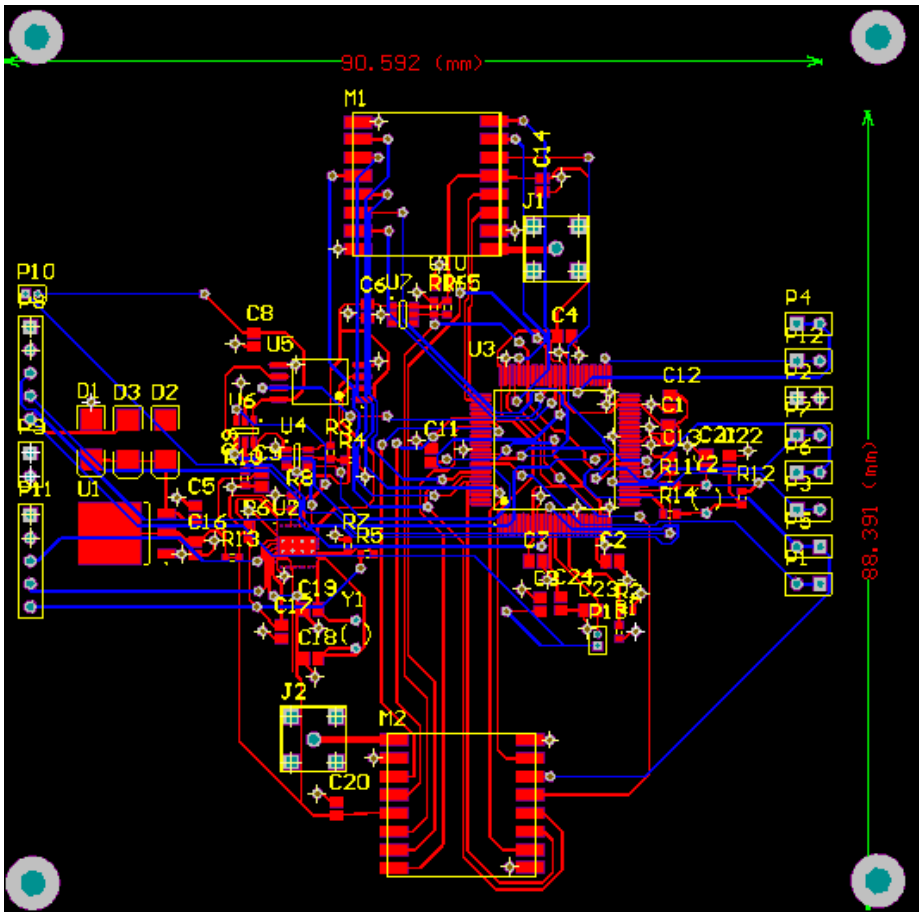


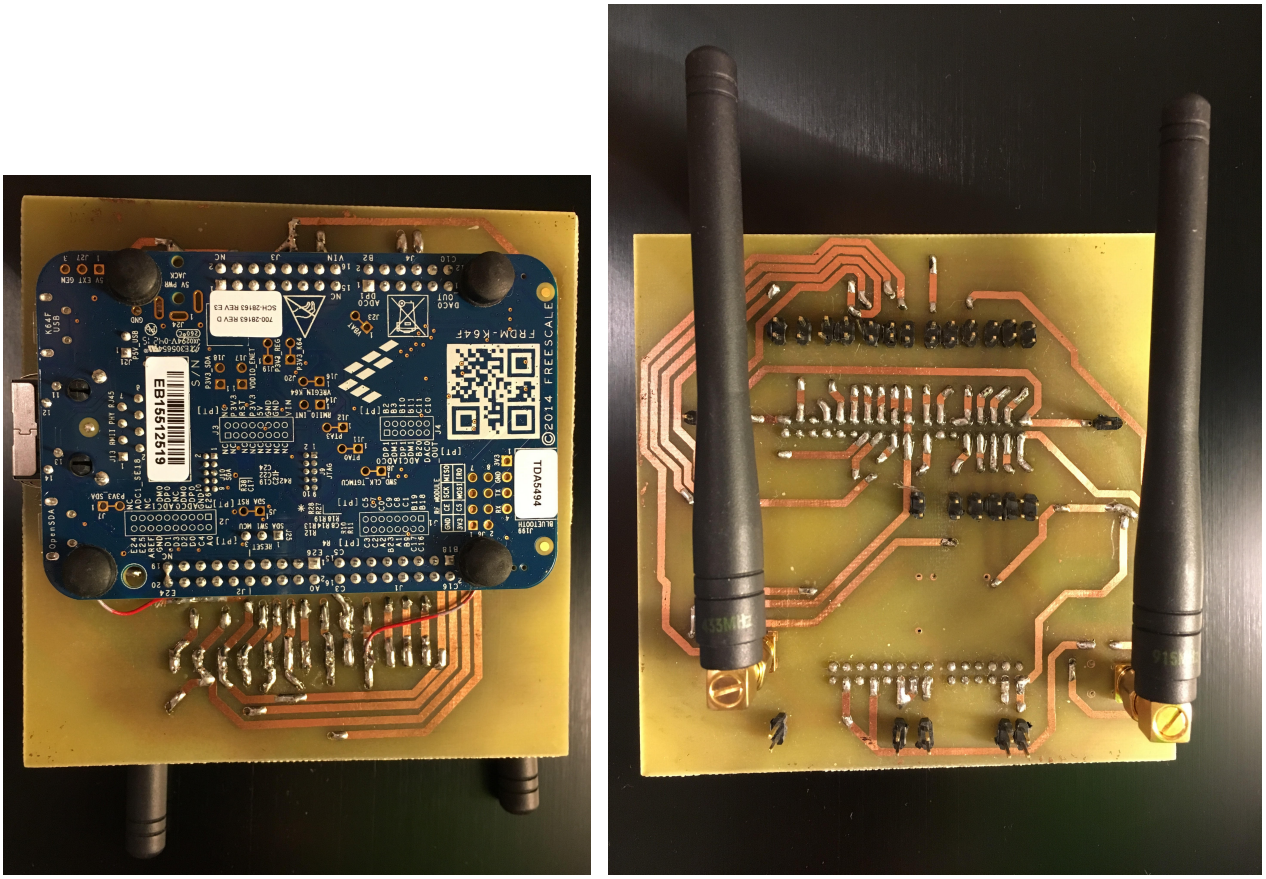
Figura 38: Layout del segundo prototipo.

18. Detalles de Construcción y Precauciones Especiales de Montaje

18.1. Primer prototipo

En este prototipo se centró la atención en el acople con la placa de desarrollo FRDM-K64F con el fin de realizar un ensamblaje robusto, y perdurable en el tiempo. A su vez se hizo foco en el testeo de las diferentes señales de importancia con el fin de poder debuggear el acople entre hardware y software.

A continuación se muestran unas fotografías del prototipo construido y ensamblado.



(a) Fotografía tomada desde abajo.

(b) Fotografía tomada desde abajo..

Figura 39: Primer prototipo ensamblado.

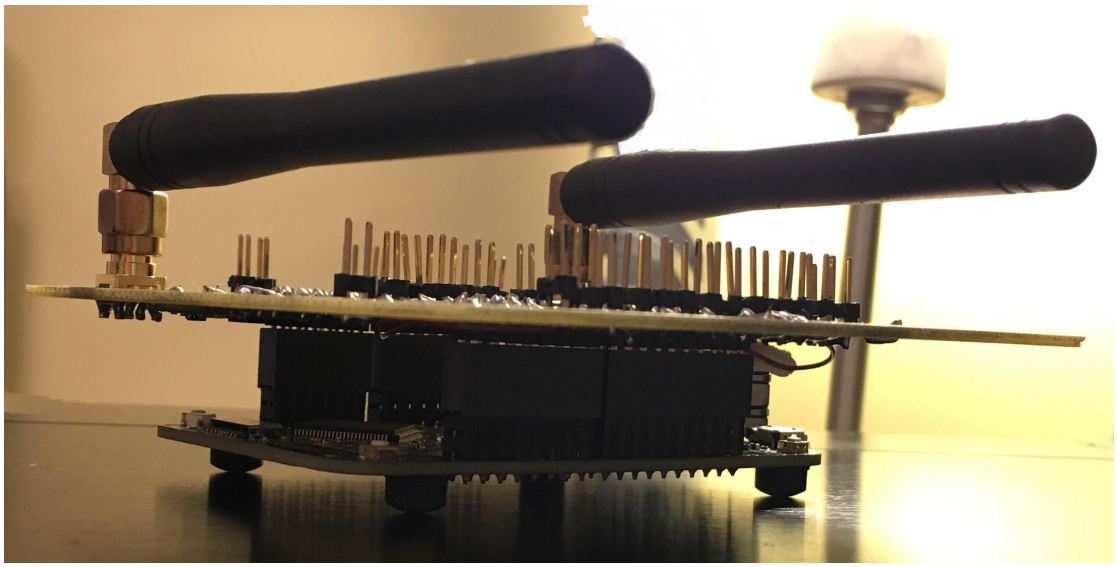
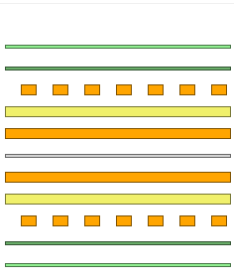


Figura 40: Primer prototipo ensamblado.

18.2. Segundo prototipo

En el momento de diagramar el layout de la placa final, se tomó en consideración el ensamblaje de la placa con el resto del sistema. Como se indicó en el requerimiento INT-MEC-01, el mismo se realizará con cuatro perforaciones ubicadas en la esquina de la placa de un diámetro acotado entre 3.2mm y 3.7mm. (consultar el cuadro 6) Es por esto que se decidió liberar las esquinas de la placa, minimizando el área ocupada. Al alejar los componentes de las esquinas también se reduce el stress generado por el agarre mecánico aplicado sobre los componentes soldados.

A continuación se muestra el diagrama de capas utilizados para el diseño final, el cual fue comentado previamente.



Layer Name	Type	Material	Thickness (mm)	Dielectric Material	Dielectric Constant	Pullback (mm)
Top Overlay	Overlay					
Top Solder	Solder Mask/Co...	Surface Material	0.01016	Solder Resist	3.5	
Top Layer	Signal	Copper	0.03556			
Dielectric 1	Dielectric	Core	0.254	FR-4	4.2	
VCC	Internal Plane	Copper	0.036			0.508
Dielectric 3	Dielectric	Prepreg	0.127		4.2	
GND	Internal Plane	Copper	0.036			0.508
Dielectric 2	Dielectric	Core	0.254		4.2	
Bottom Layer	Signal	Copper	0.03556			
Bottom Solder	Solder Mask/Co...	Surface Material	0.01016	Solder Resist	3.5	
Bottom Overlay	Overlay					

Figura 41: Diagrama de las capas utilizadas.

Part VII
Validación del Prototipo

Para verificar el correcto funcionamiento del sistema de comunicación se presentan dos métodos de validación: el primero segundo una simulación del sistema contemplando diversos tipos de atenuaciones que puede sufrir la señal y el segundo un conjunto de pruebas experimentales.

18.3. Simulación

Para sustentar el modelo teórico se desarrollo una simulación que tiene en cuenta el sistema real para simular las condiciones bajo las cuales el sistema trabajará normalmente. En la figura 42 se puede observar el sistema completo el cual cuenta con un transmisor, un receptor y la atenuación del medio. A su vez se pueden observar unos módulos para evaluar el rendimiento del sistema en sí.

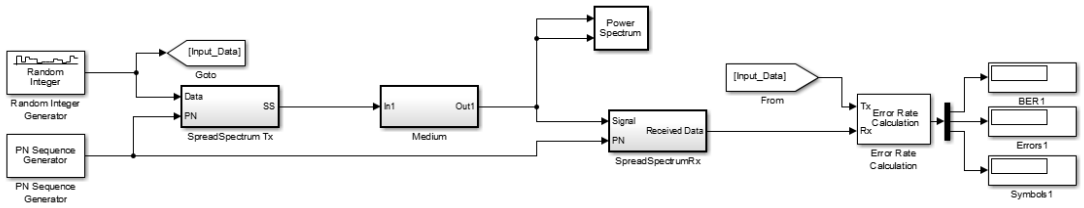


Figura 42: Diagrama en bloques de la simulación.

La simulación se encuentra dividida en tres subsistemas: el sistema de transmisión, el medio que modela los efectos que afectarán a la señal electromagnética y el sistema receptor. Los sistemas de transmisión y receptor se basan en los modelos provistos por los fabricantes de los integrados utilizados, utilizando los parámetros de las hojas de datos.

Transmisor

El sistema de transmisión consiste en un módulo de transmisión que obtiene como entrada una fuente de datos aleatorios que serán utilizados como señal de datos. El transmisor consiste en un modulador BPSK que transmite a 433Mhz, utilizando una modulación en Spread Spectrum con un ancho de banda de 60kHz. Posee un amplificador de la señal a 14dBm y un generador de ruido térmico de 300K, temperatura normal del satélite en funcionamiento.

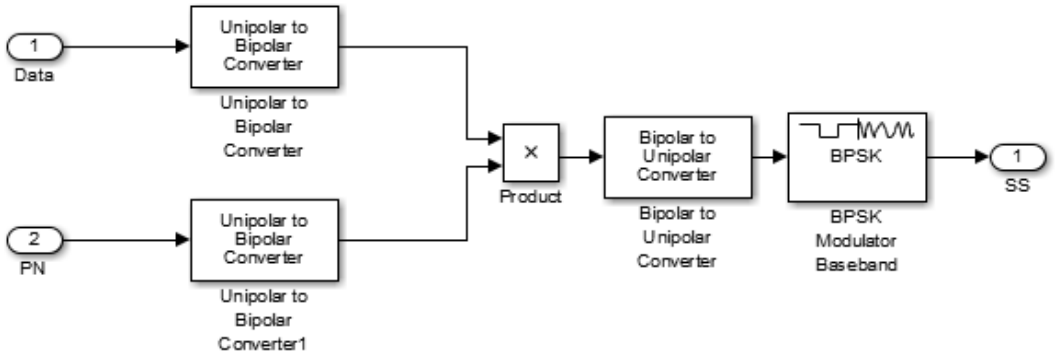


Figura 43: Diagrama en bloques del transmisor.

Medio

Para el modelado del medio se utilizaron los efectos ya mencionados en la tabla 55.

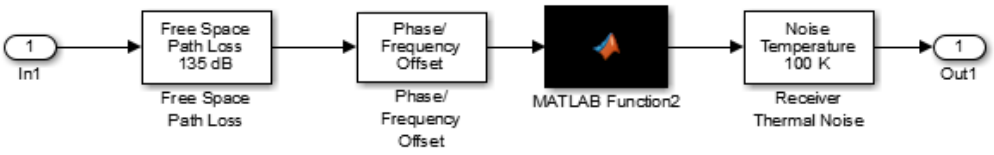


Figura 44: Esquema del medio.

Receptor

El modelo del receptor, así como el transmisor, representa el modelo provisto por el fabricante del integrado utilizado. La principal diferencia con el modelo del transmisor es que el receptor además posee un control de ganancia automático y corrección en la frecuencia de la señal.

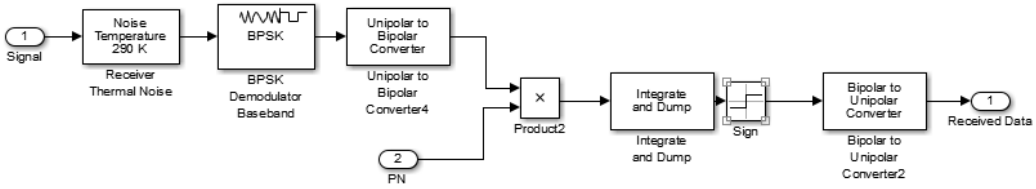


Figura 45: Diagrama en bloques del receptor.

18.3.1. Validación

Para utilizar la simulación como método de validación se correrá el sistema por treinta minutos lo cual equivale a 2.160.000 datos esperando obtener un error en el sistema menor a $1x10^{-4}$.

18.4. Pruebas experimentales

Para la verificación práctica del funcionamiento del proyecto se propone una prueba experimental del sistema. Debido a que resulta imposible replicar las condiciones reales a las cuales se someterá el proyecto, se proponen diferentes pruebas para corroborar la efectividad del diseño. Las pruebas realizadas se presentan en la tabla 48.

Prueba	Descripción	Objetivo
Verificación de la potencia de transmisión.	Utilizando un analizador de espectro y una antena receptora, se verificará que la potencia de transmisión del sistema se encuentre en el valor necesario para garantizar la comunicación.	[13,23] dBm
Verificación del ancho de banda.	Utilizando un analizador de espectro y una antena receptora, se verificará que el ancho de banda no exceda los límites propuestos por el diseño.	$<$65 kHz
Verificación de la velocidad.	Se realizarán pruebas a 2 Km de distancia en el vial de Vicente López. En esta prueba se correrá el sistema por 5 minutos para asegurarse que la velocidad de envío y recepción de paquetes sea acorde a lo especificado.	$>$120 bps
Verificación de la tasa de errores.	Para validar la tasa de errores se realizaran dos pruebas: con atenuaciones y con linea de visión. Ambas serán realizadas en el vial de Vicente López a 2 Km (la mayor distancia con línea de visión encontrada en las cercanías de la ciudad). En estas pruebas se enviarán 2.160.000 datos y se analizará la tasa de error de la misma. El sistema estará trabajando debajo de la potencia máxima nominal para simular mayor atenuación y poder realizar un contraste con el funcionamiento de la simulación en tales condiciones. En el caso de pruebas con atenuaciones, se realizaran durante un día de fuertes lluvias desde el interior de un automóvil para los vidrios y los cambios de impedancias de la señal simulen los efectos de rebote producido por las condiciones climatológicas como las nubes y niebla.	$< 1x10^{-4}$

Cuadro 48: Pruebas empíricas.

18.4.1. Verificación de tasas de errores

Pruebas de verificación de tasa de errores fallidas

Los días 16 y 17 de febrero del 2017 se realizaron pruebas en Costanera Sur, Puerto Madero, las que fueron afectadas por la cantidad de camiones en la zona. Durante las pruebas, el tráfico interrumpía la línea de visión entre el receptor y el transmisor, y generaba rebotes y atenuaciones a la señal. Durante las pruebas realizadas en la localidad de Vicente López en algunos casos se encontraron condiciones desfavorables para la emulación de las condiciones reales. Algunas pruebas fueron principalmente afectadas por el exceso de peatones, ciclistas y automóviles quienes producían rebotes y atenuaciones, afectando la potencia recibida por el receptor. Las pruebas fallidas fueron realizadas los días 21 y 22 de febrero entre las 19:00 horas y las 22:00 en el Vial de Vicente López.

En la tabla 49 se pueden encontrar la documentación de dichas pruebas.

Fecha	16/2/2017	17/2/2017	21/2/2017	22/2/2017
Lugar	Puerto Madero	Puerto Madero	Vicente López	Vicente López
Distancia	1,5 Km	1,5 Km	2,0 Km	2,0 Km
Temperatura	30 C	30 C	32 C	29 C
Humedad	70%	75%	86%	90%
Condicones Climáticas	Soleado	Soleado	Soleado	Nublado
Altura sobre el nivel del mar	0 m	0 m	0 m	0 m
Tasa de errores	6.00E-02	4.00E-01	8.30E-04	3.10E-03
Condiciones desfavorables	Excesivo tráfico de autos y algunos camiones interrumpían la línea de visión entre el transmisor y el receptor.	Excesivo tráfico de camiones con acoplado interrumpían la línea de visión entre el transmisor y el receptor.	Una gran cantidad de peatones y ciclistas interrumpían la linea de visión entre los módulos.	Una gran cantidad de peatones y autos afectaban la línea de visión entre los módulos.

Cuadro 49: Pruebas empíricas fallidas

Figura 46: Distancia utilizada en las pruebas en Puerto Madero.

Figura 47: Distancia utilizada en las pruebas en Vicente López.

Pruebas de verificación de tasa de errores exitosas

Debido a que las pruebas se vieron altamente afectadas por el tráfico en la zona, se decidió realizar las pruebas en días y horarios menos concurridos para generar menor interferencia entre los módulos. Se descartó la localización de Puerto Madero ya que en todo momento en la que se intentaron hacer pruebas tenía una gran cantidad de tráfico. Por este motivo solamente se siguieron realizando las pruebas de validación en Vicente López.

18.5. Resultados exitosos

Se realizó la validación exitosa del prototipo mediante 100 horas de prueba en el mes de Marzo de 2017 en la localidad de Vicente López. Cada prueba consistió en el envío de 130.000 datos de una cadena conocida por el receptor y el transmisor durante veinte minutos. Cada dato recibido era contrastado con el valor de esperado para determinar una falla en la comunicación. Se realizó el envío haciendo prueba de la capacidad full-dúplex por lo que la cantidad de datos efectiva enviada fue el doble ya que se validó la tasa de error del receptor y del transmisor.

Se realizó envío de más de 100.000.000 datos considerando todas las pruebas realizadas. Durante las pruebas se percibieron pequeñas interferencias como personas y autos aislados que interrumpían la línea de visión entre el transmisor y el receptor. Durante las pruebas no se tuvo ningún error en la transmisión. De acuerdo a la cantidad de datos enviada se puede estimar que la tasa de errores es menor a $1x10^{-8}$.

En la tabla 50 se muestran los resultados obtenidos para las pruebas.

Prueba	Resultado	Cumple Requerimiento
Verificación de la potencia de transmisión.	16.0 ± 0.5 dBm	Sí
Verificación del ancho de banda.	60 ± 1 kHz	Sí
Verificación de la velocidad.	130 ± 2 bps	Sí
Verificación de la tasa de errores.	$< 1x10^{-8}$	Sí

Cuadro 50: Pruebas empíricas

En los resultados obtenidos de las pruebas se pudo comprobar el correcto funcionamiento del sistema de comunicación. Cabe destacar que el rendimiento de la tasa de errores es cuatro órdenes de magnitud menor que el requerimiento PER-01.

Parte VIII
Estudios de Confiabilidad de Hardware y de Software

19. Confiabilidad de Hardware

En la siguiente sección se muestra el estudio de confiabilidad de hardware. Dado que el proyecto consiste en un módulo para comunicaciones de nanosatélites, se utiliza un modelo de fallas catastróficas. Entiéndase por falla catastróficas aquellas que son repentinas, completas y definitivas. El sistema a analizar está compuesto por la placa de desarrollo del microprocesador y el módulo de comunicaciones. Para el cálculo de confiabilidad se utiliza la norma militar $HDBK-217$. En la misma se especifica cómo calcular el $MTTF$ de cada componente, teniendo en cuenta los condicionantes de uso en cada caso. Utilizando la ecuación 3 se puede calcular cada $MTTF$, donde π_i representa cada característica de uso que afecta a cada componente.

$$MTTF_{componente}(1/FIT) = \left(1000 \cdot \lambda_{base} \cdot \prod \pi_i\right)^{-1} \tag{3}$$

Finalmente el $MTTF$ total se calcula por medio de la ecuación 4.

$$MTTF_{modulo}\,[Hs] = \left(\sum \left(\frac{1}{MTTF_{componente-i}}\right)\right)^{-1} \cdot \frac{10^9}{24} \tag{4}$$

A continuación se realiza el cálculo de confiabilidad para los distintos componentes del sistema de comunicaciones.

Resistencias

Para calcular la tasa efectiva de fallas λse utiliza la expresión de le ecuación 5. En dicha ecuación λ_b es la tasa base de fallas, π_R es el factor asociado al valor de la resistencia, $\pi_Q$el factor asociado a la calidad del componente y $\pi_E$es el factor asociado al ambiente de trabajo. En el caso de π_Q se elije el peor valor para situarce en el peor caso. Se asumen un stress del 30 % y una temperatura ambiente de 50C . En el caso de π_E, como en su funcionamiento se ubicará en la órbita terrestre debería ser S_F (*Space Flight*), pero como el sistema debe soportar el despegue y, probablemente, pruebas en Tierra se asume un ambiente de trabajoA_{UC}(*Airborne Inhabited Cargo*).

$$\lambda = \lambda_b \cdot \pi_R \cdot \pi_Q \cdot \pi_E \; \frac{Fallas}{10^6 Hs} \tag{5}$$

Encapsulado	Cantidad	λ_b	π_R	π_Q	π_E	λ_{ef}(FIT)
SMD 0805	8	0.0012	1	15	1	144
SMD 0805	2	0.0012	1.1	15	10	396
					Total	540

Cuadro 51: Cálculo de tasa de falla para las resistencias.

Capacitores

En el caso de los capacitores cerámicos el cálculo consiste en la ecuación 6. Dondeπ_{CV} es el factor asociado al valor del capacitor. En este caso se asume un stress del 30 %.

$$\lambda = \lambda_b \cdot \pi_{CV} \cdot \pi_Q \cdot \pi_E \; \frac{Fallas}{10^6 Hs} \tag{6}$$

Encapsulado	Cantidad	λ_b	π_{CV}	π_Q	π_E	λ_{ef}(FIT)
SMD - 0805	10	0.0036	1.3	10	12	5616
SMD - 0805	2	0.0036	1.6	10	12	1382.4
					Total	6998.4

Cuadro 52: Cálculo de tasa de falla para los capacitores.

Microcircuitos, arreglos lógicos y microprocesadores

Finalmente, en el caso de los circuitos integrados y los micropocesadores, la forma de calcular la tasa efectiva de fallas se encuentra descrita por la ecuación 7. Siendo C_1el factor asociado a la complejidad del integrado, C_2el factor asociado al tipo de encapsulado, y $\pi_L$el factor asociado a la cantidad de años que el circuito integrado se encuentra en producción.

$$\lambda = (C_1 \cdot \pi_T + C_2 \cdot \pi_E) \cdot \pi_Q \cdot \pi_L \ \frac{Fallas}{10^6 Hs} \tag{7}$$

Componente	Cantidad	π_Q	π_E	π_T	π_L	C_1	C_2	λ_{ef}(FIT)
MK64FN1M0VDC12	1	2	5	0.35	1	0.24	0.032	488
NPC117	1	2	5	1	1	0.01	0.0013	33
HopeRF96/98W	2	2	5	1	1	0.2	0.0026	426
							Total	947

Cuadro 53: Cálculo de tasa de falla para los microcircuitos.

De los cálculos anteriores se deduce que el sistema posee un MTTF equivalente a 13.45 años. Dicho tiempo resulta suficiente ya que se espera que el nanosatélite posea un MTTF mayor a 2 años..

20. Confiabilidad de Software

A continuación se muestra el análisis de confiabilidad para el Software. En este caso se puede enfocar el análisis de acuerdo a dos modelos. Los modelos son el de predicción y el de estimación. Por un lado, el modelo de predicción permite predecir la fiabilidad en algún momento futuro, mientras que el modelo de estimación requiere datos provenientes del desarrollo actual y estima tanto la fiabilidad presente como la futura.

20.1. Predicción

Para realizar la predicción de la confiabilidad de Software se utilizó el modelo de Musa, cuya ecuación característica es la número 8. En dicha ecuación, k es una constante cuantizada para la estructura dinámica del programa y de las máquinas, p es la estimación del número de ejecuciones por unidad de tiempo, y ω_0 es la estimación del número inicial de fallas en el programa.

$$\lambda_0 = k \cdot p \cdot \omega_0 \tag{8}$$

Como no se conoce el valor de k , se adoptó el valor típico de $4,2 \cdot 10^{-7}$. Por otro lado, p se calcula como:

$$p = \frac{r}{I} = \frac{r}{n/ER} \tag{9}$$

Donde r es el promedio de la velocidad de ejecución de instrucciones. En este caso, dado que se utiliza el lenguaje C, y se programa un microprocesador cuyo cristal es de $10\,MHz$, se la estimó en $1 \cdot 10^6 \frac{instrucciones}{seg}$. Por otro lado I, es la cantidad de instrucciones del código fuente en el nivel de máquina.

Para el cálculo de I se requiere conocer la cantidad n de líneas de código fuente, las cuales se expresan en $SLOC$ por sus siglas en inglés *Source Line Of Code*. En este trabajo, se cuenta con aproximadamente 1950 líneas de código, mientras que el *Expansion Ratio (ER),* que permite relacionar la cantidad de lineas de un lenguaje con la cantidad de instrucciones de más bajo nivel, dado que se utilizó lenguaje C, es $ER = 2,5$.

Finalmente, ω_0 se puede calcular como:

$$\omega_0 = N \cdot B \tag{10}$$

Siendo N el número total de defectos inherentes y B la proporción de defectos que se convierten en fallas. Considerando $B = 95\,\%$ como valor típico y asumiendo una relación de fallas sobre SLOC de 6/1000, se obtiene:

$$\omega_0 = \frac{6\,fallas}{1000\,SLOC} \tag{11}$$

Como consecuencia se obtiene una tasa de fallas dada por el siguiente cálculo:

$$\lambda_0 = k \cdot \left(\frac{1 \cdot 10^6 \frac{instrucciones}{seg}}{\frac{1950}{2,5} \frac{instrucciones}{SLOC}} \right) \cdot \frac{6\,fallas}{1000\,SLOC} \tag{12}$$

Como resultado se obtiene un MTTF aproximado de $309500\,Hs$, lo que es equivalente a 35 años.

20.2. Estimación

Para el modelo de estimación se utiliza el modelo de Shooman el cual tiene en cuenta el número de líneas de código. El mismo es similar al exponencial pero normaliza cada falla por el número de líneas de código.

El modelo propuesto por Shooman es el siguiente:

$$R\left(t\right) = e^{-\lambda t} \tag{13}$$

Donde λ está definido de acuerdo a la siguiente ecuación:

$$\lambda = k \cdot \varepsilon_r\left(\zeta\right) \tag{14}$$

Siendo $\varepsilon_r\left(\zeta\right)$ es la tasa residual de corrección de errores. La misma depende de la cantidad de errores corregidos al cabo del tiempo. Por lo tanto, considerando a ζcomo el tiempo de depuración de código, y de la tasa de errores totales ε_T se obtiene que:

$$\varepsilon_r\left(\zeta\right) = \varepsilon_T - \varepsilon_c\left(\zeta\right) = \frac{E_T}{I_T} - \varepsilon_c\left(\zeta\right) \tag{15}$$

Mientras queE_T es la cantidad inicial de errores e I_T es la cantidad de instrucciones en el programa (en este caso aproximadamente 1950). Se puede observar que con mayor tiempo de depuración ζ aumenta la fiabilidad del sistema.

Finalmente, el MTTF se calcula como la inversa λ, siendo λ la definida en la ecuación 14.

$$MTTF = \frac{1}{\lambda} = \frac{1}{k \cdot \left(\frac{E_T}{I_T} - \varepsilon_c\left(\zeta\right) \right)} \tag{16}$$

Dado que hay dos incógnitas, E_T y k, se tienen que plantear dos ecuaciones para poder resolver el sistema. Es por esto que se plantea el modelo en dos instantes de tiempo diferentes.

Para hacerlo se propone el siguiente desarrollo. Suponiendo que tras la ejecución n-ésima el código falló r veces. Sea t_i el tiempo de una ejecución exitosa y t_j el de una fallida. El tiempo total T de funcionamiento será:

$$T = \sum_{i=1}^{n-r} t_i + \sum_{j=1}^{r} t_j \tag{17}$$

Asumiendo que el modelo de fallas es exponencial, el MTTF estará dado por:

$$MTTF = \frac{1}{\lambda_i} = \frac{T}{r} \tag{18}$$

Finalmente, igualando las ecuaciones 16 y 18 se obtiene que, para dos instantes del proceso a y b:

$$\frac{T_a}{r_a} = \frac{1}{\lambda_a} = \frac{1}{k \cdot \left(\frac{E_T}{I_T} - \varepsilon_c\left(\zeta_a\right)\right)} \tag{19}$$

$$\frac{T_b}{r_b} = \frac{1}{\lambda_b} = \frac{1}{k \cdot \left(\frac{E_T}{I_T} - \varepsilon_c\left(\zeta_b\right)\right)} \tag{20}$$

Siendo T_i es el tiempo acumulado de ejecución durante el cual han aparecido las r_i fallas, luego de haberse corregido $\varepsilon_c\left(\zeta\right)$ errores:

$$\hat{E_T} = -\frac{\left[\frac{\lambda_b}{\lambda_a}\right] \cdot \varepsilon_c\left(\zeta_a\right) - \varepsilon_c\left(\zeta_b\right)}{\frac{\lambda_b}{\lambda_a} - 1} \tag{21}$$

$$\hat{k} = \lambda_a \frac{I_T}{\hat{E_T} - \varepsilon_c\left(\zeta_a\right)} \tag{22}$$

Una vez encontrada esta relación se toman como muestra cuatro semanas. Los datos obtenidos son los que figuran en la tabla 54.

Semanas	Errores totales acumulados	Errores corregidos	Errores residuales	Fallas catastróficos acumulados
1	15	5	10	2
2	32	25	7	5
3	40	31	9	5
4	45	44	1	9

Cuadro 54: Datos relevados durante cuatro semanas.

Para realizar la estimación se utilizan los datos de la semana 1 y 4. De esta información y haciendo uso de las ecuaciones citadas anteriormente se puede llegar a que:

$$\frac{T_a}{r_a} = \frac{1}{\lambda_a} = \frac{168}{2} = 84 \tag{23}$$

$$\frac{T_b}{r_b} = \frac{1}{\lambda_b} = \frac{4 \cdot 168}{5} = 74,667 \tag{24}$$

$$\hat{E_T} = -\frac{\left[\frac{\lambda_b}{\lambda_a}\right] \cdot \varepsilon_c\left(\zeta_a\right) - \varepsilon_c\left(\zeta_b\right)}{\frac{\lambda_b}{\lambda_a} - 1} = 307 \tag{25}$$

$$\frac{\hat{k}}{I_T} = \frac{1}{74{,}667.(307 - 48)} = 0,00005 \tag{26}$$

Finalmente, a partir de los resultados previos, se calcula un $\lambda \cong 5,09 \cdot 10^{-5}$ y un MTTF aproximado de *2,24 años*. Cabe destacar que dicho resultado no es muy confiable dado que el estimador de errores acumulados es de 307, mientras que los errores totales acumulados son 45. Esto indica que habría que tomar más muestras para tener un estimador más confiable.

21. Conclusiones de Confiabilidad

Finalmente, del cálculo de predicción se obtuvo un MTTF para el software de 35 años, mientras que del cálculo de estimación se obtuvo un MTTF de 2,24 años. En cuanto al hardware, se obtuvo un MTTF de 13,45 años. Planteando un modelo serie de ambos y estimando un uso contínuo, el MTTF resultante es de 9,71 años en caso de utilizar el resultado del método de predicción, mientras que en el otro caso es de 2,10 años. Cabe destacar que en el caso del método de estimación el resultado no era confiable por la cantidad de datos, por lo que se descarta dicho resultado. A modo de conclusión, poseer un MTTF de 9,71 años implica que en un año tenga una confiabilidad del 90.2 %.

Parte IX
Conclusiones

22. Objetivos Alcanzados

A modo de conclusión del trabajo realizado, cabe destacar que se logró construir un prototipo funcional de un sistema de comunicaciones para un nanosatélite que orbita en LEO. Dado que las especificaciones propuestas fueron cumplidas, se realizó un sistemas de comunicaciones full-dúplex, con una frecuencia de transmisión en el rango de 430Mhz a 450 Mhz, una frecuencia de recepción entre 902 Mhz y 928 Mhz, una potencia mínima de salida de 14 dBm, con un BER menor a 10^{-4}, entre otros aspectos. Sin embargo el detalle fundamental es que se consiguió desarrollar un prototipo con un precio de 500 USD. Para que sea mensurable la importancia del precio basta con analizar a la competencia, quienes ofrecen velocidades 10 veces más grandes pero por precios 18 veces mayores. Sin perder de vista que el objetivo del proyecto no era meramente económico sino que buscaba la entrada a un mercado emergente en la industria aeroespacial y la fidelización de clientes para futuros desarrollos, se considera más que importante dicho factor.

A lo largo del proyecto se analizaron las factibilidades temporal, legal, tecnológica y económica, como así también el armado del prototipo, su validación y confiabilidad.

En lo que respecta a la parte temporal es importante destacar que se demoró más tiempo que el estimado por inexperiencia en la estimación, cambios en la forma de trabajo, entre otros factores. En cuanto a lo legal, el proyecto es viable dado que no se encontró ninguna traba a la hora de la fabricación del producto. En el aspecto tecnológico se propusieron diversas formas de llevar a cabo el proyecto, eligiendo la mejor forma de implementación de acuerdo a los parámetros importantes para la aplicación. Finalmente, de la parte económica se desprenden dos ideas. Por un lado, luego de cuatro año sería necesario empezar el desarrollo de un nuevo producto para ser introducido al mercado entre los años 8 y 9 dado que ése es el punto en donde éste proyecto posee una caída en la demanda. Por otro lado, el desarrollo de éste proyecto por sí solo no posee un rendimiento económico positivo. Por lo tanto, pensándolo con un objetivo puramente económico, el proyecto no es recomendable. Esto último podría no ser de importancia en todos los casos, dado que a veces generar una penetración en un mercado es cuantificable a más largo plazo que en el ciclo de vida de un proyecto en sí.

23. Fallas o Recomendaciones Para Futuros Diseños

A pesar de haber cumplido de forma satisfactoria con los requerimientos y especificaciones propuestos, hay algunos mejoras que se le pueden hacer al prototipo o al método de diseño con el fin de mejorar el prototipo.

- **Estimación Temporal:** Se recomienda para proyectos futuros realizar un análisis de Montecarlo para lograr prever que actividades no críticas pueden convertirse en críticas y así lograr tener una mejor estimación temporal, con un intervalo de confianza.
- **Velocidad de Transmisión:** Dado que ya se consiguió un diseño con un precio de venta atractivo para el mercado, sería bueno mejorar el peor aspecto del diseño que es la velocidad.
- **Miniaturización:** Si bien se cumplieron los requerimientos y las especificaciones relacionadas a las dimensiones, la miniaturización no fue algo que se optimizó en el diseño del proyecto. Para un próximo prototipo quizás se podría hacer foco sobre dicho aspecto.
- **Incorporar Nuevos Protocolos de Comunicación:** Si bien el microprocesador elegido cuenta con módulos de comunicación distintos al UART, por falta de tiempo se implementó únicamente dicho protocolo. Para ampliar la compatibilidad del proyecto con otros diseños y así hacer mas atractivo el producto sería bueno incorporar nuevos protocolos de comunicación.
- **Reducción de Consumo:** El consumo es una característica esencial en un nanosatélite. Si bien se cumplieron los requerimientos de consumo, sería importante en cada rediseño trabajar para reducirlo. Quizás se podría utilizar un hardware de menor consumo.

- **Agregar Una Capa de Abstracción Superior al Diseño:** Si bien el prototipo tiene un pequeño protocolo para administrar los datos recibidos desde la Tierra, y luego enviarlos al resto del satélite, como así también el camino inverso, sería una mejora poder agregar un nivel de abstracción más. Una posible mejora podría ser que el sistema de comunicaciones pudiera administrar sesiones con distintos usuarios, los cuales podrían validarse con una contraseña, y así también poder identificar a distintos transmisores. Se podrían asignar políticas de turnos para que todos tengan oportunidad de tomar contacto con el nanosatélite, pero teniendo en cuenta que el enlace dura 5 minutos. Recordemos que lo importante de los nanosatélites no es el impacto que puede generar uno de ellos, sino un conjunto de ellos orbitando alrededor de la tierra. Otra función útil podría ser generar un mensaje broadcast que permita comunicar la misma información a distintos usuarios o para encontrar posibles fallas de enlace.

Referencias

[1] Página WEB - Historia del Sputnik - https://mx.tuhistory.com/

[2] Página WEB - Jet Propulsion Laboratory -https://jpl.nasa.gov

[3] Página WEB - Charla de Emiliano Kargieman - http://tedxriodelaplata.org

[4] Página WEB - Laboratorio de Procesamiento de Imagenes de la Universidad de Valladolid - https://www.lpi.tel.uva.es

[5] Página WEB - Ejemplos de Aplicación de Nanosatélites - https://ntrs.nasa.gov

[6] Página WEB - Proyecto ISGEN - https://nasa.gov

[7] Página WEB - Base de datos de Nanosatélites - http://nanosats.eu/

[8] Página WEB - Costo de Proyectos - http://www.space.com

[9] Página WEB - Proyección de 'Markets and Markets' -http://marketsandmarkets.com

[10] Tesis - 'Power Supply for the AAU Cubesat' - p25 - 20/12/2001

[11] Página WEB - Proyecto CubesatKit -http://cubesatkit.com/

[12] Página WEB - NASA - https://nasa.gov

[13] Página WEB - Conector SMA Aamphenolrf - http://amphenolrf.com

[14] Página WEB - Universidad de Liege - http://www.leodium.ulg.ac.be

[15] Página WEB - Tratado sobre los principios que deben regir las actividades de los Estados en la exploración y utilización del espacio ultraterrestre - http://unoosa.org

[16] Página WEB - IARU - http://iaru.org/

[17] Página WEB - Información Legislativa, Ministerio de la Nación - http://servicios.infoleg.gob.ar

[18] Página WEB - Universidad del estado de Nuevo Méjico

[19] Página WEB - AMSAT - http://amsat.org/

[20] Página WEB - ISIS - https://www.isispace.nl

[21] Página WEB - Clyde Space - https://clyde.space

[22] Iterative Decoding of Binary Block and Convoluional Codes

[23] Trellis Code - Communication System Design - Massachusetts Institute of Technology

[24] Thoughts on Commercial Off the Shelf (COTS) Electronics for Space - NASA

[25] Evaluation and Qualification of Commercial Off-The-Shelf Supercapacitors for Space - 2nd Space Passive Component Days (SPCD), International Symposium

[26] Thoughts on Commercial Off the Shelf (COTS) Electronics for Space - NASA

[27] Quality Assurance of Space Intruments Built with COTS - Univerdad Técnica de Dinamarca

[28] AMES Cost Model - Nasa Cost Symposium - https://www.nasa.gov

[29] Encuesta realizada: Anexo26

[30] Survey of worldwide pico and nanosatellite missions - Delft University of Technology

[31] Página WEB - IBM - https://www.ibm.com

[32] Cálculo de la atenuación en el espacio libre - recomendación UIT-R P.525-2

[33] Half-Wave Dipole Analysis - University of Toronto

[34] Link Analysis of a Telecommunication System on Earth, in Geostationary Orbit, and at the Moon: Atmospheric Attenuation and Noise Temperature Effects

[35] Digital Communications Fundamentals And Applications - Sklar

[36] HopeRF RF96W Datasheet

[37] Orbital Lifetime Analyses of Pico- and Nano-Satellites - AI-AI LUMNAY C. COJUANGCO - Univertisy of Florida - 2007

[38] RFM95/96/97/98(W) - Low power Long range transceiver Module Datasheet V1.0 - Hoperf

[39] Kinetis K64F Sub-Family Datasheet REV 6 - Freescale Semiconductor, INC

[40] Cubesat Design Specification REV 12- The Cubesat Program

[41] MAI-100 ADACS REV. G - Cubesatkit

[42] Thoughts on COTS Electronics for Space -NASA/GSFC

[43] GSFC-STD-7000A General Enviromental Verification Standard - NASA

[44] Modelo del Enlace Satelital - Capitulo 4 - Universidad Politécnica Salesiana

[45] PC/104-Plus Specification V2.3

Parte X
Anexos

24. Cálculo Básico de Enlace

Teniendo en cuenta que la ganancia de la cada antena es de 2.5dB, la atenuación en el espacio libre es de 141dB, sin tener en cuenta otros factores de atenuación, considerando que la sensibilidad es de -122dB, y el margen es 0, se puede aproximar que la potencia en el peor caso debe ser de al menos 14dBm. Esto nos dice que si la potencia del transmisor es mayor a 14dBm se garantizará el enlace, sin embargo, al bajar la sensibilidad dicho valor puede decaer, inclusive cuando se incorporen otras fuentes de atenuación.

$$Pot = -(At + Sens + 2 * Gant) \tag{27}$$

Siendo Pot la potencia de transmisor, At la atenuación en el espacio libre, Sens la sensibilidad del receptor y Gant la ganancia de las antenas.

Acontinuación se plantea un cálculo del enlace más detallado con el fin de darle un soporte teórico a la validación de la implementación realizada. En el cuadro 55 se presenta un análisis de los mismos.

Fenómeno	Descripción	Valor
Atenuación por espacio libre	El satélite se encontrará en una órbita elíptica con diagonal mayor de 300km.	-141dB[32]
Atenuación por desacople	Debido a que el modelo fue diseñado para antenas de tipo dipolo de Hertz, un cambio en la polarización afecta a la potencia de recepción.	-3.01dB[33]
Ganancia de las antenas	Ganancia producida por el tipo de antena utilizada.	3.52dB[33]
Atenuación por efectos climáticos	Efectos climáticos como fuertes lluvias, nieblas y nubosidades causan rebotes y pérdidas en la potencia de la señal.	-1.83dB[34]
Atenuación por absorción del aire	Las partículas de agua en el aire producen absorción de potencia generando pérdidas en la señal.	-1.63dB[35]
Ruido térmico	El movimiento de las partículas por temperatura en el receptor genera ruido térmico. La temperatura nominal de este componente son 300K.	-
Ruido por oscilación del aire	El movimiento de las partículas de aire produce irradiación de ondas electromagnéticas que se puede modelar como una fuente de ruido térmico a 100K.	-
Defasaje en la frecuencia	Debido al efecto Doppler se producen defasajes en la frecuencia de la portadora de hasta 3Hz.	-
Potencia de transmisión	Amplificación de la señal en el transmisor.	14dBm[36]
Ganancia por sobremodulación	Se utiliza un sistema de información redundante y codificación para mejorar la inmunidad frente al ruido.	3dB[36]

Cuadro 55: Efectos considerados

El cálculo de potencia de recepción condensa la información de la tabla 55 demostrando que la potencia del receptor es mayor que la sensibilidad del receptor para una tasa de error menor a $1x10^{-4}$.

$$BER = erfc\left(\sqrt{\frac{Eb}{No}}\right) = erfc\left(\sqrt{\frac{-120{,}9dBm}{-132{,}5dBm}}\right) = 8,7x10^{-8}$$

Donde

- Eb es la energía de bit con unidad Julios [J].
- No es la densidad espectral de ruido con unidad Watt sobre Hertz [W/Hz].
- BER es la tasa de error de bit.
- erfc es la función error complementara definida por la expresión: $\frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} dt$.

La relación $\frac{Eb}{No}$ indica la eficiencia de potencia del sistema independiente a la modulación.

25. Comparación de Nanosatélites 2009-2012

En las siguientes tablas se muestran las características de los algunos nanosatélites puestos en órbita entre los años 2009 a 2012. Lo importante a destacar es que la mayoría de los nanosatélites de servicio amateur tienen un período de vida menor al año. Las mismas fueron extraídas del paper '*A Survey of CubeSat Communication Systems 2009-2012*' - Klofas, Leveque - 2013.

Satellite	Object	Size	Radio	Frequency	Satellite Service	Power	TNC	Protocol	Baud Rate/Modulation	Downloaded	Lifetime	Antenna	Status
Minotaur 1; 19 May 2009													
AeroCube-3	35005	1U	Freeware FGRM	915 MHz	experimental	2 W	Integrated	Proprietary	77 kbaud GFSK	52 MB	7 months	patch	Deorbited
CP6	35003	1U	CC1000/RF2117	437.305 MHz	amateur	1 W	PIC18LF6720	AX.25	1200 baud FSK		4 months	dipole	Dead
HawkSat-1	35004	1U	Microhard MHX-425	437.345 MHz	amateur	1 W	Integrated	Proprietary		0 kB	0 days	monopole	DOA
PharmaSat	35002	3U	Microhard MHX-2400 Stensat (beacon)	2.4 GHz 437.405 MHz	experimental amateur	1 W 500 mW	Integrated Integrated	Proprietary AX.25	10 kbps 1200 baud AFSK	650 kB N/A	10 days 1 month	patch monopole	Dead
ISILaunch 01/PSLV-C14; 23 Sep 2009													
BEESAT-1	35933	1U		436.000 MHz	amateur[1]	500 mW	CMX969B	Mobitex	4800/9600 baud GMSK		43+ months	monopole	Alive
UWE-2	35934	1U	PR430	437.385 MHz	amateur	1 W	Internal	AX.25	1200 baud AFSK		1 week	dipole	Dead
ITUpSAT-1	35935	1U	Microhard MHX-425 BeeLine/CC1050	437.325 MHz 437.325 MHz	amateur amateur	1 W 350 mW	Integrated	Proprietary CW	19200 baud	0 kB[2] N/A	43+ months	dipole monopole	Alive
SwissCube	35932	1U	Butler oscillator/RFM110G RF2516 (beacon)	437.505 MHz 437.505 MHz	amateur amateur	1 W 100 mW	MSP430F1611 Integrated	AX.25 CW	1200 baud FSK 10 WPM	0 kB N/A	43+ months	monopole monopole	Active
H-IIA F17; 20 May 2010													
Hayato	36573	1U	Custom	13.275 GHz	Earth exploration	100 mW	Integrated		10 kbps/1 Mbps BPSK	0 kB[2]	18 days	patch	Deorbited
Waseda-SAT2	36574	1U	TXE430-301A TXE430-301A (beacon)	437.485 MHz 437.485 MHz	amateur amateur	150 mW 100 mW	HS/30529[3] HS/30529[3]	AX.25 CW	9600 baud FSK	0 kB N/A	0 days	monopole dipole	DOA Deorbited
Negai-Star	36575	1U	Data Beacon Radio	437.305 MHz 437.305 MHz	amateur amateur	150 mW 100 mW		AX.25 CW	1200 baud FSK 50 WPM	N/A	1 month	dipole dipole	Deorbited
NLS-6/PSLV-C15; 12 July 2010													
Tbsat-1	36799	1U	Alinco DJ-C6 CC1010 (beacon)	437.305 MHz 437.305 MHz	amateur amateur	500 mW 400 mW	MSP430F169 MSP430F169	AX.25 CW	1200 baud AFSK 15-110 WPM	N/A	33+ months	monopole monopole	Active
StudSat	36796	1U	CC1020 MAX1472 (beacon)	437.505 MHz 437.980 MHz	amateur amateur	500 mW 10 mW	UC3A0512[3] UC3A0512[3]	Custom AX.25 CW	4800 baud FSK 22 WPM	0 kB[2] N/A	5 days	monopole monopole	Dead
STP-S26; 19 Nov 2010													
RAX-1	37223	3U	Lithium-1	437.505 MHz	amateur	750 mW	Integrated	AX.25	9600 baud GMSK	4.8 MB	2 months	turnstile	Dead
O/OREOS	37224	3U	Microhard MHX-2400 Stensat (beacon)	2.4 GHz 437.305 MHz	experimental amateur	1 W 500 mW	Integrated Integrated	Proprietary AX.25	Variable 1200 baud AFSK	8 MB N/A	29+ months	patch monopole	Alive
NanoSail-D2	37361	3U	Microhard MHX-2400 Stensat (beacon)	2.4 GHz 437.270 MHz	experimental amateur	1 W 500 mW	Integrated Integrated	Proprietary AX.25	Variable 1200 baud AFSK	N/A	5 days[4]	patch monopole	Deorbited
Falcon 9-002; 8 Dec 2010													
Perseus (4)	37251	1.5U			government						1 month		Deorbited
QbX (2)	37249	3U	TTC	450 MHz	government	1 W			9600 baud GMSK		1 month	quadrafilar helix	Deorbited
SMDC-ONE	37246	3U	Peride	UHF	government						1 month	turnstile	Deorbited
Mayflower	37252	3U	Microhard MHX-425 Stensat (beacon)	437.000 MHz 437.600 MHz	unlicensed unlicensed	1 W 1 W	Integrated Integrated	Proprietary AX.25	Variable 1200 baud AFSK	0 kB[2] N/A	2 days	dipole	Deorbited

[1] This satellite was not coordinated through the IARU.
[2] Uplink commands were never received by this satellite.
[3] This is also the main spacecraft processor.
[4] There were no solar cells on this satellite.

Figura 48: Comparación de nanosatélites.

Satellite	Object	Size	Radio	Frequency	Satellite Service	Power	TNC	Protocol	Baud Rate/Modulation	Downloaded	Lifetime	Antenna	Status
PSLV-C18; 12 Oct 2011													
Jugnu	37839	3U	CC1070/RF5110G	437.505 MHz	amateur	1 W		AX.25	2400 baud FSK		18+ months	monopole	Alive
			MAX1472 (beacon)	437.505 MHz	amateur	10 mW		CW	20 WPM	N/A		monopole	
ELaNa-3/NPP; 28 Oct 2011													
AubieSat-1	37854	1U	Melexis TH72011	437.475 MHz	amateur	800 mW	ATmega1281[1]	CW	20 WPM	0 kB	18+ months	dipole	Alive
DICE (2)	37851	1.5U	L3 Cadet	465 MHz	meteorological	1 W	Integrated	Proprietary	2.6 Mbps BPSK	8.4 GB	18+ months	dipole	Active
HRBE	37855	1U	CC1000	437.505 MHz	amateur	850 mW		AX.25	1200 baud FSK	7.6 MB	18+ months	monopole	Active
M-Cubed	37855	1U	Lithium-1	437.485 MHz	amateur	1 W	Integrated	AX.25	1200 baud FSK	0 kB[2]	18+ months	monopole	Alive
RAX-2	37853	3U	Lithium-1	437.345 MHz	amateur	1 W	Integrated	AX.25	9600 baud GMSK	242 MB	18+ months	turnstile	Active
Vega VV01; 13 Feb 2012													
Xatcobeo	38082	1U	GomSpace U482C	437.365 MHz	amateur	500 mW	Integrated	AX.25/CW	1200 baud MSK/20 WPM		14+ months	turnstile	Active
ROBUSTA		1U	MC12181/MAX2608	437.325 MHz	amateur	800 mW	PIC18F4580[1]	AX.25	1200 baud AFSK	0 kB[3]	2 days	dipole	Dead
e-st@r	38079	1U	BHX2-437-5	437.445 MHz	amateur	500 mW	PIC16	AX.25	1200 baud AFSK	0 kB[2]	3 days	dipole	Dead
Goliat	38085	1U	Alinco DJ-C7	437.485 MHz	amateur	500 mW	FX614/MSP430	AX.25/CW	1200 baud AFSK/20 WPM		1 week	monopole	Dead
			Microhard MHX-2420	2.4 GHz		1 W	Integrated	Proprietary	Variable	0 kB[3]		patch	
PW-Sat	38083	1U	ISIS TRXUV	145.900 MHz	amateur	200 mW	Integrated	AX.25/CW	1200 baud BPSK/12 WPM		10 months	dipole	Dead
Masat-1	38081	1U	Si4432	437.345 MHz	amateur	100/400 mW	dsPIC33F[1]	Custom/CW	GFSK/120 CPM	305 MB	14+ months	monopole	Active
UniCubeSat-GG		1U	AstroDev Custom	437.305 MHz	amateur	500 mW	Integrated	AX.25/CW	9600 baud GFSK	0 kB[2]	2 days	dipole	Dead
ELaNa-6/NROL-36; 13 Sep 2012													
SMDC-ONE (2)	38766	3U	Pericle	UHF	government							turnstile	Alive
AeroCube-4 (3)	38767	1U	FreeWave MM2	915 MHz	experimental	2 W	Integrated	Proprietary	38.4 kbaud		8+ months	patch	Active
			CC1101	915 MHz	experimental	1.3 W	Integrated	Proprietary	500 kbps FSK			patch	
Aeneas	38760	3U	MHX-425	437.000 MHz	experimental	1 W	Integrated	Proprietary	Variable		8+ months	monopole	Alive
			Stensat (beacon)	437.600 MHz	amateur	1 W	Integrated	AX.25	1200 baud FSK	N/A	8+ months	monopole	
CSSWE	38761	3U	Lithium-1	437.345 MHz	experimental	1 W	Integrated	AX.25	9600 baud GFSK	60 MB	8+ months	monopole	Active
CP5	38763	1U	CC1000/RF2117	437.405 MHz	amateur	500 mW	PIC18LF6720	AX.25	1200 baud FSK	500 kB	4 months	dipole	Dead
CXBN	38762	2U	Lithium-1	437.525 MHz	amateur	1 W	Integrated	AX.25	9600 baud GFSK		8+ months	turnstile	Active
CINEMA	38764	3U	Emhiser	2200 MHz	space research	1 W	FPGA	Proprietary	1 Mbps FSK		8+ months	patch	Active
Re	38765	3U	Helium-100	915 MHz	government	1 W	Integrated	AX.25	57.6 kbps FSK			dipole	

[1] This is also the main satellite processor.

[2] Uplink commands were never received by this satellite.

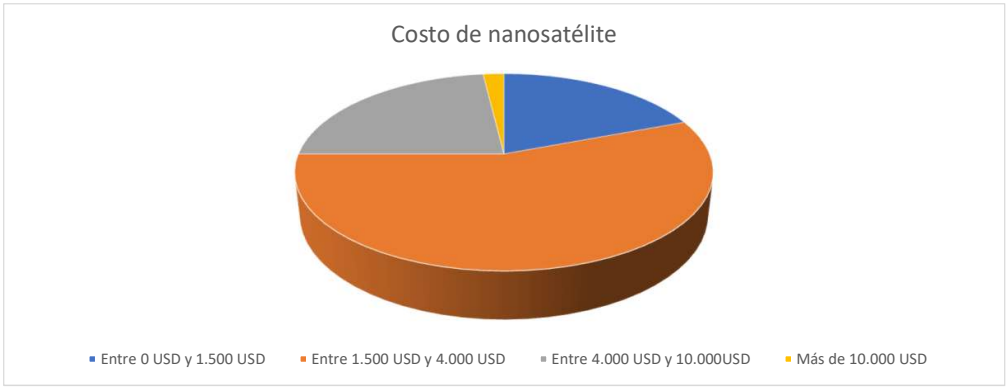
[3] This spacecraft did receive uplink commands, but it died before before downlink could be established.

Figura 49: Comparación de nanosatélites.

26. Resultados de Encuentas

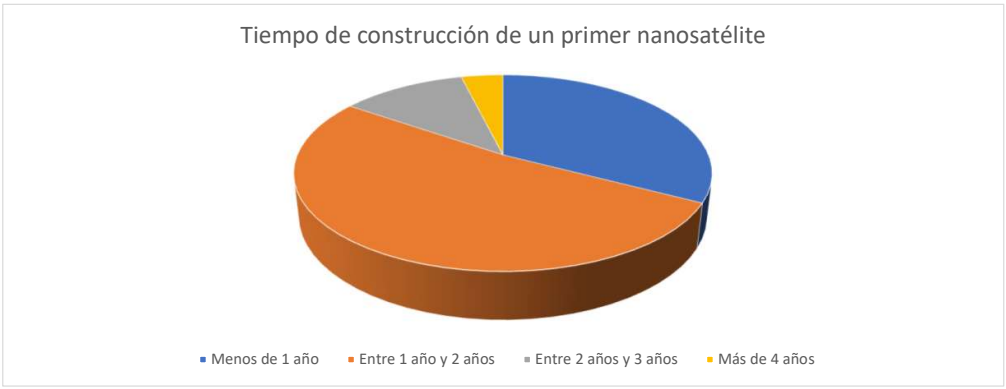
Cúanto estima que será su presupuesto para construir un prototipo de nanosatélite (Sin incluir el costo de puesta en órbita)

Entre 0 USD y 1.500 USD	10
Entre 1.500 USD y 4.000 USD	29
Entre 4.000 USD y 10.000USD	12
Más de 10.000 USD	1



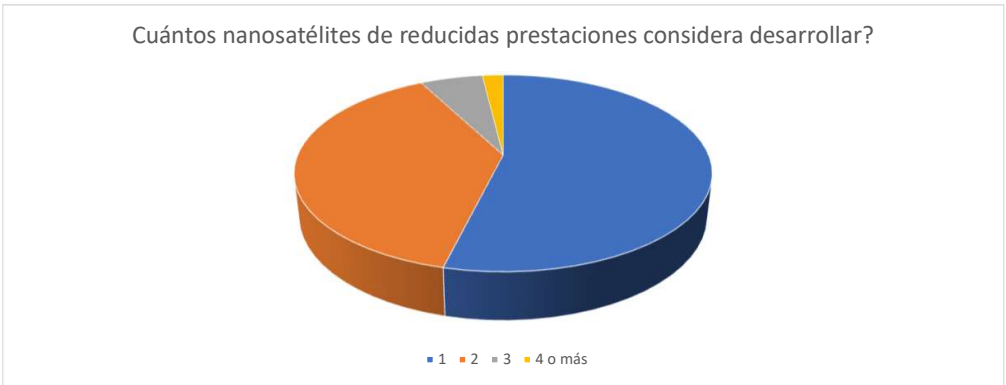
Suponiendo que nunca construyó un nanosatélite, cuánto piensa que demoraría en construir su primer nanosatélite?

Menos de 1 año	17
Entre 1 año y 2 años	27
Entre 2 años y 3 años	6
Más de 4 años	2



Cuántos nanosatélites de reducidas prestaciones considera desarrollar?

1	28
2	20
3	3
4 o más	1



En que año le gustaría empezar un nanosatélite?

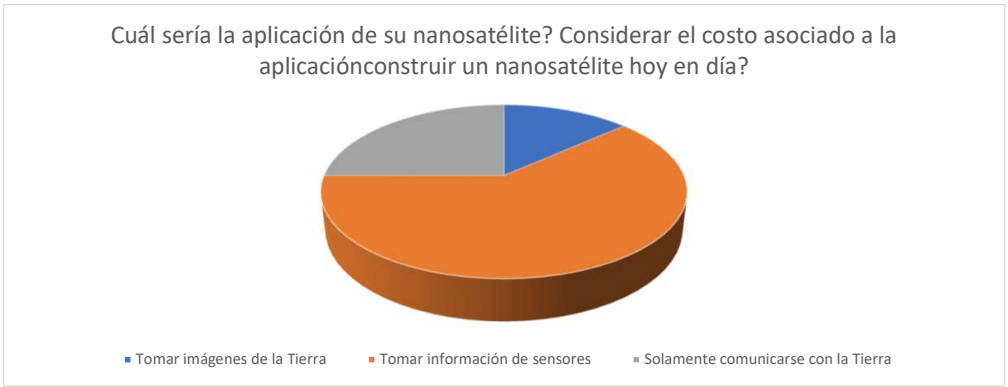
2017	5
2018	24
2019	19
2020 o más	4

Cuál siente que es el mayor limitante para construir un nanosatélite hoy en día?

Falta de interés	5
Falta de presupuesto	24
Falta de tiempo	8
Falta de conocimiento	15

Cuál sería la aplicación de su nanosatélite? Considerar el costo asociado a la aplicación

Tomar imágenes de la Tierra	7
Tomar información de sensores	32
Solamente comunicarse con la Tierra	13



Cuántos sensores considera necesarios para su nanosatélite?	
0 a 2	10
3 a 5	27
6 a 10	13
Más de 10	2

27. Rutinas de Software

A continuación se agrupan las rutinas de Software del prototipo.

```
/*
 * ADC.c
 */
#include "ADC.h"

/***
 ADCPRESCALER :
                  00 prescaler = 1
                  01 prescaler = 2
                  10 prescaler = 4
                  11 prescaler = 8

MODE:
        00 DIFF 0   8bit
        00 DIFF 1   9bit diff complento a 2
        01 DIFF 0   12bit
        01 DIFF 1   13bit diff complento a 2
        10 DIFF 0   10bit
        10 DIFF 1   11bit diff complento a 2
        11 DIFF 0   16bit
        11 DIFF 1   16bit diff complento a 2

ADICLK:
        00  Bus Clock
        01  Alternate clock 2
        10  Alternate clock
        11  Asynchronous clock

AVGS:
        00 4 samples averaged.
        01 8 samples averaged.
        10 16 samples averaged.
        11 32 samples averaged.

ADCH:
        00000 When DIFF=0, DADP0 is selected as input; when DIFF=1, DAD0 is selected
        as input.
        00001 When DIFF=0, DADP1 is selected as input; when DIFF=1, DAD1 is selected
        as input.
        00010 When DIFF=0, DADP2 is selected as input; when DIFF=1, DAD2 is selected
        as input.
        00011 When DIFF=0, DADP3 is selected as input; when DIFF=1, DAD3 is selected
        as input.
        00100 When DIFF=0, AD4 is selected as input; when DIFF=1, it is reserved.
        00101 When DIFF=0, AD5 is selected as input; when DIFF=1, it is reserved.
        00110 When DIFF=0, AD6 is selected as input; when DIFF=1, it is reserved.
        00111 When DIFF=0, AD7 is selected as input; when DIFF=1, it is reserved.
        01000 When DIFF=0, AD8 is selected as input; when DIFF=1, it is reserved.
        01001 When DIFF=0, AD9 is selected as input; when DIFF=1, it is reserved.
        01010 When DIFF=0, AD10 is selected as input; when DIFF=1, it is reserved.
        01011 When DIFF=0, AD11 is selected as input; when DIFF=1, it is reserved.
        01100 When DIFF=0, AD12 is selected as input; when DIFF=1, it is reserved.
        01101 When DIFF=0, AD13 is selected as input; when DIFF=1, it is reserved.
        01110 When DIFF=0, AD14 is selected as input; when DIFF=1, it is reserved.
        01111 When DIFF=0, AD15 is selected as input; when DIFF=1, it is reserved.
        10000 When DIFF=0, AD16 is selected as input; when DIFF=1, it is reserved.
        10001 When DIFF=0, AD17 is selected as input; when DIFF=1, it is reserved.
        10010 When DIFF=0, AD18 is selected as input; when DIFF=1, it is reserved.
        10011 When DIFF=0, AD19 is selected as input; when DIFF=1, it is reserved.
        10100 When DIFF=0, AD20 is selected as input; when DIFF=1, it is reserved.
        10101 When DIFF=0, AD21 is selected as input; when DIFF=1, it is reserved.
        10110 When DIFF=0, AD22 is selected as input; when DIFF=1, it is reserved.
        10111 When DIFF=0, AD23 is selected as input; when DIFF=1, it is reserved.
        11000 Reserved.
        11001 Reserved.
        11010 When DIFF=0, Temp Sensor (single-ended) is selected as input; when
        DIFF=1, Temp Sensor
             (differential) is selected as input.
        11011 When DIFF=0,Bandgap (single-ended) is selected as input; when DIFF=1,
        Bandgap (differential)
              is selected as input.
        11100 Reserved.
        11101 When DIFF=0,VREFSH is selected as input; when DIFF=1, -VREFSH
```

```
          (differential) is selected as
                input. Voltage reference selected is determined by SC2[REFSEL].
          11110 When DIFF=0,VREFSL is selected as input; when DIFF=1, it is reserved.
          Voltage reference
                selected is determined by SC2[REFSEL].
          11111 Module is disabled.

CONVERSION VALUE: ADC1_R(index)
DRIVER STATUS:  (ADC_SC2 & ADC_SC2_ADACT_MASK) --> "0" WHEN Conversion not in
progress, "1" WHEN Conversion in progress;
CALIBRATION STATUS: (ADC_SC3 & ADC_SC3_CALF_MASK) --> "0" WHEN Completed normally,
"1" WHEN Calibration failed;

 */
#define ADCPRESCALER 00
#define ADCMODE 00
#define ADCCLOCK 00
#define ADCCHANNEL 00001

static volatile uint8_t adcChannel = 0x00;
static volatile uint8_t convCompl0 = 0;
static volatile uint8_t convCompl1 = 0;
static volatile uint8_t measADC0 = 0;
static volatile uint8_t measADC1 = 0;
static volatile uint8_t convStartedADC0 = 0;
static volatile uint8_t convStartedADC1 = 0;

void ADC_init(ADC_MemMapPtr base){

    // Habilito la llave general de las interrupciones

    //Habilito el modulo
    if(base == ADC1_BASE_PTR){
        SIM_SCGC3 |= SIM_SCGC3_ADC1_MASK;
        NVICISER2 |= NVIC_ISER_SETENA(0x0200);
    }
    else{
        SIM_SCGC6 |= SIM_SCGC6_ADC0_MASK;
        NVICISER1 |= NVIC_ISER_SETENA(0x80);
    }
    //Configuro el clock y el funcionamiento

    ADC_CFG1_REG(base) = ADC_CFG1_ADLPC_MASK | ADC_CFG1_ADIV(ADCPRESCALER) |
    ADC_CFG1_MODE(ADCMODE) | ADC_CFG1_ADICLK(ADCCLOCK);
    //Seteo modo high speed convertion
    ADC_CFG2_REG(base) = ADC_CFG2_ADHSC_MASK;
    ADC_SC2_REG(base) = 0x00;
    ADC_SC1_REG(base,0) = ADC_SC1_ADCH(0x00);
    //
}

void ADC_calib (ADC_MemMapPtr base)
{
    uint16_t calib = 0x00;
    calib = (ADC_CLP0_REG(base) + ADC_CLP1_REG(base) + ADC_CLP2_REG(base) +
    ADC_CLP3_REG(base) + ADC_CLP4_REG(base) + ADC_CLPS_REG(base))/2;
    calib |= 1<<15;


    calib = 0x00;
    calib = (ADC_CLM0_REG(base) + ADC_CLM1_REG(base) + ADC_CLM2_REG(base) +
    ADC_CLM3_REG(base) + ADC_CLM4_REG(base) + ADC_CLMS_REG(base))/2;
    calib |= 1<<15;

    //Begins Calibration
    ADC_SC3_REG(base) |= ADC_MG_MG(calib) | ADC_PG_PG(calib);

}

void ADC_enableAverage(ADC_MemMapPtr base, uint8_t avge){
    ADC_SC3_REG(base) = ADC_SC3_AVGS(avge) | ADC_SC3_AVGE_MASK;
}

```

```
134 void ADC_disableAverage(ADC_MemMapPtr base){
135     ADC_SC3_REG(base) &= ~ADC_SC3_AVGE_MASK;
136 }
137
138 void ADC_initConversion(ADC_MemMapPtr base, uint8_t channel){
139     //Init conversion
140     ADC_SC1_REG(base,0) = ADC_SC1_AIEN_MASK | ADC_SC1_ADCH(channel);
141     if(base == ADC0_BASE_PTR){
142         convCompl0 = 0;
143         convStartedADC0 = 1;
144     }
145     else{
146         convCompl1 = 0;
147         convStartedADC1 = 1;
148     }
149 }
150
151 uint8_t ADC_convCompl(ADC_MemMapPtr base){
152     if(base == ADC0_BASE_PTR)
153         return convCompl0;
154     else
155         return convCompl1;
156 }
157
158 uint8_t ADC_convIsStarted(ADC_MemMapPtr base){
159     if(base == ADC0_BASE_PTR)
160         return convStartedADC0;
161     else
162         return convStartedADC1;
163 }
164
165 uint8_t ADC_result(ADC_MemMapPtr base){
166     if(base == ADC0_BASE_PTR){
167         convCompl0 = 0;
168         convStartedADC0 = 0;
169         return measADC0;
170     }
171     else{
172         convCompl1 = 0;
173         convStartedADC1 = 0;
174         return measADC1;
175     }
176 }
177
178 void ADC_PISR(void){
179     //adcChannel = !adcChannel; //Uso los canales 0 y 1
180     ADC_initConversion(ADC_BASE, ADC_CHANNEL);
181 }
182
183 ISR_t ADC1_IRQHandler(void){
184     //Interruption is cleared upon reading of Rn register or SC1n is written
185     //ADC_CompleteInterrupt();
186     measADC1 = ADC1_RA;
187     convCompl1 = 1;
188 }
189
190 ISR_t ADC0_IRQHandler(void){
191     //Interruption is cleared upon reading of Rn register or SC1n is written
192     //ADC_CompleteInterrupt();
193     measADC0 = ADC0_RA;
194     convCompl0 = 1;
195 }
196
197
```

```c
/*
 * ADC.h
 */

#ifndef ADC_H_
#define ADC_H_

#include "misc.h"

#define ADC_CHANNEL 0x00
#define ADC_BASE    ADC0_BASE_PTR

void ADC_init(ADC_MemMapPtr base);
void ADC_calib (ADC_MemMapPtr base);
void ADC_enableAverage(ADC_MemMapPtr base, uint8_t avge);
void ADC_disableAverage(ADC_MemMapPtr base);
void ADC_initConversion(ADC_MemMapPtr base, uint8_t channel);
uint8_t ADC_convCompl(ADC_MemMapPtr base);
uint8_t ADC_result(ADC_MemMapPtr base);
void ADC_PISR(void);
ISR_t ADC1_IRQHandler(void);
uint8_t ADC_convIsStarted(ADC_MemMapPtr base);
#endif /* ADC_H_ */

```

```c
#include "StateMachine.h"
#include "App.h"
#include "GPIO.h"
#include "ADC.h"
#include "UART.h"
#include "fsmdefines.h"

void App_init (void)
{
    //StateMachine_init();
    SPI_API_Init();

    UART_init();
    FTM_init();
    FTM_IC(RISINGEDGE);
    RH_RF95_init();
    FTM0_enable();
    GPIO_init(FORCE_PIN);
    GPIO_init(READ_PIN);
    GPIO_init(ERROR_PIN);
    GPIO_purpose(FORCE_PIN, GPIOOUT);
    GPIO_purpose(READ_PIN, GPIOOUT);
    GPIO_purpose(ERROR_PIN, GPIOOUT);
    GPIO_write(FORCE_PIN, FORCE_PIN_SHUTDOWN_OFF);
    GPIO_write(READ_PIN, READ_PIN_NO_DATA);
    GPIO_write(ERROR_PIN, ERROR_NONE_AVAILABLE);
    SysTick_init(1);
    ADC_init(ADC_BASE);

    StateMachine_init();

}

void App_run (void)
{
    for(;;){
        StateMachine_run();

    };
}

```

```
1
2   #ifndef APP_H_
3   #define APP_H_
4
5   #include "misc.h"
6   #include "FTM.h"
7   #include "UART.h"
8   #include "RH_RF95.h"
9   #include "SysTick.h"
10  #include "StateMachine.h"
11  #include "SPI_API.h"
12
13  void App_init (void);
14  void App_run (void);
15
16  #endif
17
```

```
/*
 * fsmdefines.h
 */

#ifndef FSMDEFINES_H_
#define FSMDEFINES_H_

#define     ERROR_NONE                                  0x00
#define     ERROR_AVAILABLE                             0x01
#define     ERROR_NONE_AVAILABLE                        0x00
#define     ERROR_TEMP                                  0x01
#define     ERROR_TX                                    0x02
#define     ERROR_RX                                    0x03
#define     ERROR_ALIM                                  0x04
#define     ERROR_RESET                                 0x05
#define     ERROR_RX_TEMP                               0x06
#define     ERROR_TX_TEMP                               0x07
#define     ERROR_RX_COMM                               0x08
#define     ERROR_TX_COMM                               0x09

#define     FSM_STBY_ALL_EVENT                          0x00
#define     FSM_LP_ALL_EVENT                            0x01
#define     FSM_SLEEP_ALL_EVENT                         0x02
#define     FSM_BROADCAST_EVENT                         0x03
#define     FSM_BROADCAST_STOP_EVENT                    0x04
#define     FSM_SEND_EVENT                              0x05
#define     FSM_RECEIVE_EVENT                           0x06
#define     FSM_LENGTH_EVENT                            0x16
#define     FSM_NO_EVENT                                0x07
#define     FSM_EXTERNAL_EVENT_NOT_VALID                0x08
#define     FSM_SHUTDOWN_EVENT                          0x09
#define     FSM_TURNON_EVENT                            0x0A
#define     FSM_ERROR_RESET_EVENT                       0x0B
#define     FSM_FORCE_CLEARPIN_EVENT                    0x0D

#define     RX_RECEIVE_EVENT                            0x00
#define     RX_LP_EVENT                                 0x01
#define     RX_STBY_EVENT                               0x02
#define     RX_RECEIVE_SPACE_EVENT                      0x03
#define     RX_NO_EVENT                                 0x07

#define     TX_LP_EVENT                                 0x01
#define     TX_SEND_EVENT                               0x00
#define     TX_STBY_EVENT                               0x02
#define     TX_FORCE_EVENT                              0x04
#define     TX_NO_EVENT                                 0x07

#define     STATEMACHINE_NOSTATE                        0x00
#define     STATEMACHINE_EXPECTINGLENGTH                0x01
#define     STATEMACHINE_RECEIVING                      0x02

#define     SPACE_NOSTATE                               0x00
#define     SPACE_EXPECTINGLENGTH                       0x01
#define     SPACE_RECEIVING                             0x02

#define     TXBUFFERSIZE                                1200
#define     RXBUFFERSIZE                                1200

#define     FORCE_PIN                                   0x01
#define     READ_PIN                                    0x02
#define     ERROR_PIN                                   0x00

#define     FORCE_PIN_SHUTDOWN                          0x01
#define     FORCE_PIN_SHUTDOWN_OFF                      0x00

#define     READ_PIN_DATA_AVAILABLE                     0x01
#define     READ_PIN_NO_DATA                            0x00

#define     ERROR_PIN_ERROR_AVAILABLE                   0x01
#define     ERROR_PIN_NO_ERROR                          0x00

#define     ADC_ERROR_LEVEL                             0x00

```

```
74  #define     ID_RX                                   0x01
75  #define     ID_TX                                   0x02
76  #define     ID_BOTH                                 0x03
77
78  #define     NEW_COMMAND_COMPLETE                    0x01
79  #define     NO_COMMAND                              0x02
80
81  #endif /* FSMDEFINES_H_ */
82
```

```c
#include <stdio.h>
#include "fsmRx.h"
#include "fsmtableRx.h"

extern STATE *laststate;

STATE* fsmRx(STATE *p_tabla_estado,BYTE evento_actual)
{
    while (p_tabla_estado -> evento != evento_actual        //Recorre las tablas de estado
           && p_tabla_estado -> evento != FIN_TABLA)
    ++p_tabla_estado;
    (*p_tabla_estado -> p_rut_accion) ();          /*rutina de accion correspondiente*/
    p_tabla_estado=p_tabla_estado -> prx_estado;   /*siguiente estado*/
    return(p_tabla_estado);
}
```

```
/*
 * fsmtable.h
 */
#ifndef FSMTABLE_H_
#define FSMTABLE_H_

#include "fsmMain.h"
#include "fsmtableMain.h"
#include "StateMachineDefs.h"
    //CARACTERES
    /*
    s stby
    l low power
    m mandar
    r recibi algo
    d dormir
    b broadcasting
    n no mas broadcasting
    l salgo de error
    a error alimentacion
    t error de temperatura
    x error de tx
    c error de rx
    */
/*Forward Declarations*/
extern STATE lp_state[];
extern STATE stby[];
extern STATE erroralim[];
extern STATE errortemp[];
extern STATE errortx[];
extern STATE errorRx[];
extern STATE errorBoth[];
///***PROTOTIPOS***///

void do_nothing(void);
void reset_FSM(void);
///*** lp_state***///
void lp2stby (void);
/*** stby ***/
void stby2lp_state(void);
void stby2stbysending (void);
void stby2stbybdctg(void);
void stby2stbystopbdctg(void);
void stby2erroralim (void);
void stby2errortemp(void);
void stby2errorRx(void);
void stby2errortx (void);
void stby2receive(void);

/*** errores ***/
void erroralim2stby(void);
void errortemp2stby(void);
void errortx2stby(void);
void errorRx2stby(void);
void error2errorBoth(void);
void errorBoth2stby(void);
void forceClearPin(void);
STATE *FSM_GetInitState(void);
void stby2length(void);
#endif /* FSMTABLE_H_ */

void forceShutdown (void);

```

```c
#include "fsmtableMain.h"
#include "fsmdefines.h"
#include "GPIO.h"
#include "UART.h"

extern int idOff;
extern unsigned char rxReadBuffer[RXBUFFERSIZE];

extern unsigned int rxWriteBufferPointer;
extern unsigned int rxReadBufferPointer;
extern unsigned int txWriteBufferPointer;
extern unsigned int txReadBufferPointer;

/*** tablas de estado ***/

    //CARACTERES
    /*
    s stby
    l low power
    m mandar
    r recibi algo
    d dormir
    b broadcasting
    n no mas broadcasting
    l salgo de error
    a error alimentacion
    t error de temperatura
    x error de tx
    c error de rx
    */

/*** lp_state ***/
 STATE lp_state[]=
{
    {FSM_STBY_ALL_EVENT,stby,lp2stby},
    {FSM_FORCE_CLEARPIN_EVENT,lp_state,forceClearPin},
    {FIN_TABLA,lp_state,do_nothing}
};

/*** stby ***/
STATE stby[]=
{
    {FSM_LP_ALL_EVENT,lp_state,stby2lp_state},
    {FSM_SEND_EVENT,stby,stby2stbysending},
    {FSM_RECEIVE_EVENT,stby,stby2receive},
    {FSM_LENGTH_EVENT,stby,stby2length},
    {ERROR_ALIM,erroralim,stby2erroralim},
    {ERROR_TEMP,errortemp,stby2errortemp},
    {ERROR_TX,errortx,stby2errortx},
    {ERROR_RX,errorRx,stby2errorRx},
    {FSM_FORCE_CLEARPIN_EVENT,stby,forceClearPin},
    {FSM_SHUTDOWN_EVENT,stby,forceShutdown},
    {FIN_TABLA,stby,do_nothing}
};

/*** erroralim ***/
STATE erroralim[]=
{
    {FSM_ERROR_RESET_EVENT,stby,erroralim2stby},
    {FSM_FORCE_CLEARPIN_EVENT,erroralim,forceClearPin},
    {FIN_TABLA,erroralim,do_nothing}
};
/*** errortemp ***/
STATE errortemp[]=
{
    {FSM_ERROR_RESET_EVENT,stby,errortemp2stby},
    {FSM_FORCE_CLEARPIN_EVENT,errortemp,forceClearPin},
    {FIN_TABLA,errortemp,do_nothing}
};
/*** errortx ***/
STATE errortx[]=
{
    {FSM_ERROR_RESET_EVENT,stby,errortx2stby},
```

```
    {FSM_FORCE_CLEARPIN_EVENT,errortx,forceClearPin},
    {ERROR_RX,errorBoth,error2errorBoth},
    {FIN_TABLA,errortx,do_nothing}
};
/*** errorRx ***/
STATE errorRx[]=
{
    {FSM_ERROR_RESET_EVENT,stby,errorRx2stby},
    {ERROR_TX,errorBoth,error2errorBoth},
    {FSM_FORCE_CLEARPIN_EVENT,errorRx,forceClearPin},
    {FIN_TABLA,errorRx,do_nothing}
};
/*** errorBoth ***/
STATE errorBoth[]=
{
    {FSM_ERROR_RESET_EVENT,stby,errorBoth2stby},
    {FSM_FORCE_CLEARPIN_EVENT,errorBoth,forceClearPin},
    {FIN_TABLA,errorBoth,do_nothing}
};
///=========Rutinas de accion===============
/*Dummy function*/
void do_nothing(void)
{

}

//========interfaz=================
STATE *FSM_GetInitState(void){
    return (lp_state);
    //escribir HI 11 en el teclado
}

/*Restart FSM*/
void reset_FSM(void){

}
///*** lp_state***///
void lp2stby (void){
    GPIO_write(FORCE_PIN, FORCE_PIN_SHUTDOWN_OFF);
    //UART_writeByte('a');
}

/*** stby ***/
void stby2lp_state (void){
    // DO NOTHING
}
void stby2stbysending(void) {
    // Cargar en el buffer
}

void stby2receive(void) {
    if(rxWriteBufferPointer != rxReadBufferPointer)
        UART_writeByte(rxReadBuffer[rxReadBufferPointer++]);
    if(rxReadBufferPointer == RXBUFFERSIZE)
        rxReadBufferPointer = 0;
    if(rxWriteBufferPointer == rxReadBufferPointer)
        GPIO_write(READ_PIN, READ_PIN_NO_DATA);
}

void stby2length(void){
    unsigned int aux = rxWriteBufferPointer - rxReadBufferPointer;
    if(aux < 0)
        aux += RXBUFFERSIZE;
    UART_writeByte((unsigned char) (aux & 0x00FF));
    UART_writeByte((unsigned char) ((aux & 0xFF00) >> 8));
}

void stby2stbybdctg(void) {
    // DO NOTHING
}
void stby2stbystopbdctg (void){

}
```

```
147  void stby2erroralim(void) {
148      GPIO_write(ERROR_PIN, ERROR_AVAILABLE);
149  }
150  void stby2errortemp(void) {
151      GPIO_write(ERROR_PIN, ERROR_AVAILABLE);
152  }
153  void stby2errortx(void) {
154      GPIO_write(ERROR_PIN, ERROR_AVAILABLE);
155      }
156  void stby2errorRx(void) {
157      GPIO_write(ERROR_PIN, ERROR_AVAILABLE);
158      }
159  void error2errorBoth(void) {
160      GPIO_write(ERROR_PIN, ERROR_AVAILABLE);
161  }
162  /*** errores ***/
163  void erroralim2stby(void) {
164      GPIO_write(ERROR_PIN, ERROR_NONE_AVAILABLE);
165  }
166  void errortemp2stby(void) {
167      GPIO_write(ERROR_PIN, ERROR_NONE_AVAILABLE);
168  }
169
170  void errortx2stby(void) {
171      GPIO_write(ERROR_PIN, ERROR_NONE_AVAILABLE);
172  }
173  void errorRx2stby(void) {
174      GPIO_write(ERROR_PIN, ERROR_NONE_AVAILABLE);
175  }
176
177  void errorBoth2stby(void) {
178      GPIO_write(ERROR_PIN, ERROR_NONE_AVAILABLE);
179  }
180
181  void forceClearPin(void){
182      //GPIO_write(FORCE_PIN, FORCE_PIN_SHUTDOWN_OFF);
183  }
184
185  void forceShutdown(void){
186      GPIO_write(FORCE_PIN, FORCE_PIN_SHUTDOWN);
187  }
188
```

```c
/*
 * fsm.h
 */
#ifndef FSMRX_H_
#define FSMRX_H_

#include "StateMachineDefs.h"

// Interfaz
STATE* fsmRx(STATE *p_tabla_estado,BYTE evento_actual);

#endif /* FSM_H_ */

```

```
/*
 * fsmtable.h
 */
#ifndef FSMTABLE_H_
#define FSMTABLE_H_

#include "fsmMain.h"
#include "fsmtableMain.h"
#include "StateMachineDefs.h"
    //CARACTERES
    /*
    s stby
    l low power
    m mandar
    r recibi algo
    d dormir
    b broadcasting
    n no mas broadcasting
    l salgo de error
    a error alimentacion
    t error de temperatura
    x error de tx
    c error de rx
    */
/*Forward Declarations*/
extern STATE lp_state[];
extern STATE stby[];
extern STATE erroralim[];
extern STATE errortemp[];
extern STATE errortx[];
extern STATE errorRx[];
extern STATE errorBoth[];
///***PROTOTIPOS***///

void do_nothing(void);
void reset_FSM(void);
///*** lp_state***///
void lp2stby (void);
/*** stby ***/
void stby2lp_state(void);
void stby2stbysending (void);
void stby2stbybdctg(void);
void stby2stbystopbdctg(void);
void stby2erroralim (void);
void stby2errortemp(void);
void stby2errorRx(void);
void stby2errortx (void);
void stby2receive(void);

/*** errores ***/
void erroralim2stby(void);
void errortemp2stby(void);
void errortx2stby(void);
void errorRx2stby(void);
void error2errorBoth(void);
void errorBoth2stby(void);
void forceClearPin(void);
STATE *FSM_GetInitState(void);
void stby2length(void);
#endif /* FSMTABLE_H_ */

void forceShutdown (void);

```

```c
#include <stdio.h>
#include "fsmdefines.h"
#include "fsmtableRx.h"
#include "GPIO.h"
#include "RH_RF95.h"

/*** tablas de estado ***/

    /*
    s stbyRx
    l low power
    r recibi
    */

/*** lp_stateRx ***/
 STATE lp_stateRx[]=
{
    {RX_STBY_EVENT,stbyRx,lp2stbyRx},
    {FIN_TABLA,lp_stateRx,do_nothingRx}
};

/*** stbyRx ***/
STATE stbyRx[]=
{
    {RX_LP_EVENT,lp_stateRx,stby2lp_stateRx},
    {RX_RECEIVE_SPACE_EVENT,stbyRx,stby2stbyRx},
    {FIN_TABLA,stbyRx,do_nothingRx}
};


///=========Rutinas de accion===============
/*Dummy function*/
void do_nothingRx(void)
{

}

//========interfaz=================
STATE *FSM_GetInitStateRx(void)
{
    return (lp_stateRx);
    //escribir HI 11 en el teclado
}

/*Restart FSM*/
void reset_FSMRx(void)
{

}
///*** lp_stateRx***///
void lp2stbyRx(void){
    RH_RF95_RX_STBY();
}


/*** stbyRx ***/
void stby2lp_stateRx(void){
    RH_RF95_RX_LP();
}
void stby2stbyRx(void) {
    GPIO_write(READ_PIN, READ_PIN_DATA_AVAILABLE);
}

```

```c
/*
 * fsmtable.h
 */
#ifndef FSMTABLERX_H_
#define FSMTABLERX_H_

#include "fsmRx.h"
#include "fsmtableRx.h"
    //CARACTERES
    /*
    s stby
    l low power
    r recibi
    */
/*Foward Declarations*/
extern STATE lp_stateRx[];
extern STATE stbyRx[];
///***PROTOTIPOS***///
void do_nothingRx(void);
void reset_FSMRx(void);
///*** lp_state***///
void lp2stbyRx(void);
/*** stby ***/
void stby2lp_stateRx(void);
void stby2stbyRx(void);
STATE *FSM_GetInitStateRx(void);
#endif /* FSMTABLE_H_ */

```

```c
#include "fsmtableTx.h"
#include "fsmdefines.h"
/*** tablas de estado ***/

    //CARACTERES
    /*
    s stby
    l low power
    m mandar
    f fin de mandar
    b broadcasting
    n no mas broadcasting
    */

/*** lp_state ***/

extern unsigned int rxWriteBufferPointer;
extern unsigned int rxReadBufferPointer;
extern unsigned int txWriteBufferPointer;
extern unsigned int txReadBufferPointer;

extern unsigned char txWriteBuffer[RXBUFFERSIZE];

 STATE lp_stateTx[]=
{
    {TX_STBY_EVENT,stbyTx,lp2stbyTx},
    {TX_FORCE_EVENT,lp_stateTx,lp2forceTx},
    {FIN_TABLA,lp_stateTx,do_nothingTx}
};

/*** stby ***/
STATE stbyTx[]=
{
    {TX_LP_EVENT,lp_stateTx,stby2lp_stateTx},
    {TX_SEND_EVENT,sendingTx,stby2sendingTx},
    {TX_FORCE_EVENT,stbyTx,stby2forceTx},
    {FIN_TABLA,stbyTx,do_nothingTx}
};

/*** sending ***/
STATE sendingTx[]=
{
    {TX_LP_EVENT,lp_stateTx,sending2lp_stateTx},
    {TX_STBY_EVENT,stbyTx,sending2stbyTx},
    {TX_FORCE_EVENT,sendingTx,sending2forceTx},
    {FIN_TABLA,sendingTx,do_nothingTx}
};
/*** broadcasting ***/
STATE broadcastingTx[]=
{
    {'l',lp_stateTx,broadcasting2lp_stateTx},
    {'s',stbyTx,broadcasting2stbyTx},
    {FIN_TABLA,stbyTx,do_nothingTx}
};


///=========Rutinas de accion===============
/*Dummy function*/
void do_nothingTx(void)
{

}

//========interfaz=================
STATE *FSM_GetInitStateTx(void)
{
    return (lp_stateTx);
    //escribir HI 11 en el teclado
}

/*Restart FSM*/
void reset_FSMTx(void)
{
```

```
74
75  }
76  ///*** lp_state***///
77  void lp2stbyTx (void){
78      RH_RF95_TX_STBY();
79  }
80  void lp2bdctgTx (void){
81      //prender el transmisor y decirle que mire el buffer de bcsting
82  }
83
84  /*** stby ***/
85  void stby2lp_stateTx (void){
86      RH_RF95_TX_LP();
87  }
88  void stby2sendingTx(void) {
89      unsigned int aux = txReadBufferPointer;
90      RH_RF95_SendPacket(txWriteBuffer[txReadBufferPointer++]);
91      if (txReadBufferPointer==TXBUFFERSIZE)
92          txReadBufferPointer=0;
93  }
94  void stby2bdctgTx(void) {
95
96  }
97
98  /*** sending ***/
99  void sending2lp_stateTx(void) {
100     RH_RF95_TX_LP();
101 }
102 void sending2stbyTx(void) {
103     RH_RF95_TX_STBY();
104 }
105
106 /*** broadcasting ***/
107 void broadcasting2lp_stateTx(void) {
108     //apago el tx
109 }
110 void broadcasting2stbyTx(void) {
111     //apago el tx
112 }
113
114 void stby2forceTx(void) {
115     GPIO_write(FORCE_PIN, FORCE_PIN_SHUTDOWN);
116 }
117 void lp2forceTx(void) {
118     GPIO_write(FORCE_PIN, FORCE_PIN_SHUTDOWN);
119 }
120 void sending2forceTx(void) {
121     GPIO_write(FORCE_PIN, FORCE_PIN_SHUTDOWN);
122 }
123
```

```c
/*
 * fsmtable.h
 */
#ifndef FSMTABLETX_H_
#define FSMTABLETX_H_

#include "fsmTx.h"
#include "fsmtableTx.h"
#include "StateMachineDefs.h"
    //CARACTERES
    /*
    s stby
    l low power
    m mandar
    f fin de mandar
    b broadcasting
    n no mas broadcasting
    */
/*Foward Declarations*/
extern STATE lp_stateTx[];
extern STATE stbyTx[];
extern STATE sendingTx[];
extern STATE broadcastingTx[];
///***PROTOTIPOS***///

void do_nothingTx(void);
void reset_FSMTx(void);
///*** lp_stateTx***///
void lp2stbyTx(void);
void lp2bdctgTx(void);
void lp2forceTx(void);
/*** stby ***/
void stby2lp_stateTx(void);
void stby2sendingTx(void);
void stby2bdctgTx(void);
void stby2forceTx(void);
/*** sending ***/
void sending2lp_stateTx(void);
void sending2stbyTx(void);
void sending2forceTx(void);
/*** broadcasting ***/
void broadcasting2lp_stateTx(void);
void broadcasting2stbyTx(void);
STATE *FSM_GetInitStateTx(void);
#endif /* FSMTABLE_H_ */

```

```c
#include <stdio.h>
#include "fsmTx.h"
#include "fsmtableTx.h"

extern STATE *laststate;

STATE* fsmTx(STATE *p_tabla_estado,BYTE evento_actual)
{
    printf(" >>%c<<\n ",evento_actual); //just for test (debug)
    while (p_tabla_estado -> evento != evento_actual        //Recorre las tablas de estado
           && p_tabla_estado -> evento != FIN_TABLA)
    ++p_tabla_estado;
    (*p_tabla_estado -> p_rut_accion) ();          /*rutina de accion correspondiente*/
    p_tabla_estado=p_tabla_estado -> prx_estado;   /*siguiente estado*/
    return(p_tabla_estado);
}
```

```c
/*
 * FTM.c
 */
#include "FTM.h"
#include "UART.h"
#include "RH_RF95.h"

void FTM_init  (void)
{
    // Habilito el pin PTC1 como salida del canal 0
    PORTC_PCR1 = (PORT_PCR_ISF_MASK |
                  PORT_PCR_MUX(0x04));
    PORTC_PCR2 = (PORT_PCR_ISF_MASK |
                        PORT_PCR_MUX(0x04));

    // Habilito la llave general de las interrupciones
    NVICISER1 |= NVIC_ISER_SETENA(0x0400);

    // Habilito el clock gating
    SIM_SCGC6 |= SIM_SCGC6_FTM0_MASK;

    // Deshabilito la proteccion de escritura
    FTM0_MODE       = FTM_MODE_WPDIS_MASK;

    // Apago los flag de interrupciones
    FTM0_STATUS     = (FTM0_STATUS &
                       0x00);

    // Deshabilito los clocks y borro la configuracion
    FTM0_SC         = 0x00;

    // Configuro el canal 0 como nada
    FTM0_C0SC       = 0x00;
    FTM0_C1SC       = 0x00;

    // Configuro la salida inicial en alto (los LEDs son activo-bajo)
    FTM0_MODE       = FTM_MODE_INIT_MASK;

    // Configuro el PWM a duty 25% y reinicio el contador
    FTM0_C0V        = 0x0000;
    FTM0_C1V        = 0x0000;
    FTM0_MOD        = 0x0000;
    FTM0_CNTIN      = 0x0000;
    FTM0_CNT        = 0x0000;


    // Habilito las interrupciones y los clocks, y configuro el prescaler
    FTM0_SC         = (FTM_SC_CLKS(0x01) |
                       FTM_SC_PS(0x00));
    // Habilito los contadores del modulo
    FTM0_CONF       = FTM_CONF_BDMMODE(0x03);
}


void FTM_IC(bool edge){
    FTM0_C0SC   &=  ~FTM_CnSC_MSA_MASK;
    FTM0_C0SC   &=  ~FTM_CnSC_MSB_MASK;
    FTM0_C1SC   &=  ~FTM_CnSC_MSA_MASK;
    FTM0_C1SC   &=  ~FTM_CnSC_MSB_MASK;
    FTM0_SC &=  ~FTM_SC_CPWMS_MASK;
    FTM0_COMBINE &= ~FTM_COMBINE_COMBINE0_MASK;
    FTM0_COMBINE &= ~FTM_COMBINE_DECAPEN0_MASK;


    FTM0_CnSC(SPI_CH0)  |=  FTM_CnSC_ELSA_MASK;
    FTM0_CnSC(SPI_CH0)  &=  ~FTM_CnSC_ELSB_MASK;

    FTM0_CnSC(SPI_CH1)  |=  FTM_CnSC_ELSA_MASK;
    FTM0_CnSC(SPI_CH1)  &=  ~FTM_CnSC_ELSB_MASK;

    //FTM0_CnSC(SPI_CH1)    |=  FTM_CnSC_ELSB_MASK;
    //FTM0_CnSC(SPI_CH1)    &=  ~FTM_CnSC_ELSA_MASK;

```

```
74      FTM0_C0SC   |=  FTM_CnSC_CHIE_MASK;
75      FTM0_C1SC   |=  FTM_CnSC_CHIE_MASK;
76  }
77
78
79  void FTM0_enable(void){
80      SIM_SCGC6 |= SIM_SCGC6_FTM0_MASK;
81  }
82
83  void FTM0_disable(void){
84      SIM_SCGC6 &= ~SIM_SCGC6_FTM0_MASK;
85  }
86
87  uint8_t interrupting = 0;
88
89  uint8_t isInterrupting(void){
90      return interrupting;
91  }
92
93
94  ISR_t FTM0_IRQHandler(void){
95      uint32_t C0SC = FTM0_CnSC(SPI_CH0);
96      uint32_t C1SC = FTM0_CnSC(SPI_CH1);
97
98
99      if (C0SC & FTM_CnSC_CHF_MASK)
100         RH_RF95_handleInterruptTX();
101     if (C1SC & FTM_CnSC_CHF_MASK)
102         RH_RF95_handleInterruptRX();
103     //interrupting = 1;
104
105     //RH_RF95_handleInterruptRX();
106     FTM0_C0SC &= ~FTM_CnSC_CHF_MASK;
107     FTM0_C1SC &= ~FTM_CnSC_CHF_MASK;
108 }
109
110 void setInterrupting(uint8_t value){
111     interrupting = value;
112 }
113
```

```c
/*
 * fsm.h
 */
#ifndef FSMTX_H_
#define FSMTX_H_

#include "StateMachineDefs.h"

// Interfaz
STATE* fsmTx(STATE *p_tabla_estado,BYTE evento_actual);

#endif /* FSM_H_ */

```

‘

```c
/*
 * FTM.h
 */

#ifndef FTM_H_
#define FTM_H_

#include "misc.h"

#define RISINGEDGE      0x00
#define FALLINGEDGE     0x01
#define ANYEDGE         0x02

#define TOGGLEOUTPUT    0x00
#define CLEAROUTPUT     0x01
#define SETOUTPUT       0x02

#define HIGHTRUE        0x00
#define LOWTRUE         0x01

void FTM_init  (void);
void FTM_PWMCenterAligned(bool outputTrue);
void FTM_PWMEdgeAligned(bool outputTrue);
void FTM_IC(bool edge);
void FTM_ICCONT(bool edge);
void FTM_ICSING(bool edge);
void FTM_OC(bool type);
void FTM_FTM0ISR  (void);
void FTM_OVFISR  (void);
ISR_t FTM0_IRQHandler(void);
void FTM0_setPeriod(unsigned int period);
void FTM0_setDuty(unsigned int duty);
unsigned int FTM0_C0Value(void);
void FTM0_setPrescaler(bool prescaler);
void FTM0_disable(void);
void FTM0_enable(void);
uint8_t isInterrupting(void);
void setInterrupting(uint8_t value);
#endif /* FTM_H_ */

```

```c
/*
 * GPIO.c
 */

#include "GPIO.h"

//Tabla de puertos
static volatile uint32_t MYPORT_BASE[3] = {PORTB_BASE_PTR,PORTC_BASE_PTR,PORTB_BASE_PTR};
static volatile uint32_t MYPORT_NUM[3] = {18,17,9};
static volatile uint32_t MYPORT_REG[3] = {PTB_BASE_PTR,PTC_BASE_PTR,PTB_BASE_PTR};


void GPIO_init (uint8_t port)
{
    PORT_PCR_REG((PORT_MemMapPtr)MYPORT_BASE[port],MYPORT_NUM[port]) = PORT_PCR_ISF_MASK | PORT_PCR_MUX(0x01);
}

bool GPIO_read(uint8_t port)
{
    return (bool) (0x01 & (GPIO_PDIR_REG((GPIO_MemMapPtr) MYPORT_REG[port]) >> MYPORT_NUM[port]));
    //return (bool) (0x01 & (GPIO_PDIR_REG((GPIO_MemMapPtr)MYPORT_REG[port]) >> MYPORT_NUM[port]));
    //return 1;
}

void GPIO_write (uint8_t port, bool value)
{
    if (value)
        GPIO_PDOR_REG((GPIO_MemMapPtr)MYPORT_REG[port]) |= GPIO_PDOR_PDO(1<<MYPORT_NUM[port]);
    else
        GPIO_PDOR_REG((GPIO_MemMapPtr)MYPORT_REG[port]) &= ~GPIO_PDOR_PDO(1<<MYPORT_NUM[port]);
}

void GPIO_set(uint8_t port)
{
    GPIO_PSOR_REG((GPIO_MemMapPtr)MYPORT_REG[port]) |= GPIO_PSOR_PTSO(1<<MYPORT_NUM[port]);
}

void GPIO_clear(uint8_t port)
{
    GPIO_PCOR_REG((GPIO_MemMapPtr)MYPORT_REG[port]) |= GPIO_PCOR_PTCO(1<<MYPORT_NUM[port]);
}

void GPIO_toggle (uint8_t port)
{
    GPIO_PTOR_REG((GPIO_MemMapPtr)MYPORT_REG[port]) |= GPIO_PTOR_PTTO(1<<MYPORT_NUM[port]);
}

void GPIO_purpose(uint8_t port, bool io)
{
    if(io == GPIOINPUT)
        GPIO_PDDR_REG((GPIO_MemMapPtr)MYPORT_REG[port]) &= ~GPIO_PDDR_PDD(1<<MYPORT_NUM[port]);
    else
        GPIO_PDDR_REG((GPIO_MemMapPtr)MYPORT_REG[port]) |= GPIO_PDDR_PDD(1<<MYPORT_NUM[port]);
}

```

```c
/*
 * GPIO.h
 */

#ifndef GPIO_H_
#define GPIO_H_

/*
 * GPIO.c
 *
 *  Created on: Jun 7, 2015
 *      Author: Juan
 */

#include "GPIO.h"
#include "misc.h"

#define GPIOINPUT 0
#define GPIOOUT 1
bool GPIO_read(uint8_t port);
void GPIO_init (uint8_t port);
void GPIO_write (uint8_t port, bool value);
void GPIO_set (uint8_t port);
void GPIO_clear (uint8_t port);
void GPIO_toggle (uint8_t port);
void GPIO_untoggle (uint8_t port);
void GPIO_purpose(uint8_t port, bool io);


#endif /* GPIO_H_ */

```

```c
/*
 * main.c
 */

#include "LDM.h"
#include "App.h"

int main(void)
{
    __LDM_init();

    __DI();

    /*************************/
    /* Program-specific setup */
    /*************************/
    App_init();               //
    /*************************/

    __EI();

    /*************************/
    /* Program-specific loop  */
    /*************************/
    App_run();                //
    /*************************/

    FOREVER;
}

```

```c
/*
 * MISC.h
 */

#ifndef MISC_H_
#define MISC_H_

#include "derivative.h"

typedef unsigned char           bool;
typedef signed char             int8_t;
typedef signed short int        int16_t;
typedef signed long int         int32_t;
typedef unsigned char           uint8_t;
typedef unsigned short int      uint16_t;
typedef unsigned long int       uint32_t;

#define ISR_t void __attribute__ ((interrupt))

#define FOREVER for(;;)

#define  FALSE  0x00
#define  TRUE   0x01

#define __EI() do {__asm("CPSIE f");} while(0)
#define __DI() do {__asm("CPSID f");} while(0)


#endif /* MISC_H_ */

```

```
#include "SPI_API.h"
#include "RH_RF95.h"
#include "UART.h"
#include "fsmdefines.h"
#include "GPIO.h"

//Rx DIO3
//Tx DIO0


static bool _rxBufValid = 0;
static bool _txFinish = 0;
static unsigned char _buf[RH_RF95_MAX_PAYLOAD_LEN];
static unsigned int myCounter = 0;
static unsigned int myReadCounter = 0;

extern unsigned char rxWriteBuffer[RXBUFFERSIZE];
extern unsigned char txWriteBuffer[TXBUFFERSIZE];
extern unsigned char rxReadBuffer[RXBUFFERSIZE];
extern unsigned char txReadBuffer[TXBUFFERSIZE];

extern unsigned int rxWriteBufferPointer;
extern unsigned int rxReadBufferPointer;
extern unsigned int txWriteBufferPointer;
extern unsigned int txReadBufferPointer;

void RH_RF95_init(){
    RH_RF95_writeReg(RH_RF95_REG_01_OP_MODE, RH_RF95_MODE_SLEEP | RH_RF95_LONG_RANGE_MODE,SPI_CH0);     // Put in Sleep mode
    RH_RF95_writeReg(RH_RF95_REG_0E_FIFO_TX_BASE_ADDR, 0,SPI_CH0);
    RH_RF95_writeReg(RH_RF95_REG_0F_FIFO_RX_BASE_ADDR, 0,SPI_CH0);
    RH_RF95_writeReg(RH_RF95_REG_01_OP_MODE, RH_RF95_MODE_STDBY,SPI_CH0);   // Put in OOK mode

    RH_RF95_writeReg(RH_RF95_REG_09_PA_CONFIG, RH_RF95_PA_SELECT | 0x05,SPI_CH0);
    RH_RF95_writeReg(RH_RF95_REG_1D_MODEM_CONFIG1, 0x63,SPI_CH0);
    RH_RF95_writeReg(RH_RF95_REG_1E_MODEM_CONFIG2, 0x74,SPI_CH0);
    RH_RF95_writeReg(RH_RF95_REG_26_MODEM_CONFIG3, 0x00,SPI_CH0);
    RH_RF95_writeReg(RH_RF95_REG_20_PREAMBLE_MSB, 0x00,SPI_CH0);
    RH_RF95_writeReg(RH_RF95_REG_21_PREAMBLE_LSB, 0x06,SPI_CH0);
    RH_RF95_writeReg(RH_RF95_REG_06_FRF_MSB, F_UP_1,SPI_CH0);
    RH_RF95_writeReg(RH_RF95_REG_07_FRF_MID, F_UP_2,SPI_CH0);
    RH_RF95_writeReg(RH_RF95_REG_08_FRF_LSB, F_UP_3,SPI_CH0);
    RH_RF95_writeReg(0x4D, 0x87,SPI_CH0);

    RH_RF95_writeReg(RH_RF95_REG_01_OP_MODE, RH_RF95_MODE_SLEEP | RH_RF95_LONG_RANGE_MODE,SPI_CH1);     // Put in Sleep mode

    RH_RF95_writeReg(RH_RF95_REG_0E_FIFO_TX_BASE_ADDR, 0,SPI_CH1);
    RH_RF95_writeReg(RH_RF95_REG_0F_FIFO_RX_BASE_ADDR, 0,SPI_CH1);
    RH_RF95_writeReg(RH_RF95_REG_01_OP_MODE, RH_RF95_MODE_STDBY,SPI_CH1);   // Put in OOK mode

    RH_RF95_writeReg(RH_RF95_REG_09_PA_CONFIG, RH_RF95_PA_SELECT | 0x0f,SPI_CH1);

    RH_RF95_writeReg(RH_RF95_REG_1D_MODEM_CONFIG1, 0x63,SPI_CH1);
    RH_RF95_writeReg(RH_RF95_REG_1E_MODEM_CONFIG2, 0x74,SPI_CH1);
    RH_RF95_writeReg(RH_RF95_REG_26_MODEM_CONFIG3, 0x00,SPI_CH1);
    RH_RF95_writeReg(RH_RF95_REG_20_PREAMBLE_MSB, 0x00,SPI_CH1);
    RH_RF95_writeReg(RH_RF95_REG_21_PREAMBLE_LSB, 0x06,SPI_CH1);
    RH_RF95_writeReg(RH_RF95_REG_06_FRF_MSB, F_DWN_1,SPI_CH1);
    RH_RF95_writeReg(RH_RF95_REG_07_FRF_MID, F_DWN_2,SPI_CH1);
    RH_RF95_writeReg(RH_RF95_REG_08_FRF_LSB, F_DWN_3,SPI_CH1);
    RH_RF95_writeReg(RH_RF95_REG_40_DIO_MAPPING1, 0x00,SPI_CH1); // Interrupt on TxDone
    RH_RF95_writeReg(RH_RF95_REG_12_IRQ_FLAGS, 0xff,SPI_CH1); // Clear all IRQ flags

    RH_RF95_writeReg(RH_RF95_REG_01_OP_MODE, RH_RF95_MODE_RXCONTINUOUS,SPI_CH1);        // Put in Sleep mode

}

bool RH_RF95_available(){
```

```
68      bool aux = _rxBufValid;
69
70      if(_rxBufValid)
71          _rxBufValid = false;
72
73      return aux;
74  }
75
76  bool RH_RF95_finishTx(){
77      bool aux = _txFinish;
78
79      if(_txFinish)
80          _txFinish = false;
81
82      return aux;
83  }
84
85
86  void RH_RF95_printRxBuf(){
87      //UART_write(_buf);
88  }
89
90  void RH_RF95_writeFIFO(unsigned int *mydata, unsigned int size){
91
92  }
93
94  void RH_RF95_readFIFO(unsigned char *mydata, unsigned int size){
95      *mydata = _buf[myReadCounter++];
96  }
97
98  void RH_RF95_transmit(void){
99
100     RH_RF95_writeReg(RH_RF95_REG_01_OP_MODE, RH_RF95_MODE_TX,SPI_CH0);
101     RH_RF95_writeReg(RH_RF95_REG_40_DIO_MAPPING1, 0x40,SPI_CH0); // Interrupt on
        TxDone
102     _txFinish = TRUE;
103 }
104
105 void RH_RF95_receive(void){
106
107 }
108
109 void RH_RF95_writeReg(unsigned int reg, unsigned int val,unsigned int SS){
110      reg=reg<<8;
111      reg|=val;
112      reg|=0x8000;
113      SPI0_SenData(&reg, 1,SS);
114 }
115
116 unsigned int RH_RF95_readReg(unsigned int reg,unsigned int SS){
117      unsigned int data;
118      data=0;
119      reg=reg<<8;
120      SPI0_ReceiveData(&data,1, reg,SS);
121      data&=0x00FF;
122      return data;
123 }
124
125 unsigned long int magicAuxTx = 0;
126
127 ISR_t RH_RF95_handleInterruptTX(){
128     // Read the interrupt register
129
130     magicAuxTx++;
131
132     if(magicAuxTx == 10100)
133         magicAuxTx = 0;
134
135     RH_RF95_writeReg(RH_RF95_REG_01_OP_MODE, RH_RF95_MODE_STDBY,SPI_CH0);
136     RH_RF95_writeReg(RH_RF95_REG_12_IRQ_FLAGS, 0xff,SPI_CH0); // Clear all IRQ flags
137
138     if(txWriteBufferPointer == txReadBufferPointer)
139         executeEvent(TX_STBY_EVENT);
```

```
    else{
        RH_RF95_SendPacket(txWriteBuffer[txReadBufferPointer++]);
        if (txReadBufferPointer==TXBUFFERSIZE)
            txReadBufferPointer=0;
    }
}

unsigned int returnCounter(void){
    unsigned int aux = myCounter;
    myCounter = 0;
    return aux;
}

void RH_RF95_Dummy(){
    //UART_write("Dummy");
}

void RH_RF95_Server(){
    /*if (RH_RF95_available())
        RH_RF95_printRxBuf();*/
    RH_RF95_receive();
}

void RH_RF95_RX_LP(){
    RH_RF95_writeReg(RH_RF95_REG_01_OP_MODE, RH_RF95_MODE_STDBY,SPI_CH1);
}
void RH_RF95_RX_STBY(){
    RH_RF95_writeReg(RH_RF95_REG_01_OP_MODE, RH_RF95_MODE_RXCONTINUOUS,SPI_CH1);
}
void RH_RF95_TX_LP(){
    RH_RF95_writeReg(RH_RF95_REG_01_OP_MODE, RH_RF95_MODE_STDBY,SPI_CH0);
}
void RH_RF95_TX_STBY(){
    RH_RF95_writeReg(RH_RF95_REG_01_OP_MODE, RH_RF95_MODE_STDBY,SPI_CH0);
}
void RH_RF95_SendPacket(int j){
    RH_RF95_writeReg(RH_RF95_REG_0D_FIFO_ADDR_PTR, 0,SPI_CH0);
    RH_RF95_writeReg(0x00, j,SPI_CH0);
    RH_RF95_writeReg(RH_RF95_REG_22_PAYLOAD_LENGTH, 0x01, SPI_CH0);
    RH_RF95_transmit();
}

unsigned long int magicAuxRx = 0;

ISR_t RH_RF95_handleInterruptRX(){
    int aux;
    int validHeader;
    validHeader = RH_RF95_readReg(RH_RF95_REG_12_IRQ_FLAGS,SPI_CH1);
    aux = RH_RF95_readReg(RH_RF95_REG_10_FIFO_RX_CURRENT_ADDR,SPI_CH1);
    RH_RF95_writeReg(RH_RF95_REG_0D_FIFO_ADDR_PTR, aux,SPI_CH1);
    aux = RH_RF95_readReg(RH_RF95_REG_00_FIFO,SPI_CH1);
    if(validHeader == 0x50){
        magicAuxTx++;
        if(magicAuxTx == 1010)
            magicAuxTx = 0;
        SPACE_receiveFromSpace(aux);
    }
    RH_RF95_writeReg(RH_RF95_REG_12_IRQ_FLAGS, 0xff,SPI_CH1); // Clear all IRQ flags
}
unsigned int RH_RF95_TX_CHECK_TEMP(){
    int aux;
    aux = RH_RF95_readReg(RH_RF95_REG_3C_TEMP,SPI_CH0);

    return (aux > MAX_TEMP);
}
unsigned int RH_RF95_RX_CHECK_TEMP(){
    int aux;
    aux = RH_RF95_readReg(RH_RF95_REG_3C_TEMP,SPI_CH1);

    return (aux > MAX_TEMP);
}
bool RH_RF95_TX_CHECK_COMM(){
    int aux;
```

```
213         aux = RH_RF95_readReg(RH_RF95_REG_01_OP_MODE,SPI_CH0);
214
215         return (aux == 0xFF);
216     }
217     bool RH_RF95_RX_CHECK_COMM(){
218         int aux;
219         aux = RH_RF95_readReg(RH_RF95_REG_01_OP_MODE,SPI_CH1);
220
221         return (aux == 0xFF);
222     }
223
224
225
```

```
/*
 * RFM.h
 */

#ifndef RFM_H_
#define RFM_H_

#define T434R915    0
#define T915R434    ~T434R915
#define MAX_TEMP    0xFF
#if T434R915 == 0
    #define SPI_CH0 0x01
    #define SPI_CH1 0x00

    #define F_DWN_1 0xE4
    #define F_DWN_2 0xC0
    #define F_DWN_3 0x00

    #define F_UP_1  0x6c
    #define F_UP_2  0x80
    #define F_UP_3  0x00
#else
    #define SPI_CH0 0x00
    #define SPI_CH1 0x01

    #define F_DWN_1 0x6C
    #define F_DWN_2 0x80
    #define F_DWN_3 0x00

    #define F_UP_1  0xE4
    #define F_UP_2  0xC0
    #define F_UP_3  0x00
#endif

#define RH_RF95_FXOSC 32000000.0

// The Frequency Synthesizer step = RH_RF95_FXOSC / 2^^19
#define RH_RF95_FSTEP  (RH_RF95_FXOSC / 524288)


// Register names (LoRa Mode, from table 85)
#define RH_RF95_REG_00_FIFO                                0x00
#define RH_RF95_REG_01_OP_MODE                             0x01
#define RH_RF95_REG_02_RESERVED                            0x02
#define RH_RF95_REG_03_RESERVED                            0x03
#define RH_RF95_REG_04_RESERVED                            0x04
#define RH_RF95_REG_05_RESERVED                            0x05
#define RH_RF95_REG_06_FRF_MSB                             0x06
#define RH_RF95_REG_07_FRF_MID                             0x07
#define RH_RF95_REG_08_FRF_LSB                             0x08
#define RH_RF95_REG_09_PA_CONFIG                           0x09
#define RH_RF95_REG_0A_PA_RAMP                             0x0a
#define RH_RF95_REG_0B_OCP                                 0x0b
#define RH_RF95_REG_0C_LNA                                 0x0c
#define RH_RF95_REG_0D_FIFO_ADDR_PTR                       0x0d
#define RH_RF95_REG_0E_FIFO_TX_BASE_ADDR                   0x0e
#define RH_RF95_REG_0F_FIFO_RX_BASE_ADDR                   0x0f
#define RH_RF95_REG_10_FIFO_RX_CURRENT_ADDR                0x10
#define RH_RF95_REG_11_IRQ_FLAGS_MASK                      0x11
#define RH_RF95_REG_12_IRQ_FLAGS                           0x12
#define RH_RF95_REG_13_RX_NB_BYTES                         0x13
#define RH_RF95_REG_14_RX_HEADER_CNT_VALUE_MSB             0x14
#define RH_RF95_REG_15_RX_HEADER_CNT_VALUE_LSB             0x15
#define RH_RF95_REG_16_RX_PACKET_CNT_VALUE_MSB             0x16
#define RH_RF95_REG_17_RX_PACKET_CNT_VALUE_LSB             0x17
#define RH_RF95_REG_18_MODEM_STAT                          0x18
#define RH_RF95_REG_19_PKT_SNR_VALUE                       0x19
#define RH_RF95_REG_1A_PKT_RSSI_VALUE                      0x1a
#define RH_RF95_REG_1B_RSSI_VALUE                          0x1b
#define RH_RF95_REG_1C_HOP_CHANNEL                         0x1c
#define RH_RF95_REG_1D_MODEM_CONFIG1                       0x1d
#define RH_RF95_REG_1E_MODEM_CONFIG2                       0x1e
#define RH_RF95_REG_1F_SYMB_TIMEOUT_LSB                    0x1f
```

```
74  #define RH_RF95_REG_20_PREAMBLE_MSB                     0x20
75  #define RH_RF95_REG_21_PREAMBLE_LSB                     0x21
76  #define RH_RF95_REG_22_PAYLOAD_LENGTH                   0x22
77  #define RH_RF95_REG_23_MAX_PAYLOAD_LENGTH               0x23
78  #define RH_RF95_REG_24_HOP_PERIOD                       0x24
79  #define RH_RF95_REG_25_FIFO_RX_BYTE_ADDR                0x25
80  #define RH_RF95_REG_26_MODEM_CONFIG3                    0x26
81  #define RH_RF95_REG_3C_TEMP                             0x3C
82
83  #define RH_RF95_REG_40_DIO_MAPPING1                     0x40
84  #define RH_RF95_REG_41_DIO_MAPPING2                     0x41
85  #define RH_RF95_REG_42_VERSION                          0x42
86
87  // RH_RF95_REG_01_OP_MODE                               0x01
88  #define RH_RF95_LONG_RANGE_MODE                         0x80
89  #define RH_RF95_ACCESS_SHARED_REG                       0x40
90  #define RH_RF95_MODE                                    0x07
91  #define RH_RF95_MODE_SLEEP                              0x00
92  #define RH_RF95_MODE_STDBY                              0x01
93  #define RH_RF95_MODE_FSTX                               0x02
94  #define RH_RF95_MODE_TX                                 0x03
95  #define RH_RF95_MODE_FSRX                               0x04
96  #define RH_RF95_MODE_RXCONTINUOUS                       0x05
97  #define RH_RF95_MODE_RXSINGLE                           0x06
98  #define RH_RF95_MODE_CAD                                0x07
99
100 // RH_RF95_REG_09_PA_CONFIG                             0x09
101 #define RH_RF95_PA_SELECT                               0x80
102 #define RH_RF95_OUTPUT_POWER                            0x0f
103
104 // RH_RF95_REG_0A_PA_RAMP                               0x0a
105 #define RH_RF95_LOW_PN_TX_PLL_OFF                       0x10
106 #define RH_RF95_PA_RAMP                                 0x0f
107 #define RH_RF95_PA_RAMP_3_4MS                           0x00
108 #define RH_RF95_PA_RAMP_2MS                             0x01
109 #define RH_RF95_PA_RAMP_1MS                             0x02
110 #define RH_RF95_PA_RAMP_500US                           0x03
111 #define RH_RF95_PA_RAMP_250US                           0x0
112 #define RH_RF95_PA_RAMP_125US                           0x05
113 #define RH_RF95_PA_RAMP_100US                           0x06
114 #define RH_RF95_PA_RAMP_62US                            0x07
115 #define RH_RF95_PA_RAMP_50US                            0x08
116 #define RH_RF95_PA_RAMP_40US                            0x09
117 #define RH_RF95_PA_RAMP_31US                            0x0a
118 #define RH_RF95_PA_RAMP_25US                            0x0b
119 #define RH_RF95_PA_RAMP_20US                            0x0c
120 #define RH_RF95_PA_RAMP_15US                            0x0d
121 #define RH_RF95_PA_RAMP_12US                            0x0e
122 #define RH_RF95_PA_RAMP_10US                            0x0f
123
124 // RH_RF95_REG_0B_OCP                                   0x0b
125 #define RH_RF95_OCP_ON                                  0x20
126 #define RH_RF95_OCP_TRIM                                0x1f
127
128 // RH_RF95_REG_0C_LNA                                   0x0c
129 #define RH_RF95_LNA_GAIN                                0xe0
130 #define RH_RF95_LNA_BOOST                               0x03
131 #define RH_RF95_LNA_BOOST_DEFAULT                       0x00
132 #define RH_RF95_LNA_BOOST_150PC                         0x11
133
134 // RH_RF95_REG_11_IRQ_FLAGS_MASK                        0x11
135 #define RH_RF95_RX_TIMEOUT_MASK                         0x80
136 #define RH_RF95_RX_DONE_MASK                            0x40
137 #define RH_RF95_PAYLOAD_CRC_ERROR_MASK                  0x20
138 #define RH_RF95_VALID_HEADER_MASK                       0x10
139 #define RH_RF95_TX_DONE_MASK                            0x08
140 #define RH_RF95_CAD_DONE_MASK                           0x04
141 #define RH_RF95_FHSS_CHANGE_CHANNEL_MASK                0x02
142 #define RH_RF95_CAD_DETECTED_MASK                       0x01
143
144 // RH_RF95_REG_12_IRQ_FLAGS                             0x12
145 #define RH_RF95_RX_TIMEOUT                              0x80
146 #define RH_RF95_RX_DONE                                 0x40
```

```
147  #define RH_RF95_PAYLOAD_CRC_ERROR                       0x20
148  #define RH_RF95_VALID_HEADER                            0x10
149  #define RH_RF95_TX_DONE                                 0x08
150  #define RH_RF95_CAD_DONE                                0x04
151  #define RH_RF95_FHSS_CHANGE_CHANNEL                     0x02
152  #define RH_RF95_CAD_DETECTED                            0x01
153
154  // RH_RF95_REG_18_MODEM_STAT                            0x18
155  #define RH_RF95_RX_CODING_RATE                          0xe0
156  #define RH_RF95_MODEM_STATUS_CLEAR                      0x10
157  #define RH_RF95_MODEM_STATUS_HEADER_INFO_VALID          0x08
158  #define RH_RF95_MODEM_STATUS_RX_ONGOING                 0x04
159  #define RH_RF95_MODEM_STATUS_SIGNAL_SYNCHRONIZED        0x02
160  #define RH_RF95_MODEM_STATUS_SIGNAL_DETECTED            0x01
161
162  // RH_RF95_REG_1C_HOP_CHANNEL                           0x1c
163  #define RH_RF95_PLL_TIMEOUT                             0x80
164  #define RH_RF95_RX_PAYLOAD_CRC_IS_ON                    0x40
165  #define RH_RF95_FHSS_PRESENT_CHANNEL                    0x3f
166
167  // RH_RF95_REG_1D_MODEM_CONFIG1                         0x1d
168  #define RH_RF95_BW                                      0xc0
169  #define RH_RF95_BW_125KHZ                               0x00
170  #define RH_RF95_BW_250KHZ                               0x40
171  #define RH_RF95_BW_500KHZ                               0x80
172  #define RH_RF95_BW_RESERVED                             0xc0
173  #define RH_RF95_CODING_RATE                             0x38
174  #define RH_RF95_CODING_RATE_4_5                         0x00
175  #define RH_RF95_CODING_RATE_4_6                         0x08
176  #define RH_RF95_CODING_RATE_4_7                         0x10
177  #define RH_RF95_CODING_RATE_4_8                         0x18
178  #define RH_RF95_IMPLICIT_HEADER_MODE_ON                 0x04
179  #define RH_RF95_RX_PAYLOAD_CRC_ON                       0x02
180  #define RH_RF95_LOW_DATA_RATE_OPTIMIZE                  0x01
181
182  // RH_RF95_REG_1E_MODEM_CONFIG2                         0x1e
183  #define RH_RF95_SPREADING_FACTOR                        0xf0
184  #define RH_RF95_SPREADING_FACTOR_64CPS                  0x60
185  #define RH_RF95_SPREADING_FACTOR_128CPS                 0x70
186  #define RH_RF95_SPREADING_FACTOR_256CPS                 0x80
187  #define RH_RF95_SPREADING_FACTOR_512CPS                 0x90
188  #define RH_RF95_SPREADING_FACTOR_1024CPS                0xa0
189  #define RH_RF95_SPREADING_FACTOR_2048CPS                0xb0
190  #define RH_RF95_SPREADING_FACTOR_4096CPS                0xc0
191  #define RH_RF95_TX_CONTINUOUS_MOE                       0x08
192  #define RH_RF95_AGC_AUTO_ON                             0x04
193  #define RH_RF95_SYM_TIMEOUT_MSB                         0x03
194
195
196  #define false   0
197  #define true    0
198  /*
199  #define RH_RF95_REG_00_FIFO                                 0x00
200  #define RH_RF95_REG_01_OP_MODE                              0x01
201  #define RH_RF95_REG_06_FRF_MSB                              0x06
202  #define RH_RF95_REG_07_FRF_MID                              0x07
203  #define RH_RF95_REG_08_FRF_LSB                              0x08
204  #define RH_RF95_REG_09_PA_CONFIG                            0x09
205  #define RH_RF95_REG_3E_IRQ_Flag1                            0x3E
206  #define RH_RF95_REG_3F_IRQ_Flag2                            0x3F
207  #define RH_RF95_REG_40_DIO_MAPPING1                         0x40
208  #define RH_RF95_REG_30_PACK_CONFIG1                         0x30
209  #define RH_RF95_REG_31_PACK_CONFIG2                         0x31
210  #define RH_RF95_REG_35_FIFO_THRESH                          0x35
211
212  // RH_RF95_REG_01_OP_MODE                               0x01
213  #define RH_RF95_MODE_SLEEP                              0x00
214  #define RH_RF95_MODE_RXCONTINUOUS                       0x25
215  #define RH_RF95_MODE_STDBY                              0x21
216  #define RH_RF95_MODE_TX                                 0x23
217  #define RH_RF95_OOK_MODE                                0x20
218
219  // RH_RF95_REG_09_PA_CONFIG                             0x09
```

```
#define RH_RF95_PA_SELECT                               0x80
#define RH_RF95_OUTPUT_POWER                            0x0f
*/
#define RH_RF95_FIFO_SIZE 2000
#define RH_RF95_MAX_PAYLOAD_LEN RH_RF95_FIFO_SIZE

void RH_RF95_receive(void);
void RH_RF95_transmit(void);
void RH_RF95_readFIFO(unsigned char *mydata, unsigned int size);
void RH_RF95_writeFIFO(unsigned int *mydata, unsigned int size);
void RH_RF95_printRxBuf();
bool RH_RF95_available();
void RH_RF95_init();
void RH_RF95_writeReg(unsigned int reg, unsigned int val,unsigned int SS);
unsigned int RH_RF95_readReg(unsigned int reg,unsigned int SS);
ISR_t RH_RF95_handleInterruptRX();
ISR_t RH_RF95_handleInterruptTX();
void RH_RF95_Dummy();
void RH_RF95_Client();
void RH_RF95_Server();
unsigned int returnCounter(void);

bool RH_RF95_RX_CHECK_COMM();
bool RH_RF95_TX_CHECK_COMM();
unsigned int RH_RF95_RX_CHECK_TEMP();
unsigned int RH_RF95_TX_CHECK_TEMP();
void RH_RF95_RX_LP();
void RH_RF95_RX_STBY();
#endif /* RFM_H_ */
```

```c
#include "SPI_API.h"
#include "SPI_HAL.h"
#include <MK64F12.h>
#include "misc.h"
#include <stdio.h>


//***MODO DE ARRANQUE***//
//SPICR1
#define SPI_SPIE_INIT 0x80   //arranca en modo interrupcion
#define SPI_SPTIE_INIT 0x20  //arranca con interrupciones de transmicion tambien
#define SPI_CPOL_INIT 0x08   //arranca el clock en activo alto
#define SPI_LSBFE_INIT 0x00  //arranca mandando primero MSB
//SPICR2
#define SPI_MODFEN_INIT 0x10 //If the SPI is in master mode and
                             //MODFEN is cleared, then the SS port pin
                             //is not used by the SPI.

//***DEFINICIONES***//
//SPICR1
#define SPI_SPEMASK 0x40
#define SPI_MSTRMASK 0x10
#define SPI_SSOEMASK 0x01
#define SPI_CPHEMASK 0x00   //Le importan los flancos impares
//SPICR2
#define SPI_BIDIROE 0x00    //Toda la parte bidireccional esta
desactivada                           //(BIDIROE y SPC0)
//SPIBR
#define SPI_SPPR 0x04           //16MHz de clock
#define SPI_SPR  0x06           //1.6Mhz de Baud Rate

//GENERALES
#define MAXDATA          24
#define IDLE             0
#define INT              1
#define Polling          2
#define Wait2SendGood    3
#define Wait2SendBad     4
#define Wait2Ignore      5
#define Wait2Save        6


//***VARIABLES***///
static volatile unsigned int
SPI0_APICounter=0,SPI0_APIdatasize=0,SPI0_APIReceiveindex=0,SPI0_MaxRec=0;
static volatile unsigned int
SPI0_APIData[MAXDATA],SPI1_APIData[MAXDATA],SPI2_APIData[MAXDATA];
static volatile unsigned char SPI0_APIGlobalStatus=IDLE,SPI0_APISemiStatus=IDLE;
static volatile unsigned char SPI1_APIGlobalStatus=IDLE,SPI1_APISemiStatus=IDLE;
static volatile unsigned char SPI2_APIGlobalStatus=IDLE,SPI2_APISemiStatus=IDLE;
static volatile unsigned char *SPI0_RecVector;

void SPI_API_Init (void){
  int a;
  //SPI0_ClrINT();
  SPI0_HAL_Init();          //deberia inicializar cada modulo que deseara usar;
  //SPI0_ClrINT();
  for(a=0; a<MAXDATA;a++){  //deberia inicializar un buffer por modulo;
    SPI0_APIData[a]=0;
  }

  //SPI0_APIGlobalStatus=INT;
  SPI0_APIGlobalStatus=Polling;
  //SPI0_SetHL();

}

void SPI0_SetAsPolling(void){
  SPI0_ClrSPTIE();
  SPI0_ClrSPIE();
  SPI0_APIGlobalStatus=Polling;
}

```

```
71  void SPI0_SetAsINT(void){
72    SPI0_APIGlobalStatus=INT;
73  }
74
75  void SPI0_SetINT(void){
76    SPI0_SetSPIE();
77    SPI0_SetSPTIE();
78  }
79
80  void SPI0_ClrINT(void){
81    SPI0_ClrSPTIE();
82    SPI0_ClrSPIE();
83  }
84
85  void SPI0_SetCkAL(void){
86    SPI0_SetCPOL();
87  }
88
89  void SPI0_SetCkAH(void){
90    SPI0_ClrCPOL();
91  }
92
93  void SPI0_SetHL(void){
94    SPI0_ClrLSBFE();
95  }
96
97  void SPI0_SetLH(void){
98    SPI0_SetLSBFE();
99  }
100
101
102 void SPI0_SetOddEdges(void){
103   SPI0_ClrCPHA();
104 }
105
106 void SPI0_SetEvenEdges(void){
107   SPI0_SetCPHA();
108 }
109
110
111 unsigned char SPI0_GetGlobalStatus(void){
112   return SPI0_APIGlobalStatus;
113 }
114
115  unsigned char SPI0_GetSemiStatus(void){
116   return SPI0_APISemiStatus;
117 }
118
119
120  void SPI0_ReceiveData(unsigned int *mydatareg, unsigned int size, unsigned int
     registro,unsigned int SS){   //
     size=1,2,3,4...
121    int a;
122    unsigned int aux;
123    uint32_t aux2;
124    unsigned int temp;
125    for(a=0;a<size;a++){
126      temp=registro|(0xFF);
127
128      SPI0_SR|= SPI_SR_TCF_MASK;
129
130      if (SS==0x01){
131          SPI0_WriteDataRegS1(temp);
132      }
133      else{
134          SPI0_WriteDataRegS2(temp);
135      }
136
137      aux=SPI0_CheckSPTEF();
138      while(!(aux==SPI_SR_TCF_MASK)){
139         aux=SPI0_CheckSPTEF();
140      }
141
```

```
142         aux2=SPI0_ReadDataReg();
143         *(mydatareg+a)=aux2;
144       }
145     }
146
147
148   void SPI0_SenData(unsigned int *mydata, unsigned int size,unsigned int SS){   //
      size=1,2,3,4...
149       uint32_t aux2;
150       unsigned int aux;
151       int a;
152         for(a=0;a<size;a++){
153
154           SPI0_SR|=SPI_SR_TCF_MASK;
155
156           if (SS==0x01){
157                SPI0_WriteDataRegS1(*(mydata+a));
158            }
159            else{
160                SPI0_WriteDataRegS2(*(mydata+a));
161            }
162
163           aux=SPI0_CheckSPTEF();
164           while(!(aux==SPI_SR_TCF_MASK)){
165               aux=SPI0_CheckSPTEF();
166           }
167
168           aux2=SPI0_ReadDataReg();
169         }
170       }
171
172   ISR_t SPI0_IRQHandler (void){   //Funcion por
      interrupcion
173     char temp;
174     SPI0_ClrINT();
175     //TERMIO_PutChar('g');
176     if(SPI0_CheckSPIF()){
177       if(SPI0_APISemiStatus==Wait2Ignore){
178         temp=SPI0_ReadDataReg();
179         if(SPI0_APICounter==SPI0_APIdatasize){
180           SPI0_APISemiStatus=IDLE;
181           SPI0_ClrINT();
182         }else{
183           SPI0_APISemiStatus=Wait2SendGood;
184         }
185       }
186       if(SPI0_APISemiStatus==Wait2Save){
187         SPI0_RecVector[SPI0_APIReceiveindex]=SPI0_ReadDataReg();
188         SPI0_APIReceiveindex++;
189
190         if(SPI0_APIReceiveindex==SPI0_MaxRec){
191           SPI0_APISemiStatus=IDLE;
192           SPI0_ClrINT();
193         }else{
194           SPI0_APISemiStatus=Wait2SendBad;
195         }
196
197       }
198     }
199
200     if(SPI0_CheckSPTEF()){
201       if(SPI0_APISemiStatus==Wait2SendGood){
202         SPI0_WriteDataReg(SPI0_APIData[SPI0_APICounter]);
203         SPI0_APICounter++;
204         SPI0_APISemiStatus=Wait2Ignore;
205       }
206
207       if(SPI0_APISemiStatus==Wait2SendBad){
208         temp=0x00;
209         SPI0_WriteDataReg(temp);
210       }
211
212     }
```

```
213
214     if(SPI0_APISemiStatus!=IDLE)
215       SPI0_SetINT();
216
217   }
```

```
#ifndef SPI_API_H
#define SPI_API_H
#include "misc.h"
/*
 Definitions:
SS    Slave Select
SCK   Serial Clock
MOSI  Master Output, Slave Input
MISO  Master Input, Slave Output
MOMI  Master Output, Master Input
SISO  Slave Input, Slave Output
*/

/*
SPICR1: [7 6 5 4 3 2 1 0]
7->SPIE: SPI Interrupt Enable Bit.This bit enables SPI interrupt requests, if SPIF or MODF status flag is set.
          0 SPI interrupts disabled.
          1 SPI interrupts enabled.
6->SPE: SPI System Enable Bit. This bit enables the SPI system and dedicates the SPI port pins to SPI system
        functions. If SPE is cleared, SPI is disabled and forced into idle state,
        status bits in SPISR register are reset.
          0 SPI disabled (lower power consumption).
          1 SPI enabled, port pins are dedicated to SPI functions.
5->SPTIE: SPI Transmit Interrupt Enable. This bit enables SPI interrupt requests, if SPTEF flag is set.
          0 SPTEF interrupt disabled.
          1 SPTEF interrupt enabled.
4->MSTR:  SPI Master/Slave Mode Select Bit. This bit selects whether the SPI operates in master or slave mode.
          Switching the SPI from master to slave or vice versa forces the SPI system into idle state.
          0 SPI is in slave mode.
          1 SPI is in master mode.
3->CPOL:  SPI Clock Polarity Bit. This bit selects an inverted or non-inverted SPI clock. To transmit data between SPI
          modules, the SPI modules must have identical CPOL values. In master mode, a change of this bit will abort a
          transmission in progress and force the SPI system into idle state.
          0 Active-high clocks selected. In idle state SCK is low.
          1 Active-low clocks selected. In idle state SCK is high.
2->CPHA:  SPI Clock Phase Bit — This bit is used to select the SPI clock format. In master mode, a change of this bit will
          abort a transmission in progress and force the SPI system into idle state.
          0 Sampling of data occurs at odd edges (1,3,5,...,15) of the SCK clock.
          1 Sampling of data occurs at even edges (2,4,6,...,16) of the SCK
          clock.
1->SSOE:  Slave Select Output Enable — The SS output feature is enabled only in master mode, if MODFEN is set, by
          asserting the SSOE as shown in Table 12-2. In master mode, a change of this bit will abort a transmission in
          progress and force the SPI system into idle state.
0->LSBFE: LSB-First Enable — This bit does not affect the position of the MSB and LSB in the data register. Reads and
          writes of the data register always have the MSB in bit 7. In master mode, a change of this bit will abort a
          transmission in progress and force the SPI system into idle state.
          0 Data is transferred most significant bit first.
          1 Data is transferred least significant bit first.

Config: [x 1 x 1 0 0 1 0]
*/

/*
SPICR2: [7 6 5 4 3 2 1 0]
         0 0 0     0
4->MODFEN:  Mode Fault Enable Bit — This bit allows the MODF failure to be detected. If the SPI is in master mode and
            MODFEN is cleared, then the SS port pin is not used by the SPI. In slave mode, the SS is available only as an
            input regardless of the value of MODFEN. For an overview on the impact of the MODFEN bit on the SS port pin
```

```
            configuration, refer to Table 12-4. In master mode, a change of this bit
            will abort a transmission in progress and
            force the SPI system into idle state.
            0 SS port pin is not used by the SPI.
            1 SS port pin with MODF feature.
3->BIDIROE: Output Enable in the Bidirectional Mode of Operation — This bit controls
the MOSI and MISO output buffer
            of the SPI, when in bidirectional mode of operation (SPC0 is set). In
            master mode, this bit controls the output
            buffer of the MOSI port, in slave mode it controls the output buffer of
            the MISO port. In master mode, with SPC0
            set, a change of this bit will abort a transmission in progress and
            force the SPI into idle state.
            0 Output buffer disabled.
            1 Output buffer enabled.
1->SPISWAI: SPI Stop in Wait Mode Bit — This bit is used for power conservation
while in wait mode.
            0 SPI clock operates normally in wait mode.
            1 Stop SPI clock generation when in wait mode.
0->SPC0:  Serial Pin Control Bit 0 — This bit enables bidirectional pin
configurations as shown in Table 12-4. In master
          mode, a change of this bit will abort a transmission in progress and force
          the SPI system into idle state.
Config: [0 0 0 1 1 0 0 0]
*/


/*
SPIBR: [7 6 5 4 3 2 1 0]
        0       0
6-4-> SPPR[2:0]:SPI Baud Rate Preselection Bits—These bits specify the SPI baud
rates as shown in Table 12-6. In master
      mode, a change of these bits will abort a transmission in progress and force
      the SPI system into idle state.
2-0->SPR[2:0]:SPI Baud Rate Selection Bits—These bits specify the SPI baud rates as
shown in Table 12-6. In master mode,
      a change of these bits will abort a transmission in progress and force the SPI
      system into idle state.

BaudRateDivisor = (SPPR + 1). (2^(SPR + 1))

Baud Rate = BusClock / BaudRateDivisor
*/

/*
SPISR: [7 6 5 4 3 2 1 0]
           0     0 0 0 0
7->SPIF:  SPIF Interrupt Flag — This bit is set after a received data byte has been
transferred into the SPI data register.
          This bit is cleared by reading the SPISR register (with SPIF set) followed
          by a read access to the SPI data
          register.
          0 Transfer not yet complete.
          1 New data copied to SPIDR.
5->SPTEF: SPI Transmit Empty Interrupt Flag — If set, this bit indicates that the
transmit data register is empty. To clear
          this bit and place data into the transmit data register, SPISR must be
          read with SPTEF = 1, followed by a write
          to SPIDR. Any write to the SPI data register without reading SPTEF = 1, is
          effectively ignored.
          0 SPI data register not empty.
          1 SPI data register empty.
4->MODF:  Mode Fault Flag—This bit is set if the SS input becomes low while the SPI
is configured as a master and mode
          fault detection is enabled, MODFEN bit of SPICR2 register is set. Refer to
          MODFEN bit description in
          Section 12.3.2.2, “SPI Control Register 2 (SPICR2)”. The flag is cleared
          automatically by a read of the SPI status
          register (with MODF set) followed by a write to the SPI control register 1.
          0 Mode fault has not occurred.
          1 Mode fault has occurred.
Config: [0 0 1 0 0 0 0 0]
*/
```

```
110
111  /*
112  SPIDR: [7 6 5 4 3 2 1 0]
113
114  The SPI data register is both the input and output register for SPI data. A write to
     this register
115  allows a data byte to be queued and transmitted. For an SPI configured as a master,
     a queued data
116  byte is transmitted immediately after the previous transmission has completed. The
     SPI transmitter
117  empty flag SPTEF in the SPISR register indicates when the SPI data register is ready
     to accept new
118  data.
119  Received data in the SPIDR is valid when SPIF is set.
120  If SPIF is cleared and a byte has been received, the received byte is transferred
     from the receive
121  shift register to the SPIDR and SPIF is set.
122  If SPIF is set and not serviced, and a second byte has been received, the second
     received byte is
123  kept as valid byte in the receive shift register until the start of another
     transmission. The byte in the
124  SPIDR does not change.
125  If SPIF is set and a valid byte is in the receive shift register, and SPIF is
     serviced before the start of
126  a third transmission, the byte in the receive shift register is transferred into the
     SPIDR and SPIF
127  remains set (see Figure 12-8).
128  If SPIF is set and a valid byte is in the receive shift register, and SPIF is
     serviced after the start of
129  a third transmission, the byte in the receive shift register has become invalid and
     is not transferred
130  into the SPIDR (see Figure 12-9).
131
132  */
133
134
135
136  //***SERVICIOS***//
137
138  void SPI_API_Init(void);
139  void SPI0_SetAsPolling(void);
140  void SPI0_SetAsINT(void);
141  void SPI0_SetCkAL(void);
142  void SPI0_SetCkAH(void);
143  void SPI0_SetHL(void);
144  void SPI0_SetLH(void);
145  void SPI0_SetOddEdges(void);
146  void SPI0_SetEvenEdges(void);
147  unsigned char SPI0_GetGlobalStatus(void);
148  unsigned char SPI0_GetSemiStatus(void);
149  void SPI0_SetINT(void);
150  void SPI0_ClrINT(void);
151  void SPI0_SenData(unsigned int *mydata, unsigned int size,unsigned int SS);
152  void SPI0_ReceiveData(unsigned int *mydatareg, unsigned int size, unsigned int
     registro,unsigned int SS);
153  ISR_t SPI0_IRQHandler (void);
154
155  #endif
156
```

```c
#include "SPI_HAL.h"
#include <MK64F12.h>

//***MODO DE ARRANQUE***//
//SPICR1
#define SPI_SPIE_INIT 0x80   //arranca en modo interrupcion
#define SPI_SPTIE_INIT 0x20  //arranca con interrupciones de transmicion tambien
#define SPI_CPOL_INIT 0x08   //arranca el clock en activo alto
#define SPI_LSBFE_INIT 0x00  //arranca mandando primero MSB
//SPICR2
#define SPI_MODFEN_INIT 0x10 //If the SPI is in master mode and
                             //MODFEN is cleared, then the SS port pin
                             //is not used by the SPI.

//***DEFINICIONES***//
//SPICR1
#define SPI_SPEMASK 0x40
#define SPI_MSTRMASK 0x10
#define SPI_SSOEMASK 0x01
#define SPI_CPHEMASK 0x00   //Le importan los flancos impares
//SPICR2
#define SPI_BIDIROE 0x00    //Toda la parte bidireccional esta
desactivada                          //(BIDIROE y SPC0)
//SPIBR
#define SPI0_SPPR 0x00           //40MHz de clock
#define SPI0_SPR  0x07           //40Mhz/1024 de Baud Rate

#define SPI1_SPPR 0x04           //16MHz de clock
#define SPI1_SPR  0x06           //1.6Mhz de Baud Rate

#define SPI2_SPPR 0x04           //16MHz de clock
#define SPI2_SPR  0x06           //1.6Mhz de Baud Rate

typedef unsigned int byte;


///*** ACA EMPIEZA TODO PARA EL MODULO 0 ***//
void SPI0_HAL_Init(void){

    SIM_SCGC6 |= SIM_SCGC6_SPI0_MASK;       // Habilito el clock gating

    PORTD_PCR0|=PORT_PCR_ISF_MASK|PORT_PCR_MUX(0x02);//SPI0_PCS0
    PORTC_PCR3|=PORT_PCR_ISF_MASK|PORT_PCR_MUX(0x02);//SPI0_PCS1

    PORTD_PCR1|=PORT_PCR_ISF_MASK|PORT_PCR_MUX(0x02);//SCK
    PORTD_PCR2|=PORT_PCR_ISF_MASK|PORT_PCR_MUX(0x02);//Sout
    PORTD_PCR3|=PORT_PCR_ISF_MASK|PORT_PCR_MUX(0x02);//Sin

    PORTC_PCR5|=PORT_PCR_ISF_MASK|PORT_PCR_MUX(0x02);//SCK
    PORTC_PCR6|=PORT_PCR_ISF_MASK|PORT_PCR_MUX(0x02);//Sout
    PORTC_PCR7|=PORT_PCR_ISF_MASK|PORT_PCR_MUX(0x02);//Sin


    //SPI0_CTAR0|=SPI_CTAR_FMSZ(0x07)|SPI_CTAR_LSBFE_MASK|SPI_CTAR_PBR(0x02)|SPI_CTAR_
    BR(0x0B);// DBR=0, PBR=5, BR=2048=> SCK baud rate= 9765
    SPI0_CTAR0|=SPI_CTAR_FMSZ(0x07)|SPI_CTAR_PBR(0x02)|SPI_CTAR_BR(0x05);// DBR=0,
    PBR=5, BR=2048=> SCK baud rate= 9765
    SPI0_MCR&=~(1<<SPI_MCR_MDIS_SHIFT);  // Esto hace que ande el clock
    SPI0_MCR|=SPI_MCR_MSTR_MASK|SPI_MCR_DIS_TXF_MASK|SPI_MCR_DIS_RXF_MASK;
    SPI0_MCR&=~(1<<SPI_MCR_HALT_SHIFT); // Esto le dan ON al modulo
    SPI0_MCR|=0x30000;
}

void SPI0_ClrSPTIE(void){
  SPI0_RSER&=~(1<<SPI_RSER_TFFF_RE_SHIFT);

}
void SPI0_SetSPTIE(void){
  SPI0_RSER|=SPI_RSER_TFFF_RE_MASK;
}

void SPI0_ClrSPIE(void){
  SPI0_RSER&=~(1<<SPI_RSER_TFFF_RE_SHIFT);
```

```
}
void SPI0_SetSPIE(void){
 // SPI0CR1_SPIE=1;
  SPI0_RSER|=SPI_RSER_RFDF_RE_MASK;
}

unsigned int SPI0_ReadDataReg(void){
    uint32_t aux;
    aux=SPI0_POPR;
    return aux;
}

void SPI0_WriteDataRegS1(unsigned int midata){
    SPI0_PUSHR=0x10000|SPI_PUSHR_CTAS(0x00)|SPI_PUSHR_TXDATA(midata);
}
void SPI0_WriteDataRegS2(unsigned int midata){
  SPI0_PUSHR=0x20000|SPI_PUSHR_CTAS(0x00)|SPI_PUSHR_TXDATA(midata);
}

void SPI0_WriteDataReg(unsigned int midata){
    SPI0_PUSHR=0x10000|SPI_PUSHR_CTAS(0x00)|SPI_PUSHR_TXDATA(midata);
}

unsigned int SPI0_CheckSPTEF(void){
    int hola;
    hola=SPI0_SR & SPI_SR_TCF_MASK;
    return (hola);
}

unsigned char SPI0_CheckSPIF(void){
    return (SPI0_SR & SPI_SR_TCF_MASK);
}
```

```
#ifndef SPI_HAL_H
#define SPI_HAL_H
/*
 Definitions:
SS    Slave Select
SCK   Serial Clock
MOSI  Master Output, Slave Input
MISO  Master Input, Slave Output
MOMI  Master Output, Master Input
SISO  Slave Input, Slave Output
*/

/*
SPICR1: [7 6 5 4 3 2 1 0]
7->SPIE: SPI Interrupt Enable Bit.This bit enables SPI interrupt requests, if SPIF or MODF status flag is set.
          0 SPI interrupts disabled.
          1 SPI interrupts enabled.
6->SPE: SPI System Enable Bit. This bit enables the SPI system and dedicates the SPI port pins to SPI system
        functions. If SPE is cleared, SPI is disabled and forced into idle state, status bits in SPISR register are reset.
          0 SPI disabled (lower power consumption).
          1 SPI enabled, port pins are dedicated to SPI functions.
5->SPTIE: SPI Transmit Interrupt Enable. This bit enables SPI interrupt requests, if SPTEF flag is set.
          0 SPTEF interrupt disabled.
          1 SPTEF interrupt enabled.
4->MSTR:  SPI Master/Slave Mode Select Bit. This bit selects whether the SPI operates in master or slave mode.
          Switching the SPI from master to slave or vice versa forces the SPI system into idle state.
          0 SPI is in slave mode.
          1 SPI is in master mode.
3->CPOL:  SPI Clock Polarity Bit. This bit selects an inverted or non-inverted SPI clock. To transmit data between SPI
          modules, the SPI modules must have identical CPOL values. In master mode, a change of this bit will abort a
          transmission in progress and force the SPI system into idle state.
          0 Active-high clocks selected. In idle state SCK is low.
          1 Active-low clocks selected. In idle state SCK is high.
2->CPHA:  SPI Clock Phase Bit — This bit is used to select the SPI clock format. In master mode, a change of this bit will
          abort a transmission in progress and force the SPI system into idle state.
          0 Sampling of data occurs at odd edges (1,3,5,...,15) of the SCK clock.
          1 Sampling of data occurs at even edges (2,4,6,...,16) of the SCK
          clock.
1->SSOE:  Slave Select Output Enable — The SS output feature is enabled only in master mode, if MODFEN is set, by
          asserting the SSOE as shown in Table 12-2. In master mode, a change of this bit will abort a transmission in
          progress and force the SPI system into idle state.
0->LSBFE: LSB-First Enable — This bit does not affect the position of the MSB and LSB in the data register. Reads and
          writes of the data register always have the MSB in bit 7. In master mode, a change of this bit will abort a
          transmission in progress and force the SPI system into idle state.
          0 Data is transferred most significant bit first.
          1 Data is transferred least significant bit first.

Config: [x 1 x 1 0 0 1 0]
*/

/*
SPICR2: [7 6 5 4 3 2 1 0]
         0 0 0     0
4->MODFEN:  Mode Fault Enable Bit — This bit allows the MODF failure to be detected. If the SPI is in master mode and
            MODFEN is cleared, then the SS port pin is not used by the SPI. In slave mode, the SS is available only as an
            input regardless of the value of MODFEN. For an overview on the impact of the MODFEN bit on the SS port pin
            configuration, refer to Table 12-4. In master mode, a change of this bit
```

```
            will abort a transmission in progress and
            force the SPI system into idle state.
            0 SS port pin is not used by the SPI.
            1 SS port pin with MODF feature.
3->BIDIROE: Output Enable in the Bidirectional Mode of Operation — This bit controls the MOSI and MISO output buffer
            of the SPI, when in bidirectional mode of operation (SPC0 is set). In master mode, this bit controls the output
            buffer of the MOSI port, in slave mode it controls the output buffer of the MISO port. In master mode, with SPC0
            set, a change of this bit will abort a transmission in progress and force the SPI into idle state.
            0 Output buffer disabled.
            1 Output buffer enabled.
1->SPISWAI: SPI Stop in Wait Mode Bit — This bit is used for power conservation while in wait mode.
            0 SPI clock operates normally in wait mode.
            1 Stop SPI clock generation when in wait mode.
0->SPC0:  Serial Pin Control Bit 0 — This bit enables bidirectional pin configurations as shown in Table 12-4. In master
          mode, a change of this bit will abort a transmission in progress and force the SPI system into idle state.
Config: [0 0 0 1 1 0 0 0]
*/


/*
SPIBR: [7 6 5 4 3 2 1 0]
        0       0
6-4-> SPPR[2:0]:SPI Baud Rate Preselection Bits—These bits specify the SPI baud rates as shown in Table 12-6. In master
      mode, a change of these bits will abort a transmission in progress and force the SPI system into idle state.
2-0->SPR[2:0]:SPI Baud Rate Selection Bits—These bits specify the SPI baud rates as shown in Table 12-6. In master mode,
      a change of these bits will abort a transmission in progress and force the SPI system into idle state.

BaudRateDivisor = (SPPR + 1). (2^(SPR + 1))

Baud Rate = BusClock / BaudRateDivisor
*/

/*
SPISR: [7 6 5 4 3 2 1 0]
          0     0 0 0 0
7->SPIF:  SPIF Interrupt Flag — This bit is set after a received data byte has been transferred into the SPI data register.
          This bit is cleared by reading the SPISR register (with SPIF set) followed by a read access to the SPI data
          register.
          0 Transfer not yet complete.
          1 New data copied to SPIDR.
5->SPTEF: SPI Transmit Empty Interrupt Flag — If set, this bit indicates that the transmit data register is empty. To clear
          this bit and place data into the transmit data register, SPISR must be read with SPTEF = 1, followed by a write
          to SPIDR. Any write to the SPI data register without reading SPTEF = 1, is effectively ignored.
          0 SPI data register not empty.
          1 SPI data register empty.
4->MODF:  Mode Fault Flag—This bit is set if the SS input becomes low while the SPI is configured as a master and mode
          fault detection is enabled, MODFEN bit of SPICR2 register is set. Refer to MODFEN bit description in
          Section 12.3.2.2, “SPI Control Register 2 (SPICR2)”. The flag is cleared automatically by a read of the SPI status
          register (with MODF set) followed by a write to the SPI control register 1.
          0 Mode fault has not occurred.
          1 Mode fault has occurred.
Config: [0 0 1 0 0 0 0 0]
*/

```

```
/*
SPIDR: [7 6 5 4 3 2 1 0]

The SPI data register is both the input and output register for SPI data. A write to this register
allows a data byte to be queued and transmitted. For an SPI configured as a master, a queued data
byte is transmitted immediately after the previous transmission has completed. The SPI transmitter
empty flag SPTEF in the SPISR register indicates when the SPI data register is ready to accept new
data.
Received data in the SPIDR is valid when SPIF is set.
If SPIF is cleared and a byte has been received, the received byte is transferred from the receive
shift register to the SPIDR and SPIF is set.
If SPIF is set and not serviced, and a second byte has been received, the second received byte is
kept as valid byte in the receive shift register until the start of another transmission. The byte in the
SPIDR does not change.
If SPIF is set and a valid byte is in the receive shift register, and SPIF is serviced before the start of
a third transmission, the byte in the receive shift register is transferred into the SPIDR and SPIF
remains set (see Figure 12-8).
If SPIF is set and a valid byte is in the receive shift register, and SPIF is serviced after the start of
a third transmission, the byte in the receive shift register has become invalid and is not transferred
into the SPIDR (see Figure 12-9).

*/

/*
En la barracuda:

        Puerto H4           PINES ->El puerto empieza en PIN 1

                  MISO0      9
                  MOSI0      10
                  SCK0       11
                  SS0        12
*/

//***SERVICIOS***//

typedef unsigned int byte;

//***ARRANCA MODULO 0 ***//
void SPI0_HAL_Init(void);
void SPI_InitSPI0CR1(void);
void SPI_InitSPI0CR2(void);
void SPI0_SetBaudRate(void);
void SPI0_ClrSPTIE(void);
void SPI0_SetSPTIE(void);
void SPI0_ClrSPIE(void);
void SPI0_SetSPIE(void);
void SPI0_ClrCPOL(void);
void SPI0_SetCPOL(void);
void SPI0_ClrLSBFE(void);
void SPI0_SetLSBFE(void);
void SPI0_ClrCPHA(void);
void SPI0_SetCPHA(void);
unsigned int SPI0_ReadDataReg(void);
void SPI0_WriteDataReg(unsigned int midata);
void SPI0_WriteDataRegS1(unsigned int midata);
void SPI0_WriteDataRegS2(unsigned int midata);
unsigned int SPI0_CheckSPTEF(void);
unsigned char SPI0_CheckSPIF(void);

#endif
```

```c
/*
 * StateMachine.c
 *
 *  Created on: Nov 1, 2016
 *      Author: jpfag
 */

#include "StateMachine.h"
#include "StateMachineDefs.h"
#include "fsmdefines.h"
#include "fsmMain.h"
#include "fsmTx.h"
#include "fsmRx.h"
#include "fsmtableTx.h"
#include "fsmtableRx.h"
#include "fsmtableMain.h"

STATE *p2stateMain = NULL_STATE;    /*Used to store FSM state*/
STATE *p2stateRx = NULL_STATE;  /*Used to store FSM state*/
STATE *p2stateTx = NULL_STATE;  /*Used to store FSM state*/

unsigned char myExternalEvent = FSM_NO_EVENT;
unsigned char myMainEvent = FSM_NO_EVENT;
unsigned char spaceExternalEvent = STATEMACHINE_NOSTATE;
unsigned char myRxEvent = RX_NO_EVENT;
unsigned char myTxEvent = TX_NO_EVENT;
unsigned char newStateCommand = NO_COMMAND;
unsigned char myErrorEvent = ERROR_NONE;
unsigned char rxWriteBuffer[RXBUFFERSIZE];
unsigned char txWriteBuffer[TXBUFFERSIZE];
unsigned char rxReadBuffer[RXBUFFERSIZE];
unsigned char txReadBuffer[TXBUFFERSIZE];
unsigned int rxWriteBufferPointer = 0x00;
unsigned int rxReadBufferPointer = 0x00;
unsigned int txWriteBufferPointer = 0x00;
unsigned int txReadBufferPointer = 0x00;
int idOff;
int idOn;

void StateMachine_init(){
    p2stateMain = FSM_GetInitState();// Inicializo la FSM con el estado inicial
    p2stateRx = FSM_GetInitStateRx();// Inicializo la FSM con el estado inicial
    p2stateTx = FSM_GetInitStateTx();// Inicializo la FSM con el estado inicial
}

void StateMachine_getEventMain(){
    unsigned char aux = myExternalEvent;

    if(myExternalEvent != 0x07)
        aux = 0;

    if(newStateCommand == NEW_COMMAND_COMPLETE){
        if(myErrorEvent == ERROR_NONE){
            switch(myExternalEvent){
                case FSM_SEND_EVENT:
                    myMainEvent = FSM_SEND_EVENT;
                    break;
                case FSM_RECEIVE_EVENT:
                    myMainEvent = FSM_RECEIVE_EVENT;
                    break;
                case FSM_LENGTH_EVENT:
                    myMainEvent = FSM_LENGTH_EVENT;
                    break;
                case FSM_LP_ALL_EVENT:
                    myMainEvent = FSM_LP_ALL_EVENT;
                    break;
                case FSM_STBY_ALL_EVENT:
                    myMainEvent = FSM_STBY_ALL_EVENT;
                    break;
                case FSM_ERROR_RESET_EVENT:
                    myMainEvent = FSM_ERROR_RESET_EVENT;
                    break;
                case FSM_SHUTDOWN_EVENT:
```

```
74                      myMainEvent = FSM_SHUTDOWN_EVENT;
75                      break;
76                  case FSM_FORCE_CLEARPIN_EVENT:
77                      myMainEvent = FSM_FORCE_CLEARPIN_EVENT;
78                      break;
79                  default:
80                      myMainEvent = FSM_NO_EVENT;
81                      break;
82              }
83          }
84          else{
85              // Error Analysis --> event generated by interruptions?
86              switch(myErrorEvent){
87              case ERROR_ALIM:
88                  myMainEvent=ERROR_ALIM;
89                  break;
90              case ERROR_RX_TEMP:
91                  myMainEvent=ERROR_TEMP;
92                  break;
93              case ERROR_TX_TEMP:
94                  myMainEvent=ERROR_TEMP;
95                  break;
96              case ERROR_TX_COMM:
97                  myMainEvent=ERROR_TX;
98                  break;
99              case ERROR_RX_COMM:
100                 myMainEvent=ERROR_RX;
101                 break;
102             default:
103                     break;
104             }
105         }
106     }
107 }
108
109 void StateMachine_getEventRx(){
110     if(newStateCommand == NEW_COMMAND_COMPLETE){
111         if(myMainEvent == FSM_STBY_ALL_EVENT && ( (idOn==ID_RX) || (idOn==ID_BOTH) ))
112             myRxEvent = RX_STBY_EVENT;
113
114         if(myMainEvent == FSM_LP_ALL_EVENT && ( (idOff==ID_RX) || (idOff==ID_BOTH) ))
115             myRxEvent = RX_LP_EVENT;
116
117         if(myMainEvent == FSM_RECEIVE_EVENT && rxWriteBufferPointer !=
            rxReadBufferPointer)
118             myRxEvent = RX_RECEIVE_EVENT;
119
120         if((myMainEvent == FSM_ERROR_RESET_EVENT)&& ( (p2stateMain==errorRx) ||
            (p2stateMain==errorBoth) ))
121             myRxEvent = RX_STBY_EVENT;
122
123         if((myErrorEvent == ERROR_ALIM) ||(myErrorEvent == ERROR_RX_TEMP)
            ||(myErrorEvent ==  ERROR_RX_COMM))
124             myRxEvent = RX_LP_EVENT;
125     }
126 }
127
128 void StateMachine_getEventTx(){
129     if(newStateCommand == NEW_COMMAND_COMPLETE){
130
131         if(myMainEvent == FSM_STBY_ALL_EVENT && ( (idOn==ID_TX) || (idOn==ID_BOTH) ))
132             myTxEvent = TX_STBY_EVENT;
133
134         if(myMainEvent == FSM_SHUTDOWN_EVENT)
135             myTxEvent = TX_FORCE_EVENT;
136
137         if(myMainEvent == FSM_LP_ALL_EVENT && ( (idOff==ID_TX) || (idOff==ID_BOTH) ))
138             myTxEvent = TX_LP_EVENT;
139
140         if((p2stateTx == sendingTx || p2stateTx == stbyTx) && txWriteBufferPointer
            != txReadBufferPointer)
141             myTxEvent = TX_SEND_EVENT;
142
```

```
143         if((myMainEvent == FSM_ERROR_RESET_EVENT)&& ( (p2stateMain==errortx) ||
            (p2stateMain==errorBoth) ))
144             myTxEvent = TX_STBY_EVENT;
145
146         if((myErrorEvent == ERROR_ALIM) ||(myErrorEvent == ERROR_TX_TEMP)
            ||(myErrorEvent ==  ERROR_TX_COMM))
147             myTxEvent = TX_LP_EVENT;
148     }
149 }
150
151 void StateMachine_setEventMain(unsigned char state){
152     myMainEvent = state;
153 }
154
155 void StateMachine_setEventTx(unsigned char state){
156     myTxEvent = state;
157 }
158
159 void StateMachine_setEventRx(unsigned char state){
160     myRxEvent = state;
161 }
162
163 unsigned char StateMachine_InputState = STATEMACHINE_NOSTATE;
164 unsigned char StateMachine_Counter = 0x00;
165 unsigned char StateMachine_InputState_State = 0x00;
166
167 unsigned char Space_InputState = SPACE_NOSTATE;
168 unsigned char Space_Counter = 0x00;
169 unsigned char Space_InputState_State = 0x00;
170
171 unsigned char rxSendCounter = 0x00;
172
173 unsigned char checkExternalEvent(unsigned char externalEvent){
174     if(externalEvent == FSM_STBY_ALL_EVENT || externalEvent == FSM_LP_ALL_EVENT ||
      externalEvent == FSM_SEND_EVENT || externalEvent == FSM_RECEIVE_EVENT ||
      externalEvent == FSM_LP_ALL_EVENT || externalEvent == FSM_TURNON_EVENT ||
      externalEvent==FSM_SHUTDOWN_EVENT || externalEvent==FSM_SHUTDOWN_EVENT ||
      externalEvent == FSM_FORCE_CLEARPIN_EVENT || externalEvent
      ==FSM_ERROR_RESET_EVENT || externalEvent == FSM_LENGTH_EVENT)
175         return 1;
176     else
177         return 0;
178 }
179
180 unsigned char checkSpaceExternalEvent(unsigned char externalEvent){
181     if(externalEvent == FSM_STBY_ALL_EVENT || externalEvent == FSM_LP_ALL_EVENT ||
      externalEvent == FSM_SEND_EVENT || externalEvent == FSM_RECEIVE_EVENT ||
      externalEvent == FSM_LP_ALL_EVENT || externalEvent == FSM_TURNON_EVENT ||
      externalEvent==FSM_SHUTDOWN_EVENT)
182         return 1;
183     else
184         return 0;
185 }
186
187 void StateMachine_putEvent(unsigned char externalEvent){
188
189     switch(StateMachine_InputState){
190         case STATEMACHINE_NOSTATE:
191             if(checkExternalEvent(externalEvent)){
192                 myExternalEvent = externalEvent;
193                 StateMachine_InputState_State = externalEvent;
194                 if((myExternalEvent== FSM_RECEIVE_EVENT) ||
                  (myExternalEvent==FSM_ERROR_RESET_EVENT) ||
                  (myExternalEvent==FSM_SHUTDOWN_EVENT)||
                  (myExternalEvent==FSM_SHUTDOWN_EVENT) || myExternalEvent ==
                  FSM_FORCE_CLEARPIN_EVENT || myExternalEvent == FSM_LENGTH_EVENT){
195                     StateMachine_InputState = STATEMACHINE_NOSTATE;
196                     newStateCommand = NEW_COMMAND_COMPLETE;
197                 }
198                 else{
199                     StateMachine_InputState = STATEMACHINE_EXPECTINGLENGTH;
200                 }
201                 if(myExternalEvent== FSM_SEND_EVENT ||
```

```
                (myExternalEvent==FSM_SHUTDOWN_EVENT)){
                    txWriteBuffer[txWriteBufferPointer++] = externalEvent;
                    if (txWriteBufferPointer==TXBUFFERSIZE)
                        txWriteBufferPointer=0;
                }
            }
            else{
                myExternalEvent = FSM_EXTERNAL_EVENT_NOT_VALID;
            }
            break;
        case STATEMACHINE_EXPECTINGLENGTH:
            StateMachine_Counter = externalEvent;
            switch(StateMachine_InputState_State){
                case FSM_SEND_EVENT:
                    StateMachine_InputState = STATEMACHINE_RECEIVING;
                    txWriteBuffer[txWriteBufferPointer++] = externalEvent;
                    if (txWriteBufferPointer==TXBUFFERSIZE)
                        txWriteBufferPointer=0;
                    break;
                case FSM_LP_ALL_EVENT:
                    StateMachine_InputState = STATEMACHINE_NOSTATE;
                    idOff=StateMachine_Counter;
                    newStateCommand = NEW_COMMAND_COMPLETE;
                    break;
                case FSM_STBY_ALL_EVENT:
                    StateMachine_InputState = STATEMACHINE_NOSTATE;
                    idOn=StateMachine_Counter;
                    newStateCommand = NEW_COMMAND_COMPLETE;
                    break;
                default:
                    StateMachine_InputState = STATEMACHINE_NOSTATE;
                    newStateCommand = NEW_COMMAND_COMPLETE;
                    break;
            }
            // Validar la longitud del comando
            break;
        case STATEMACHINE_RECEIVING:
            txWriteBuffer[txWriteBufferPointer++] = externalEvent;
            if (txWriteBufferPointer==TXBUFFERSIZE)
                txWriteBufferPointer=0;
            if(!(--StateMachine_Counter)){
                StateMachine_InputState = STATEMACHINE_NOSTATE;
                newStateCommand = NEW_COMMAND_COMPLETE;
            }
            break;

        default:
            break;
    }
}

unsigned char isSpaceNewByte = 0;

void SPACE_receiveFromSpace(unsigned char externalEvent){
    int aux;

    if (externalEvent != 0x05)
        aux = externalEvent;

    switch(Space_InputState){
        case SPACE_NOSTATE:
            if(checkSpaceExternalEvent(externalEvent)){
                isSpaceNewByte = 1;
                spaceExternalEvent = externalEvent;
                Space_InputState_State = externalEvent;
                if(spaceExternalEvent== FSM_SHUTDOWN_EVENT){
                    Space_InputState = SPACE_NOSTATE;
                    myMainEvent = externalEvent;
                    myExternalEvent = externalEvent;
                    newStateCommand = NEW_COMMAND_COMPLETE;
                }
                else{
                    Space_InputState = SPACE_EXPECTINGLENGTH;
```

```
274                 }
275             }
276             else{
277                 spaceExternalEvent = FSM_EXTERNAL_EVENT_NOT_VALID;
278             }
279             break;
280         case SPACE_EXPECTINGLENGTH:
281             isSpaceNewByte = 1;
282             Space_Counter = externalEvent;
283             Space_InputState = SPACE_RECEIVING;
284             break;
285         case SPACE_RECEIVING:
286             isSpaceNewByte = 1;
287             rxReadBuffer[(rxWriteBufferPointer++)] = externalEvent;
288             if(rxWriteBufferPointer == RXBUFFERSIZE)
289                 rxWriteBufferPointer=0;
290             if(!(--Space_Counter)){
291                 aux = rxWriteBufferPointer;
292                 aux = externalEvent;
293                 Space_InputState = SPACE_NOSTATE;
294                 GPIO_write(READ_PIN, READ_PIN_DATA_AVAILABLE);
295             }
296             break;
297 
298         default:
299             break;
300     }
301 }
302 
303 unsigned char spaceNewByte(){
304     return isSpaceNewByte;
305 }
306 
307 void spaceClearByte(){
308     isSpaceNewByte = 0;
309 }
310 
311 void spaceRestart(){
312     Space_InputState = SPACE_NOSTATE;
313 }
314 
315 void clearEvents(){
316     newStateCommand = NO_COMMAND;
317     myMainEvent = FSM_NO_EVENT;
318     myRxEvent = RX_NO_EVENT;
319     myTxEvent = TX_NO_EVENT;
320 }
321 
322 void executeEvent(unsigned char localEvent){
323     p2stateTx=fsm(p2stateTx,localEvent);      //Se lo paso a la maquina de estados
324 }
325 
326 void StateMachine_run(){
327     unsigned char auxMain = myMainEvent;
328     unsigned char auxRx = myRxEvent;
329     unsigned char auxTx = myTxEvent;
330     if(newStateCommand == NEW_COMMAND_COMPLETE){
331             auxRx = myRxEvent;
332             auxTx = myTxEvent;
333             StateMachine_getEventMain();
334             auxMain = myMainEvent;
335             auxRx = myRxEvent;
336             auxTx = myTxEvent;
337             p2stateMain=fsm(p2stateMain,myMainEvent);      //Se lo paso a la maquina
            de estados
338             auxRx = myRxEvent;
339             auxTx = myTxEvent;
340             StateMachine_getEventTx();
341             auxTx = myTxEvent;
342             StateMachine_getEventRx();
343             auxRx = myRxEvent;
344             auxTx = myTxEvent;
345             p2stateRx=fsm(p2stateRx,myRxEvent);      //Se lo paso a la maquina de
```

```
                estados
346             p2stateTx=fsm(p2stateTx,myTxEvent);      //Se lo paso a la maquina de
                estados
347             clearEvents();
348         }
349     }
350
```

```c
/*
 * StateMachine.h
 */

#ifndef STATEMACHINE_H_
#define STATEMACHINE_H_

#include "fsmTx.h"
#include "fsmtableTx.h"
#include "fsmRx.h"
#include "fsmtableRx.h"
#include "fsmMain.h"
#include "fsmtableMain.h"

void StateMachine_run();
void StateMachine_init();
void StateMachine_putEvent(unsigned char externalEvent);
void executeEvent(unsigned char localEvent);
#endif /* STATEMACHINE_H_ */

```

```c
/*
 * StateMachineDefs.h
 */

#ifndef STATEMACHINEDEFS_H_
#define STATEMACHINEDEFS_H_

#define NULL_STATE 0x00;
#define FIN_TABLA 0xFF

typedef int BYTE;
typedef struct tabla_estado STATE;

struct tabla_estado
{
    BYTE  evento;
    STATE *prx_estado;
    void (*p_rut_accion)(void);
};

#endif /* STATEMACHINEDEFS_H_ */

```

```c
#include "SysTick.h"
#include "fsmdefines.h"
#include "RH_RF95.h"
#include "ADC.h"

static unsigned int constTime;
extern unsigned char myErrorEvent;
extern unsigned char newStateCommand;

uint8_t timeInterrupt = 0;

uint8_t isTimeterrupting(){
    return timeInterrupt;
}

void timeTerrupting(uint8_t value){
    timeInterrupt = value;
}

ISR_t SysTick_Handler(void)
{
    uint8_t aux;
    myErrorEvent = ERROR_NONE;
    /*if(!ADC_convIsStarted(ADC0_BASE_PTR))
        ADC_PISR();*/
    if(ADC_convCompl(ADC0_BASE_PTR)){
        if(ADC_result(ADC0_BASE_PTR) <= ADC_ERROR_LEVEL){
            aux = ADC_result(ADC0_BASE_PTR);
            myErrorEvent = ERROR_ALIM;
            newStateCommand = NEW_COMMAND_COMPLETE;
        }
    }
    // Check ADC
    if(RH_RF95_RX_CHECK_TEMP()){
        myErrorEvent = ERROR_RX_TEMP;
        newStateCommand = NEW_COMMAND_COMPLETE;
    }
    if(RH_RF95_TX_CHECK_TEMP()){
        myErrorEvent = ERROR_TX_TEMP;
        newStateCommand = NEW_COMMAND_COMPLETE;
    }
    if(RH_RF95_TX_CHECK_COMM()){
        myErrorEvent = ERROR_TX_COMM;
        newStateCommand = NEW_COMMAND_COMPLETE;
    }
    if(RH_RF95_RX_CHECK_COMM()){
        myErrorEvent = ERROR_RX_COMM;
        newStateCommand = NEW_COMMAND_COMPLETE;
    }
}

void SysTick_init(unsigned int msTime)
{
  SYST_CSR = 0x00;

  constTime = msTime - 1;

  SYST_RVR = SysTick_RVR_RELOAD(849999U); // 10ms
  SYST_CVR = SysTick_CVR_CURRENT(0);
  SYST_CSR = SysTick_CSR_CLKSOURCE_MASK |
             SysTick_CSR_TICKINT_MASK |
             SysTick_CSR_ENABLE_MASK;
}
```

```
#ifndef SYSTICK_H_
#define SYSTICK_H_

#include "misc.h"

ISR_t SysTick_Handler (void);

void SysTick_init (unsigned int msTime);
void timeTerrupting(uint8_t value);
uint8_t isTimeterrupting();

#endif
```

```c
/*
 * UART.c
 */
#include "UART.h"
#include "StateMachine.h"
/**HABILITAR FIFO READ y WRITE
 * LEERLO BIEN A VER DONDE LO ESCRIBE
 *  **/

uint8_t dataIndex = 0;
uint8_t dataFlag = 0;
uint8_t dataBuffer[30];

void UART_init(void){

    // Habilito la llave general de las interrupciones

    NVICISER0 |= NVIC_ISER_SETENA(1<<31);

    //Habilito los cuatro modulos de UART
    SIM_SCGC4 |= SIM_SCGC4_UART0_MASK;
    SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK;
    // Habilito el pin PTC1 como salida del canal 0
    PORTB_PCR16 = (PORT_PCR_ISF_MASK |
                                   PORT_PCR_MUX(0x03));  //Tx
    PORTB_PCR17 = (PORT_PCR_ISF_MASK |
                                   PORT_PCR_MUX(0x03)); //Rx

    //Habilito por interrupcion, un bit de stop y baudrate
    UART_C1_REG(UART0_BASE_PTR) = 0x00;

    UART_BDH_REG(UART0_BASE_PTR) = /*UART_BDH_RXEDGIE_MASK |
    */UART_BDH_SBR(BAUDRATEHIGH);
    UART_BDL_REG(UART0_BASE_PTR) = UART_BDL_SBR(BAUDRATELOW);

    UART_C2_REG(UART0_BASE_PTR) = UART_C2_TIE_MASK | UART_C2_TE_MASK |
    UART_C2_RE_MASK | UART_C2_RIE_MASK;
    UART_C3_REG(UART0_BASE_PTR) = UART_C3_TXDIR_MASK;
    UART_PFIFO_REG(UART0_BASE_PTR) = UART_PFIFO_TXFE_MASK |
    UART_PFIFO_TXFIFOSIZE(0x06) | UART_PFIFO_RXFE_MASK | UART_PFIFO_RXFIFOSIZE(0x01);
    UART_S2_REG(UART0_BASE_PTR) = 0x00;
}

void UART_writeByte(uint8_t data){
    UART_D_REG(UART0_BASE_PTR) = data;
    UART_C2_REG(UART0_BASE_PTR) |= UART_C2_TIE_MASK;
}

uint8_t UART_readByte(void){
    return UART_D_REG(UART0_BASE_PTR);
}

void UART_TransferComplete(){
    UART_C2_REG(UART0_BASE_PTR) &= ~UART_C2_TIE_MASK;
}

void UART_ReceptionComplete(){
    uint8_t aux = UART_readByte();
    StateMachine_putEvent(aux);
}

ISR_t UART0_Status_IRQHandler(void){
    uint8_t uartFlag = UART_S1_REG(UART0_BASE_PTR);
    UART_S1_REG(UART0_BASE_PTR) = 0x00;

    if(uartFlag & UART_S1_TC_MASK){
            UART_TransferComplete();
        }
    if(uartFlag & UART_S1_RDRF_MASK){
            UART_ReceptionComplete();

    }
}
```

```
71
72  uint8_t UART_isDataAvailable(void){
73      return dataFlag;
74  }
75
76  void UART_readDataBuffer(uint8_t *dataStore){
77      uint8_t index = 0;
78      // dataBuffer[index] != 10 &&
79      for(index = 0; index < 30; index++){
80          dataStore[index] = dataBuffer[index];
81      }
82      dataFlag = 0;
83  }
84  /**
85   * UARTx_BDH
86  RXEDGIE
87      RxD Input Active Edge Interrupt Enable
88      Enables the receive input active edge, RXEDGIF, to generate interrupt requests.
89      0 Hardware interrupts from RXEDGIF disabled using polling.
90      1 RXEDGIF interrupt request enabled.
91  SBNS
92      Stop Bit Number Select
93      SBNS selects the number of stop bits present in a data frame. This field valid
        for all 8, 9 and 10 bit data
94      formats available. This field is not valid when C7816[ISO7816E] is enabled.
95      0 Data frame consists of a single stop bit.
96  Baudrate
97      UART module clock / (16 × (SBR[12:0] + BRFD))
98  * UARTx_BDL
99  SBR
100     UART_BDH_SBR
101     UART_BDL_SBR
102 * UARTx_C1
103 LOOPS
104     Loop Mode Select
105     When LOOPS is set, the RxD pin is disconnected from the UART and the transmitter
        output is internally
106     connected to the receiver input. The transmitter and the receiver must be
        enabled to use the loop function.
107     0 Normal operation.
108     1 Loop mode where transmitter output is internally connected to receiver input.
        The receiver input is
109     determined by RSRC.
110 UARTSWAI
111     UART Stops in Wait Mode
112     0 UART clock continues to run in Wait mode.
113     1 UART clock freezes while CPU is in Wait mode.
114 RSRC
115     Receiver Source Select
116     This field has no meaning or effect unless the LOOPS field is set. When LOOPS is
        set, the RSRC field
117     determines the source for the receiver shift register input.
118     0 Selects internal loop back mode. The receiver input is internally connected to
        transmitter output.
119     1 Single wire UART mode where the receiver input is connected to the transmit
        pin input signal.
120 M
121     9-bit or 8-bit Mode Select
122     This field must be set when C7816[ISO_7816E] is set/enabled.
123     0 Normal—start + 8 data bits (MSB/LSB first as determined by MSBF) + stop.
124     1 Use—start + 9 data bits (MSB/LSB first as determined by MSBF) + stop.
125 WAKE
126     Receiver Wakeup Method Select
127     Determines which condition wakes the UART:
128     • Address mark in the most significant bit position of a received data
        character, or
129     • An idle condition on the receive pin input signal.
130     0 Idle line wakeup.
131     1 Address mark wakeup.
132 ILT
133     Idle Line Type Select
134     Determines when the receiver starts counting logic 1s as idle character bits.
        The count begins either after
```

a valid start bit or after the stop bit. If the count begins after the start bit, then a string of logic 1s preceding
the stop bit can cause false recognition of an idle character. Beginning the count after the stop bit avoids
false idle character recognition, but requires properly synchronized transmissions.
NOTE: • In case the UART is programmed with ILT = 1, a logic of 1'b0 is automatically shifted after a
received stop bit, therefore resetting the idle count.
• In case the UART is programmed for IDLE line wakeup (RWU = 1 and WAKE = 0), ILT has
no effect on when the receiver starts counting logic 1s as idle character bits. In idle line
wakeup, an idle character is recognized at anytime the receiver sees 10, 11, or 12 1s
depending on the M, PE, and C4[M10] fields.
0 Idle character bit count starts after start bit.
1 Idle character bit count starts after stop bit.
PE
Parity Enable
Enables the parity function. When parity is enabled, parity function inserts a parity bit in the bit position
immediately preceding the stop bit. This field must be set when C7816[ISO_7816E] is set/enabled.
0 Parity function disabled.
1 Parity function enabled.
PT
Parity Type
Determines whether the UART generates and checks for even parity or odd parity. With even parity, an
even number of 1s clears the parity bit and an odd number of 1s sets the parity bit. With odd parity, an odd
number of 1s clears the parity bit and an even number of 1s sets the parity bit. This field must be cleared
when C7816[ISO_7816E] is set/enabled.
0 Even parity.
1 Odd parity.
* UARTx_C2
TIE
Transmitter Interrupt or DMA Transfer Enable.
Enables S1[TDRE] to generate interrupt requests or DMA transfer requests, based on the state of
C5[TDMAS].
NOTE: If C2[TIE] and C5[TDMAS] are both set, then TCIE must be cleared, and D[D] must not be written
unless servicing a DMA request.
0 TDRE interrupt and DMA transfer requests disabled.
1 TDRE interrupt or DMA transfer requests enabled
TCIE
Transmission Complete Interrupt or DMA Transfer Enable
Enables the transmission complete flag, S1[TC], to generate interrupt requests . or DMA transfer requests
based on the state of C5[TCDMAS]
NOTE: If C2[TCIE] and C5[TCDMAS] are both set, then TIE must be cleared, and D[D] must not be
written unless servicing a DMA request.
0 TC interrupt and DMA transfer requests disabled.
1
TC interrupt or DMA transfer requests enabled.
RIE
Receiver Full Interrupt or DMA Transfer Enable
Enables S1[RDRF] to generate interrupt requests or DMA transfer requests, based on the state of
C5[RDMAS].
0 RDRF interrupt and DMA transfer requests disabled.
1 RDRF interrupt or DMA transfer requests enabled.
ILIE
Idle Line Interrupt DMA Transfer Enable
Enables the idle line flag, S1[IDLE], to generate interrupt requestsor DMA transfer requests based on the
state of C5[ILDMAS].
0 IDLE interrupt requests disabled. and DMA transfer
1 IDLE interrupt requests enabled. or DMA transfer

```
TE
    Transmitter Enable
    Enables the UART transmitter. TE can be used to queue an idle preamble by
    clearing and then setting TE.
    When C7816[ISO_7816E] is set/enabled and C7816[TTYPE] = 1, this field is
    automatically cleared after
    the requested block has been transmitted. This condition is detected when
    TL7816[TLEN] = 0 and four
    additional characters are transmitted.
    0 Transmitter off.
    1 Transmitter on.
RE
    Receiver Enable
    Enables the UART receiver.
    0 Receiver off.
    1 Receiver on.
RWU
    Receiver Wakeup Control
    This field can be set to place the UART receiver in a standby state. RWU
    automatically clears when an
    RWU event occurs, that is, an IDLE event when C1[WAKE] is clear or an address
    match when C1[WAKE]
    is set. This field must be cleared when C7816[ISO_7816E] is set.
    NOTE: RWU must be set only with C1[WAKE] = 0 (wakeup on idle) if the channel is
    currently not idle.
    This can be determined by S2[RAF]. If the flag is set to wake up an IDLE event
    and the channel
    is already idle, it is possible that the UART will discard data. This is because
    the data must be
    received or a LIN break detected after an IDLE is detected before IDLE is
    allowed to reasserted.
    0 Normal operation.
    1 RWU enables the wakeup function and inhibits further receiver interrupt
    requests. Normally, hardware
    wakes the receiver by automatically clearing RWU.
SBK
    Send Break
    Toggling SBK sends one break character from the following: See Transmitting
    break characters for the
    number of logic 0s for the different configurations. Toggling implies clearing
    the SBK field before the break
    character has finished transmitting. As long as SBK is set, the transmitter
    continues to send complete
    break characters (10, 11, or 12 bits, or 13 or 14 bits, or 15 or 16 bits).
    Ensure that C2[TE] is asserted
    atleast 1 clock before assertion of this bit.
    • 10, 11, or 12 logic 0s if S2[BRK13] is cleared
    • 13 or 14 logic 0s if S2[BRK13] is set.
    • 15 or 16 logic 0s if BDH[SBNS] is set.
    This field must be cleared when C7816[ISO_7816E] is set.
    0 Normal transmitter operation.
    1 Queue break characters to be sent.
* UARTx_D
    Reads return the contents of the read-only receive data register and writes go
    to the write-only transmit
    data register.
* UARTx_S1

 */

//hoja 1609
/**
 * To initiate a UART transmission:
1. Configure the UART.
a. Select a baud rate. Write this value to the UART baud registers (BDH/L) to
begin the baud rate generator. Remember that the baud rate generator is disabled
when the baud rate is zero. Writing to the BDH has no effect without also
writing to BDL.
b. Write to C1 to configure word length, parity, and other configuration bits
(LOOPS, RSRC, M, WAKE, ILT, PE, and PT). Write to C4, MA1, and MA2 to
configure.
c. Enable the transmitter, interrupts, receiver, and wakeup as required, by writing to
C2 (TIE, TCIE, RIE, ILIE, TE, RE, RWU, and SBK), S2 (MSBF and BRK13),
```

```
and C3 (ORIE, NEIE, PEIE, and FEIE). A preamble or idle character is then
shifted out of the transmitter shift register.
2. Transmit procedure for each byte.
a. Monitor S1[TDRE] by reading S1 or responding to the TDRE interrupt. The
amount of free space in the transmit buffer directly using TCFIFO[TXCOUNT]
can also be monitored.
b. If the TDRE flag is set, or there is space in the transmit buffer, write the data
to
be transmitted to (C3[T8]/D). A new transmission will not result until data exists
in the transmit buffer.
3. Repeat step 2 for each subsequent transmission.
 */


```