UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

PROGRAMA DE MAESTRIA Y DOCTORADO EN INGENIERIA

DESARROLLO DE COMPUTADORA DE VUELO Y SOFTWARE TERRESTRE DE OPERACIONES PARA EL SATÉLITE HUMSAT

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN INGENIERIA

INGENIERÍA ELÉCTRICA - S I S T E M A S
E L E C T R Ó N I C O S
P R E S E N T A :

IGNACIO MENDOZA NUCAMENDI

TUTOR:
Dr. ESAÚ VICENTE VIVAS

INSTITUTO DE INGENIERÍA UNAM

2012

JURADO ASIGNADO:

Presidente: Dr. Bohumil Psenicka

Secretario: Dr. Salvador Landeros Ayala

Vocal: Dr. Esaú Vicente Vivas

1er Suplente: Dra. Fátima Moumtadi

2do Suplente: Dr. Mario Peña Cabrera

Lugar donde se realizó la tesis:

INSTITUTO DE INGENIERÍA, UNAM

TUTOR DE TESIS

Dr. Esaú Vicente Vivas

_____

FIRMA

Dedicatoria

A mis padres, que con su esfuerzo y amor me han apoyado a lo largo de todo este tiempo para dar un paso más.

Agradecimientos

Quiero agradecer a todos los que me han animado, apoyado y compartido el desarrollo de este proyecto. A ustedes dedico este espacio para hacer notar que el éxito puede medirse no sólo con los logros obtenidos, sino también con las relaciones que cuidas y el apoyo de la gente que te quiere.

A Dios, quien me ha dado identidad, recursos, propósito y destino.

A mis padre Ignacio Mendoza y Josefa Nucamendi, a quienes con su apoyo y amor me permitieron concluir con esta etapa.

A mis hermanos Esperanza, Jorge y Luis Enrique, quienes han compartido conmigo todos estos años.

A mis compañeros y amigos del Instituto de Ingeniería, por compartir sus conocimientos, experiencia y por el tiempo que hemos convivido.

A mi asesor Dr. Esaú Vicente Vivas, por su confianza y apoyo incondicional, por compartir conmigo la visión de un sueño para México en el desarrollo tecnológico.

A mi casa de estudios, la UNAM.

Índice de capítulos

Índice General

Capítulo 1. Generalidades del proyecto

Índice de figuras

Índice de Tablas

Resumen

El presente trabajo muestra el diseño y validación del subsistema de Computadora de Vuelo (CV) para el Nanosatélite HumSAT-México en la Plataforma Satelital Educativa SATEDU (Satélite Educativo), un satélite bajo proceso de patente diseñado, fabricado y validado en el Instituto de Ingeniería de la UNAM.

El subsistema de CV permite la interacción entre la Estación en Tierra y la Plataforma Satelital, así como la ejecución autónoma de un programa de operaciones de vuelo que además ofrezca capacidades de automatización para la carga útil (en caso de que lo requiera).

Para el diseño del nuevo modelo de CV se retomaron experiencias y trabajo de tesis previas realizadas en 2007 y en 2009 para la Plataforma Satelital SATEDU, con los siguientes resultados:

Primero, se propuso un nuevo diseño en hardware para la CV que provee al subsistema de interfaces adecuadas para realizar la conectividad con la plataforma SATEDU, pero con recursos actualizados suficientes que permiten su aplicación directa al satélite HumSAT-México, además de su interacción con diferentes recursos periféricos.

Segundo, se rediseñó el *firmware* original de la CV de SATEDU, para poder ejecutarlo en un procesador Cortex3-ARM de 32 bits, el cual es el núcleo del nuevo prototipo de CV del satélite HumSAT-México.

Tercero, se implementó una Interfaz Gráfica de Usuario (GUI, por sus siglas en Inglés) con un diseño modular multi-hilo como Software Básico de Operaciones en Tierra desarrollada en la plataforma .NET, además de otras aplicaciones que fueron integradas finalmente cómo módulos del la GUI.

Abstract

This paper shows the design and validation of the flight computer (FC) subsystem for the HumSAT-Mexico satellite which is to be validated on the Educative Satellite (SATEDU) Platform. SATEDU was designed, manufactured and validated at the Institute of Engineering UNAM and at present time is under patent process at the same institution..

The FC subsystem allows interaction between the ground station and the satellite platform. It also allows the autonomous execution of satellite operations, and provides automation resources for payload operations (when required). The design of the new model of FC took advantage of previous thesis work performed in 2007 and 2009 for SATEDU satellite. This achieved the following results:

First, it was proposed a new FC hardware design which allows SATEDU to integrate better connectivity interfaces. In addition it will add adequate resources for operations automation to the HumSAT-Mexico satellite and its peripheral subsystems.

Second, it was redesigned the original SATEDU FC firmware to run on a Cortex3-ARM 32-bit processor, which is the core system for the new HumSAT- Mexico FC.

Third, it was implemented a Graphical User Interface (GUI) with a modular multi-threaded design which will be used as basic software for SATEDU´s Ground Station. The latter was developed on .NET platform along with other applications that were integrated as GUI modules.

Capítulo 1. Generalidades del Proyecto

En el presente capítulo se comentan los antecedentes del proyecto HumSAT, las actividades desarrolladas por el Instituto de Ingeniería de la UNAM en el desarrollo del satélite HumSAT-México, así como una breve descripción del estado del arte en el desarrollo de satélites pequeños. Finalmente, se comentan los requerimientos y metodología de diseño para el desarrollo de la Computadora de Vuelo de este satélite.

1.1. Tecnología Espacial

Cuando se habla de Tecnología Espacial suele relacionarse con viajes a la luna, naves espaciales y astronautas. Sin embargo, la Tecnología Espacial se encuentra presente en la vida cotidiana, trayendo beneficios a la sociedad.

En términos generales, la tecnología puede definirse como "la aplicación de conocimiento para construir objetos que satisfacen necesidades humanas", mientras que la Tecnología Espacial se encarga de dar solución a problemas y necesidades relacionadas con actividades humanas en el espacio , es decir, sobre la frontera que se sitúa a 100km de altura sobre el nivel del mar.

La Tecnología Espacial ha dado pie a resolver problemas no sólo de carácter espacial, sino que algunas soluciones han sido adaptadas a soluciones de la vida cotidiana conocidas como *spin off*. Ejemplo de artículos de la vida cotidiana que fueron diseñados inicialmente para ser usados en el espacio son los nuevos materiales, los fármacos de gran calidad, lo pañales desechables, los sistemas de conservación de alimentos, la tecnología médica, entre otros ejemplos.

1.1.1. Satélites Artificiales

Los satélites artificiales son vehículos espaciales que orbitan alrededor de la Tierra. Fue el Sputnik-1, un satélite de diseño ruso, el primer satélite que orbitó con éxito la Tierra en 1957.

El diseño y construcción de sistemas y subsistemas de un satélite, así como el satélite mismo, requieren del uso de tecnología con características especiales y mejores que los que se usan comúnmente para diseñar y construir dispositivos de consumo comercial, ya que las condiciones a las que estarán sometidos son extremas. Por lo cual son también de gran importancia otros aspectos especiales como el consumo de energía, el tipo de materiales empleados y el peso.

Clasificación de los satélites artificiales

Los satélites artificiales pueden clasificarse en tres grupos dependiendo de características específicas como son la órbita en la que operan, su masa y su aplicación.

En lo que se refiere al tipo de órbita, se pueden considerar tres grupos que abarcan su forma, la inclinación y altura. La forma de una órbita se puede clasificar en elíptica (también llamada excéntrica) o en circular, que es un caso espacial de una órbita elíptica donde su excentricidad es cero.

En lo referente a la inclinación de la órbita, está se mide con respecto al plano perpendicular al eje de rotación del cuerpo celeste que orbite, es decir, se toma con referencia la línea del ecuador. Si la órbita tiene una inclinación cercana a los 90º se considera una órbita polar, mientras que si es cercana a los 0º se le conoce con orbita ecuatorial. Cuando la órbita no es polar ni ecuatorial se le llama inclinada.

Al considerar la altura como un parámetro de clasificación se toma como referencia el nivel medio del mar y este parámetro sólo sirve para clasificar satélites que orbitan la Tierra. Los satélites de órbita baja son aquellos que se encuentran a una altura sobre nivel medio del mar de menos de 1000Km, mientras que los de órbita media se encuentran alrededor de los 10,000Km y los de órbita geoestacionaria se encuentran a 36,000 Km.

Las orbitas geoestacionarias tiene la característica de que al colocar un satélite, éste se moverá a la velocidad de rotación de la Tierra, por lo que parecerá que el cuerpo permanece inmóvil con respecto a la Tierra.

Entre las aplicaciones de los satélites de órbita baja se encuentran los usados para percepción remota, es decir, los que tienen una cámara o radar como carga útil para observar la Tierra, así como los militares y aquellos que realizan algún tipo de experimento relacionado con condiciones atmosféricas.

Los satélites de órbita media son usados generalmente para aplicaciones de navegación como el sistema de posicionamiento global (GPS), mientras que algunas de las aplicaciones de los satélites de órbita geoestacionaria son las telecomunicaciones y la meteorología.

La clasificación por su masa define las siguientes categorías. Los picosatélites son aquellos que pesan hasta 1 Kg. Los nanosatélites agrupan a los que se encuentran en el rango de 1 a 10Kg y al día de hoy son los más adecuados para realizar misiones en corto tiempo y recursos relativamente accesibles. Los microsatélites tienen un peso que va de los 10 a los 100Kg, mientras que los minisatélites van de 500 a 500Kg, los satélites medianos de 500 a 1000Kg y los macrosatélites de más de 1000Kg.

Aplicaciones de los satélites artificiales

Existen diferentes áreas donde se hace uso de satélites artificiales para ofrecer servicios que den solución a diversos problemas. La aplicación o misión está muy relacionada con la carga útil que le es colocada al satélite. De manera general podemos hablar de satélites científicos, de comunicaciones, meteorológicos, de exploración de recursos naturales, de navegación, militares, de percepción remota, entre otras aplicaciones más.

Los más conocidos suelen ser los satélites de comunicaciones, usados para la retransmisión de señales de televisión, voz y datos. Este tipo de satélites se encuentra por lo general en una órbita geoestacionaria aunque es posible hacer uso de constelaciones de satélites de órbita baja para esta aplicación.

Los satélites meteorológicos llevan como carga útil, instrumentos de percepción remota que permiten observar desde el espacio las condiciones atmosféricas, con lo que es posible tomar medidas de prevención en caso de huracanes, tormentas o para detección de incendios.

Las satélites de navegación envían señales a la Tierra para que ciertos receptores puedan mostrar la latitud, longitud y altura con la que se determina la posición del receptor en Tierra. El sistema más conocido de satélites de navegación es el GPS o Sistema de Posicionamiento Global.

Los satélites militares son empleados por los servicios de defensa para aplicaciones de transmisiones especiales o satélites espías.

Los nano y picosatélites tienen una tendencia a trabajar en arreglos llamados clúster espaciales o constelaciones, con lo que es posible que su uso pueda resolver problemas en diferentes áreas del interés científico, social y económico, entre otros como la detección temprana de sismos, detección de incendios, bancos de peses, monitoreo de cambio climático, tele-salud y ayuda humanitaria.

1.3. Estado del Arte en el Desarrollo de Satélites Pequeños

1.3.1. Antecedentes del desarrollo satelital en México

México ha participado de la tecnología satelital desde los años 80 en la compra, operación y uso de satélites geoestacionarios comerciales de comunicaciones. En los 80 se lanzaron al espacio los sistemas Morelos 1 y 2 (1985), en los 90 los sistemas Solidaridad 1 (1993) y 2 (1994), el Satmex 5 (1998) y el Satmex 6 se orbitó en 2006. Todos estos satélites se construyeron, validaron y lanzaron fuera de México.

Los primeros satélites pequeños que han sido diseñados en México fueron el UnamSat A y B, construidos en la UNAM, lanzados el 28 de marzo de 1995 y el 5 de septiembre de 1996 con sistemas de lanzamiento Ruso. El primero se perdió al explotar el sistema de transporte espacial que lo conducía a órbita y el segundo se colocó en el espacio pero sólo operó por un día.

El diseño de ambos vehículos pertenece a la compañía internacional AMSAT a quien la UNAM compró un paquete tecnológico abierto para reproducirlo y modificarlo de acuerdo con sus necesidades, no obstante, la tecnología que emplearon en cuanto a la plataforma satelital sólo pertenece a AMSAT.

Posteriormente se desarrolló en México el proyecto Microsatelital Satex de 55Kg, que pretendía desarrollar un satélite con tecnología mexicana en cada uno de sus subsistemas. Participaron instituciones como el CICESE[1], BUAP, CITEDI[2] (IPN[3]), ESIME Ticomán, ESIME[4] Zacatenco (IPN),

[1] Centro de Investigación Científica y de Educación Superior de Ensenada
[2] Centro de Investigación y Desarrollo Tecnológico Digital
[3] Instituto Politécnico Nacional
[4] ESIME Escuela Superior de Ingeniería Mecánica y Electrónica

CIMAT[5], INAOE[6] y la UNAM[7]. Este satélite no llegó a terminarse, sin embargo varias instituciones lograron terminar sus subsistemas, entre ellos el CIMAT (Estabilización satelital) y la UNAM (Instrumentación y software de vuelo, así como software de estación terrestre).

En 2005 la UNAM inició la adquisición del nanosatélite UnamSat III, de diseño y manufactura rusa. En su construcción y pruebas se esperó la participación de estudiantes doctorales mexicanos que estudiaron en Rusia. El proyecto persigue realizar investigación sobre la detección temprana de sismos.

1.3.2. Desarrollo de Satélites en Universidades

El Estándar CubeSat

CubeSat es la designación para un tipo de satélites pequeños enfocados principalmente a la investigación, innovación, validación de tecnologías y formación de recursos humanos. Los satélites del tipo CubeSat forman parte de un proyecto desarrollado por el Dr. Jordi Puig-Suari de la Universidad Politécnica Estatal de California (Cal Poly) en San Luis Obispo y el Prof. Robert Twiggs de la Universidad de Stanford en Palo Alto, California.

Entre sus especificaciones establecidas en 1999 se encuentran que el volumen del sistema no debe exceder el volumen de 1 litro y el peso de 1Kg.

Otra característica interesante de los satélites CubeSat es que son construidos habitualmente con los mejores elementos y componentes comerciales, con lo cual es posible reducir significativamente el costo final por cada satélite, lo que representa una ayuda a las universidades de todo el mundo para realizar investigación, innovación y desarrollo tecnológico espacial con una inversión relativamente reducida.

[5] Centro de Investigación en Matemáticas
[6] Instituto Nacional de Astrofísica, Óptica y Electrónica
[7] Universidad Nacional Autónoma de México

XATCobeo

XATCobeo es un proyecto español desarrollado en la Universidad de Vigo. El proyecto es dirigido por el Dr. Fernando Aguado en colaboración con el Instituto Nacional de Técnica Aeroespacial (INTA), el apoyo de la empresa Retegal y la junta de Galicia. Se estima que la vida útil del dispositivo aeroespacial será de 6 a 12 meses y que su costo aproximado es de 1.2 millones de euros. El principal objetivo de la misión es el desarrollo del satélite en sí mismo bajo el estándar CubeSat, junto a una Estación en Tierra en la Universidad de Vigo.

La carga útil del satélite consiste en un programa definido de radio reconfigurable (*software defined reconfigurable radio o SRAD*) y un sistema de medición de cantidad de radiación ionizante *(ionizing radiation system* o RDS). Se incluye también un sistema experimental de despliegue de paneles solares. Con el desarrollo de este proyecto se favorece un escenario donde profesores, investigadores y estudiantes pueden enfrentarse a la resolución de problemas en áreas como electrónica, ingeniería, comunicación y desarrollo de software, para la construcción del vehículo espacial.

El diseño del satélite se apega al estándar CubeSat con una dimensión de $10cm^3$ y aproximadamente un kilogramo de peso. Además, ha sido necesario que se apegue a estándares de documentación y seguimiento de proyectos propuestos por la Agencia Espacial Europea para garantizar su correcto funcionamiento en el espacio así como su seguridad durante el vuelo.

Figura 1. 1 Ensamblado del XATCobeo en cámara limpia.

SATEDU

SATEDU es un proyecto mexicano primeramente al entrenamiento de recursos humanos en el manejo, diseño y operación de satélites pequeños; y adicionalmente, como un sistema de apoyo para el desarrollo y validación tanto de nuevos sistemas satelitales como de cargas útiles. SATEDU es el resultado de años de trabajo de equipo multidisciplinario en el Instituto de Ingeniería de la UNAM para generar un satélite educativo que pueda ser usado en las aulas de laboratorios escolares, tecnológicos, universidades y centros de investigación.

Uno de los objetivos del proyecto SATEDU es atraer a los jóvenes al mundo de la ciencia y la Tecnología, mientras que a la par puede realizarse el desarrollo y validación de nuevos subsistemas satelitales que serán empelados en satélites pequeños reales.

Figura 1. 2. Plataforma Satelital para entrenamiento de recursos humanos SATEDU

El proyecto SATEDU es compatible en tamaño y operación respecto al estándar propuesto en el CubeSAT y cuenta con todos los subsistemas que forman a un satélite comercial como son: Estructura, Potencia, Computadora de Vuelo, Comunicaciones Inalámbricas, Sensores de Plataforma Satelital, Estabilización por Rueda Inercial y Sensores de Navegación Inercial. Adicionalmente, se encuentran en desarrollo otros sistemas de mayor complejidad en el campo de estabilización y autonomía satelital, comunicaciones y carga útil.

Para su operación, SATEDU cuenta con un software distribuido en cada uno de sus subsistemas que se ejecuta en diferentes microcomputadoras que operan en cada módulo. Dispone también de software para la comunicación entre la Estación Terrena y la Plataforma Satelital el cual corre en una PC.

Una de las principales características del desarrollo de SATEDU es su bajo costo, que se encuentra alrededor de los 25,000 pesos. Este monto es inferior al precio de sistemas similares, en particular el desarrollado por la Fuerza Aérea Norteamericana y que actualmente comercializa la compañía *Colorado Satellite Services* el cual tiene un costo aproximado de 120,000 pesos.

Subsistema de Computa de Vuelo

La Computadora de Vuelo es la encargada de ejecutar el programa de misión, así como de administrar los diferentes recursos de la plataforma satelital y la carga útil. La primera versión de SATEDU cuenta con un microprocesador SAB de Siemens como corazón de su sistema de CV, además de 32 MB de memoria Flash, 3 sensores de temperatura y 3 arreglos de efecto *latch-up* como protección para radiación.

Figura 1. 3. Primera versión de Computadora de Vuelo de SATEDU

Subsistema de Potencia

El Subsistema de Potencia es el encargado de suministrar la alimentación eléctrica al resto de los subsistemas. Para realizar un consumo inteligente de los recursos energéticos, cuenta con una microcomputadora que se mantiene en comunicación con la CV para realizar el encendido y apagado de las fuentes de energía de los demás subsistemas, además del recargado de las fuentes de energía primaria y secundaria que son 4 baterías de Li-Ion a través de celdas solares o por medio de un tomacorriente externo al satélite.

Figura 1. 4. Subsistema de Potencia de SATEDU

Subsistema de Estabilización

El Subsistema de Estabilización es el encargado de corregir la posición del satélite. Para ello ocupa dos métodos de estabilización activa que son una rueda inercial y bobinas de torque magnético.

Figura 1. 5. Subsistema de estabilización por rueda inercial de SATEDU

Subsistema de Comunicaciones

Consiste en dos tarjetas para la comunicación inalámbrica entre un equipo de cómputo, que hace las veces de Estación Terrena, con la Plataforma Satelital. La tarjeta conectada a la PC hace uso de un cable USB para obtener la alimentación eléctrica y establecer la comunicación, mientras que la tarjeta de comunicaciones de SATEDU se conecta al tándem de tarjetas de SATEDU. La frecuencia de operación del Subsistema de Comunicaciones es de 2.4GHz.

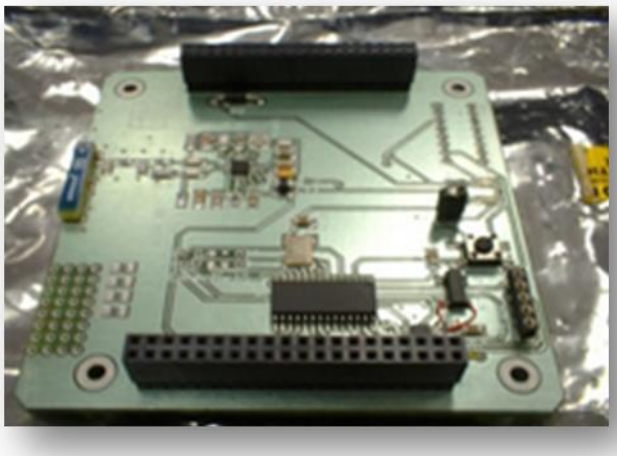


Figura 1. 6. Subsistema de Comunicaciones de SATEDU

Subsistema de sensores de Navegación Inercial

Para realizar las tareas de control y apuntamiento, es necesario tener cierta información de la plataforma satelital que permita monitorear su movimiento. Los sensores que componen al

subsistema de Sensores son una brújula electrónica, un acelerómetro triaxial y 3 giróscopos electrónicos dispuestos de manera ortogonal para monitorear los 3 ejes del satélite.

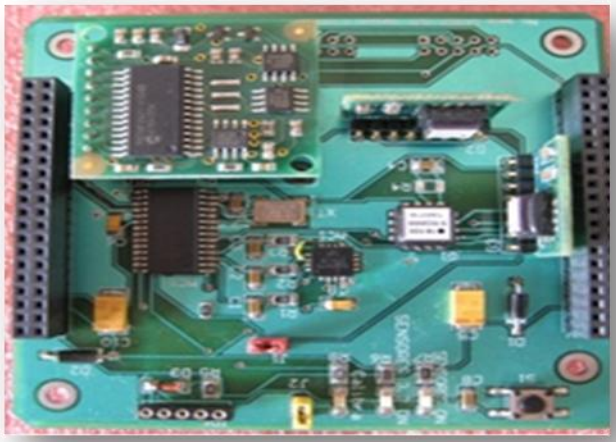


Figura 1. 7. Subsistema de sensores de Navegación Inercial de SATEDU

1.4. El proyecto HumSAT-México

El proyecto de Constelación Internacional de Satélites llamada HumSAT (humanitarian satellite constellation) será una constelación de satélites pequeños y contendrá además el soporte de estaciones base en Tierra. Su objetivo será prover servicios de ayuda humanitaria (servicios básicos de tele-medicina a zonas aisladas de todo el planeta) y de monitoreo de redes de sensores para diversas aplicaciones (estudio del cambio climático, contaminación en ríos, lagunas y mares, etcétera). Particularmente, el proyecto HumSAT-México persigue el desarrollo de Tecnología Nacional en prácticamente todos los subsistemas satelitales. Sin embargo, hace uso de software y estándares aceptados por la comunidad espacial para promover una compatibilidad internacional.

HumSAT-México espera ser uno de los primeros satélites de está constelación. Hasta el momento hay más de 21 Universidades de todo el mundo interesadas en fabricar un satélite para esta constelación. En el caso de España, su satélite HUMSAT será herencia del proyecto Xatcobeo.

1.4.1. Generalidades del proyecto HumSAT

HumSAT es un proyecto liderado por España, particularmente por la Universidad de Vigo. El proyecto lo inició con Calpoly (Dr. Jordi Puig Sauri) y después se unió el Instituto de Ingeniería UNAM por México, posteriormente se invitó al Dr. Sergio Camacho, Secretario General del

CRECTEALC (Centro Regional de Educación en Ciencia y Tecnología Espacial para América Latina y el Caribe) para integrarse al proyecto. Con iniciativas del Dr. Camacho se logró el apoyo de la Oficina de las Naciones Unidas para Asuntos del Espacio Ultraterrestre (UNOOSA). El proyecto está dirigido a la creación de una constelación de satélites pequeños (no importa la masa) en cooperación cualquier Universidad y país interesado en la iniciativa.

El proyecto HumSAT pretende hacer uso prácticamente de cualquier tipo de satélite y hará fuerte uso del sistema GENSO[8], una red mundial de Estaciones en Tierra de radioaficionados y universidades que dará soporte a proyectos universitarios.

El objetivo principal del sistema HumSAT es el desarrollo de un sistema basado en satélites para ofrecer servicios básicos de telesalud y el acceso a una red de sensores distribuidos en todo el mundo que será desplegada paulatinamente antes y después de los lanzamientos de los satélites.

Los sensores se encargarán de adquirir datos del usuario y de transmitirlos a los satélites a través de una interfaz estándar de radio (SSI). Los usuarios serán capaces de definir sus propios sensores, para el monitoreo de diferentes tipos de parámetros, por ejemplo, contaminación en cuerpos de agua, la temperatura del agua, etcétera.

Para recuperar datos de los satélites, la red de estaciones terrestres GENSO será uno de los componentes principales del sistema de distribución de datos. Varias universidades de diferentes estados miembros de la AEE[9], Japón y EEUU están cooperando en este proyecto, cuya segunda versión (R2) se espera que proporcione las funcionalidades que requerirá el sistema HumSAT.

Una vez que los datos sean transportados por los satélites HumSAT, los usuarios autorizados podrán acceder a ella a través de una conexión a Internet. Se aplicará una serie de restricciones de seguridad para garantizar un acceso correcto a los datos recogidos.

La figura8 ofrece un panorama del sistema HumSAT.

[8] www.genso.org
[9] Agencia Espacial Europea

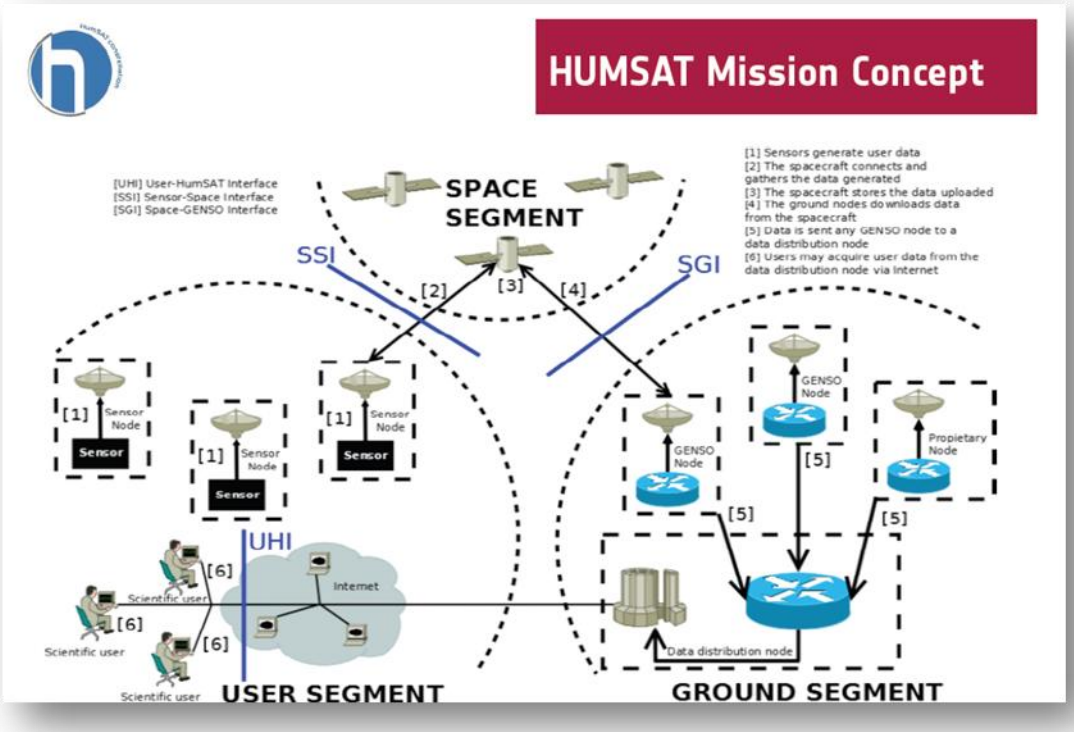


Figura 1. 8. Diagrama que representa la operación del sistema HumSAT.

1.4.2. De SATEDU al Proyecto HumSAT-México

El trabajo realizado en el Instituto de Ingeniería de la UNAM inicialmente en el proyecto SATEX y particularmente en SATEDU, ha permito sentar los precedentes para el diseño y desarrollo de un proyecto tan ambicioso como el satélite HumSAT-México.

Durante el desarrollo de los proyectos mencionados se ha contribuido en la formación de recursos humanos en diferentes áreas de la ingeniería, principalmente comunicaciones, mecatrónica, eléctrico-electrónica, mecánica y computación. Este trabajo se ha visto documentado en diversas tesis de licenciatura y posgrado, lo cual ha permitido sentar las bases para el trabajo de nuevos integrantes del equipo de trabajo del proyecto SATEDU y en particular del proyecto HumSAT-México.

Actualmente se cuenta con una segunda versión de la plataforma satelital SATEDU, la cual ha sido corregida de diversos errores, principalmente en el ruteo de los impresos, mientras se trabaja en un tercera versión en la cual se actualizan todos los subsistemas para que incorporen tecnologías más actuales como es el caso de un nuevo bus de comunicaciones principal bajo el protocolo I^2C o

SPI, una nueva tarjeta de comunicaciones inalámbrica vía Bluetooth (ya concluido), un sistema de estabilización basado en una arquitectura reconfigurable con tecnología FPGA, un nuevo sistema de comunicaciones basado en técnicas de "software radio" y una nueva versión de Computadora de Vuelo basada en un microprocesador más moderno, entre otras mejoras y el desarrollo de una carga útil de telemedicina.

Es importante destacar que los conocimientos y experiencia adquirida en el diseño de la plataforma satelital SATEDU, así como sus subsistemas satelitales permiten una rápida migración a una plataforma satelital real, como es el caso del proyecto HumSAT- México, por lo cual se considera que los tiempos de diseño y desarrollo se verán abatidos no sólo por apegarse a un estándar satelital pequeño, sino por la experiencia de los miembros del equipo que desarrolló SATEDU , con lo cual se tiene una base firme para acelerar cualquier proyecto satelital Mexicano.

Figura 1. 9. A la izquierda SATEDU ver 2, a la derecha SATEDU ver 1.

1.4.1. Planteamiento del problema y propuesta de solución

Los proyectos de desarrollo e innovación espacial son importantes para México porque buscan reducir la brecha tecnológica que se tiene respecto a otros países, de igual forma, pretenden sentar las bases para conformar un programa local de desarrollo de investigación, en particular en el campo satelital.

La presente tesis busca contribuir con la innovación y el desarrollo tecnológico, que pueda dar base a conocimiento y experiencias para ser trasmitidas al diseño de nuevos satélites con tecnología nacional para aplicaciones como la percepción remota y de comunicaciones, a la par de que son fuente de oportunidad para el entrenamiento, formación y certificación de recursos humanos.

La UNAM, a través del Instituto de Ingeniería, es pionero participante en el proyecto HumSAT-México, proyecto en el cual persigue el desarrollo y validación de un satélite pequeño de 3 Kg estabilizado en 3 ejes y con sistema de comunicaciones de bajada, directivo, de amplio ancho de banda. Además tendrá los sistemas satelitales convencionales de: potencia, estabilización, sensores, comunicaciones y Computadora de Vuelo.

La presente tesis queda delimitada a diseñar y validar el prototipo de subsistema de Computadora de Vuelo (CV) del satélite HumSAT-México y su Software Básico de Operaciones, con base en la plataforma satelital SATEDU. El objetivo principal de esta tesis es diseñar y validar un prototipo del subsistema de Computadora de Vuelo, basado en un procesador AT91SAM3U4E para actuar como elemento inteligente de control de un satélite de 3kgs, así como de la operación de su carga útil. Para lograr este objetivo principal, se plantearon los siguientes objetivos específicos:

- Determinar los requerimientos del prototipo de la Computadora de Vuelo.
- Evaluar las características de los microprocesadores de la familia ARM3 de 32 bits.
- Evaluar las herramientas de desarrollo para aplicaciones con microprocesadores ARM.
- Construcción del primer prototipo.
- Evaluar los diferentes sistemas de desarrollo en software para el desarrollo de la interfaz de usuario.
- Diseño de una tarjeta de circuito impreso para el prototipo de la tarjeta de la Computadora de Vuelo.

Con el trabajo de tesis se espera que el prototipo de Computadora de Vuelo propuesto contribuya a mejorar el rendimiento del satélite HumSAT-México, sobre todo al compararlo con el antiguo modelo de computadora empleada por el satélite SATEDU versión 2.

El método elegido para el desarrollo del proyecto ha sido la implementación de un plan Investigación-Acción. Álvarez[1] considera que la Investigación-Acción se compone de pasos seriados de acción que son: planificación, identificación de hechos, ejecución y análisis. McKernan

[2] propone un modelo de proceso para la Investigación-Acción el cual queda resumido en la figura 1.10.

Cada ciclo de acción inicia con la definición de la situación y el problema, para después evaluar las necesidades. Una vez que el plan de acción es puesto en práctica se evalúa y se toman las decisiones pertinentes para continuar con un segundo ciclo de acción.

Durante el segundo ciclo de acción se redefine el problema con base en los resultados obtenidos, con lo que se produce una redefinición del problema y una nueva evaluación de las necesidades a partir de las cuales surgen nuevas ideas o hipótesis que llevan a la revisión del plan de acción. La siguiente etapa consiste en poner en práctica el plan de acción revisado. Pueden agregarse cuantos ciclos sean necesarios según los resultados obtenidos.

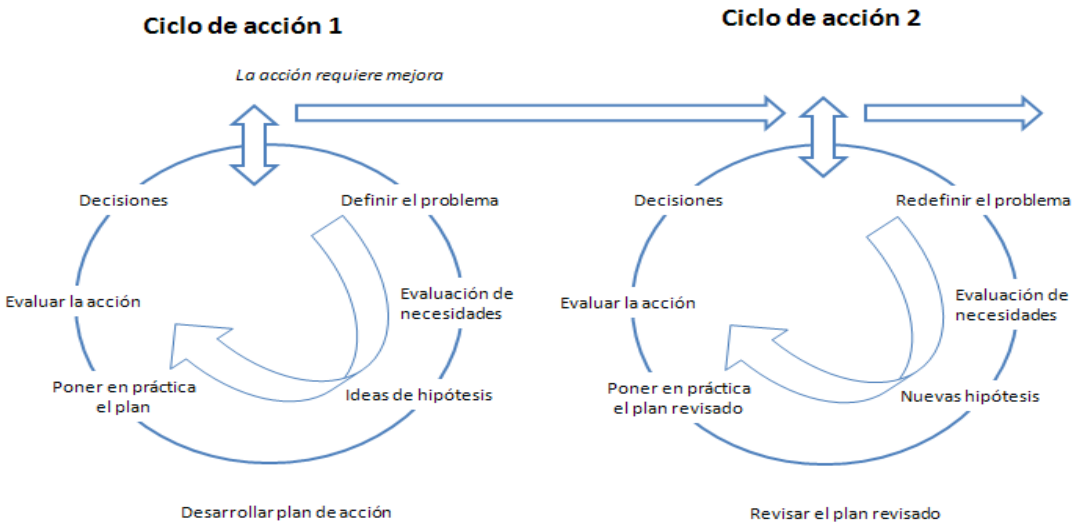


Figura 1. 10. Proceso temporal de la investigación-acción.

El proyecto de diseño y desarrollo de la Computadora de Vuelo fue concebido como 3 ciclos de investigación-acción para atender a tres líneas en el desarrollo del proyecto que contemplan el hardware, el firmware y el software de operaciones en Tierra. Cada ciclo está formado por 4 etapas que son: diseño, implementación, validación y evaluación.

Los resultados obtenidos a la salida de cada etapa son la entrada de la siguiente, por lo que después de realizar la medición y evaluación del sistema se realiza un rediseño del sistema. A este método de diseño, donde las etapas se repiten y auto-alimentan, se le conoce como diseño

iterativo. En la tabla 1 se muestra cada etapa, así como las descripciones de las tareas de cada una de ellas.

Tabla 1. Descripción de etapas y tareas

Etapa	**Descripción de Tareas**
Diseño	Hardware • Investigación y estudio de la información correspondiente a los diferentes modelos de microprocesador ARM3. • Determinación de requerimientos del prototipo. Firmware • Análisis de requerimientos del firmware. Software • Determinación de requerimientos de software. • Organización de tareas.
Implementación	Hardware • Construcción del primer prototipo. Firmware • Evaluación de las herramientas de desarrollo de bajo costo para aplicaciones con microprocesadores ATMEL SAM3 (ARM). Software • Evaluación de lenguajes de programación para el desarrollo de interfaces de usuario. • Construcción del primer prototipo.
Validación	Hardware • Pruebas de validación en SATEDU Firmware • Valoración de mejoras respecto a la versión anterior. Software • Generación de prototipos de interfaz de usuario. • Aplicación de pruebas.
Evaluación	• Generación de nuevas metas.

En el siguiente capítulo se describirán los requerimientos y características del prototipo de CV del satélite HumSAT-México.

Capítulo 2. Arquitectura de la Computadora de Vuelo del satélite HumSAT

El presente capítulo describe los requerimientos y características del subsistema de CV para el satélite HumSAT-México, así como de los módulos que lo forman.

2.1. Resumen del subsistema

El subsistema de Computadora de Vuelo (CV) es un elemento inteligente de control y administración del los recursos del satélite HumSAT-México y de su carga útil. Es en este subsistema donde se ejecuta el programa de la misión y opera tanto la activación como la desactivación de diferentes rutinas para los demás subsistemas.

2.2. Características de la Computadora de Vuelo

La Computadora de Vuelo para el satélite HumSAT-México posee una forma cuadrada de 8.9cm x 8.9cm con componentes por ambos lados.

El microcontrolador principal es un ATMEL AT91SAM3U4 el cual posee un núcleo ARM Cortex 3 con dos bancos de memoria Flash de 128 KB y 52 KB de memoria SRAM, además de diversos periféricos para comunicaciones.

La CV se encuentra integrada por diferentes módulos que permiten su interacción con los demás subsistemas de la plataforma satelital, como son procesador principal, memoria principal y secundaria, potencia auxiliar y bus principal de comunicaciones.

Además, dispone de dos conectores de costillas de 40 pines a cada lado para la interconexión con otros subsistemas. A través de estos conectores se envían los buses con señales de control, estado y tensión de alimentación. La figura 2.1 muestra el diagrama de bloques del subsistema de CV.

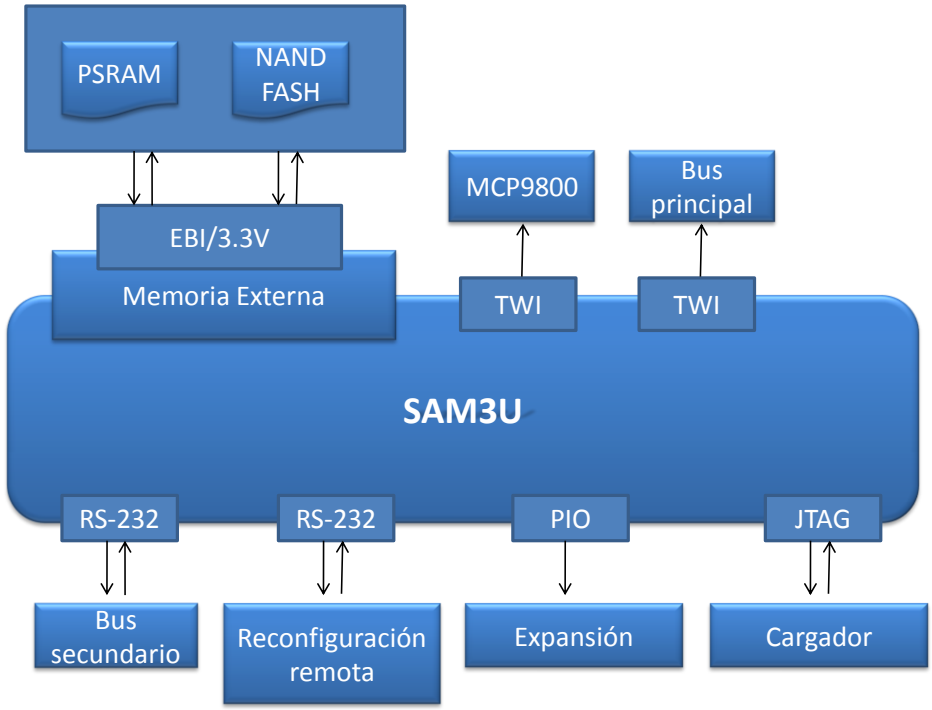


Figura 2.1 Diagrama de bloques de la CV del satélite HumSAT

2.2.1. Procesador

La serie SAM3U forma parte de la familia de microcontroladores con memoria Flash de Atmel basados en los procesadores RISC de 32 bits de alto rendimiento ARM Cortes-M3. En particular se seleccionó el modelo AT92SAM3U4 por ser compatible pin a pin con su antecesor, el SAM7, lo que permitirá una mejora a nivel de hardware e implementación del firmware y aprovechar las recomendaciones de diseño realizadas en proyectos anteriores en el Instituto de Ingeniería de la UNAM como lo es el proyecto SATEDU.

El AT91SAM3U4 opera a una frecuencia máxima de 96Mhz. Posee dos bancos de memoria Flash de 128KB y 52KB de memoria SRAM, además de 96 pines digitales para la entrada y salida repartidos en 3 puertos y multiplexados con otros recursos periféricos.

Entre el conjunto de recursos periféricos que incorpora pueden enumerarse dos convertidores analógico- digital de 10 y 12 bits con 8 canales para cada uno, 4 módulos USART, dos módulos TWI, una Interfaz de Bus Externo de 8 o 16 canales, para seleccionar hasta 4 chips con 24 bits de dirección, un RTC, entre otros.

La arquitectura SAM3U está especialmente diseñada para sostener transferencias de datos de alta velocidad, incluye una matriz de bus multicapa para manejar de manera simple bancos de memoria SRAM, PCD y canales DMA que posibilitan ejecutar tareas en paralelo y maximizar el rendimiento. Puede operar con niveles de 1.62V a 3.6V y posee un encapsulado LQFP de 144 pines. La tabla 2 resume las principales características del microcontrolador AT91SAM3U4.

Tabla 2. Características del microcontrolador AT91SAM3U4

Parámetro	Valor
Flash (Kbytes):	256
SRAM (Kbytes):	50
EEPROM (Bytes):	0
Memoria auto programable	Si
Interfaz de bus externa	1
DRAM	No
Interfaz NAND	Si
Parámetros de temperatura	-40 to 85°C
Alimentación (V)	1.62/3.6
Frecuencia máxima de operación (MHz)	96
Número de pines	144
Terminales de entrada y salida (I/O Pins)	96
Fuentes de interrupción externa	96
Transceptor USB	1
Velocidad USB	Hi-Speed
Interfaz USB	Device
SPI:	5
TWI:	2
UART:	5
SD / eMMC:	1
Canales ADC	16
Resolución ADC	10-bit, 12-bit
Velocidad ADC	1000, 384
Timers	3

PWM Channels	4
RTC 32kHz	Si
Oscilador RC Calibrado	Si
Tipo de encapsulado	LQFP 144

La figura 2.2 muestra el diagrama a bloques de la arquitectura del microcontrolador AT91SAM3U4.

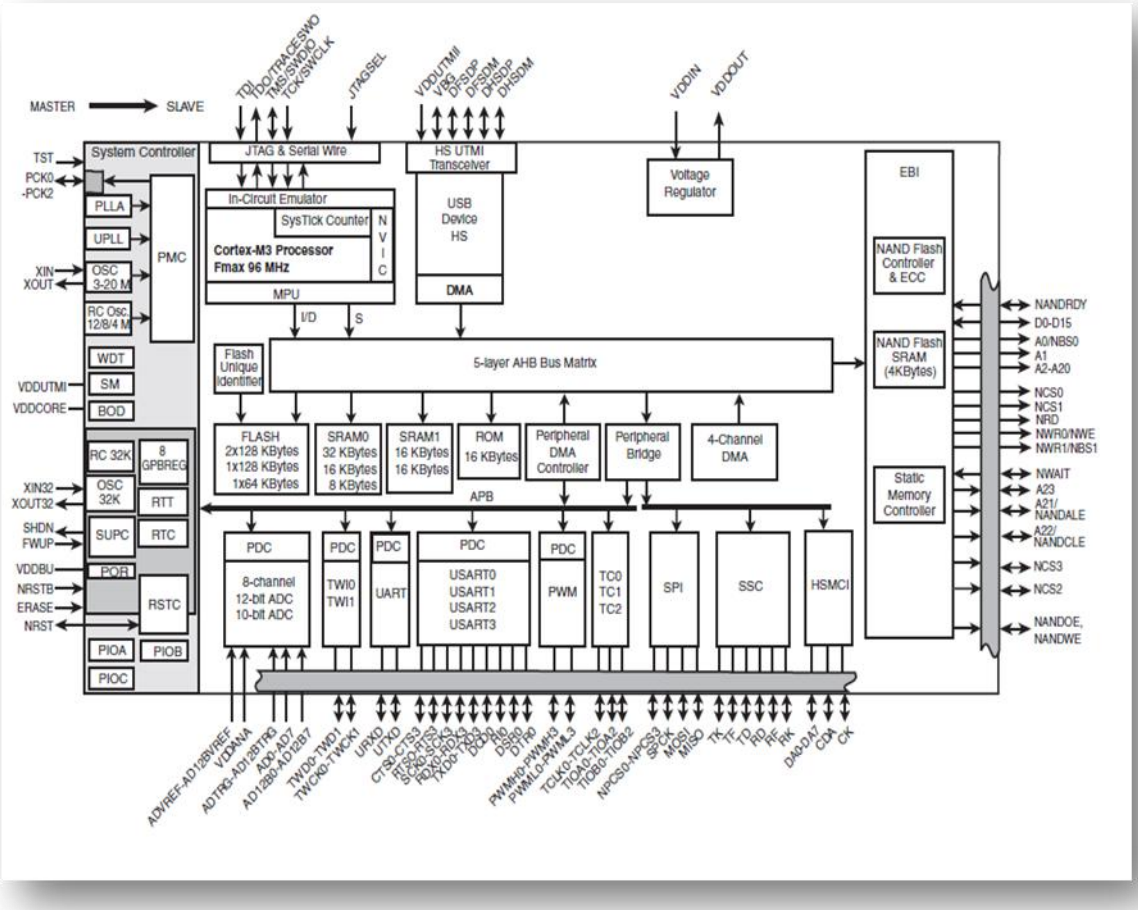


Figura 2.2 Diagrama de bloques del microcontrolador AT91SAM3U4

El AT91SAM3U4 posee diferentes tipos de líneas de entrada y salida, como son las entradas y salidas de propósito general (GPIO) , así como las entradas y salidas del sistema. Las GPIO pueden tener más de una función ya que se encuentran multiplexadas a otros recursos periféricos. Con pocas excepciones, las entradas y salidas tienen entradas *Schmitt Trigger*.

Las líneas GPIO son manejadas por los controladores PIO. Todas las entradas y salidas tienen numerosas modos de entrada o salida como pueden ser *pull-up*, entradas *Schmitt Trigger*, *multi-drive* (*open-drain*), *glitch filter*. Es posible operar línea a línea de manera independiente.

El núcleo del procesador está basado en un procesador ARM Cotex-M3 con las siguientes características:

- Versión 2.0
- *Thumb*-2 (ISA), con juego de instrucciones de 16 y 32 bits.
- Procesador de arquitectura Harvard, lo que permite la búsqueda de una instrucción y la carga o almacenamiento de datos de manera simultánea.
- Pipeline de 3 etapas.
- Multiplicador de 32 bits en un ciclo simple.
- Hardware modular
- Estados de depuración y *Thumb*
- Modos *Handler* y *Thread*.
- Entrada y salida de baja latencia para ISR[10]

Los microcontroladores SAM3U incorporan dos puentes separados APB/AHB de alta y baja velocidad. Esta arquitectura permite hacer accesos concurrentes en ambos puentes. Todos los periféricos se encuentran conectados al puente de baja velocidad con excepción de SPI[11], SSC y HSMCI. El UART, el ADC[12], TWI[13] 0-1, USART[14] 0-3, PWM[15] tiene canales dedicados pero pueden usar el DMA con FIFO interno.

[10] *Interruption Service Routine*
[11] *Serial Peripheral Interface*
[12] *Analog to Digital Converter*
[13] *Two-Wire Interface*. Equivale al protocolo I^2C.

2.2.2. Memoria

El procesador SAM3U4E poseen 256KB de memoria Flash, 48 KB de SRAM[16] en dos bancos, 16 KB de ROM[17] para rutinas de *bootloade*[18]*r* a través de la UART o USB[19] y rutinas API[20].

El SAM3U4E posee un Bus de Interfaz Externo (EBI)[21] que permite interconectar memorias externas y virtualmente cualquier periférico con comunicación en paralelo. En este caso se agregan un integrado de NAND Flash MT29F2G16ABD y otro más de PSRAM.

Las señales de selección de chip NCS0 y NCS1 son usadas para activar la PSRAM y la NAND Flash respectivamente.

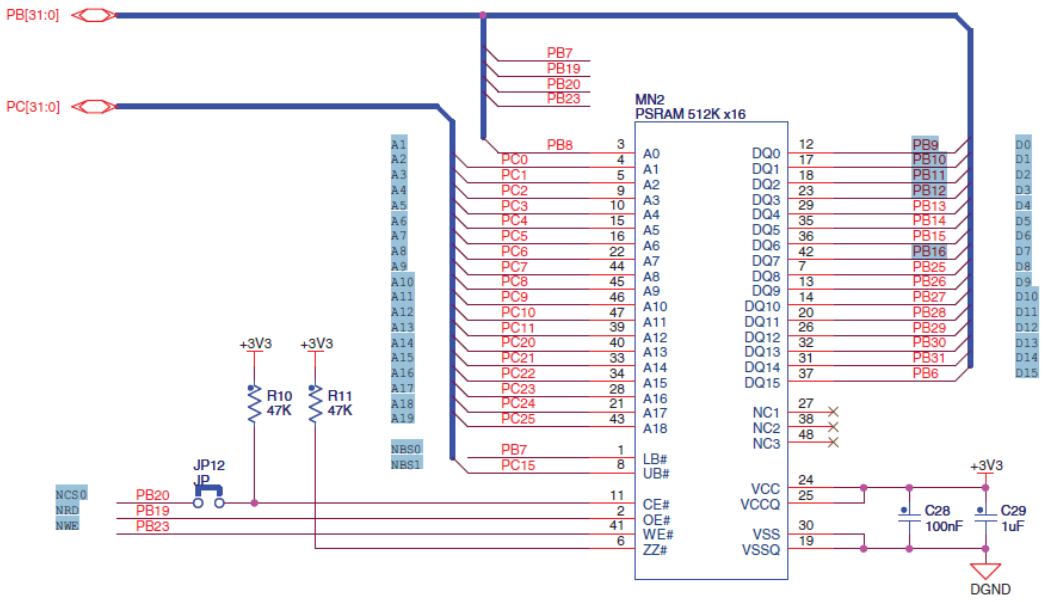


Figura 2.3 Esquemático de conexión de la memoria PSRAM

[14] *Universal Receiver-Transmitter Synchronous -Asynchronous*
[15] *Pulse-Width Modulation*
[16] *Static Random Access Memory*
[17] *Read-Only Memory*
[18] También llamado cargador de arranque
[19] *Universal Serial Bus*
[20] *In-Application Programming*
[21] *External Bus Interface*

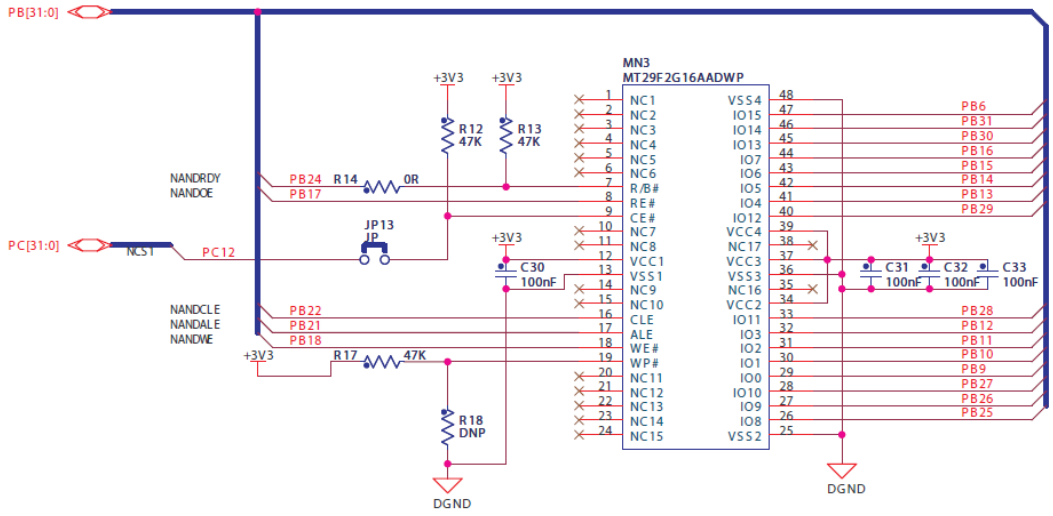


Figura 2.4. Esquemático de conexión de la memoria NAND Flash externa

2.2.3. Circuito de reloj

El circuito para el oscilador y generación de la señal de reloj está definido por:

- Un cristal de baja velocidad y bajo consumo en modo de bypass.
- Un cristal de 12 MHz, seleccionado para poder usar el módulo USB opcionalmente.
- Un oscilador interno RC de alta velocidad ajustable a una de tres posibles frecuencias: 4, 8 o 12 MHz.
- Un módulo UTM PLL que provee la señal de reloj de 480 MHz usada por el controlador USB de alta velocidad, si este es requerido.
- Un PLL programable de 96 a 196MHz con entradas de 8 a 16 MHz, capaz de proveer la señal principal de reloj (MCK) al procesador y a los circuitos periféricos.

El cristal de 32.768KHz puede ser usado como fuente de reloj del RTC, mientras que el oscilador interno RC de alta velocidad o el oscilador externo de 12MHz proporcionan la base de tiempo al procesador y al resto de periféricos.

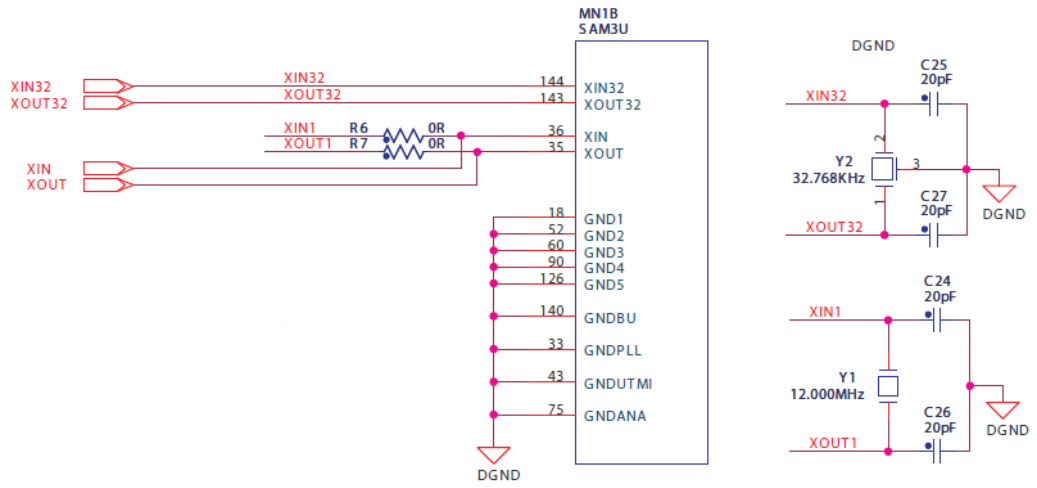


Figura 2.5. Esquemático de la señal de reloj para los osciladores de 32.768KHz y 12MHz.

2.2.4. Circuito de reinicio (Reset) y arranque (wake-up)

Para proveer a la tarjeta de un mecanismo de reinicio manual en caso de fallas durante los periodos de depuración y pruebas que se realicen en Tierra, se han dispuesto de 3 botones en las terminales NRST, NRSTB y FWUP.

El pin NRST es bidireccional y es manejado por controlador de reinicio integrado *(on-chip reset controller*). Cuando es puesto en nivel lógico bajo, genera una señal de reinicio para el microcontrolador, es decir el estado de su núcleo y periféricos, sin modificar la región de *Backup* (RTC[22], RTT[23] y controlador de alimentación). Este pin está conectado al puerto JTAG[24] e integra una resistencia interna *pull-up* de 100KΩ conectada permanentemente a VDDIO.

El pin NRSTN es una terminal de sólo entrada que habilita el reinicio asíncrono cuando es puesto a nivel lógico bajo. El pin NRSTB integra una resistencia *pull-up* permanente de 15KΩ. Esta entrada permite al usuario reiniciar el dispositivo incluyendo la región de *Backup*, con lo cual se logra un estado similar al que se genera cuando se realiza un reinicio por encendido (*Power-on Reset*). Un capacitor de 10nF conectado entre NRSTB y VDDIO permite una estabilidad en la señal que se genera al pulsar el botón.

[22] *Real Time Clock*
[23] *Real Time Timer*
[24] *Join Test Action Group*

El pin FWUP fuerza una entrada activa en bajo para el modo Wake-Up. Necesita de una resistencia *pull-up* externa de 100KΩ para funcionar. Si el pin FWUP es puesto en alto por un periodo mayor que el configurado para la eliminación de rebotes (100µS, 1mS, 16mS, 128mS o 1S) la fuente de poder del núcleo del despertador (wake-up) es inicializada.

2.2.5. Módulo de cargado de firmware

Para realizar la programación por primera vez del microcontrolador se hará uso de un programador SAM-ICE de Atmel. Este emulador JATG para ARM posee un conector de 20 pines con lo cual es necesario usar un adaptador para reducir el número de terminales que se usan a sólo 4 señales básicas del estándar JTAG más la referencia a Tierra y señal de reinicio NRST.

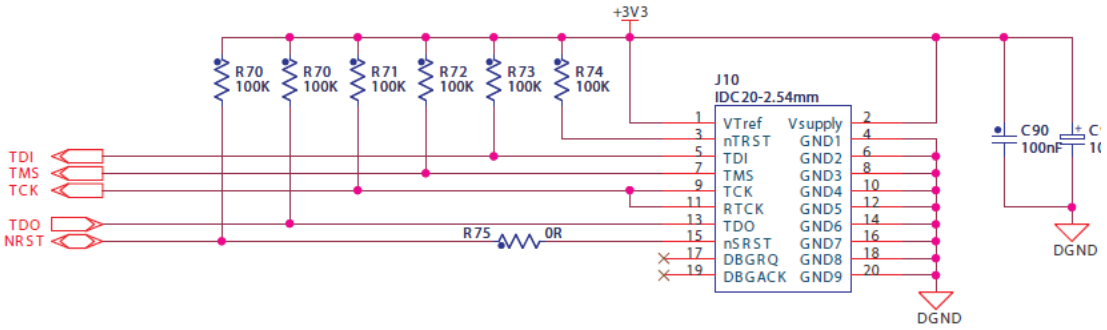


Figura 2.6. Esquemático del adaptador para el emulador SAM-ICE

Para realizar la reconfiguración parcial del sistema o subir un nuevo programa de misión se hace a través de una interfaz gráfica de usuario diseñada especialmente para tal propósito, a través de uno de los módulos UART que se encuentran multiplexados entre un microcontrolador PIC182550 y un circuito MAX3232.

Inicialmente, la memoria ROM del SAM3U4E debe ser configurada con un *bootloader*, lo que permitirá la carga remota de nuevas versiones de firmware. El MAX3232 es usado para realizar pruebas y depuración del *firmware* y del *bootloader* con una PC. Una vez que una versión estable ha sido obtenido, está es embebida en el microcontrolador PIC para que se encargue del proceso de configuración remota donde el proceso será como se describe a continuación.

El microcontrolador PIC controla un banco de memorias EEPROM serie en el cual se almacena temporalmente el nuevo *firmware* a cargar en el SAM3U4E.

Este programa es enviado desde la estación terrena a través del Subsistema de Comunicaciones, pasando por la Computadora de Vuelo y entregándolo finalmente al módulo de cargado, para que el *firmware* sea almacenado inicialmente en la memoria EEPROM serie.

Posteriormente se inicia una rutina de cargado que modificará el contenido de la memoria FLASH a través de iniciar al SAM3U4E en modo de *bootloader*. El módulo de cargado de nuevo *firmware* también incorpora la capacidad de reprogramar otros subsistemas de la plataforma satelital que hagan uso del estándar JTAG como puede ser la nueva versión del Subsistema de Estabilización que incorpora un FPGA como elemento de procesamiento.

El cargado de nuevo programa de misión se hace mediante el cargado de un *script* a la memoria FLASH de la CV, el cual es independiente del *firmware* por lo cual no es necesario la intervención del módulo de cargado. La figura 2.7 muestra el diagrama de bloques del módulo de reconfiguración remota.

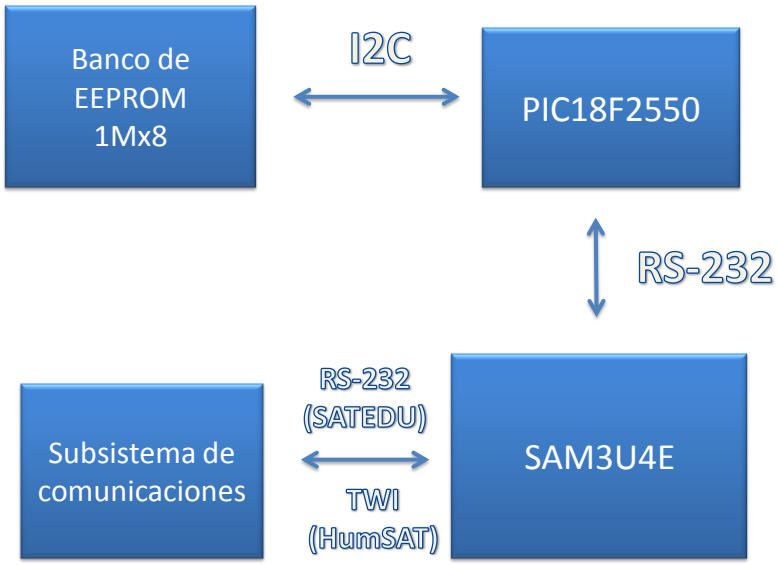


Figura 2.7. Diagrama de bloques del módulo de reconfiguración remota

2.2.6. Sensor de temperatura

A través de uno de los dos buses TWI se accede a un sensor de temperatura MCP9800. Este dispositivo cuenta con un pin de salida en colector abierto llamado ALERTA. El MCP9800 emite una señal de alarma cuando la temperatura ambiente va más allá del límite de temperatura programado por el usuario.

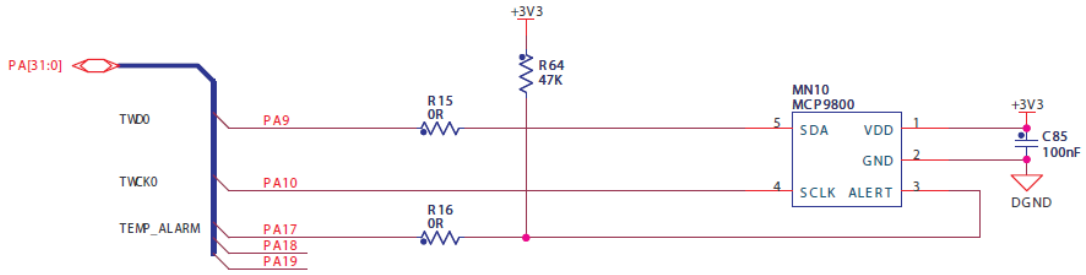


Figura 2.8. Esquemático del sensor de temperatura

2.2.7. Indicadores

Para realizar pruebas de y procesos de depuración es necesario contar con indicadores luminosos conectados a terminales de propósito general. Estos elementos pueden ser omitidos en un diseño final, pero son de gran importancia durante la fase de pruebas, un ejemplo es el indicador de encendido que muestra que la CV se encuentra en funcionamiento.

Se dispone de tres diodos led como indicadores luminosos, de los cuales dos son controlados por pines PB0 y PB1, mientras que el tercer led indica que la fuente de alimentación se encuentra habilitada, es decir, muestra una señal de encendido. Esta señal puede ser controlada por el pin PB2, pero de manera predeterminada es controlada por una resistencia pull-up de 100KΩ conectada a la fuente de 3.3V debido a que de manera predeterminada el GPIO se encuentra deshabilitado.

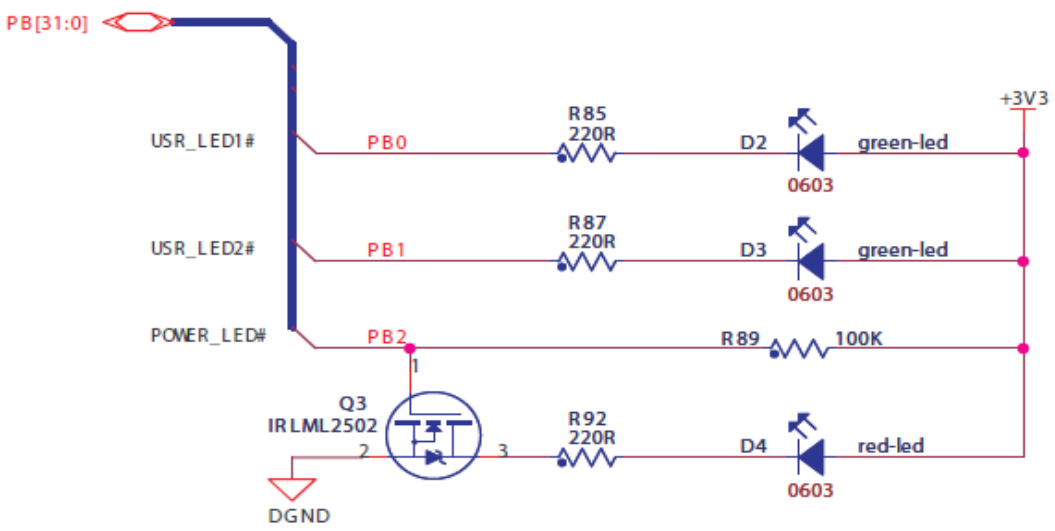


Figura 2.9. Esquemático de indicadores de usuario y encendido.

2.2.8. Alimentación eléctrica

Para mantener una compatibilidad con la plataforma satelital SATEDU en la cual se realizaron las pruebas de validación, se hizo uso de las terminales de alimentación que el Subsistema de Potencia provee. Como medidas de protección se empleó un diodo PolyZen ZEN056V130A24LS y un filtro LCBNX002-01.

Un regulador ajustable MIC29152WU es usado para obtener la fuente principal de alimentación de 3.3V. Las señales VDDUTMI, VDDANA, VDDIO, VDDIN están conectadas directamente a la salida del regulador de 3.3V, sin embargo, el núcleo operará con una tensión de 1.8V que se obtiene del regulador interno, por lo cual deben conectarse los pines VDDCORE y VDDPLL a la salida del regulador interno.

El pin VDDBU es alimentado por la fuente principal y por una batería de respaldo de 3.3V a través de dos diodos Schottky.

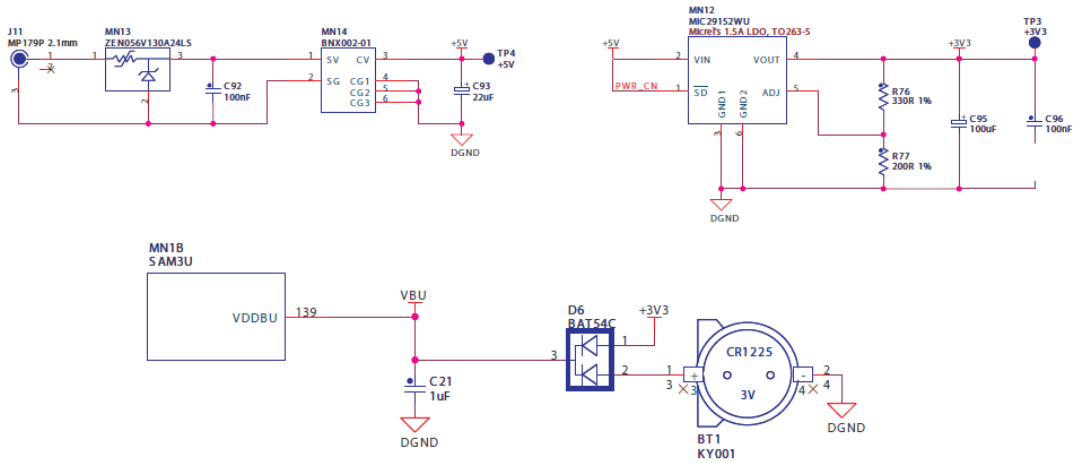


Figura 2.10. Arriba a la izquierda. Circuito de protección. Arriba a la derecha etapa de regulador. Abajo circuito de respaldo.

Capítulo 3. Software básico de operaciones de la Computadora de Vuelo

En este capítulo se presentan las características del firmware para el microcontrolador SAM3U4E así como una breve descripción de las herramientas usadas para su implementación

3.1. Análisis de requerimientos del firmware básico de operaciones de CV

La Computadora de Vuelo es el subsistema encargado de ejecutar el programa de misión. Entre sus tareas pueden enumerarse la recepción de comandos desde la Estación Terrena, el envío de telemetría y la comunicación por medio de comandos con el resto de los subsistemas de la plataforma satelital.

Al tratarse de una aplicación que requiere hacer un uso eficiente de los recursos de la plataforma, el diseño del firmware pretende ahorrar al máximo los recursos como la carga de las baterías, el control de la temperatura y el consumo de corriente, entre otros.

El sistema trabaja mediante interrupciones para atender los eventos que suceden en la plataforma satelital de la manera más rápida posible, por lo que, cuando no se está atendiendo a ningún evento, el microprocesador entra en modo de bajo consumo o *idle*.

Para realizar la validación tanto del firmware como de la arquitectura de la CV se empleó la plataforma para entrenamiento de recursos humanos SATEDU. Esto implica que es necesario guardar compatibilidad no solo a nivel de hardware sino también de firmware con la plataforma SATEDU.

Una mejora que se prepone a la CV del satélite HumSAT comparada con la versión actual de la plataforma SATEDU es un bus principal de comunicaciones basado en un bus TWI (I2C). Este bus fue valido independientemente, ya que el bus principal de la plataforma SATEDU se compone de un bus bajo el estándar RS-232 multiplexado a cada tarjeta.

Se hace uso de un tercer puerto UART para la comunicación con el modulo de reconfiguración. El *firmware* del microcontrolador SAM3U4E realizar tareas de *bypass* cuando llega un comando para el sistema de reconfiguración, lo mismo cuando recibe un nuevo *firmware*. Finalmente da la instrucción de iniciar el proceso de reconfiguración, lo que provoca un reinicio por hardware para arrancar el bootloader.

El firmware debe incorporar comandos para la lectura del sensor de temperatura, el RTC embebido en el microcontrolador y para la comunicación con el resto de los subsistemas.

3.2. Diseño del software básico de operaciones de CV

3.2.1 Herramientas de desarrollo para Microcontroladores AT91SAM

Muchas de las soluciones de desarrollo usadas para microcontroladores basados en núcleos de ARM son paquetes comerciales como IAR EWARM o ARM *RealView*. Sin embargo, recientemente soluciones de código abierto como GNU *toolchain* se han convertido en sistemas más competitivos y paquetes de más fácil uso para el usuario final.

El objetivo de esta sección es presentar el panorama de las herramientas necesarias para el diseño y desarrollo de firmware básico de operaciones de la Computadora de Vuelo del satélite HumSAT-México. De manera general se describirán las características de:

- Compilación y construcción de proyectos usando el compilador GNU *toolchain*.
- Depuración usando SAM-ICE *emulator* y SEGGER GDB-*server*.
- Programación de la memoria usando SAM-BA.
- Integración de estas herramientas en un ambiente de desarrollo integrado de código abierto como lo es Eclipse.

YAGARTO

YAGARTO es el acrónimo de *Yet another GNU ARM toolchain*. Es un proyecto que busca desarrollar herramientas bajo la licencia GPL/GNU para el desarrollo de aplicaciones con procesadores ARM. Una de las ventajas de usar YAGARTO es que es posible utilizar otras herramientas y aplicaciones también de código abierto como lo es el compilador GCC y el ambiente de desarrollo integrado Eclipse. YAGARTO está dividido en tres paquetes:

1. Una interfaz de depuración JTAG como J-Link *GDB Server* o *Open On-Chip Debugger*.
2. Binutils, Newlib, compilador GCC y GDB debugger.
3. Compatibilidad con Eclipse CDT.

Actualmente la versión más actual de YAGARTO es la 2.0.[25]

SAM-ICE

Atmel SAM-ICE es un emulador JTAG diseñado para microcontroladores AT91SAM ARM7/ARM9/Cortex-M3. Soporta velocidades de descarga de hasta 720KB por segundo y una velocidad máxima de JTAG de hasta 12MHz. Soporta *Serial Wire Debug* (SWD) y *Serial Wire Viewer* (SWV) procesos de depuración en circuito.

No necesita alimentación eléctrica externa debido a que se conecta al puerto USB de una PC. Posee un cable plano con un conector estándar de 20 hilos para su conexión a bancos de pruebas.

Figura 3.1. Emulador JTAG SAM-ICE de Atmel

[25] http://www.yagarto.de/

SAM-BA Bootloader

Atmel provee un *bootloader* llamado SAM-BA. Este bootloader puede cargar programas desde diferentes periféricos como lo son la UART o el puerto USB.

El SAM-BA *Bootloader* puede interactuar en dos modos. En el modo interactivo, el *bootloader* responde con un símbolo de sistema que consiste en el carácter '>' después de cada comando. En el modo no interactivo, no envía más datos que los devueltos por el comando enviado de manera individual. Cada comando enviado al *Bootloader* debe terminar con los caracteres ASCII # (0x23) y LF (0x0A). Una línea recibida como respuesta siempre termina con CR+LF (0x0D 0x0A).

El primer argumento del comando es la dirección, mientras que el segundo es el valor de la palabra, media palabra u octeto a ser escrito, así como la longitud de los datos que se escriben o leen. La tabla 3 resume los comandos individuales del SAM-BA Bootloader.

Tabla 3. Comandos del SAM-BA Bootloader

Comando	Significado	Respuesta (Interactivo)	Respuesta (no interactivo)
T	cambiar al modo interactivo	CRLF CRLF + + ayuda	CRLF + ayuda
N	cambiar a modo no interactivo	CRLF, ningún mensaje	nada
V	obtener la cadena de versión (por ejemplo, "v1.4 10 de noviembre 2004 14:49:33")	versión de la cadena + CRLF + ayuda	versión de la cadena + CRLF
W <8 dígitos hexadecimales>, <8 dígitos hexadecimales>	escribir la palabra	CRLF + ayuda	nada
w <8 dígitos hexadecimales>, 4	leer la palabra	número hexadecimal (como "0x1234ABCD") + CRLF + ayuda	4 octetos
H <8 dígitos hexadecimales>, <4 dígitos hexadecimales>	escribir media palabra	CRLF + ayuda	nada
h <8 dígitos hexadecimales>, 2	leer media palabra	número hexadecimal (como "0x12AB") + CRLF + ayuda	2 octetos

O <8 dígitos hexadecimales>, <2 dígitos hexadecimales>	escribir octeto	CRLF + ayuda	nada
o <8 dígitos hexadecimales>, 1	leer octeto	número hexadecimal (como "0x1A") + CRLF + ayuda	1 octeto
S <8 dígitos hexadecimales>, <8 dígitos hexadecimales>	escribir un grupo de datos. Los datos que se escriben deben seguir el terminador del comando.	CRLF + ayuda	nada
R <8 dígitos hexadecimales>, <8 dígitos hexadecimales>	leer un grupo de datos.	CRLF + como de octetos como se especifica el segundo parámetro + ayuda	como de octetos como se especifica el segundo parámetro
G <8 dígitos hexadecimales>	Ejecutar código en la dirección dada	CRLF (antes del salto a la dirección indicada), sin preguntar	nada

Los comandos w/h/o no se pueden utilizar para leer varias palabras, medias palabras u octetos en un solo comando. En el modo interactivo, el número hexadecimal devuelto por los comandos w/h/o se deriva de los datos en la dirección indicada en *Little-endian*.

La secuencia de inicialización utilizada por la versión 2.8 del programa Atmel SAM-BA es enviar primero un comando T y esperar a que se le solicite un argumento. Inmediatamente comienza el envío de un comando N seguido de los distintos comandos de lectura o escritura.

Eclipse

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma. Emplea módulos, en inglés llamados *plug-ins*, para proporcionar su funcionalidad a diferencia de los entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el desarrollador o no. Eclipse permite extender su funcionalidad a diferentes lenguajes como C/C++ y Python- Estas características permiten integrarlo a las herramientas GNU-toolchain para ARM provistas con YAGARTO.

Figura 3.2. Eclipse es un entorno de desarrollo integrado de código abierto

3.2.2. Secuencia de inicialización en ensamblador para aplicaciones en código C

Por razones de modularidad y portabilidad muchos códigos de aplicación para los microcontroladores basados en la familia AT91 ARM son escritos en C. Sin embargo, la secuencia de arranque requerida para inicializar el procesador ARM y algunos periféricos clave es altamente dependiente de la arquitectura de registros y del mapa de memoria del microprocesador.

Por esta razón la secuencia de arranque para aplicaciones en C es escrita en ensamblador. Esta secuencia de arranque es activada durante el encendido y después de un reinicio.

Secuencia de arranque en C

Una consideración importante en el diseño de una aplicación embebida ARM es el diseño del mapa memoria, en particular la memoria que se encuentra en la dirección 0x00. Después de ser reiniciado, el procesador comienza a buscar instrucciones desde la dirección 0x00, por lo tanto, debe existir código ejecutable desde esa dirección colocado, por lo menos inicialmente, en una memoria no volátil o NVM (*Non Volatile Memory*).

El diseño más simple se lleva a cabo mediante direccionamiento de la memoria ROM desde la dirección 0 en el mapa de memoria. La aplicación puede entonces brincar al punto de entrada real cuando se ejecuta su primera instrucción en el vector de reinicio (*reset vector*) en la dirección 0x0.

Sin embargo, existen desventajas con este diseño. La ROM suele ser estrecha (8 o 16 bits) y lenta comparada con la RAM, por lo que se requieren más estados de espera para acceder a ella. Esto hace más lento el manejo de las excepciones del microprocesador, especialmente las interrupciones, a través de la tabla de vectores. Por otro parte, si la tabla de vectores se encuentra en ROM, no puede ser modificada por el código.

Dado que la memoria RAM es normalmente más rápida y amplia que la ROM, es mejor si la dirección 0x0 está en RAM para la tabla de vectores y manejadores de interrupciones. Aunque es necesario que la memoria RAM se localice en la dirección 0x00 durante la ejecución normal, si se encuentra en la dirección 0x00 cuando se encienda, en ella no existirá una instrucción válida en el vector de inicio. Por lo tanto la ROM debe ubicarse en la dirección 0x00 en el momento del encendido para asegurar que existe un vector de reinicio válido. El reinicio del mapa de memoria normal se realiza mediante la reasignación de comandos.

Muchas aplicaciones escritas para sistemas basados en ARM son aplicaciones embebidas que son almacenadas en ROM y ejecutadas después de un reinicio. Hay una serie de factores que deben ser considerados al escribir aplicaciones que se ejecutan con o sin sistemas operativos embebidos, estas son:

- Reasignación de ROM a RAM para mejorar la velocidad de ejecución.
- Inicializar el entorno de ejecución, tales como vectores de excepción, pilas, puertos de estrada-salida.
- Inicialización de la aplicación. Por ejemplo, copiar los valores de inicialización de las variables inicializadas a partir de ROM y restablecer todas las demás variables a cero.
- Vinculación de una imagen ejecutable embebida a otro código y datos ubicados en localidades especificas de la memoria.

Para una aplicación embebida sin sistema operativo, el código en ROM debe proveer una forma a la aplicación de inicializarse y comenzar su ejecución. Una inicialización no automática se lleva a cabo al reiniciar, por lo tanto, el punto de entrada de la aplicación debe realizar algún tipo de inicialización antes de que pueda llamar a cualquier código en C.

El código de inicialización, que se encuentra en la dirección cero después del reinicio debe:

- Marcar el punto de entrada para el código de inicialización.
- Establecer los vectores de excepción.
- Inicializar el sistema de memoria.
- Inicializar los registros del puntero de pila.
- Inicializar cualquier dispositivo crítico de entrada y salida.
- Inicializar las variables de memoria RAM requeridas por el sistema de interrupciones.
- Habilitar las interrupciones (si se manejan por el código de inicialización).
- Cambiar el modo del procesador si es necesario.
- Cambiar el estado del procesador si es necesario.

Después de que el proceso de inicialización se ha completado, la secuencia continúa con la aplicación y debe introducir el código en C.

Área de definición y punto de entrada para el código de inicialización

En un archivo fuente en lenguaje ensamblador ARM, la sección de inicio es marcada por la directiva AREA. Esta directiva nombra la sección y establece sus atributos. Los atributos se colocan después del nombre, separados por comas. El ejemplo de código mencionado anteriormente define una sección de código de sólo lectura denominada *reset*.

Una imagen ejecutable debe tener un punto de entrada, una imagen embebida que se puede colocar en la memoria ROM suele tener un punto de entrada en la dirección 0x00. Un punto de entrada se puede definir en el código de inicialización mediante la directiva de ensamblador ENTRY.

```
;-------------------------------------------------------------------------
;- Area Definition
;-------------------------------------------------------------------------
AREA reset, CODE, READONLY
;-------------------------------------------------------------------------
;- Define the entry point
;-------------------------------------------------------------------------
ENTRY
```

Configuración de vectores de excepción

Los vectores de excepción se configuran de forma secuencial a través del espacio de direcciones con enlaces a subrutinas. Durante el flujo normal de ejecución de un programa, los incrementos del contador de programa permiten al procesador manejar eventos generados por fuentes internas o externas.

Las excepciones del procesador ocurren cuando el flujo normal de ejecución se desvía. Ejemplo de estos sucesos son:

- Alarmas generadas externamente.
- Un intento del microprocesador por ejecutar una instrucción indefinida.

Es necesario preservar el estado previo del procesador al manejar estas excepciones, por lo que la ejecución del programa que se estaba ejecutando cuando se produjo la excepción pueda reanudarse cuando la rutina de excepción apropiada se ha completado.

El código de inicialización debe configurar los vectores de excepción específica. Si la ROM se encuentra en la dirección 0, los vectores consisten en una secuencia de instrucciones codificadas en rama al controlador para cada excepción.

Estos vectores se asignan en la dirección 0 antes de volverse a asignar. Deben estar en modo de direccionamiento relativo a fin de garantizar un salto válido. Después de ser mapeados nuevamente, estos vectores se asignan en la dirección 0x01000000 y sólo pueden cambiar de nuevo por un reinicio interno o NRST.

Tabla 4. Vectores de excepción

Excepción	**Descripción**
Reinicio (*Reset)*	Se produce cuando el pin de reinicio del procesador se afirma. Esta excepción sólo se espera que se produzca para la señalización de encendido o para restablecer como si el procesador fuera encendido. Un reinicio por software se puede hacer al brincar al vector de reinicio (0x0000).
Instrucción indefinida (*Undefined Instruction*)	Se produce cuando ni el procesador o cualquier coprocesador reconocen la instrucción que se está ejecutando.
Software Interrup (SWI)	Esto es una instrucción de interrupción síncrona definida por el usuario. Permite a un programa que se ejecuta en modo usuario, por ejemplo, solicitar operaciones privilegiadas que se ejecutan en modo supervisor, como lo es una función de RTOS.
Prefecht Abort	Ocurre cuando el procesador intenta ejecutar una instrucción que ha sido buscada desde una dirección ilegal.
Data Abort.	Se produce cuando una instrucción de transferencia de datos intenta cargar o almacenar datos en una dirección ilegal.
IRQ	Se produce cuando el pin de solicitud de interrupción externa del procesador es activado (en bajo) y el bit F en el registro CPSR es puesto a cero.
FIQ	Se produce cuando el pin de solicitud de interrupción externa rápida del procesador es activado (en bajo) y el bit F del registro CPSR es limpiado.

El manejo de excepciones del procesador es controlado por una tabla de vectores. La tabla de vectores es un área reservada de 32 bytes, por lo general en la parte inferior del mapa de memoria. Tiene una sola palabra de espacio asignado a cada tipo de excepción y una palabra que está reservada debido a que no hay espacio suficiente para contener el código completo de una rutina de servicio, el vector de entrada para cada tipo de excepción contiene una instrucción de

salto o carga al contador de programa (PC) para continuar la ejecución con la rutina de servicio adecuada.

```
;-------------------------------------------------------------------------
;- Exception vectors ( before Remap )
;-------------------------------------------------------------------------
B InitReset ; reset
undefvec
B undefvec ; Undefined Instruction
swivec
B swivec ; Software Interrupt
pabtvec
B pabtvec ; Prefetch Abort
dabtvec
B dabtvec ; Data Abort
rsvdvec
B rsvdvec ; reserved
irqvec
B irqvec ; reserved
fiqvec
B fiqvec ; reserved
```

3.3. Actualización del firmware

A fin de extender las prestaciones que un *firmware* puede prestar a un sistema embebido o ya sea para reparar errores de programación, la actualización de *firmware* en campo es la técnica más usada hoy en día para realizar una reconfiguración en software de un sistema embebido.

El concepto de reconfiguración en campo es muy simple:

1. El fabricante diseña un dispositivo con una versión inicial de *firmware*
2. El dispositivo es vendido al cliente
3. El fabricante desarrolla una nueva versión de *firmware*
4. La nueva versión es distribuida al cliente
5. El cliente actualiza su dispositivo con la nueva versión del *firmware*

Si no se toman ciertas medidas, este proceso puede resultar muy peligroso y dejar el dispositivo inservible. Algunas situaciones comunes que se generan durante el proceso de reconfiguración y que pueden generar un error son la perdida de energía o de la conexión durante la transmisión del

nuevo *firmware*. Este problema es llamado problema de seguridad del dispositivo *(safety of the device*) y queda resumido en el siguiente ejemplo.

1. El fabricante diseña un dispositivo con un *firmware* inicial.
2. El dispositivo es vendido al cliente.
3. El fabricante desarrolla una nueva versión del firmware.
4. La nueva versión del *firmware* es distribuida al usuario.
5. El cliente actualiza su dispositivo con la nueva versión del *firmware*.
6. Ocurre un error durante la transmisión, lo que provoca que el dispositivo no pueda funcionar.

La seguridad suele ser otro problema, donde la integridad del *firmware* es importante por lo cual puede agregarse código de detección de errores pero también de codificación para que no pueda ser usado por un competidor. Este problema queda resumido como sigue:

1. El fabricante libera una nueva versión del *firmware* para su dispositivo
2. El competidor obtiene la nueva versión de *firmware*
3. Se realiza ingeniería inversa para obtener el código original

3.3.1. Actualización en campo

Desarrollar un *firmware* para un microcontrolador es una tarea relativamente fácil cuando se cuenta con las herramientas necesarias y un banco de pruebas. Entre el conjunto de estas herramientas pueden enumerarse puertos de depuración como JTAG, dispositivos programadores, depuradores en circuito, entre otros. Sin embargo, estas herramientas no suelen ser usadas para actualizar el *firmware* de un dispositivo por el usuario final, ya que pueden presentarse varios escenarios bastante obvios que incluyen desde el no contar con estas herramientas hasta la falta de experiencia técnica por parte del usuario. Por esta razón, la actualización en campo suele realizarse mediante otros mecanismos.

Bootloader

Muchos microcontroladores modernos usan memoria NOR Flash para almacenar el código de la aplicación, la principal ventaja de la memoria Flash es que puede ser modificada por una rutina de *software* durante la ejecución del programa. Esta rutina que se agrega junto al programa principal es la encargada de reemplazar la versión anterior del *firmware* con una nueva llamada *bootloader*.

Un *bootloader* reside siempre en la memoria, lo que permite que el dispositivo pueda ser actualizado en cualquier momento. Sin embargo debe ser lo más pequeño posible ya que no es deseable invertir una gran cantidad de un recurso tan preciado como lo es la memoria en una rutina que no tendrá una funcionalidad directa para el usuario final.

Para descargar un nuevo *firmware* en el dispositivo debe idearse un mecanismo que permita llamar al *bootloader* preparase para la transferencia. Por lo general existen dos condiciones de disparo, por *hardware* y por *software*. Una condición de disparo por *hardware* puede ser presionar un botón durante el inicio del dispositivo, mientras que una condición por *software* puede ser la falta de una aplicación válida en el sistema.

Cuando el sistema inicia, el *bootloader* checa las condiciones de disparo predefinidas y si alguna de ellas es válida trata de conectarse con un anfitrión y espera por un nuevo *firmware*. Este anfitrión puede ser cualquier dispositivo, sin embargo suele usarse una computadora personal con el *software* como anfitrión. La transmisión del *firmware* puede ser hecha a través de cualquier medio soportado tanto por el anfitrión como por el dispositivo de destino, por ejemplo RS232, USB, CAN entro otros.

El proceso de actualización usando un bootloader queda resumido como sigue

1. El fabricante libera una nueva versión del firmware
2. La nueva versión del firmware es distribuida a los usuarios
3. La condición de disparo es activada por el usuario
4. El bootloader se conecta con el anfitrión
5. El anfitrión enviar el nuevo firmware
6. La aplicación existente es reemplazada
7. La nueva aplicación es ejecutada

Una vez que la transferencia termina, el bootloader ha reemplazado la versión antigua del firmware por la nueva versión, por lo que la nueva aplicación ha sido cargada. La figura 3.3 muestra el proceso descrito.

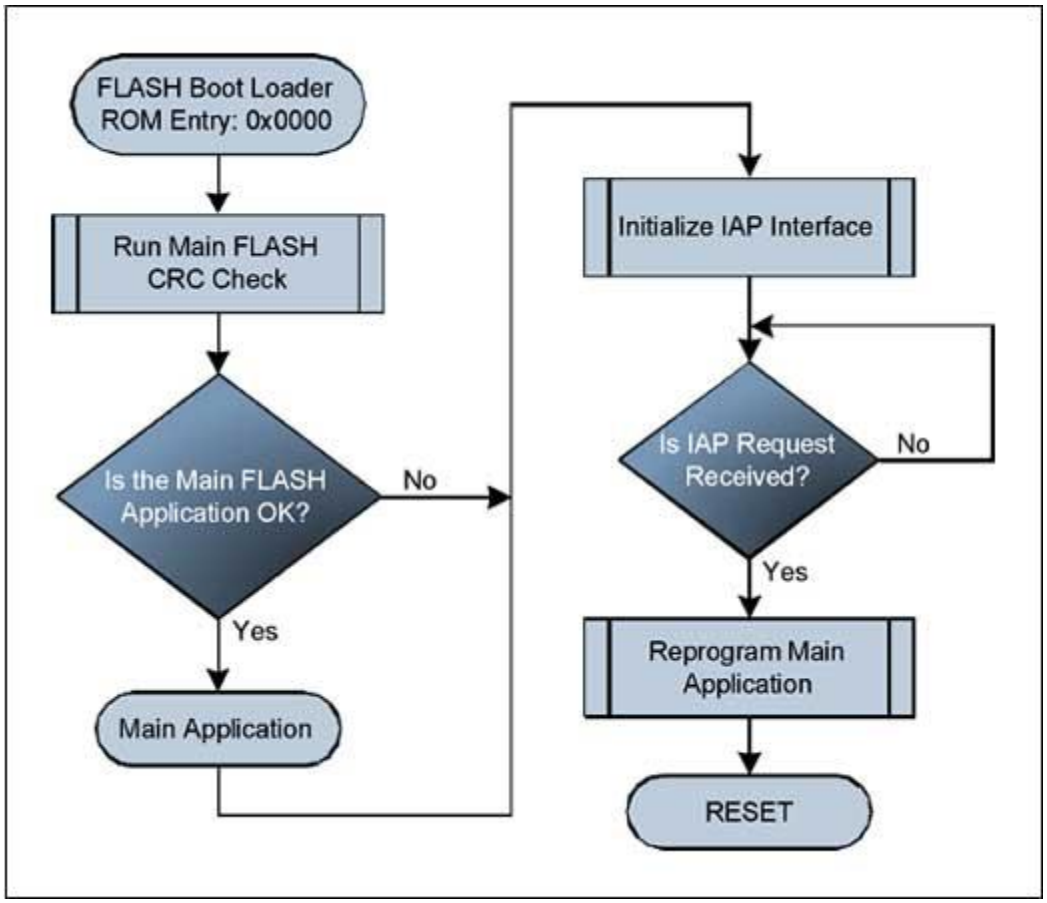


Figura 3.3. Diagrama de bloques del proceso de cargado de nuevo firmware mediante *Bootloader*

La ventaja de hacer uso de un *bootloader* en lugar de diseñar una rutina de actualización en el programa principal es liberar al usuario final de tener que usar herramientas especiales y a los diseñadores de *firmware* de tener que programar rutinas dedicadas, lo que favorece la simplicidad del programa final si se considera una actualización.

Existen diferentes problemas asociados al uso de un bootloader simple, como se describió anteriormente y se presentan de acuerdo con el área a la que están relacionados, es decir, a la seguridad del dispositivo o a la seguridad del *firmware*

De manera general, los problemas de seguridad del dispositivo pueden ocurrir cuando existe un error en la transmisión, por lo cual el *firmware* se corrompe; por lo que el *firmware* se trunca o una perdida en la información durante la transmisión, generando que el *firmware* no sea ejecutable por el dispositivo.

Los problemas que tienen que ver con la seguridad del *firmware* se dividen en 4 posibles casos en los cuales se puede hacer ingeniería inversa para la obtención del código fuente, un uso no autorizado del *firmware*, modificaciones del *firmware* o un uso del *firmware* en un dispositivo no autorizado.

El uso de un *bootloader* puede resultar una solución problemática cuando el nuevo *firmware* no se ha instalado apropiadamente, con lo cual se compromete el comportamiento del sistema. Por lo que el área de código de la aplicación estaría entonces corrompida e inutilizable. Esto puede ser considerado un problema de falla en la transferencia. Un resultado similar se obtiene si se pierde la conexión con el anfitrión durante la transmisión.

Alternativamente, si un error de transmisión ocurre durante la transferencia, parte del código está corrompido . El resultado puede variar de un error menor de cálculo hasta colapsar todo el sistema, dependiendo de la instrucción que ha sido modificada y como ha sido cambiada. Finalmente, algunos datos pueden perderse durante la transmisión del *firmware*, con lo que se corrompería completamente el código después de la parte perdida.

Asegurar un sistema significa que nuestro *firmware* debe ser protegido, integro y auténtico. La protección significa que una pieza de datos no puede ser leída por usuarios o dispositivos no autorizados. Los microcontroladores incorporan generalmente un mecanismo que imposibilita la lectura de la memoria de programa a usuarios maliciosos. Sin embargo, para la actualización de *firmware* en campo, el fabricante debe proporcionar el nuevo *firmware* al usuario final para que sea él quien inicie el proceso de actualización. Lo que significa que una persona con los conocimientos, herramientas y experiencia necesarias puede ser capaz de obtener el código original mediante un proceso de decompilación.

La autenticidad permite verificar que el *firmware* proviene del fabricante y no de alguien más. Un *firmware* genuino puede ser usado por dispositivos diferentes para el que fue diseñado. Este dispositivo podría ser una copia no autorizada del *hardware* del producto, o un dispositivo para propósitos de *hacking*. Este es nuevamente un problema de autenticidad.

Finalmente, la integridad es requerida para detectar la modificación de un dato, por ejemplo, un *firmware* autorizado puede ser ligeramente modificado, por lo que podría parecer genuino pero ser usado para fines diferentes al que fue diseñado.

Sin ningún tipo de función de seguridad, un *firmware* estará sujeto a todos los ataques de privacidad, integridad y autenticidad. Por lo tanto, es necesario tomar algunas medidas para hacer cumplir estos tres aspectos.

Para prevenir problemas de seguridad del dispositivo pueden emplearse pilas de protocolos de comunicaciones. Las pilas de protocolos son usadas en muchos estándares de comunicaciones, junto a otras características, para mantener una fiabilidad de la transferencia. Esta fiabilidad es importante para un *bootloader* ya que el *firmware* no debe corromperse durante la transmisión.

En el modelo OSI, un sistema de comunicaciones es dividido en siete capas, que forman la pila de protocolos. Cada capa es responsable de proveer un conjunto de características, por ejemplo, la capa física es la responsable de la interconexión física de los dispositivos. La fiabilidad es típicamente implementada en la capa de transporte.

La fiabilidad del transporte es usualmente obtenida por usar diferentes técnicas de detección y corrección de errores, bloques numerados y paquetes de reconocimiento.

Detección y corrección de errores

Es común utilizar códigos de detección y corrección de errores en la transmisión de datos. En realidad, muchas operaciones no pueden ser manejadas si se reciben datos corrompidos , por ejemplo, al cargar un nuevo *firmware*, al enviar un archivo a través de la red, entre otras. Sin embargo, muchos códigos han sido diseñados para posibilitar la detección y en algunos casos corregir errores de transmisión.

Los códigos de detección usan propiedades matemáticas simples para calcular un valor sobre el dato que será enviado. Este valor es transmitido junto con el dato original, cuando el dispositivo de destino recibe el dato, se recálcula el valor y se compara con el que ha sido recibido. Si ambos valores son iguales, la trasferencia ha sido exitosa, en caso contrario, existe uno o más bits no válidos.

Los códigos de corrección trabajan de manera similar, con la diferencia de que además de poder detectar un error en los datos son capaces de recuperar la información original, lo que agiliza la transmisión ya que no es necesario que el transmisor reenvíe la información.

Usualmente, la detección y corrección de errores no es aplicada sobre el *firmware* completo, sino que se trabaja sobre pequeñas piezas del archivo, lo que permite aplicar lo algoritmos de transmisión y detección de errores sobre cada trama del código, para verificar si existe un error y corregirlo o pedir la retransmisión de la trama.

Existen limitaciones en los códigos de detección y corrección de errores que dependen del algoritmo empleado para construirlos. Por otro lado, existen casos en los cuales no es realmente necesario un código de corrección de errores, ya que puede ser más práctico solicitar el reenvío de la trama incorrecta.

El propósito de dividir un archivo en bloques numerados es el de prevenir la perdida de algún bloque además de prevenir la llegada de dos bloques en un orden erróneo. Esto es crítico durante la transferencia de archivos en redes distribuidas donde los procesos de enrutamiento pueden causar la pérdida de algún paquete, lo que resultará en un archivo corrupto y en nuestro caso en un *firmware* inutilizable.

Como su nombre lo sugiere, para dividir un archivo en bloques numerados basta con agregar un número de identificación a cada bloque transmitido, este número de bloque se incrementa con cada envío. De esta manera, el receptor puede detectar fácilmente que dos bloques han sido intercambiados si recibe el bloque 3 en lugar de bloque 2 de manera similar puede determinar la perdida del paquete 4 si primero ha recibido el paquete 5 inmediatamente después del paquete 3.

Otra técnica consiste en hacer uso de paquetes con reconocimiento. Los paquetes con reconocimiento trabajan de la siguiente forma. Cada vez que el transmisor envía un bloque de datos, éste espera por recibir un paquete de reconocimiento (*acknowledge*). Por ejemplo, un paquete que indique que ha sido recibido correctamente. Si no se recibe ninguna respuesta después de un periodo de tiempo, entonces el transmisor asume que el paquete se ha perdido y lo retransmite.

Ningún dato es enviado mientras el transmisor espera por la señal de ACK, por lo que los paquetes no pueden llegar fuera de orden si son transmitidos uno a la vez.

Seguridad del firmware

Verificar la integridad significa validar si el *firmware* ha sufrido una modificación accidental o intencionada. Una modificación accidental es un problema de seguridad y es típicamente resuelto por usar códigos de detección de errores. Existen diferentes formas de revisar si el *firmware* ha sido modificado, revisaremos algunas de ellas.

El objetivo de una función *hash* es proporcionar una huella digital en un archivo. Esto significa que por el contrario a un sistema de detección de errores, cada archivo tendrá su propia huella digital. Para verificar la integridad de un *firmware*, su huella digital es calculada y adjuntada en el archivo. Cuando el *bootloader* recibe tanto el *firmware* como su huella digital, éste recálcula la huella digital y compara con la original. Si ambos son idénticos, entonces el *firmware* no ha sido alterado.

En la práctica, una función *hash* toma una cadena de cualquier longitud como una entrada y procesa una salida de tamaño fijo llamado resumen del mensaje (*message digest*). También tiene varias propiedades importantes, por ejemplo, una buena difusión que es la habilidad de producir un resultado completamente diferente, incluso si sólo un bit de la entrada cambia .

Dado que la longitud de salida es fija independientemente de la entrada, no es posible generar un resumen diferente para cada archivo. Sin embargo, las funciones *hash* aseguran que es casi imposible encontrar dos mensajes distintos que tengan el mismo resumen, con esto se logra casi el mismo resultado que la singularidad, por lo menos en la práctica.

La desventaja de usar una función de *hashing* simple en el *firmware* es que cualquiera puede hacerlo. Esto significa que un atacante podría modificar el archivo y volver a calcular el resumen, con lo que el bootloader sería incapaz de determinar si una alteración fue realizada.

Sin embargo, una función *hash* por si misma puede utilizarse para verificar la integridad del *firmware* en tiempo de ejecución, para evitar la ejecución de una aplicación dañada.

Otra técnica empleada es con base en una firma digital de un *firmware* calculado usando una función hash y encriptado con una llave pública criptográfica. Esto produce una firma digital, similar a las firmas utilizadas en la vida real.

El cifrado de clave pública o asimétrica se basa en el uso de dos claves, el fabricante utiliza su clave publica que es secreta para encriptar la firma, mientras que el dispositivo utiliza la clave pública correspondiente para descifrarla.

Ya que sólo la llave cifrada puede cifrar los datos, nadie más que el fabricante puede producir la firma. De esta forma, un usuario malicioso no sería capaz de realizar el ataque descrito anteriormente, pero nadie puede comprobar la firma usando la llave publica del fabricante.

3.3.2. Formato del archivo de Firmware

Los compiladores soportan una amplia variedad de formatos de archivo de salida. El más básico de ellos es el formato binario (.bin) que es una imagen simple del firmware. Muchos formatos incluyen información adicional como la dirección del paquete, ejemplos de ellos el Motorola s-record e Intel .hex, o con información de depuración como el *Executable and Linking Format* (.elf).

El compilador gcc puede entregar el archivo final en formato binario o en formato Intel HEX32, el cual ha sido elegido para representar la información del *firmware* cuando se envía a la CV para realizar una reconfiguración remota.

Formato Intel Hex 32

El formato Intel Hex es un formato de archivo usado para representar la información con la que se programarán microcontroladores, memorias y otros circuitos integrados. Consiste en un archivo de texto cuyas líneas contienen valores hexadecimales que codifican los datos , y su *offset* o dirección de memoria.

Cada línea se forma con los siguientes campos:

1. Código de inicio, un símbolo ':'

2. Longitud del registro, dos dígitos hexadecimales con la cantidad de bytes de datos que contiene el paquete, por lo regular 16 o 32 bytes.

3. Dirección, cuatro dígitos hexadecimales en *big endian*, con la dirección de inicio de los datos. Para direcciones mayores 0xFFFF se emplean otros tipos de registro.

4. Datos, duplas de dígitos hexadecimales, de 00 a 05 que definen el tipo de campo de datos.

6. Checksum o suma de comprobacion, son dos dígitos hexadecimales con el complemento a dos de la suma de todo los campos anteriores con excepción del campo de inicio (:).

Existen seis tipos de paquetes:

00. Paquete de datos, contiene una dirección de 16 bits y los datos correspondientes.

01. Fin de archivo, no contiene datos y debe estar al final del archivo.

02. Dirección Extendida de Segmento, indica el offset o dirección de inicio del segmento para acceder a direcciones con más de 16 bits. Este valor se desplaza 4 bits a la izquierda y se le suma a la dirección proporcionada por los registros de datos. Su campo de longitud debe ser valor 02 y el de dirección 0000.

03. Dirección de Comienzo de Segmento, especifica los valores iniciales de los registros CS:IP, para procesadores 80x86. El campo de dirección es 0000, longitud 04 y los datos contienen dos bytes para el segmento de código y otros dos para el *instruction pointer.*

04, Dirección lineal extendida. Permite dirigirse a 32 bits de memoria al contener los 16 bits superiores de la dirección. Su campo de dirección vale 0000 y el de longitud 02.

05. Comienzo de Dirección Lineal. Contiene 4 bytes que se cargan en el registro EIP de los procesadores 80386 y superiores. Su campo de dirección vale 0000 y el de longitud 04.

En particular, en *firmware* es representado en formato Intel Hex32 que hace uso de los paquete 0, 1 y 4 para poder manejar programas de hasta 2GB.

3.3.3 Firmware del modulo de reconfiguración remota

El diseño del *firmware* para el microcontrolador PIC18F2550 que es el corazón del módulo de reconfiguración remota se basa en una arquitectura de bomba de mensajes.

Inicialmente se encuentra en modo de bajo consumo y es despertado por una interrupción a través de UART. El primer mensaje corresponde a un mensaje de asentamiento que revisa la disponibilidad del módulo de cargado a través de un código que el microcontrolador PIC envía como respuesta.i el mensaje es recibido con éxito, se procede a iniciar uno de los 5 posibles estados que son:

Tabla 5. Comandos para la bomba de mensajes de reconfiguración remota

Comando	Descripción	Valor
Check Comunication	Regresa un comando de verificación indicando que el módulo se encuentra listo para recibir un comando	=
Program Memory	Pone al módulo en modo de cargado de programa con lo que se inicia la recepción del archivo en formato Intel Hex	!
Read Memory	Regresa un archivo en formato Intel Hex con el contenido de la memoria EEPROM	"
Erase Memory	Inicializa y verifica el correcto borrado de la memoria EEPROM	$
Idle Mode	Indica que el módulo de cargado no recibirá más comandos por lo que debe entrar en modo de bajo consumo	&
Verify Memory	Verifica el contenido de la memoria EEPROM	%

Los comandos están codificados en formato ASCII, con símbolos que no están presentes en el archivo Intel HEX para evitar posibles problemas de ejecución o interpretación durante el cargado de un nuevo programa.

La figura 3.4 muestra el diagrama de flujo de la bomba de mensajes usada por el firmware del módulo de reconfiguración remota. Una vez que el archivo Intel Hex ha sido cargado a la memoria EEPROM.

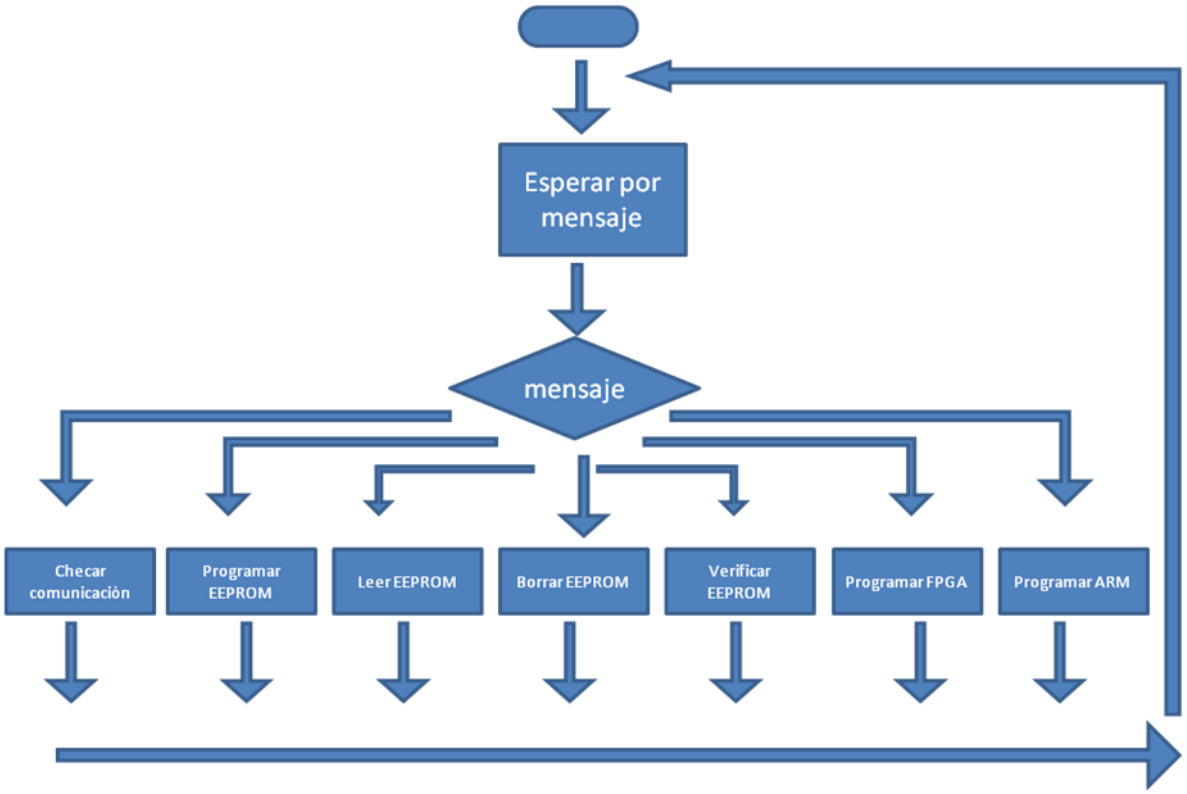


Figura 3.4. Diagrama de flujo de la bomba de mensajes del módulo de reconfiguración remota.

3.4. Modo de carga de software

La CV cuenta con un microcontrolador PIC18F2550 que se encarga de reconfigurar en campo el microcontrolador SAM3U4E.

3.4.1. AT91SAM Boot Program

El Boot Program integra aplicaciones que permiten la carga de software en la distintas memorias del microcontrolador SAM3 ya sea por una interfaz USB o por medio de la unidad serial DBGU conectada a una PC, el modo de funcionamiento de Boot Program, utilizando la unidad serial es la siguiente.

El Boot Program comienza inicializando la unidad DBGU a 115200 bauds, 8 bits, 1 bit de paro y sin paridad. Después comprueba el auto-baudaje de acuerdo a los pasos descritos en el diagrama de flujo de la figura 3.4.

Una vez que realiza la comproboación del baudaje y resulta satisfactoria, espera de forma continua la recepción de comandos provenientes del software SAM-BA (Boot Asistant) o en este caso, del módulo de reconfiguración vía UART.

El Boot Program se copia de la memoria ROM a la SRAM del SAM 3U4E por lo que ocupa una cantidad de memoria para variables y *stacks*, con lo que deja disponible para el usuario un espacio de 24576 bytes. El área del usuario se extiende de la dirección 0x202000 a 0x20800. Los pines utilizados, para la unidad serial DBUG, son: DRXD y DTXD.

SAM-BA, como se mencionó en la sección 3.1, es un software basado en una biblioteca de enlace dinámico (DLL[26]) AT91Boot_DLL.dll de dominio público que permite elaborar aplicaciones particulares para la comunicación entre el *Boot Program* del microcontrolador y el software de estación terrena. Con ella se puede generar un software bajo diseño que permita comunicar una PC con el microcontrolador SAM3U4E a través de un puerto serial.

SAM-BA puede ejecutarse desde una terminal de comandos o a través de una interfaz gráfica de usuario (GUI), operando de la siguiente forma:

[26] Common Dynamic Linked Library)

- Se conectan las líneas DRXD y DTXD de la CV a través del conector dispuesto para tal fin a través del puerto serie, que a su vez se conecta a un transceptor MAX3232 para acoplar niveles lógicos y de tensión.
- SAM-BA utiliza la función *Scan* de AT91Boot_DLL.dll para determinar los dispositivos que se encuentran conectados a la PC.
- SAM-BA despliega una ventana donde se selecciona el dispositivo conectado a la PC y su interfaz de comunicación, "COMX" para el caso de la unidad serial DBGU.
- SAM-BA despliega un área de memoria máxima de 1024 Bytes en los formatos ASCII , 8 bits, 16 bits o 32 bits. El buffer de la memoria ROM, FLASH o SRAM puede ser editado.
- Para la lectura o despliegue de memoria es necesario especificar la dirección de inicio, el tamaño en bytes (menos a 1024) y el formato para finalmente presionar el botón *refresh* y actualizar el área de despliegue de la memoria.
- La edición del contenido de memoria, disponible sólo para la SRAM, se realiza seleccionando la dirección de memoria donde se desea modificar su contenido y anotando el nuevo valor en la ventana emergente, para luego presionar por último el boton OK.
- SAM-BA ofrece un área donde puede seleccionar la memoria SRAM o Flash interna del microcontrolador y subir un programa o descargar un programa a través de un archivo binario.
- Una vez seleccionado el archivo binario a subir al microcontrolador, se habilita la memoria SRAM o Flash, luego se selecciona la dirección de memoria donde se comenzará a subir el programa y se presiona el botón de envío (*Send File*)
- De manera similar se puede recibir un archivo con la variante de que debe especificarse el tamaño en bytes que se desean leer de la memoria del microcontrolador. Además, puede compararse un área de memoria a partir de una dirección específica y con base al tamaño de un archivo especificado en el campo *Send File*. Una ventana aparece indicando si el archivo y el contenido de la memoria son idénticos o no, la comparación permite verificar si un archivo se subió correctamente al microcontrolador.

Los comandos y sus respuestas aparecerán en un área denominada TLC[27], donde pueden ejecutarse en modo terminal los comandos específicos que proporcionan otro método para ejecutar las tareas ya descritas. Para más información puede consultarse *SAM-BA Boot Asistant* en el apartado de bibliografía.

[27] Tool Command Language

Capítulo 4. Software Básico de Operaciones de la Estación Terrestre

En este capítulo se describen los diferentes módulos diseñados para operar y validar el subsistema de CV. El software se implementó en diferentes lenguajes y se diseñaron programas tanto en ambiente Windows como en consola, que después se integrarían en la Interfaz Gráfica de Usuario Final.

4.1. La tecnología .NET

La tecnología .NET es un *framework* de Microsoft que hace un énfasis en la transparencia de redes, con independencia de plataforma de hardware y que permite un rápido desarrollo de aplicaciones. Basado en ella, Microsoft intenta desarrollar una estrategia horizontal que integre todos sus productos, desde el sistema operativo hasta las herramientas de mercado.

.NET podría considerarse como una respuesta de Microsoft al creciente mercado de los negocios en entornos Web, como competencia a la plataforma Java de Oracle Corporation y a los diversos *framework* de desarrollo web basados en PHP. Su propuesta es ofrecer una manera rápida, económica, pero a la vez segura y robusta, de desarrollar aplicaciones a las que denomina soluciones, con una integración más rápida y ágil entre empresas y un acceso más simple y universal a todo tipo de información desde cualquier tipo de dispositivo.

4.1.1. Microsoft Visual C# Express Edition

C# es un lenguaje de programación orientado a objetos de escritura segura, que es simple pero poderoso, permitiendo a los programadores construir una amplia gama de aplicaciones.

Visual C# forma parte de Visual Studio, el cual es un Ambiente de Desarrollo Integrado (IDE por sus siglas en inglés) que asiste a los desarrolladores en la creación de programas en varios lenguajes, poniendo a su disposición un gran número de herramientas invaluables en la programación.

Las *Express Editions* de Visual Studio son una extensión de su línea de productos que incluyen herramientas simples y fáciles de aprender enfocadas a desarrolladores no profesionales como estudiantes o aficionados a la programación que deseen construir aplicaciones Windows dinámicas.

El desarrollo de Software en Microsoft Visual C# *Express Edition* es completamente gratuito sin necesidades de licencia o contratos.

4.1.2. Microsoft .NET Framework

La plataforma .NET Framework de Microsoft es una gran biblioteca de clases de uso común que permite la interoperabilidad entre los diversos lenguajes que lo soportan. Esta le permite a desarrolladores usar un mismo set de habilidades para rápidamente construir aplicaciones para la web, dispositivos, servicios y más.

.NET Framework provee un ambiente de programación orientado a objetos consistente que promueve el uso de código seguro y elimina problemas de rendimiento comunes minimizando conflictos entre versiones y distribución hacia diversas plataformas.

Actualmente, Microsoft .NET Framework es el estándar de desarrollo para aplicaciones basadas en Microsoft Windows.

4.1.3. Windows Forms

Windows Forms es una interfaz de programación gráfica incluida como parte de Microsoft .NET Framework, el cual provee acceso a los elementos de interfaz nativos de Microsoft Windows al envolver el existente Windows API en código administrado. Windows Forms es una forma fácil y

rápida de proveer componentes de interfaz gráfica a programas desarrollados utilizando .NET Framework.

En *Windows Forms*, todas las partes que formarán nuestra interfaz gráfica se denominan *Componentes*. Todo *Menú*, *Botón*, *Casilla de Texto*… etc., es un *Componente*, y para lograr tener una aplicación funcional se suelen requerir docenas de estos.

Mientras que la mayoría de los *Componentes* no son indispensables para el funcionamiento de nuestros programas, toda aplicación de *Windows Forms* debe consistir de al menos un *Form* (Formulario). El resto de los *Componentes* deberán entonces ser "agregados" a un *Form* para poder desplegarlos en pantalla y trabajar con ellos.

Cabe destacar que una aplicación de *Windows Forms* puede consistir de varios *Forms*, pero siempre existirá un *Form* raíz del cual se deriven todos, el cual será nuestra Ventana Principal.

Los *Forms* constan de tres áreas generales:

1. TitleBar: *Barra de* Título. Parte superior de todo *Form*. Por defecto, cuenta con un *Menú de Click Derecho* que representa comandos básicos del sistema operativo para el manejo de aplicaciones, algunos de los cuales también podremos encontrar en la *Control Box* (Caja de Controles). De izquierda a derecha, identificamos:

1. Icon : *Ícono*. Imagen que representa al programa.
2. Title : *Título*. Nombre de nuestro programa.
3. Control Box: *Caja de Controles*. Área derecha de la Barra de Título, consta de tres components.
    - Minimize : *Minimizar*. Oculta la ventana hacia la barra de tareas.
    - Maximize : *Maximizar*. Expande la ventana, abarcando la totalidad del escritorio.
    - Close : *Cerrar*. Destruye nuestra ventana, finalizando el programa.

4.1.4. Manipulación del Puerto Serie en Visual C#

El puerto serie es una interfaz de comunicación que se basa en un protocolo serial, o sea, los datos son transmitidos y recibidos uno por uno. Visual C# 2010 contiene una herramienta que es de gran ayuda para trabajar con los puertos: el Componente *SerialPort*.

Para que una aplicación existente pueda comunicarse con el Puerto Serie, lo único que necesitamos es:

1. Agregar un Componente *SerialPort* a nuestro *Form*.
2. Modificar las Propiedades del *SerialPort*. Para el caso de este Componente, sus Propiedades definen las características con las que la comunicación se llevará a cabo, por ejemplo, el nombre del puerto COM a utilizar, el tamaño del buffer y el baudaje entre otros.
3. Dentro del código de nuestro programa (por ejemplo, en un Evento Click de un Componente *Button*), se utilizan diversos Métodos y Propiedades del Componente *SerialPort* para *realizar la comunicación.*

En seguida, se muestra una lista de algunos de los *Métodos* y *Propiedades* más útiles de *SerialPort*:

- *SerialPort.Open()* – Abre un canal de comunicación con las características especificadas en las *Propiedades* de dicho *Componente*.
- *SerialPort.Close()* – Cierra el canal de comunicación actual, impidiendo que se envíen o reciban nuevos datos.
- *SerialPort.Write(string text)* – Envía la cadena de caracteres *text* a través de un Puerto Serie.
- *SerialPort.ReadExisting()* – Regresa una cadena de caracteres proveniente de un Puerto Serie.
- *SerialPort.IsOpen* – *Propiedad* cuyo valor será *true* o *false* dependiendo del estado actual del canal de comunicación.

4.2. Análisis de requerimientos del software básico de estación terrestre

La interfaz de usuario requiere de una aplicación ejecutable en una plataforma Windows. Fácil de usar para usuarios inexpertos en el uso de plataformas satelitales, pero a su vez con las herramientas necesarias para realizar procesos de depuración, mantenimiento y actualización del firmware, monitoreo del estado del satélite y envío de comandos a los subsistemas de la

plataforma satelital así como recepción de telemetría. Se debe realizar un sistema modular, que permita la reutilización de código, por lo cual se recomienda un diseño orientado a objetos. El diseño adecuado de clases permitirá crear rápidamente nuevas versiones de software o subprogramas para depuración y cargado de programa.

Un diseño adecuado multicapa permitirá también separar la funcionalidad de la vista. Esto se logra al realizar un diseño orientado a objetos, separando la funcionalidad de la vista, donde los controles como menús y botones harán llamadas a funciones de clases base y no implementarán directamente la funcionalidad para la comunicación del satélite como se haría en un programa monolítico.

4.3. Desarrollo del software básico de estación terrestre

El software básico de la estación terrestre está compuesto por diferentes submódulos probados inicialmente de manera individual. Estos módulos detallan funciones como convertir archivos en formato binario (.bin y .XSVF) a formato Intel para ser enviados al módulo de reconfiguración. De igual forma se dispone de un módulo que transforma archivos en formato Intel Hex a formato binario.

Para realizar pruebas iníciales y validar la correcta transmisión del firmware al modulo de reconfiguración, se dispone de una interfaz que inicialmente fue monolítica, que posteriormente implementó el diseño multicapa para poder reutilizar el código de la funcionalidad del software, el cual es independiente de la vista, en la interfaz final.

4.3.1. EEPROM Loader-Debugger

Este modulo fue diseñado para validar el firmware que posee el microcontrolador PIC18F2520 del módulo de reconfiguración. Inicialmente las pruebas se realizaban con una interfaz para puerto serie como lo es la Hyper Terminal, pero esta opción pronto quedó rebasada ya que en ocasiones se requería enviar tramas de datos en formato ASCII y en otras valores binarios o hexadecimales, por lo que se optó por desarrollar una interfaz simple y amigable a la medida que permitiera la reutilización de las clases usadas en el diseño de la funcionalidad.

La figura 4.1 muestra la pantalla principal de la interfaz de cargado a la memoria en la cual, del lado derecho se selecciona el puerto serie al cual está conectado el módulo de reconfiguración. Se utiliza una conexión a puerto serie debido a que el módulo de reconfiguración se comunicará con

el microcontrolador SAM3U4E a través de la UART. Del lado derecho podemos encontrar los controles necesarios para la configuración del puerto así como para su conexión y desconexión.

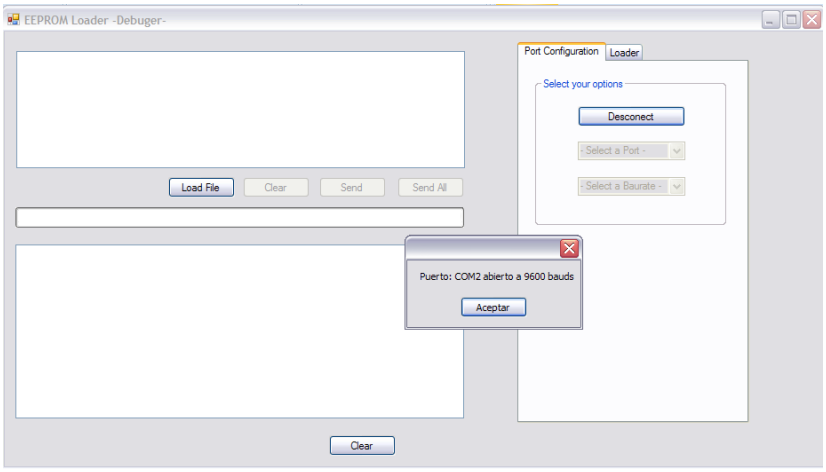


Figura 4. 1. Pantalla inicial de configuración de puerto serie

Una vez que los parámetros del puerto serie han sido configurados y el puerto ha sido abierto, los controles del lado derecho se intercambian al enfocar la pestaña *Loader*. La figura 4.2 muestra los controles para la depuración del firmware.

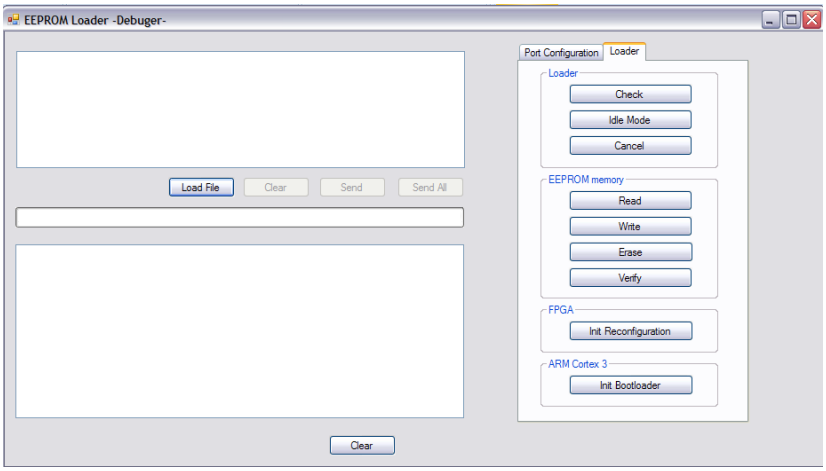


Figura 4. 2. Pantalla con los controles de depuración del firmware para el módulo de reconfiguración

Del lado izquierdo de la pantalla se encuentran dos controles donde se mostrará la información que es enviada desde un archivo en formato Intel Hex. El cuadro de la parte inferior muestra la información que ha sido enviada y recibida a manera de una bitácora.

Entre los comandos que podemos encontrar en la barra de la parte derecha se encuentran checar comunicación, iniciar modo de bajo consumo, cancelar la última operación. El segundo bloque de comandos envía los comandos para cargar el firmware en la memoria EEPROM como son: inicia modo de lectura, escritura, borrar y verificar la memoria. Los comandos para *Init Reconfiguration* e *Init Bootloader* sirven para que el modulo de reconfiguración inicie la rutina del intérprete de archivos XSFV para un FPGA que es el cerebro de la nueva tarjeta de estabilización, mientras que el segundo carga un nuevo programa a través del *bootloader* al microcontrolador SAM3U4E

Para cargar el *buffer* de salida con un archivo se hace uso del botón *Open File*, mientras que para limpiar el buffer se utiliza el botón *Clear*. Para iniciar un paquete, es decir la línea que esté seleccionada en el buffer de transmisión, se usa el botón *Send*, mientras que el botón *Send All* envía todo el contenido del buffer.

4.3.2. JTAG Programmer

Esta interfaz es la propuesta para cargar un nuevo *firmware* al microcontrolador SAM3U4E o al FPGA del Subsistema de Estabilización. A diferencia de la interfaz EEPROM *Loader Debugger*, esta interfaz realiza el proceso de manera más inteligente y autónoma ya que realiza el envío de comandos con base en una maquina de estados a diferencia de la interfaz anterior que sólo enviaba comandos o tramas cuando el usuario lo solicitaba.

El diseño ofrece una herramienta independiente del software básico de operaciones, de menor tamaño y modular. La figura 4.3 muestra la interfaz de usuario JTAG Programmer.

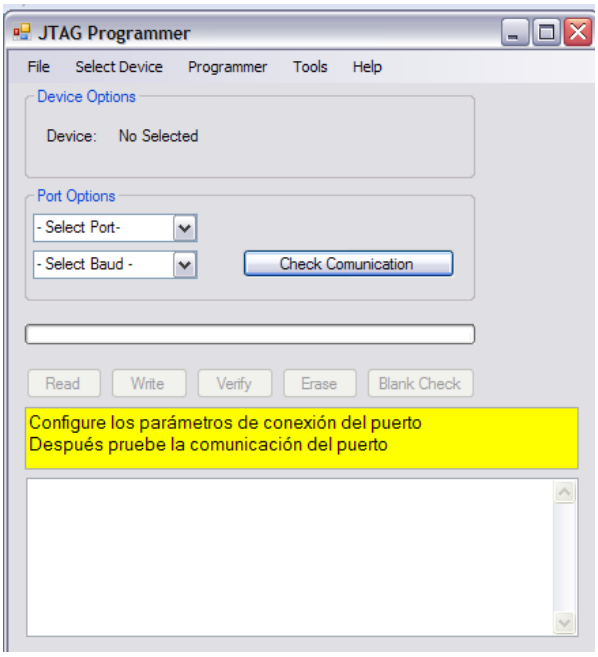


Figura 4. 3. Pantalla principal de la aplicación JTAG *Programmer*

Uno de los requisitos es que esta interfaz sea fácil de usar, por lo cual se tiene un control que indica cual es la condición faltante para funcionar o cual es el siguiente paso que puede generarse. Como se ve en la imagen, la interfaz indica al usuario que existe un parámetro faltante, en este caso de configuración del puerto serie, y una opción para resolver el problema. Además se hace una llamada de alerta colocando este control en color amarillo.

4.3.3. Bin2Hex

Este programa es una aplicación de consola, diseñada con el propósito de probar la clase que se encarga de transformar archivos entre diferentes formatos como lo son binario, xsvf e Intel Hex. En particular, esta aplicación solo implementa las rutinas de conversión de formato binario a formato Intel Hex32. La figura 4.4 muestra un ejemplo de la aplicación.

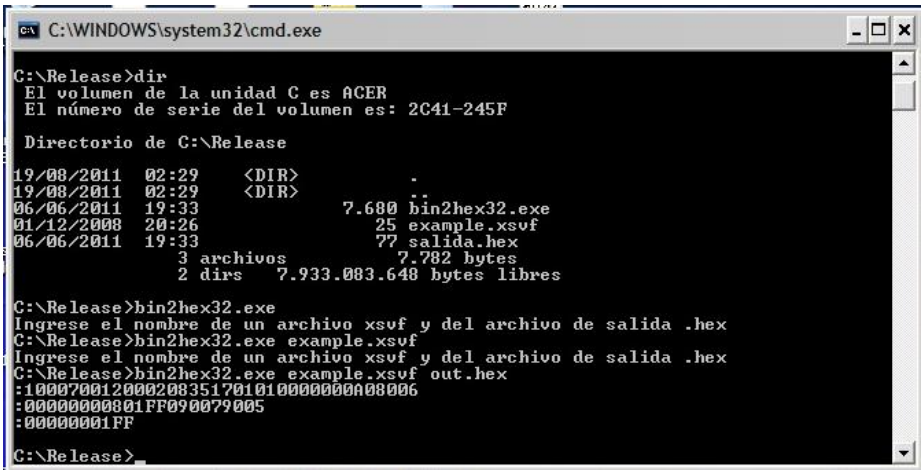


Figura 4.4. Aplicación Bin2Hex

Para ejecutar la aplicación es necesario invocarla con el nombre bin2hex junto a los parámetros de nombre del archivo de origen y nombre del archivo destino, ambos con extensión.

Capítulo 5. Pruebas operativas entre Software de Estación Terrestre y Computadora de Vuelo

En este capítulo se presentan las pruebas operativas realizadas para validar el diseño de la Computadora de Vuelo (CV) y probar la operación de los subsistemas de la plataforma satelital SATEDU.

5.1. Herramientas usadas en la validación del diseño de CV

La validación de la comunicación con los subsistemas se realizó de manera virtual con la herramienta VSM Proteus que forma parte de la suit Proteus 7 Professional de Labcenter Electronics, con un simulador de circuitos electrónicos y con una tarjeta de desarrollo de Atmel, la SAM3U-EK.

Estas herramientas permitieron validar el software básico de operaciones y otras aplicaciones que se ejecutarán en Tierra, como el *firmware* para el módulo de reconfiguración remota y el programa básico de operaciones de la CV.

5.1.1. MPLAB IDE

MPLAB IDE[28] es un Entorno de Desarrollo Integrado para productos de la marca Microchip. Soporta todos los microcontroladores Microchip, entre ellos los usados en los subsistemas de la plataforma satelital SATEDU, además de integrar un editor modular y permitir la interacción con otras herramientas como compiladores en lenguajes de alto y bajo nivel, depuradores en circuito y simuladores, además de programadores, entre otras herramientas. MPLAB IDE corre bajo la plataforma Microsoft Windows.

MPLAD IDE fue usado en el proceso de validación del diseño de la CV del satélite HumSAT-México para constituir un entorno de desarrollo desde el cual se invocarían otras herramientas como lo son el compilador, simulador y programador.

Para iniciar el trabajo con MPLAB ID es necesario seguir la siguiente secuencia:

1. Crear un nuevo proyecto para trabajar con el compilador PIC18
2. Crear y agregar un archivo de código fuente al proyecto
3. Capturar el código fuente del subsistema a validar
4. Compilar el código fuente y depurar cualquier error de sintaxis

Una vez que se han realizado estos pasos, es posible usar otra herramientas para la simulación interactiva.

5.1.2. Proteus Professional 7

Proteus Professional es una suit de programas de diseño y simulación para aplicaciones en electrónica, desarrollado por Labcenter Electronics que consta de dos programas principales: Ares, usado para el diseño de circuitos impresos e Isis, usado para el diseño de esquemáticos y simulación en tiempo real. Proteus también incorpora los módulos VSM y Electra.

Isis es el acrónimo de *Intelligent Schematic Input System* o Sistema de Ruteo de Esquemáticos Inteligente. Isis permite diseñar el plano eléctrico o esquemático del circuito. Además de incorporar componentes tan comunes como resistencias y capacitores, las bibliotecas de Isis incorporan microcontroladores e instrumentos virtuales, además de componentes interactivos para realizar simulaciones en tiempo real con el uso del módulo VSM.

[28] Integrated Develoment Enviroment

El módulo VSM es una de las prestaciones de Proteus que permite la simulación de esquemáticos en tiempo real. VSM son las siglas de *Virtual System Modeling* o Sistema Virtual de Modelado. Pueden simularse diferentes microcontroladores conectados a recursos periféricos interactivos como pantallas LCD, teclados, motores y muchos otros componentes.

VSM puede ser ejecutado desde MPLAB IDE una vez que el circuito esquemático ha sido creado en Isis. Para realizar una simulación intereactiva desde MPLAB con VMS de Proteus deben seguirse los siguientes pasos.

1. Crear el esquemático en Isis de Proteus
2. Crear un proyecto en MPLAB si este no existe, si este existe y ya se posee un archivo compilado debe saltarse al paso 4
3. Agregar un archivo de código fuente y compilarlo
4. Abrir VSM de Proteus desde el menú *Debugger*
5. Cargar el archivo del esquemático
6. configurar los parámetros del simulador como son velocidad del reloj, entre otros, e iniciar la simulación.

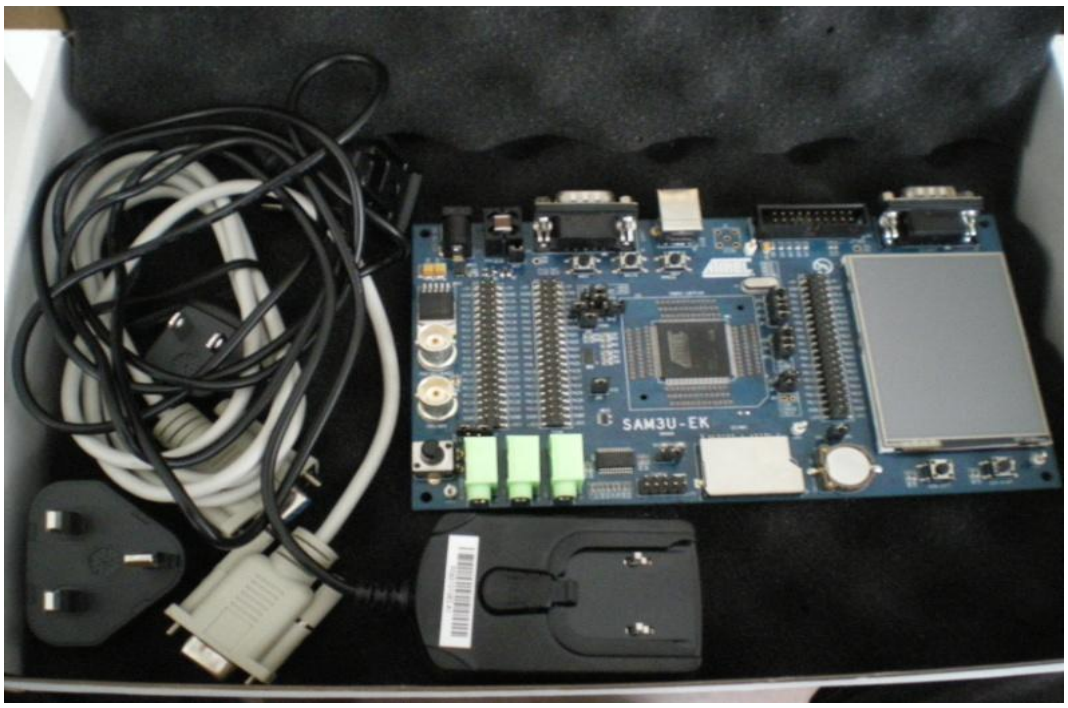
5.1.3. Sistema de desarrollo SAM3U-EK

El sistema de desarrollo SAM3U-EK es una plataforma para desarrollar de manera rápida código para aplicaciones que se ejecutan en procesadores SAM3U de Atmel. Entre las características y recursos que incorpora se pueden enumerar:

1. Microcontrolador SAM3U4E QFP
2. Cristal de 12 MHz
3. Cristal de 32.768 KHz
4. PSRAM
5. NAND Flash
6. Batería de respaldo de 3.3V
7. Pantalla táctil LCD a color de 2.8 pulgadas
8. UART con transceptor de voltaje
9. USART con transceptor de voltaje

10. Codificador de audio con conectores de entrada y salida para audífonos, línea estereofónica de entrada y micrófono monoaural.
11. Interfaz SD/MMC
12. Acelerómetro 3D
13. Sensor de temperatura
14. Puerto JTAG
15. Conector BNC par aentrada al ADC
16. Potenciómetro conectado al ADC
17. conector ZigBee

La figura 5.1 muestra el sistema de desarrollo SAM3U-EK.



5. 1. Sistema de desarrollo SAM3-EK

5.2. Pruebas operativas del firmware de la CV

El *firmware* de la Computadora de Vuelo está implementado en lenguaje C e implementado haciendo uso del entorno de desarrollo Eclipse con la suit YAGARTO para GNU ARN toolchain. El *firmware* guarda compatibilidad en la operación y comunicación con los subsistemas de la plataforma SATEDU y es ejecutado en un sistema de desarrollo Atmel SAM3U-EK.

Las pruebas operativas del *firmware* de la CV que se realizaron comprenden la correcta recepción de comandos así como su ejecución desde un programa de terminal remota, que se ejecuta en una PC, por comunicación por puerto serie que hace las veces de la Interfaz de usuario.

Una vez que hubo respuesta por parte del *firmware* del procesador SAM3, se procedió a validar su interacción con el subsistema de comunicaciones de la plataforma satelital SATEDU. Esta validación consistió en construir un esquemático en el programa Isis de Proteus para luego realizar una simulación interactiva en tiempo real en una PC.

Para que el esquemático y la tarjeta de desarrollo pudieran interactuar se hizo uso del componente COMPIN, el cual permite la comunicación con los componentes del VSM de Proteus en el esquemático y un puerto serie físico o virtual.

El resultado final fue la interacción entre el *firmware* de la CV, que se ejecutaba en el sistema de desarrollo, y la tarjeta de comunicaciones simulada en el VSM de Proteus. Realizar esta primera validación haciendo uso de estas herramientas permitió una depuración de los sistemas sin necesidad de hacer uso de otras herramientas como los depuradores en circuito.

Finalmente se realizaron pruebas con la interfaz de cargado de nuevo programa y el módulo de reconfiguración remota.

5.3. Pruebas operativas del firmware del módulo de reconfiguración remota

Para el módulo de reconfiguración remota se ha implementado una interfaz que permite el cargado de un programa en un archivo en formato binario o en Intel hex32 a una memoria EEPROM donde se almacena temporalmente con ayuda de un microcontrolador PIC18F2520, para luego ser volcada a la memoria FALSH del microcontrolador SAM3.

Para la validación de este sistema de cargado remoto se empleó software de simulación de la suit Proteus, posteriormente fue diseñado y armado un prototipo que pudiera interactuar con el sistema de desarrollo SAM3U-EK.

Falta referenciar las siguientes dos figuras en el texto.

Se han realizado pruebas del cargado remoto de nuevo programa, con lo que se han tenido resultados favorables. La comunicación inalámbrica entre la interfaz de usuario y la Computadora de Vuelo es transparente ya que el sistema de comunicaciones se encarga de validar las tramas de información y detección de errores. Para realizar esta prueba se emplearon radiomodems que serán posteriormente reemplazados por las tarjetas de comunicaciones de la plataforma SATEDU y para el satélite HumSAT por el sistema de comunicaciones de la estación terrena.

Una vez armado el prototipo del circuito impreso de la tarjeta de Computadora de Vuelo, le será cargado el firmware ya validado, lo que permitirá verificar la correcta operación del hardware.

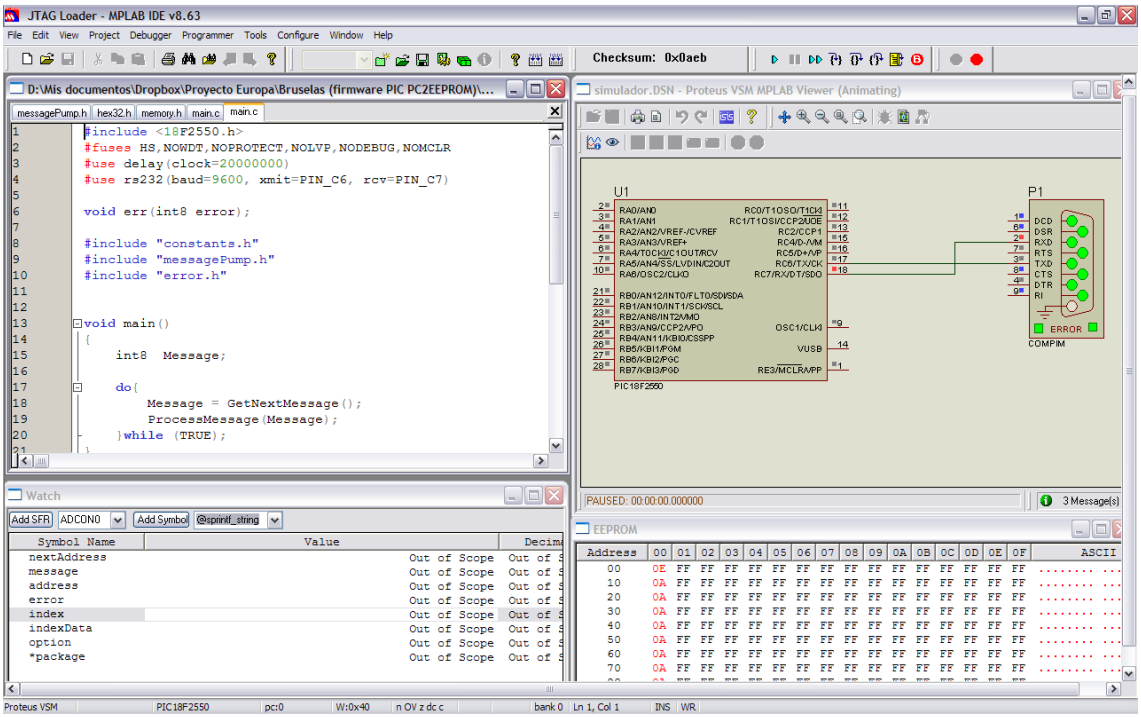


Figura 5.2. Validación del firmware mediante simulación

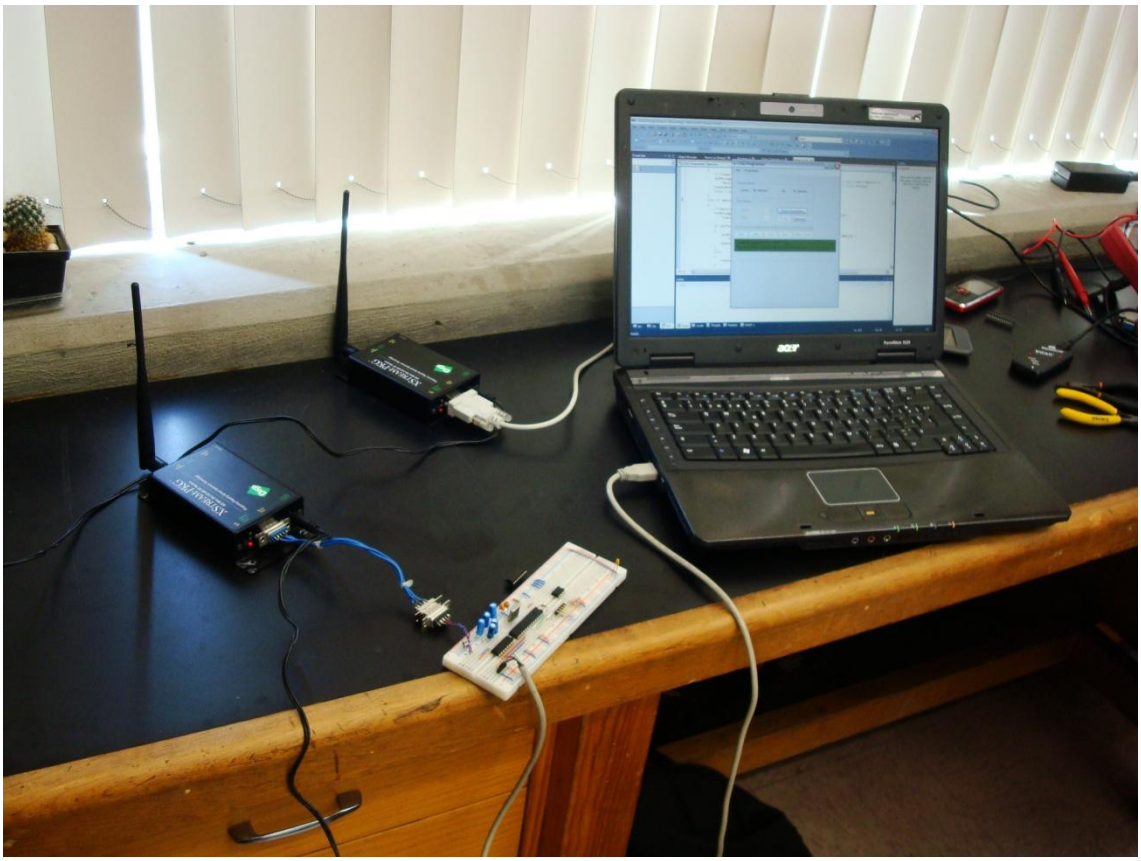


Figura 5.3. Validación del modulo de reconfiguración de radiomodems.

Por medio del uso de las herramientas de diseño y desarrollo mencionadas para la validación del firmware de la CV, del módulo de reconfiguración remota y del software de operaciones se realizó un trabajo concurrente en las tres vertientes del proyecto, sin necesidad de tener terminada totalmente una parte para poder continuar con la siguiente. Con esto se logra un mejor diseño modular, lo que significa un ahorro de tiempo y recursos.

Capítulo 6. Conclusiones y recomendaciones

6.1. Resultados

Se efectuó el diseño del prototipo de Computadora de Vuelo para el satélite HumSAT-México cumpliendo los objetivos planteados al inicio de su desarrollo. Se obtuvieron resultados en el ahorro de energía, ampliación de los recursos de hardware y en compatibilidad con los subsistemas de la plataforma satelital SATEDU.

Se sustituyo el microcontrolador principal de la plataforma satelital SATEDU SAB80C166 por el microcontrolador SAM3U4E de Atmel, basado en un núcleo Cortex 3 de 32 bits de ARM, con juego de instrucciones RISC, arquitectura Harvard y una alimentación de 3.3V para sus periféricos y de 1.8V para su núcleo.

Se mantiene compatibilidad en el canal de comunicación de la plataforma SATEDU vía RS-232, pero se incluye un segundo bus de comunicaciones para la plataforma HumSat-México que haga uso del protocolo TWI (I2C).

Se diseñó un sistema de reconfiguración remota basado en un microcontrolador PIC18LF2550 para realizar el cargado de nuevo *firmware* a través de un *bootloader* compatible con SAM-BA. Este

módulo de cargado también puede reconfigurar el controlador de la nueva tarjeta de Estabilización y Control de Apuntamiento basado en un FPGA a través de JTAG.

Se diseñó una interfaz en Visual C# que aprovecha la reutilización de código gracias a la implementación del paradigma orientado a objetos que permitirá la rápida migración de código a otros lenguajes como lo es C++ o Java si fuera necesario.

El diseño de la interfaz es modular, con lo cual se favorece su mantenimiento al separar la funcionalidad de la vista y particularmente, permitirá dotar a la interfaz de nuevas funcionalidades sin caer en un sistema monolítico de software, lo que la hace una opción muy versátil.

Se actualizó el *firmware* para la CV de SATEDU para que pudiera ser ejecutado en un procesador SAM3U4E con herramientas de código abierto, lo que permite reducir costos de producción.

6.2. Conclusiones

El diseño propuesto de Computadora de Vuelo para el satélite HumSAT-México cumple preliminarmente con los objetivos marcados al principio del proyecto.

Con este nuevo diseño de CV se plantea una base para una nueva plataforma satelital basada en SATEDU que mejora las prestaciones ofrecidas por este último al contar con un nuevo bus de comunicaciones en red, un mejor núcleo para procesamiento y menor consumo de energía.

La principal aportación es que al hacer uso de un procesador SAM3U4E se libera al usuario de realizar el proceso de cargado que anteriormente se realizaba en SATEDU para el microprocesador SAB80C166, el cual no contaba con memoria flash interna, por lo que la reconfiguración o cargado de programa era necesario cada vez que se iniciaba la operación del satélite.

6.3. Recomendaciones

Algunas mejoras para continuar con el trabajo futuro que repercutirá en una mejor operación de la CV en el satélite HumSAT-México son:

Primero, para mejorar el rendimiento del microcontrolador y aprovechar los recursos del mismo se propone el uso de un sistema operativo en tiempo real. Existen diferentes opciones tanto de

código abierto como de software comercial. Un RTOS permitirá un mejor manejo de memoria, de uso de CPU y división de procesos en hilos para evitar así el uso directo de interrupciones.

Para realizar pruebas en Tierra entre la tarjeta de CV y el software básico de operaciones puede sustituirse la interfaz RS-232 por un puerto USB. Esto permitirá que una vez cargado el bootloader, el sistema pueda ser actualizado desde el programa SAM-BA vía puerto USB en lugar del puerto serie, ya que este último está tendiendo a desaparecer de equipos portátiles principalmente. La otra forma de subir nuevo programa puede aprovechar la nueva tarjeta de comunicaciones de SATEDU que emplea el estándar Bluetooth, el cual está disponible en buena parte de las computadoras portátiles.

La UART dedicada a la tarjeta de comunicaciones y al bus principal debe ser sustituida por el bus TWI conforme se diseñen los nuevos módulos para el satélite HumSAT-México.

Apéndice A

Firmware para el microcontrolador PIC18F2520 del módulo de reconfiguración remota.

Archivo main.c

```
#include <18F4520.h>
#include "string.h"
#fuses HS, NOWDT, NOMCLR
#use delay(clock=20000000)
#use rs232(baud=19200, xmit=PIN_C6, rcv=PIN_C7)

// * * * * * * * * * * * * * * * * * * *
// Comunication constant
// * * * * * * * * * * * * * * * * * * *

#define ACK    0x06
#define NL     0x0A
#define CR      0x0D
#define XON    0x11
#define XOFF   0x13
#define NAK    0x15

#define SendAck()       printf("ACK\n");
#define SendNack()      printf("NACK\n");
#define StopSender()    printf("XOFF\n")
#define StartSender()   printf("XON\n")

#define OnLedProg()     output_high(LED_PROG);
#define OffLedProg()    output_low(LED_PROG);

#define OnLedBusy()     output_high(LED_BUSY);
#define OffLedBusy()    output_low(LED_BUSY);

// * * * * * * * * * * * * * * * * * * *
// Bomba de mensajes del servidor
// * * * * * * * * * * * * * * * * * * *
#define CHECK_COMUNICATION '='
#define IDLE_MODE          '&'
#define PROGRAM_MEMORY     '!'
#define READ_MEMORY        '"'
#define ERASE_MEMORY       '$'
#define VERIFY_MEMORY      '%'

//TO DO: Agregar más comandos
#define CANCEL                  '*'
```

```
#include "memory.h"
#INT_RDA
void isr_rad(void)
{
    disable_interrupts(INT_RDA);
}

void main()
{
       int8  message;
    init_ext_eeprom();

       do{
             message = getc();
        switch(message)
        {
             case CHECK_COMUNICATION:
                    SendAck();
                    break;
             case PROGRAM_MEMORY:
                    ProgramMemory();
                    break;
             case READ_MEMORY:
                    ReadMemory();
                    break;
             case ERASE_MEMORY:
                    EraseMemory();
                    break;
             case IDLE_MODE:
                  enable_interrupts(int_rda);
                  SendAck();
                  sleep();
                  break;
             case VERIFY_MEMORY:
                  VerifyMemory();
                  break;
             default:
                    SendNack();
        }
       }while (TRUE);
}
```

Archivo hex32

```
//////////////////////////////////////////////////////////////////////
// Archivo:   hex32.h
// Descripcion: Este archivo contiene la esctrucuta de datos y funciones
//              asociadas para manejar paquetes en formato IntelHex32.
//
// int8 InitPackage(Hex32 *package)      - Inicializar estructura Hex32.
// int8 Hex2Package(Hex32 *package)      - Inicializa una estructura Hex32 a
//                                         partir del contenido de su variable
//                                         stringHex32.
// int8 Bin2Package(Hex32 *package,      - Inicializa una estructura Hex32 a
//                  int8  lineType=1,      partir del contenido de su variable
//                  int8 count=0,          data y los parametros de la funcion.
//                  int32 address=0)
```

```
// int8 RecivePackage(Hex32 *package)    - Recibe un paquete IntelHex32 desde UART
//                                         y lo alamacena en la estructura Hex32.
// int8 SendPackage(Hex32 *package)      - Envia un paquete IntelHex32 a traves de
//                                         la UART.
// int8 atoi(char *s)                    - Convierte una cadena hexadecimal a una
//                                         variablede 8 bits.
//
// Liberacion:
//  - Ver 0.1 Implementacion de las funciones
//    - int8 InitPackage(Hex32 *package)
//    - int8 Hex2Package(Hex32 *package)
//    - int8 Bin2Package(Hex32 *package,
//                       int8  lineType=1,
//                       int8 count=0,
//                       int32 address=0)
//    - int8 RecivePackage(Hex32 *package)
//    - int8 SendPackage(Hex32 *package)
//    - int8 atoi(char *s)
//
//  - Ver 0.1b Implementacion de la funcion 09/03/2011

// LastUpdate: 13/07/2011
//*****************************************************************

#define RECORD_SIZE  0x10             // Tamaño del paquete estandar Intel Hex32
#define BUFFER_SIZE  0x41             // Tamaño del buffer binario

#define LINE_DATA    0x00             // Paquete  de datos
#define LINE_LAST    0x01             // Ultimo paquete
#define LINE_EXTEN   0x04             // paquete extendido

#define ERR_NONE     0x00
#define ERR_INVALID  0x00             // No se construyo el paquete
#define ERR_FORMAT   0x02             // No es un paquete que inicie con ':'
#define ERR_CHECKSUM 0x03                  // No coincide CS calculado con el del
paquete

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
// INTEL HEX32 Package
// - Esta estructura guarda un paquete de datos en formato
//   INTEL HEX32.
//   El paquete es decodificado y sus parametros son
//   almacenados en los elementos de la estructura.
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

typedef struct Hex32
{
      int8    stringHex32[BUFFER_SIZE];        //  Buffer  con  paquete  en  formato
IntelHEX32
      int8  data[RECORD_SIZE];           // Datos decodficados en binario
      int8  checksum;                    // Suma de compoibacion de errores
      int8  lineType;                    // Tipo de paquete
      int8  count;                       // Num de lementos en el paquete
    int32 address;                     // Direccion de inicio del paquete
    int8  error;                       // Codigo de error en el paquete
}Hex32;

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

```
// Funciones portotipo
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

void InitPackage(Hex32 *package);
int8 Hex2Package(Hex32 *package);
int8 Bin2Package(Hex32 *package,
                 int8  lineType=1,
                 int8 count=0,
                 int32 address=0);
int8 RecivePackage(Hex32 *package);
int8 SendPackage(Hex32 *package);
int32 SendPackage4(int32 address);
int8 atoi(char *s);

//*****************************************************************
// Nombre: InitPackage
// Argumentos: Hex32*
// Retorno: None
// Descripcion: Esta funcion dinicializa los elementos de la
//              estrucutara pasada como argumento a los valores:
//                stringHex32[] = ":00000001FF\0"
//                data[]   = 0xFF...0xFF;
//                checksum = 0xFF
//                lineType = 0x00
//                count    = 0x00
//                address  = 0x0000
//                error    = ERR_NONE
//*****************************************************************

void InitPackage(Hex32 *package)
{
       package->stringHex32[0]  = ':';
       package->stringHex32[1]  = '0';
       package->stringHex32[2]  = '0';
       package->stringHex32[3]  = '0';
       package->stringHex32[4]  = '0';
       package->stringHex32[5]  = '0';
       package->stringHex32[6]  = '0';
       package->stringHex32[7]  = '0';
       package->stringHex32[8]  = '1';
       package->stringHex32[9]  = 'F';
       package->stringHex32[10] = 'F';
       package->stringHex32[11] = null;

       package->data[0]  = 0xFF;
       package->data[1]  = 0xFF;
       package->data[2]  = 0xFF;
       package->data[3]  = 0xFF;
       package->data[4]  = 0xFF;
       package->data[5]  = 0xFF;
       package->data[6]  = 0xFF;
       package->data[7]  = 0xFF;
       package->data[8]  = 0xFF;
       package->data[9]  = 0xFF;
       package->data[10] = 0xFF;
       package->data[11] = 0xFF;
       package->data[12] = 0xFF;
       package->data[13] = 0xFF;
```

```
        package->data[14] = 0xFF;
        package->data[15] = 0xFF;

        package->checksum = 0xFF;
        package->lineType = 0x01;
        package->count    = 0x00;
        package->address  = 0x0000;
        package->error    = ERR_NONE;
}

//*********************************************************************
// Nombre: RecivePackage
// Argumentos: Hex32*
// Retorno: codigo de error
//
// Descripcion: Esta funcion recibe un paquete en formato IntelHex32
//              y lo decodifica en una estructura Hex32.
//*********************************************************************

int8 RecivePackage(Hex32 *package)
{
    int8 error;
    int8 index;
    int1 _continue;

    error = ERR_NONE;
    _continue = true;
    index = 0x00;

    //printf("Recibiendo paquete\n");

    // Escribir en stringHex32 mientras:
    //   - No sea haya recibido un paquete
    //   - No se cancele la operacion
    //   - Haya espacio en el buffer
    StartSender();
    do{
        package->stringHex32[index] = getc();

        //Paquete recibido
        if(package->stringHex32[index] == NL)
        {
               error = ERR_NONE;
               package->stringHex32[index] = null;
               _continue = false;

               //printf("paquete recibido\n");
        }
        //Operacion cancelada
        else if(package->stringHex32[index] == CANCEL)
        {
               error = ERR_COM_0;
              initPackage(package);
              _continue = false;
              //printf("operacion cancelada\n");
        }
        //Buffer desbordado
        else if(index >= BUFFER_SIZE)
```

```
        {
                error = ERR_COM_1;
                initPackage(package);
                _continue = false;
                //printf("se sobrepaso el tamaño del buffer\n");
        }
        index++;
    } while (_continue);
    StopSender();

    //Construir estructura Hex32
    if(error == ERR_NONE)
        error = Hex2Package(package);

    return error;
}

//**********************************************************************
// Nombre: SendPackage
// Argumentos: Hex32*
// Retorno: codigo de error
// Descripcion: Esta funcion envia un paquete en formato IntelHex32
//**********************************************************************

int8 SendPackage(Hex32 *package)
{
    int8 index;

    //Enviar un paquete valido
    if(package->error == ERR_NONE)
    {
        index = 0;
        while(package->stringHex32[index] != null)
        {
            putc(package->stringHex32[index++]);
        }
        putc(NL);  //Envio de salto de linea
    }

    return package->error;
}
//**********************************************************************
// Name: ReadPackage
// Parameters:
//         - package: Puntero a un arreglo entero donde se almacenaran los datos
//                    leidos de la memoria
//         - adress:  Es la direccion de inicio desde donde se llenara el arreglo
//                    package con un numero igual a MAX_LENG_PACKAGE de elementos
// Return: Codigo de error
// Description: Esta funcion obtiene un paquete de bytes desde la memoria
//**********************************************************************
int8 ReadPackage(Hex32 *package, int32 address)
{
    int8 error;
    int8 count;
    int1 last;   //Para detectar si el bloque esta vacio (igual a 0xFF)

    last = true;
```

```
    error = ERR_NONE;

    if( address >= EEPROM_SIZE )
    {
        error = Bin2Package(package);
    }
    else
    {
        //Leer un bloque RECORD_SIZE de la memoria
        for(count=0;  count  <  RECORD_SIZE  &&  address  <  EEPROM_SIZE;  count++,
address++)
        {
            package->data[count] = read_ext_eeprom(address);
            if(last == true && package->data[count] != 0xFF)
                last = false;
        }

        //Si todos los elementos de data son 0xFF
        if(last == true)          //Generar ultimo paquete
            error = Bin2Package(package);
        else
            error = Bin2Package(package,0, count, address-count);
    }
    return error;
}

//*********************************************************************
// Nombre: WritePackage
// Argumentos: Hex32*
// Retorno: codigo de error
// Descripcion: Esta funcion escribe un bloque de memoria
//              con los datos almacenados en la estructura Hex32
//*********************************************************************
int8 WritePackage(Hex32 *package)
{
    int8 index;
    int32 address;

    address = package->address;
    if(package->error == ERR_NONE)
        for(index=0; index<package->count; index++)
        {
           write_ext_eeprom(address, package->data[index]);
           //write_ext_eeprom(address,0xAA);
           //printf("a:%2X d:%2X\n",address,read_ext_eeprom(package->data[index]));
           address++;
        }
           // write_ext_eeprom(address++, 0xAA);

    return package->error;
}

//*********************************************************************
// Nombre: Bin2Package
// Retorno: codigo de error
// Descripcion: Esta funcion genera una estructura Hex32
//              a partir del contenido del arreglo data
//*********************************************************************
```

```
int8 Bin2Package(Hex32 *package,
                 int8 lineType = LINE_LAST,
                 int8 count    = 0x00,
                 int32 address = 0x0000)
{
    int8  buffer[5];
    int8  checksum;
    int8  error;
    int16 addressH;
    int16 addressL;
    int8  index;
    int8 _count;

    checksum = 0x00;
    index    = 0x00;
    error    = ERR_NONE;

    package->count    = count;
    package->address  = address;
    package->lineType = lineType;

    //Generar paquetes
    if(lineType == 1)
    {
        sprintf(package->stringHex32,":00000001FF\0");
        package->checksum = 0xFF;
    }
    else if(lineType == 0)
    {
        //calcular checksume
        package->stringHex32[index++] = ':';

        //Definir tamanio del paquete
           sprintf(buffer,"%2X",count);
           package->stringHex32[index++] = buffer[0];
           package->stringHex32[index++] = buffer[1];

           //Definir la direccion del paquete
        sprintf(buffer,"%4LX",address);
        package->stringHex32[index++] = buffer[0];
        package->stringHex32[index++] = buffer[1];
        package->stringHex32[index++] = buffer[2];
        package->stringHex32[index++] = buffer[3];

        //Definir tipo de paquete
        sprintf(buffer,"%2X",lineType);
        package->stringHex32[index++] = buffer[0];
        package->stringHex32[index++] = buffer[1];

        //Agregar datos al arreglo
        for(_count = 0; _count < count ; _count++)
       {
              sprintf(buffer,"%2X",package->data[_count]);
              package->stringHex32[index++] = buffer[0];
              package->stringHex32[index++] = buffer[1];
       }

        //Calcular checksum
```

```
        addressL = address & 0x00FF;
        addressH = address >> 8;
        addressH = addressH & 0x00FF;

        checksum = 0;
        checksum = count + lineType + addressH + addressL;

        for(_count = 0; _count < count; _count++)
        {
             checksum += package->data[_count];
        }

        checksum = 0xFF - checksum + 1;
        sprintf(buffer,"%2X",checksum);
        package->stringHex32[index++] = buffer[0];
        package->stringHex32[index++] = buffer[1];
        package->stringHex32[index++] = null;
    }
    else
    {
        //Error en el tipo de paquete
        package->error = true;
    }

    return error;
}

//**********************************************************************
// Nombre: Hex2Package
// Retorno: Codigo de error
// Descripcion: Esta funcion genera una estructura Hex32
//              a partir del contenido del arreglo stringHex32
//**********************************************************************

int8 Hex2Package(Hex32 *package)
{
    int8 error;
    int8 index;
    int8 indexData;

    error = ERR_NONE;

    if(package->stringHex32[0] == ':')
    {
        package->error = ERR_NONE;
        package->count = atoi(&package->stringHex32[1]);
        package->address    =    make16(atoi(&package->stringHex32[3]),atoi(&package-
>stringHex32[5]));
        package->lineType = atoi(&package->stringHex32[7]);

        if (package->lineType == 0x01)
        {
            //Agregar datos al arreglo igual a 0xFF
            for(index = 0; index < RECORD_SIZE ; index++)
              {
              package->data[index] = 0xFF;
              }
              package->checksum = 0xFF;
```

```
        }
        else
        {
            if(package->lineType == 0x00)
            {
                //Calculas checksume
                package->checksum = 0x00;
                indexData = package->count*2+8;
                for (index = 1; index< indexData ; index += 2)
                {
                    package->checksum += atoi(&package->stringHex32[index]);
                }
                package->checksum = ~package->checksum + 1;

                if(package->checksum != atoi(&package->stringHex32[index]))
                {
                    package->error = ERR_CHECKSUM;
                }
                else
                {
                    //Extraer los datos
                    for (indexData=0, index=9; indexData < package->count; index +=
2, indexData++)
                    {
                        package->data[indexData]          =          atoi(&package-
>stringHex32[index]);
                    }
                }
            }
        }
    }
    else
    {
        package->error = ERR_FORMAT;
    }
    return package->error;
}

int8 atoi(char *s) {  // Convert two hex characters to a int8
   int8 result = 0;
   int8 i;

   for (i=0; i<2; i++,s++)  {
      if (*s >= 'A')
         result = 16*result + (*s) - 'A' + 10;
      else
         result = 16*result + (*s) - '0';
   }

   return(result);
}

int32 SendPackage4(int32 address)
{
       char buffer[15];
       int32 offset;
       int8 sum;
       offset = address >> 16;
```

```
        offset &= 0x0000FFFF;
        sum = offset>>8;
        sum += offset & 0xFF;
        sum += 6;
        sum = 0 - sum;
        sprintf(buffer,":02000004%4X%2X",offset,sum);
        printf("%s\n",buffer);
}
```

Archivo memory.h

```
// * * * * * * * * * * * * * * * * * *
// Error of Comunication Type: 1
// * * * * * * * * * * * * * * * * * *

#define ERR_COM_0 0x10    // La recepcion fue cancelada
#define ERR_COM_1 0x11    //
#define ERR_COM_2 0x12
#define ERR_COM_3 0x13
#define ERR_COM_4 0x14
#define ERR_COM_5 0x15
#define ERR_COM_6 0x16
#define ERR_COM_7 0x17
#define ERR_COM_8 0x18
#define ERR_COM_9 0x19

#include "24lc512.h"
#include "hex32.h"

void ProgramMemory(void);
void ReadMemory(void);
void EraseMemory(void);

void err(int8 error)
{
        switch (error)
        {
                case ERR_FORMAT:

                    break;
        }
}

//*****************************************************************
// Name: ProgramMemory
// Parameters:
//         - package: Puntero a un arreglo entero donde se almacenaran los datos
//                    leidos de la memoria
//         - adress:  Es la direccion de inicio desde donde se llenara el arreglo
//                    package con un numero igual a MAX_LENG_PACKAGE de elementos
// Return: El valor de la ultima direccion leida
// Description: Esta funcion obtiene un paquete de bytes desde la memoria
//*****************************************************************
void ProgramMemory(void)
{
        Hex32 package;
        int1  _continue;
```

```
        int8  error;
        int32 offset;
        offset = 0;

        _continue = true;
        error = ERR_NONE;

        while(_continue == true)
        {
                //Recibiendo paquete
                error = RecivePackage(&package);

                //Ocurrio un error en la recepcion
                if( error != ERR_NONE )
                {
                        //Manejador de errores de programación
                        //printf("Ocurrio un error en la recepcion\n");
                        SendNack();
                        _continue = true;
                }
/*              else if( package.lineType == 4 )
                {
                //      obtener los datos -atoi-
                //      offset <- datos;
                //      offset << 16 & 0xFFFF0000

                }*/
                //Programacion terminada
                else if( package.lineType == 1 )
                {
                //printf("Programacion terminada\n");
                StartSender();
                SendAck();
                _continue = false;
         }
                //Se recibio un paquete valido
                else
                {
                //printf("Escribiendo paquete en la memoria\n");
                        package.address &= offset;
              error = WritePackage(&package);
                }
        }
}

//*********************************************************************
// Name: ReadMemory
// Parameters: Ninguno
// Return: ninguno
// Description: Esta funcion lee el contenido de la memoria
//*********************************************************************
void ReadMemory(void)
{
        Hex32 package;
        int1  _continue;
        int8  error, option;
        int32 nextAddress;
```

```
	_continue = true;
	error = ERR_NONE;
	nextAddress = 0x00;

	while( _continue )
	{
		if( !kbhit() )
		{
			error = ReadPackage(&package, nextAddress);	//leer paquete
de memoria
			//Generar paquete 4
			if(nextAddress % 0x10000 == 0)
			{
				SendPackage4(nextAddress);
			}
			// Si courrio un error
			if( error != ERR_NONE )
			{
			sendNack();
			_continue = false;
			}
			// Si se trata del ultimo paquete
			else if( package.lineType == 1 )
			{
			error = SendPackage(&package);
			sendAck();
			_continue = false;
			}
			// Se leyo un paquete valido
			else
			{
		  error = SendPackage(&package);		//Enviar paquete
		  nextAddress = package.address + RECORD_SIZE;	//Calcular la
siguiente direccion
			}
		}
		else
		{
		// Validar por cancelar
			option = getc();
			if(option == CANCEL)
			{
			SendNack();
			_continue = false;
		}
		  }
	}
}

//*****************************************************************
// Name: EraseMemory
// Parameters: Ninguno
// Return: ninguno
// Description: Esta funcion borra el contenido de la memoria
//*****************************************************************
void EraseMemory(void)
{
	int32 address;
```

```
	int8  _continue;
	int8  option;
	address = 0x00;
	_continue = true;

	while(_continue)
	{
		if(!kbhit())
		{
		write_ext_eeprom(address,0xFF);
		address++;
			if(address % 0x40 == 0x00 || address == 0x00)
			{
			StopSender();
			}

			if(address > EEPROM_SIZE)
			{
			//printf("Borrado de la memoria terminado\n");
			StartSender();
				SendAck();
				_continue = false;
			}
		}
		else
		{
			option = getc();
			if(option == CANCEL)
			{
			//printf("Operacion de borrado cancelada\n");
			StartSender();
				SendNack();
				_continue = false;
			}
		}
	}
}


void VerifyMemory(void)
{
	Hex32 RecPack;
	Hex32 SavedPack;
	int1  _continue;
	int8  error;
	int8 index;

	_continue = true;
	error = ERR_NONE;

	while(_continue == true)
	{
		//Recibiendo paquete
		error = RecivePackage(&RecPack);

		//Ocurrio un error en la recepcion
		if( error != ERR_NONE )
		{
```

```
                    //Manejador de errores de programación
                    //printf("Ocurrio un error en la recepcion\n");
                    SendNack();
                    _continue = true;
              }
              //Programacion terminada
              else if( RecPack.lineType == 1 )
              {
              //printf("Programacion terminada\n");
              //verificar que el siguiente registro sea vacio
              StartSender();
              SendAck();
              _continue = false;
        }
              //Se recibio un paquete valido
              else
              {
              //Leer un paquete de la memoria
            error = ReadPackage(&SavedPack, RecPack.address);      //leer paquete de
memoria
                    // Si courrio un error
                    if( error != ERR_NONE )
                    {
                    sendNack();
                    _continue = false;
                    }
                    // Si se trata del ultimo paquete
                    else
                    {
                    //Validar que todo sea chevere
                    index = 0;

                    do{
                           if(RecPack.stringHex32[index]                          ==
SavedPack.stringHex32[index])
                           {
                           _continue = true;
                           }
                           else
                           {
                           _continue = false;
                           SendNack();
                           }
                    }while(RecPack.stringHex32[index++]  !=  null  &&  _continue  ==
true);
                    }
              }
       }
}
```

Apéndice B

Este archivo muestra el código fuente del programa C# para la interfaz de cargado y depuración para el módulo de reconfiguración remota.

Archivo progran.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.IO.Ports;
using System.Threading;

namespace Debuger_EEPROM_Loader
{
    public partial class Form1 : Form
    {
        public bool _continue;
        public Form1()
        {
            InitializeComponent();
            _continue = false;
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            foreach (string s in SerialPort.GetPortNames())
            {
                cbPort.Items.Add(s);
            }
            cbPort.SelectedIndex = 0;
            cbBaudrate.SelectedIndex = 0;
            tabControl1.SelectedIndex = 0;
        }

        private void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {
            if (serialPort1.IsOpen)
                serialPort1.Close();
            _continue = false;
        }
```

```
        private void btEnviar_Click(object sender, EventArgs e)
        {
            if (lbBuffer.SelectedIndex < lbBuffer.Items.Count)
            {
                string buffer;
                buffer = lbBuffer.SelectedItem.ToString();
                serialPort1.WriteLine(buffer);

                if (lbBuffer.SelectedIndex < lbBuffer.Items.Count-1)
                    lbBuffer.SelectedIndex++;
                progressBar1.Increment(1);
                lbLog.Items.Add("Enviado: " + buffer);
            }
        }

        private         void         serialPort1_DataReceived(object          sender,
System.IO.Ports.SerialDataReceivedEventArgs e)
        {
            string buffer;
            buffer = serialPort1.ReadLine();
            lbLog.Items.Add("Recibido: "+buffer);
            lbLog.SelectedIndex = lbLog.Items.Count - 1;
            btClearLog.Enabled = true;

            if (buffer == "XON")
                _continue = true;
            else if (buffer == "XOFF")
                _continue = false;
        }

        private void btCheck_Click(object sender, EventArgs e)
        {
            serialPort1.Write("=");
            lbLog.Items.Add("Enviado: CHECK_COMUNICATION");
        }

        private void btWrite_Click(object sender, EventArgs e)
        {
            serialPort1.Write("!");
            lbLog.Items.Add("Enviado: PROGRAM_MEMORY");
        }

        private void btRead_Click(object sender, EventArgs e)
        {
            serialPort1.Write("\"");
            lbLog.Items.Add("Enviado: READ_MEMORY");
        }

        private void btClearBuffer_Click(object sender, EventArgs e)
        {
            lbBuffer.Items.Clear();
            btClearBuffer.Enabled = false;
            btEnviar.Enabled = false;
            btSendAll.Enabled = false;
        }

        private void btClearLog_Click(object sender, EventArgs e)
        {
```

```
        lbLog.Items.Clear();
        btClearLog.Enabled = false;
    }

    private void btLoadFile_Click(object sender, EventArgs e)
    {
        OpenFileDialog Dlg = new OpenFileDialog();
        if (Dlg.ShowDialog() ==  DialogResult.OK)
        {
            StreamReader sr = new StreamReader(Dlg.FileName);
            string buffer = "";
            do
            {
                buffer = sr.ReadLine();
                if(buffer != null)
                    lbBuffer.Items.Add(buffer);
            } while (buffer != null);
            sr.Close();
            //inicializar la barra de prograso
            progressBar1.Maximum = lbBuffer.Items.Count;
            lbBuffer.SelectedIndex = 0;
        }

        btClearBuffer.Enabled = true;
        btEnviar.Enabled = true;
        btSendAll.Enabled = true;
    }

    private void lbBuffer_SelectedIndexChanged(object sender, EventArgs e)
    {
        progressBar1.Value = lbBuffer.SelectedIndex;
    }

    private void btCancel_Click(object sender, EventArgs e)
    {
        serialPort1.Write("*");
        lbLog.Items.Add("Enviado: CANCEL");
        _continue = false;
    }

    private void btConect_Click(object sender, EventArgs e)
    {
        if (!serialPort1.IsOpen)
        {
            serialPort1.Open();
            btConect.Text      = "Desconect";
            btCheck.Enabled    = true;
            btWrite.Enabled    = true;
            btRead.Enabled     = true;
            btCancel.Enabled   = true;
            btInitARM.Enabled  = true;
            btInitRec.Enabled  = true;
            btIdle.Enabled     = true;
            btErase.Enabled    = true;
            btVerify.Enabled   = true;
            cbBaudrate.Enabled = false;
            cbPort.Enabled     = false;
            MessageBox.Show("Puerto: " + serialPort1.PortName
```

```
                + " abierto a " + serialPort1.BaudRate.ToString() +" bauds");
            tabControl1.SelectedIndex = 1;
            btLoadFile.Focus();
        }
        else
        {
            serialPort1.Close();
            btConect.Text      = "Conect";
            btCheck.Enabled    = false;
            btWrite.Enabled    = false;
            btRead.Enabled     = false;
            btCancel.Enabled   = false;
            btInitARM.Enabled  = false;
            btInitRec.Enabled  = false;
            btIdle.Enabled     = false;
            btErase.Enabled    = false;
            btVerify.Enabled   = false;
            cbBaudrate.Enabled = true;
            cbPort.Enabled     = true;
        }
    }

    private void cbPort_SelectedIndexChanged(object sender, EventArgs e)
    {
        if (cbPort.SelectedIndex > 0)
        {
            serialPort1.PortName = cbPort.SelectedItem.ToString();
        }
    }

    private void cbBaudrate_SelectedIndexChanged(object sender, EventArgs e)
    {
        if (cbBaudrate.SelectedIndex > 0)
        {
           serialPort1.BaudRate=Int32.Parse(cbBaudrate.SelectedItem.ToString());
        }
    }

    private void button1_Click(object sender, EventArgs e)
    {
        serialPort1.Write("&");
        lbLog.Items.Add("Enviado: IDLE MODE");
    }

    private void lbLog_SelectedIndexChanged(object sender, EventArgs e)
    {
        btClearLog.Enabled = true;
    }

    private void btErase_Click(object sender, EventArgs e)
    {
        serialPort1.Write("$");
        lbLog.Items.Add("Enviado: ERASE");
    }

    private void btWrite_Click_1(object sender, EventArgs e)
    {
        serialPort1.Write("!");
```

```
            lbLog.Items.Add("Enviado: WRITE");
        }

        private void btVerify_Click(object sender, EventArgs e)
        {
            serialPort1.Write("%");
            lbLog.Items.Add("Enviado: VERIFY");
        }

        private void btSendAll_Click(object sender, EventArgs e)
        {
            Thread sendAll = new Thread(SendAllThread);
            sendAll.Start();
        }

        private void SendAllThread()
        {
            string buffer;
            int index = 0;

            while (index < lbBuffer.Items.Count)
            {
                buffer = lbBuffer.SelectedItem.ToString();

                while (_continue == false) ;
                serialPort1.WriteLine(buffer);
                lbLog.Items.Add("Enviado: " + buffer);
                _continue = false;

                if (lbBuffer.SelectedIndex < lbBuffer.Items.Count-1)
                {
                    lbBuffer.SelectedIndex++;
                    progressBar1.Increment(1);
                }
                index++;
            }
            MessageBox.Show("Programación finalizada");
        }
    }
}
```

Referencias

Coporation, A. (octubre de 2006). *In-system Programmer (ISP):User Guide.* Recuperado el abril de 2010, de http://atmel.com

Coporation, A. (2006). *SAM Boot Asistant (SAM-BA): User Guide.* Recuperado el abril de 2010, de http://atmel.com

García Illescas, M. Á. (2009). *Actualización de una computadora de vuelo par aun satélite educativo.* Mexico DF: Universidad Nacional AUtónoma de México.

Gómez Islas, F. J. (2007). *Computadora de vuelo para un sistema de capacitación de recursos humanos en el manejo de satélites.* México DF: Universidad Nacional Autónoma de México.

Ingeniería, I. d., UNAM, & SATEDU. (2009). *Satélite Educativo Mexicano*. Recuperado el diciembre de 2009, de http://proyectos.iingen.unam.mx/satedu

University, A. (2001). Recuperado el junio de 2010, de AAU CUBE SAT: http://www.cubesat.auc.dk

University, A. (2005). *AAUSAT II.* Obtenido de http://aausatii.space.aau.dk/homepage/index.php

University, A. (2009). *AAUSAT3.* Recuperado el junio de 2010