UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE INGENIERÍA MECÁNICA

"DISEÑO, SIMULACIÓN E IMPLEMENTACIÓN DEL MÓDULO DE CONTROL CENTRAL Y MONITOREO DE INFORMACIÓN DEL NANOSATÉLITE DE INVESTIGACIÓN CHASQUI-I DE LA UNIVERSIDAD NACIONAL DE INGENIERÍA"

TESIS

PARA OPTAR EL TÍTULO PROFESIONAL DE: INGENIERO MECATRÓNICO

PROMOCIÓN 2011-I

LIMA-PERÚ

2012

Dedico la presente tesis a mi familia, mis padres Juan y Marina y a mis hermanos Mario y Lisseth quienes siempre se mostraron insistentes por mi desarrollo profesional. Así también quiero agradecer a todos los integrantes del proyecto nanosatélite Chasqui con quienes compartí tres años de investigación. Finalmente agradezco a los docentes de la UNI que siempre se han preocupado por formar buenos ingenieros para nuestro país.

INDICE

LISTA DE FIGURAS

LISTA DE TABLAS

Prólogo

Un satélite artificial es un objeto enviado al espacio para que cumpla una determinada función, principalmente transmisión de señales de datos. Independiente de su escala, un satélite está conformado internamente por 5 subsistemas principales o módulos. El primero tiene como objetivo el suministro de potencia usando paneles solares como fuente principal de energía, además de asegurar que la temperatura interna del satélite no llegue a valores que afecten su normal funcionamiento. El segundo subsistema es la carga útil, como es el caso de cámaras fotográficas. El tercero se encarga del control de orientación y estabilidad espacial del satélite. El cuarto es denominado sistema de comunicaciones y se encarga de que la información registrada pueda llegar desde el espacio exterior hacia una estación terrena. Finalmente existe una última, denominada control central y manejo de información (CCMI), encargada de hacer que todos los demás puedan interactuar eficientemente, también se encarga de administrar la información útil de cada módulo y almacenarla en memorias, además cumple la función de controlar el modo de operación del satélite, es este último subsistema de todo satélite el que se desarrolla en esta tesis.

El módulo CCMI debe ser robusto y capaz de solucionar problemas, generados en el espacio exterior, de forma autónoma. Ésta característica es obvia

debido a que todo satélite se encuentra a cientos de kilómetros de la Tierra y la intervención del hombre para alguna corrección o mantenimiento del sistema es imposible para satélites pequeños. Además, el módulo de CCMI debe ser altamente confiable, es decir, debe asegurarse que cumpla los objetivos. Éstas características son bastante importantes por los altos costos de puesta en operación de un satélite.

Actualmente, el avance de la microelectrónica ha permitido el desarrollo de una nueva generación de dispositivos de aplicaciones espaciales, el más importante para nuestra investigación es el microcontrolador MCU (*microcontroller unit*), el cual es un circuito integrado programable. Además del desarrollo de la microelectrónica, también se han desarrollado técnicas para los algoritmos que ejecutan los microprocesadores, estas técnicas permiten aprovechar al máximo las capacidades de dicho circuito integrado, haciendo que ejecute múltiples tareas, administrando sus prioridades y tiempos de ejecución. Un ejemplo del desarrollo alcanzado, a nivel de uso de recurso, está en los sistemas operativos en tiempo real denominados RTOS [1], bastante usados en aplicaciones donde al sistema se le exige correcciones en sus respuestas bajo ciertas restricciones de tiempo. En este proyecto se implementan dichas técnicas de programación para el desarrollo del CCMI del Chasqui-I, orientando el diseño de nuestros algoritmos a un RTOS. Esto es muy usado en los sistemas satelitales debido a que el CPU del microcontrolador debe ejecutar muchos procesos simultáneamente. El orientar nuestro trabajo a un RTOS nos permite ejecutar procesos en función de sus prioridades y tiempos de ejecución. Además de implementar nuestro código de forma más ordenada y comprensible,

facilitando así los continuos cambios de nuestros algoritmos y la documentación del mismo.

Los satélites deben registrar información útil para los usuarios, ésta contiene datos que dependen de la misión del satélite, por ejemplo la toma de fotografías de la Tierra, o la adquisición de datos de sensores de temperatura. Debido a que el tiempo de operación del satélite en el espacio va desde varios meses hasta decenas de años, la información a almacenar es abundante, por lo que se emplean dispositivos de almacenamiento de información digital denominados “Memorias” para guardar los datos. Un ejemplo de estos dispositivos es la tarjeta de memoria SD (*secure digital*) [2] que es controlada desde el módulo de CCMI.

Se ha desarrollado también en estos últimos años, dispositivos altamente eficientes y con un muy bajo consumo de potencia eléctrica, esto es muy beneficioso para aplicaciones espaciales debido a que la única fuente de energía se da a través de paneles solares. El uso de estos dispositivos hace que la energía sea un recurso mejor aprovechado. Así, cuando el satélite tenga una baja energía se busca establecer un modo de muy bajo consumo para poder recargar nuevamente su sistema de almacenamiento de energía.

Los satélites antes de ser enviados al espacio, usando cohetes denominados *lanzadores,* deben cumplir con certificaciones internacionales exigidas por dichos cohetes lanzadores. Estas certificaciones están principalmente relacionadas los componentes electrónicos empleados, el comportamiento de éstos bajo condiciones de vacío térmico y el comportamiento de todo el nanosatélite sometido a vibraciones mecánicas. Estas pruebas se realizan en laboratorios especializados [3].

Otro aspecto que se tiene en cuenta para el desarrollo de los algoritmos del módulo de CCMI, es la determinación de los modos de operación del nanosatélite. Estos modos de operación indican etapas del funcionamiento del sistema; habitualmente se definen usando un diagrama de estados. Actividades a ser incluidas en el diagrama de estados y que permitirán la definición de los algoritmos de todos los modos de operación del nanosatélite Chasqui-I [4] son el lanzamiento al espacio, despliegue de las antenas, transmisión de datos, toma de fotografía de la Tierra, orientación espacial, control de la energía del sistema, lectura de sensores, etc.

Capítulo 1

1 Introducción

La presente tesis trata sobre el diseño, simulación e implementación del módulo de control central y monitoreo de información, llamado módulo CCMI, del proyecto “Nanosatélite de investigación Chasqui-I”. Este módulo tiene por objetivo el desarrollo de capacidades en la UNI [5] en el área de tecnologías satelitales a través del diseño, análisis, ensamblaje, integración, prueba, lanzamiento, y operación de un nanosatélite con tecnología *CubeSat* [6]. El nanosatélite Chasqui I tomará fotografías de la Tierra y colectará datos de sensores para luego transmitirlas a dos estaciones terrestres, una ubicada en las instalaciones del CTIC-UNI [7] y otra en el INICTEL-UNI [8].

El módulo CCMI gestiona y monitorea la información entre todos los subsistemas del Chasqui-I. Para cumplir con estos objetivos el Chasqui-I debe tener dentro de su circuito electrónico un microprocesador, el cual debe ser capaz de manejar eficientemente los datos y los protocolos de comunicación que servirán para la integración de todas las tarjetas que conforman el nanosatélite. El diseño del

módulo CCMI consiste en el desarrollo del correspondiente software y hardware, los cuales deberán cumplir exigencias propias de los equipos espaciales para asegurar el funcionamiento del nanosatélite y además evitar poner en riesgo al lanzador del nanosatélite.

La presente tesis está dividida en 8 capítulos, los cuales son detallados a continuación:

Capítulo 1: Se describe puntualmente el trabajo desarrollado en esta tesis, además se describe brevemente el entorno del sistema donde funcionará el Chasqui y la forma en la que éste interactúa con los demás subsistemas. Con esta información se definen los objetivos de la tesis. Además se muestran los antecedentes del trabajo desarrollado.

Capítulo 2: Este capítulo se centra en explicar el planteamiento del problema, este punto es importante ya que el sistema desarrollado en esta tesis tendrá aplicación espacial, por ello los requerimientos del hardware y software son bastante exigentes, éstos últimos también son detallados en este capítulo.

Capítulo 3: Se muestra el estado del arte del proyecto, en este aspecto hay cientos de proyectos a nivel mundial bastante interesantes basados en un mismo estándar. En este capítulo se muestran los principales proyectos que han servido de base para el desarrollo del nanosatélite Chasqui.

Capítulo 4: Se muestra el marco teórico que servirá para comprender toda la tesis, por ello los conceptos mostrados se centran únicamente en lo que se usó en el desarrollo de la tesis.

Capítulo 5: Se trata sobre todos los subsistemas del Chasqui, ya que la mayoría de éstos interactúan con el módulo CCMI. Además que todos los módulos son mencionados en varias partes de la tesis.

Capítulo 6: Se muestra el proceso de diseño e implementación del software. Además se muestran los diagramas de flujo que debe ejecutar el CCMI y los modos de operación del nanosatélite, los cuales son controlados también por CCMI.

Capítulo 7: Se muestra el proceso de diseño e implementación del hardware, se explican los planos, selección de componentes, técnicas para la manufactura de la tarjeta y consideraciones para la integración de todos los subsistemas con la tarjeta CCMI.

Capítulo 8: Se muestran las pruebas de funcionamiento del módulo CCMI, del software y del hardware del mismo, estas pruebas principalmente son: funcionamiento de CCMI sometido a vacío térmico, prueba de globo cautivo y prueba de tiempos para el RTOS del MCU.

1.1 Objetivos

1.1.1 Objetivo principal

- Diseñar, simular, implementar y probar el módulo de Control Central y Monitoreo de Información CCMI para el nanosatélite Chasqui-I de la UNI

1.1.2 Objetivos específicos

- Determinar los requerimientos para el diseño del hardware y software del módulo CCMI.
- Establecer las características de la comunicación entre el módulo CCMI y los otros que conforman el nanosatélite Chasqui-I.
- Determinar las funciones de control con los demás módulos
- Establecer los modos de operación del nanosatélite Chasqui-I desde el momento de su lanzamiento, considerando todos los posibles estados del mismo.
- Seleccionar los componentes electrónicos para el diseño del módulo CCMI y hallar el costo total para la fabricación de una tarjeta CCMI.
- Probar la operación del módulo CCMI sometido a vacío térmico y globo cautivo.

1.2 Antecedentes

El nanosatélite Chasqui-I, es un proyecto de la UNI sin precedentes en el país, lo cual explica la falta de especialistas con experiencia en esta área. Este proyecto nace como un proyecto de investigación dentro de las diversas áreas de interés del Centro de Tecnologías en Información y Comunicaciones CTIC. Éste centro ha venido desarrollando proyectos en muchas áreas de las Tecnologías en Información y Comunicaciones TICs, como son los sistemas embebidos, antenas, tratamiento de señales, procesamiento de imágenes, etc, sin embargo era necesario desarrollar un proyecto que sea capaz de incluir diversas áreas para así potencializar capacidades de los investigadores y ampliar la visión de los mismos desarrollando un proyecto multidisciplinario como lo es el proyecto nanosatélite de investigación Chasqui I. El

proyecto Chasqui-I cuenta con la participación de estudiantes de diversas áreas de la ingeniería, como son estudiantes de ingeniería mecánica, electrónica, mecatrónica, telecomunicaciones, sistemas, física, etc.

Analizando la experiencia adquirida en el proyecto nanosatelital desarrollado por la Universidad de *Standford* [9], quienes desarrollaron un nanosatélite de pequeñas dimensiones y de bajo costo, ideal para la investigación en diversas áreas de la ingeniería orientada al desarrollo de las TICs, se consideró conveniente desarrollar un nanosatélite de diseño propio denominado Chasqui-I.

El estándar CubeSat, ver Subsección 3.1.1, recomienda la existencia de un subsistema de control central capaz de monitorear y gestionar la información del satélite, por ello el módulo CCMI se crea desde el inicio del proyecto Chasqui para cumplir con el estándar.

1.3 Alcances

El proyecto satelital Chasqui-I busca desarrollar capacidades para el desarrollo de tecnología satelital en la UNI, con este objetivo se forman grupos que investigan cada módulo del nanosatélite en base a la formación recibida en sus respectivas facultades.

El proyecto mostrado en esta tesis desarrolla el módulo CCMI, el cual será capaz de gestionar todos los sub-sistemas del nanosatélite de investigación Chasqui-I, además gestionará el almacenamiento de información de los datos de los sensores y fotografías capturadas desde el nanosatélite, el módulo CCMI será sometido a vacío térmico de 0.02 mbar y temperaturas de -20°C a 80°C.

El módulo CCMI almacenará 50 fotos en su memoria, una vez alcanzada esta cantidad, las fotos más antiguas serán remplazadas por las nuevas. Caso similar se da con los datos de sensores, ya que cuando un bloque de datos sea transmitido a Tierra, la memoria que ocupaba será liberada, si dicho bloque no se transmite y la memoria continúa llenándose entonces la información de dicho bloque de memoria será remplazado por información actualizada, Para garantizar una buena tomas de fotos se harán pruebas de globo cautivo a más de 150 m.

Será implementado también la lógica de control de los modos de operación del nanosatélite, los cuales son: modo lanzamiento, modo espera, modo normal y modo emergencia.

1.4 Limitaciones

Debido a que nuestro nanosatélite es desarrollado con fines de investigación, las limitaciones son muchas comparadas con los satélites convencionales, el nanosatélite Chasqui no cumplirá ninguna función de servicio social, comercial o militar.

El nanosatélite tendrá un tiempo de vida promedio de 2 meses, debido a las limitaciones de las baterías empleadas en el sistema.

Las fotografías tomadas tendrán una resolución de 640x480, según cálculo realizado por el módulo SIMA. En una foto tomada por el nanosatélite Chasqui cada pixel corresponde a 1 km^2 de imagen terrestre aproximadamente, esta información no es útil para las aplicaciones típicas satelitales.

Debido a los recursos con los que contamos en la UNI, las pruebas se limitan a condiciones de vacío térmico con temperaturas que van de -20°C a 80°C y 0.02 mbar. Se debe tener en cuenta que las pruebas exigidas a nivel internacional no necesariamente son las mismas, ver Sección 5.13. Además el módulo CCMI no será sometido a la prueba de radiación, ya que tampoco se cuenta con un laboratorio que nos permita desarrollar dicha prueba, se espera solucionar posibles problemas de radiación mediante el software de CCMI.

Los proyectos satelitales necesitan ser ensamblados en laboratorios acondicionados especialmente para evitar contaminación con partículas aéreas en suspensión. En nuestro caso tampoco se cuenta con dicho ambiente de trabajo.

1.5 Importancia

La inclusión en un satélite de un módulo encargado únicamente de controlar y gestionar la información es importante ya que simplifica en gran medida el trabajo de los demás módulos, los cuales se dedicarán a ejecutar sus tareas únicamente cuando CCMI se los indique, esta forma de trabajo se semejanza a la configuración *Master-Slave* de los sistemas embebidos, esta configuración reduce el riesgo de fallas de software en el sistema ya que el *Master* tiene mayor jerarquía y puede dar órdenes a los demás. El módulo CCMI incluso es capaz de solucionar problemas en los demás módulos.

1.6 Método de solución

Para la solución del problema primero se decidió hacer un diseño básico del módulo, al cual llamaremos CCMI v1, éste desarrollará funciones que sean posibles en relación a la disponibilidad de componentes y limitación de conocimientos

obtenidos hasta el momento de su fabricación. Posteriormente se irán añadiendo funcionalidades al sistema y además se irán cambiando y añadiendo componentes si es que son necesarios para dar fruto a CCMI v2, así se irán haciendo nuevas versiones hasta llegar a cumplir con el objetivo final del módulo.

Para agilizar el aprendizaje se asistió a capacitaciones realizadas por investigadores nacionales e internacionales, esto fue gestionado por el módulo MGP, ver Subsección 4.4.10, del proyecto Chasqui.

Capítulo 2

2 Planteamiento del problema

2.1 Formulación del problema

El módulo de control central y manejo de información (CCMI) debe ser capaz de almacenar datos relevantes para el sistema y gestionar los protocolos de comunicación con los demás módulos que integran el nanosatélite Chasqui. Para lograr esto, se debe diseñar el hardware y software del módulo CCMI, el cual debe cumplir muchos requerimientos generales del sistema.

Para el caso del hardware el problema consiste en la selección de los componentes adecuados para almacenamiento de datos y control del sistema, estos componentes deben funcionar en modo de ultra bajo consumo de energía; se debe cumplir además exigencias dimensionales, funcionamiento bajo condiciones de vacío térmico y vibraciones mecánicas con el objetivo de simular, hasta cierto grado, el ambiente espacial. Otro problema del hardware se origina al momento de la integración de todo el nanosatélite, debido a las limitaciones de tamaño del nanosatélite, motivo por el cual surge la necesidad de incluir el circuito del sistema

de adquisición de imágenes en la tarjeta del módulo CCMI del Chasqui. En otras palabras, el diseño de la tarjeta de control central debe incluir el circuito de cámaras con el objetivo de cumplir las exigencias dimensionales.

Para el caso del software, el problema consiste en el diseño del algoritmo que ejecutará nuestro controlador, este debe considerar los modos de operación del nanosatélite, los cuales son: lanzamiento, espera, normal y emergencia, además debe implementar los protocolos de comunicación que permitan leer los datos para luego almacenarlos en una memoria y además manejar el flujo de comandos con los demás módulos del Chasqui.

Los protocolos de comunicación deben permitir la seguridad y eficiencia del flujo de información, este punto es importante ya que para el caso de la presente tesis, se debe implementar todos los protocolos necesarios para el correcto funcionamiento de todo el sistema y su interacción con el módulo CCMI, incluso se deben diseñar protocolos internos entre módulos, como son el SPI o I2C, sino a niveles más altos de software, es decir, deberá establecerse los comandos para transmitir y recibir datos, la velocidad de la transmisión misma, el formato de las tramas, su reacción ante una posible pérdida de comunicación, la detección de errores, etc.

En aplicaciones satelitales de este tipo se exige que el sistema cuente con triple nivel de contingencia [10], tanto para el hardware como para el software, es decir, todos los módulos del nanosatélite deben estar preparados ante posibles fallas de algún dispositivo o del algoritmo del sistema.

2.2 Diagrama de bloques del nanosatélite Chasqui

La intención de esta sección es mostrar el contexto de funcionamiento del módulo CCMI dentro del nanosatélite. Como vemos en la Figura 2-1, el módulo CCMI interactúa activamente con otros cuatro módulos, internos a la estructura; la conexión eléctrica existente entre ellos se representa mediante las líneas etiquetadas como *power bus, control bus, camera bus*.

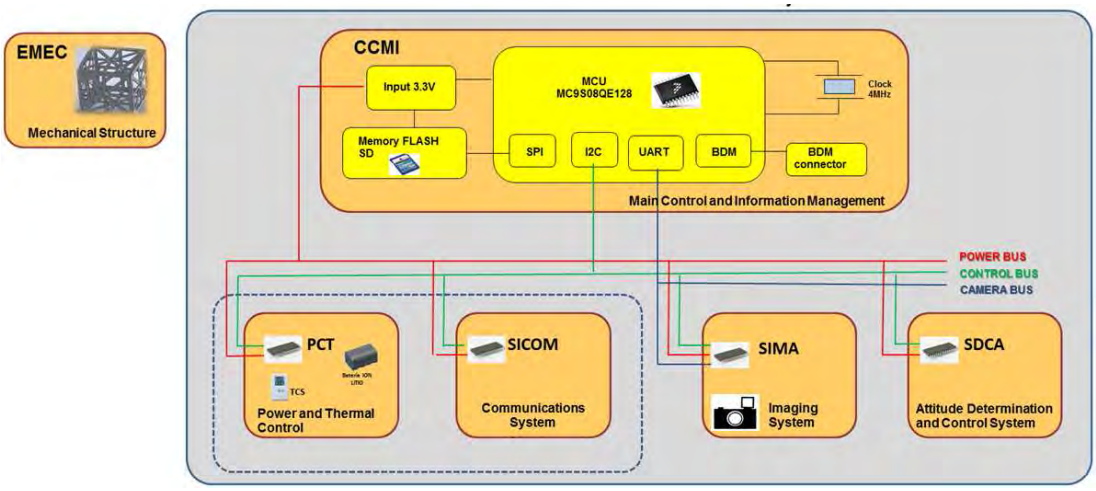


Figura 2-1. Diagrama de bloques general

La funcionalidad de cada módulo será detallada ampliamente en la sección 4.4. A continuación se presenta un resumen básico de cada módulo que integra el nanosatélite.

El módulo CCMI gestiona y monitorea la información entre todos los subsistemas del Chasqui I. Este módulo para cumplir los objetivos planteados debe tener dentro de sí un procesador (llamado OBC: *On Board Computer*), el cual realiza las siguientes tareas en relación a cada modulo:

- Cámara (SIMA): Ordena la captura de imágenes satelitales y su almacenamiento en una memoria externa.
- Actitud (SDCA): Ordena y confirma solicitud de estabilización y orientación espacial.
- Potencia (PCT): Gestiona los estados del nanosatélite y monitorea variables físicas como temperatura, voltaje y corriente.
- Comunicaciones (SICOM): Recibe órdenes de estación terrena y envía la información de datos de cámara, sensores y estados del nanosatélite

2.3 Requerimientos del sistema

El nanosatélite debe cumplir requisitos de hardware y de software, los cuales son bastante exigentes y buscan asegurar el funcionamiento del nanosatélite en el espacio, sometido a las condiciones propias del ambiente de operación.

2.3.1 Requerimientos del hardware

Incluye los requisitos de la forma física de la tarjeta, la característica de los componentes electrónicos y las pruebas que debe cumplir.

2.3.1.1 Requerimientos generales del Módulo CCMI

La Tabla 2-1 muestra los requerimientos iniciales del módulo, estos fueron determinados teniendo en cuenta el estándar *CubeSat* [6], el cual menciona que el nanosatélite debe tener forma de cubo de 10 cm de arista. Para cumplir con esta exigencia y teniendo en cuenta el posicionamiento de algunos componentes en las caras internas al cubo, se acordó que cada módulo se limitaba a un área de diseño

electrónico de 85 mm^2. Otra condición del estándar *CubeSat* es la masa del nanosatélite, la cual no debe sobrepasar un 1 kg. Este requerimiento del peso no será problema en la tarjeta CCMI ya que está compuesta de dispositivos pequeños tipo superficial, éstos se caracterizan por su bajo peso. Para la selección del material de la tarjeta CCMI se eligió fibra de vidrio tipo FR4 ya que fue usada en otros proyectos satelitales y cumple con las recomendaciones de la ESA [10]. Para la electrónica, todos los dispositivos deben seleccionarse en base a su consumo energético promedio, estos son detallados en las hojas técnicas de los fabricantes. Para el caso del procesador de CCMI se decidió emplear uno de 8 bits en modo *low power* y baja frecuencia de funcionamiento, ya que así se logra un mínimo consumo, además debido a que CCMI tendrá solo la función de gestionar el funcionamiento de los demás módulos, lo cual no implica cálculos complejos, por ello se eligió un procesador sencillo. El consumo energético es un factor importante para el buen funcionamiento del satélite, ya que se tiene una única fuente de energía basada en paneles solares, por ello cada módulo debe limitarse a un consumo de energía cuyo valor fue establecido según estudio del módulo de potencia y control térmico – PCT [11], para el caso del módulo CCMI el valor promedio máximo es de 80 mW.

Tabla 2-1: Requerimientos generales del módulo CCMI

Requerimientos generales para diseño del módulo CCMI	
Área	85 mm^2
Masa límite	100 g
Material base de tarjeta	Fibra de vidrio FR4
Procesador	8 bits, *low power*
Voltaje	3.3 V
Frecuencia de trabajo	4 MHz
Consumo máximo de energía	80 mW

2.3.1.2 Funcionalidad del módulo CCMI sometido a vacío térmico

Además de cumplir con las características mencionadas en el punto anterior, cada tarjeta debe funcionar estando sometida a vacío térmico, esto debido a que las condiciones del ambiente de operación de los satélites se caracterizan por presentar alto grado de vacío, es decir presión muy baja, 10^{-7}mbar. Además debido a las condiciones de temperaturas, las cuales varían en un rango considerablemente alto, para nuestro caso dichas temperaturas de operación del módulo CCMI deben estar entre -20 hasta 80°C. Estos datos fueron proporcionados por el módulo de integración y pruebas [3]. Cumpliendo estos requerimientos podemos tener mayor confianza de que nuestro sistema pueda funcionar en condiciones espaciales, por ello es parte del estándar que comparten las instituciones de fabricación satelital.

2.3.1.3 Funcionalidad del módulo CCMI sometido a vibraciones mecánicas

Cuando el nanosatélite es transportado al espacio, se usan los cohetes como medio de transporte, estos cuando salen de la atmósfera, generan vibraciones mecánicas las cuales afectan a la carga útil, incluyendo a los satélites que son transportados en su interior, es por ello que el diseño mecánico de todo el nanosatélite debe soportar el rango de frecuencias de vibración propias de cada cohete lanzador. El módulo MIP desarrolló un vibrador mecánico, cuya frecuencia aplicada al nanosatélite puede ser programada, todo el nanosatélite debe soportar las frecuencias requeridas por el lanzador.

2.3.2 Requerimientos del software

El software del módulo CCMI debe ser altamente confiable, ya que se encarga de gestionar el funcionamiento de los demás módulos. Para asegurar esto se debe desarrollar un software sencillo pero que incluya un sistema de seguridad contra errores de funcionamiento, esto se conseguirá habilitando el módulo de protección *watchdog*, incluido en el microcontrolador.

Un problema del ambiente de trabajo de los sistemas digitales para satélites es el nivel de radiación espacial, el cual es capaz de alterar los valores lógicos de las señales de corriente, por ello el sistema puede comportarse de forma extraña ya que las señales digitales que se transmitan serán alteradas por la radiación. Para prevenir este problema el programa debe incluir triple nivel de protección en partes estratégicas del software.

La velocidad de la ejecución de las instrucciones del software debe ser la mínima posible, ya que en general los sistemas digitales consumen mayor energía cuando trabajan a altas frecuencias de *clock*.

El módulo CCMI, además de su sistema interno, debe atender a los demás módulos, para cumplir eficientemente esta tarea debe dedicar el tiempo adecuado a cada uno de estos, además existen funciones que deben ejecutarse a tiempos relativamente precisos, una de estas funciones es la de pasar a modo bajo consumo cuando no tenga tareas que realizar. Para cumplir con estos requerimientos se debe orientar el diseño del software hacia un modelo de sistema operativo en tiempo real para sistemas embebidos.

El módulo CCMI también debe almacenar los datos enviados por cada módulo y los adquiridos el él mismo, esto incluye datos de sensores y de fotografías, dichos datos se actualiza constantemente durante todo el tiempo de operación del nanosatélite. Esta actividad implica la programación de un organizador de memoria que continuamente borre los datos que considere innecesarios y los remplace por datos actualizados o de mayor interés. Cada módulo requiere cantidades distintas de memoria y que también debe tenerse en cuenta.

Capítulo 3

3 Estado del arte

3.1 Experiencias en el Mundo

La aparición de proyectos satelitales basados en el estándar *CubeSat*, ver Subsección 3.1.1, han motivado a muchas instituciones de investigación a desarrollar nanosatélites, ya que dicho estándar permite desarrollar nanosatélites de pequeñas dimensiones y de bajo costo. Estas características hicieron que muchas instituciones de investigación se involucren en el área satelital, generando así una comunidad científica a nivel mundial que busca el desarrollar los *CubeSat* para fines de investigación. Además se busca simplificar aún más los diseños para que puedan ser implementados por estudiantes universitarios e incluso escolares, tal es el caso del proyecto *CanSat* [12] originado en Japón.

La Tabla 3-1 muestra algunos de los tantos nanosatélites cuyo funcionamiento fue exitoso, también se muestra el nombre del nanosatélite, quienes lo fabricaron, cuál fue su misión en el espacio, la fecha de lanzamiento y el nombre del cohete lanzador, notamos que en este último varios nanosatélites coinciden, esto es debido a que los lanzadores suelen llevar varios nanosatélites como carga.

Tabla 3-1: Proyectos basados en el estándar CubeSat de funcionamiento exitoso.

Nombre	Organización	Misión	Lanzamiento	Lanzador
Compass-I	FH-Aachen	Demostración de funcionamiento de componentes y toma de fotos.	28/04/2008	PSLV-CA
Cute-1.7 + APD II	Instituto tecnológico de Tokio	Demostración del sistema y prueba de sensor experimental de fotodiodo	28/04/2008	PSLV-CA
AAUSAT-II	Universidad de Aalborg de Dinamarca	Prueba sistema ADCS y detector gamma ray	28/04/2008	PSLV-CA
Delfi-C3	Universidad Tecnológica de Delft, Países Bajos	Testeo de celdas solares y sensor de sol inalámbrico	28/04/2008	PSLV-CA
KySat-I	Kentucky Space	Educacional, testeo de tecnologías	04/03/2011	Taurus-XL
Quakesat	Universidad de Stanford	Detección de terremotos	30/06/2003	2003-031F
GeneSat-1	NASA y Universidad de santa Clara	Demostración de la tecnología e investigación biológica	16/12/2006	Minotaur
CP-4	Universidad politécnica de california	Demostración del sistema y prueba de telemetría	17/04/2007	Dnepr-1

3.1.1 El estándar CubeSat – EEUU

Las universidades siempre han tenido las limitaciones de investigar en tecnologías satelitales por los altos costos que implica, es por ello que la universidad de *Standford* [9]logró diseñar un primer modelo de nanosatélite miniaturizado al cual denominaron *CubeSat* [6]. Este nanosatélite se caracteriza por sus pequeñas dimensiones y bajo costo, para luego formar parte de un estándar del cual muchas instituciones educativas, principalmente universidades, han tomado como modelo para el desarrollo de tecnología satelital.

Basados en el estándar *CubeSat* se implementó un módulo educativo denominado *Cubesat kit* [13], el cual cuenta con la estructura y toda la electrónica necesaria para pruebas de funcionamiento en laboratorios. Pronto algunas

universidades adquirieron dicho *kit* y comenzaron a desarrollar sus nanosatélites, cada uno con sus propias aplicaciones pero siempre cumpliendo el estándar *CubeSat*. Las aplicaciones de los nanosatélites están siendo cada vez más útiles para la comunidad científica y en muchos casos es más práctico en uso de los nanosatélites en lugar de los nanosatélites convencionales.

La Figura 3-1 muestra la estructura mecánica incluida en el *CubeSat Kit*, el cual es comercializado desde su página web, donde también se muestran los planos de los diseños mecánicos, esta estructura cumple el estándar *CubeSat*.

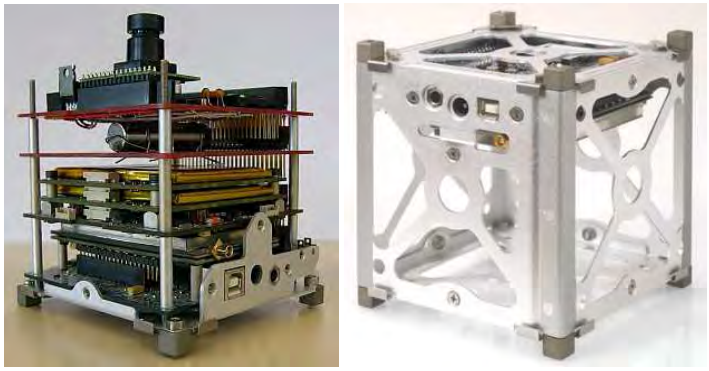


Figura 3-1. Modelo de la estructura de un CubeSat desarrollado por CubeSat Kit

El estándar *CubeSat* ofrece las siguientes características:

- Diseño estándar de la estructura, tarjetas y pautas de diseño.
- Diseño del mecanismo del lanzador P-POD.
- Documentación necesaria de funcionamiento y licencias.
- Envío del equipo al lugar de lanzamiento y la integración de LV.
- Verificación de la integración y la información de la telemetría

La tarjeta de control central del *CubeSat Kit*, ver Figura 3-2: Tarjeta de control central del CubeSat Kit., hace uso de un MCU de *Texas Instrument* TI de

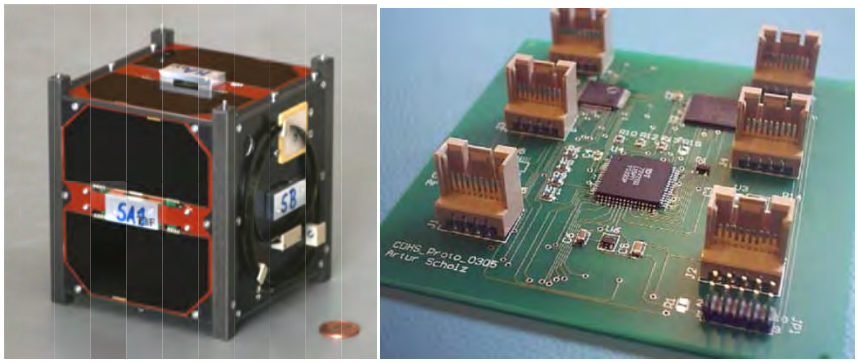
32bits, el cual se conecta con un adaptador a la tarjeta, además cuenta con un bus común tipo PC104 de 4x26 pines y opción de alimentación externa y por batería.

Figura 3-2: Tarjeta de control central del CubeSat Kit.

3.1.2 El proyecto Compass-I – Alemania

El proyecto Compass-I [14], ha servido de buena referencia para el desarrollo del proyecto Chasqui gracias a la buena documentación de sus trabajos, cada módulo del nanosatélite representa una tesis publicada en la página web del Compass-I [14].Para el caso de la tarjeta de control central, el diseño de su tarjeta, ver Figura 3-3, difiere bastante en forma a la tarjeta de control central del CCMI del Chasqui.

La tarjeta de control central del Compass-I emplea 6 conectores separados para cada módulo, un conector de grabación y una memoria tipo NAND-flash con su driver controlador que hace de interfaz entre el MCU y la memoria.

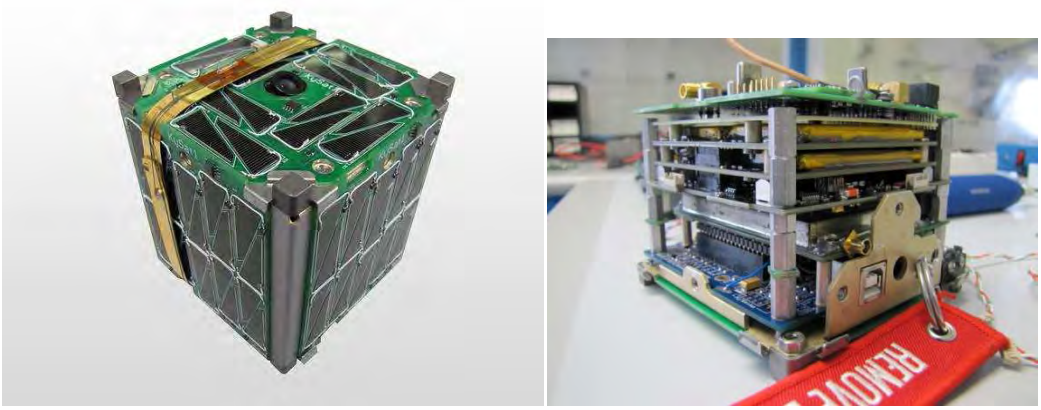


(a) *(b)*

Figura 3-3:(a) Nanosatélite Compass-I de Alemania. (b) Tarjeta del sistema de Control y manejo de datos.

3.1.3 El proyecto KySat – EEUU

Este proyecto KySat [15]está basado en el Cubesat-kit, tiene como misión principal la formación de capacidades en varias universidades de Kentucky. El nanosatélite KySat usa la misma tarjeta de control central del *CubeSat kit* como base de todas las demás tarjetas, esto lo podemos ver en la Figura 3-4(a), donde la tarjeta de control, de color azul, se ubica como base conectada a las demás por su bus PC104. Este nanosatélite fue diseñado para poder ser activado manualmente, es decir no usará el P-POD, por ello lleva una cinta roja en el *switch* de encendido, ver Figura 3-4(b).

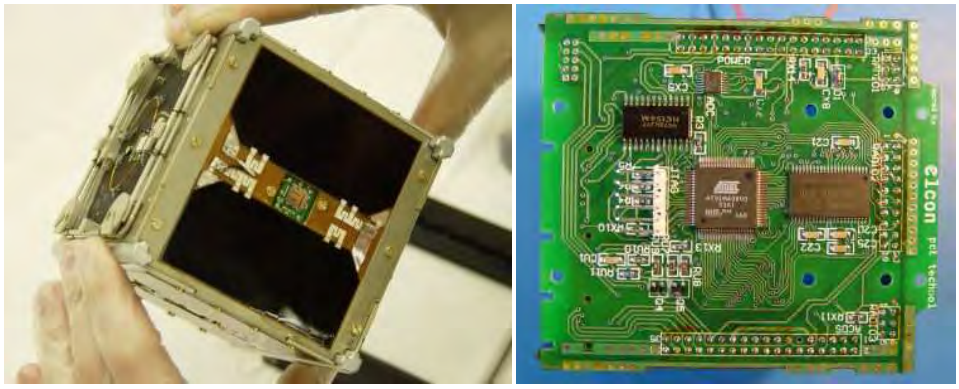


(a) (b)

Figura 3-4:(a) Nanosatélite KySat de USA. (b) Tarjetas integradas a la estructura.

3.1.4 El proyecto DTUsat – Dinamarca

En el proyecto DTUsat [16], el módulo equivalente a CCMI en el DTUsat, ver Figura 3-5, se denomina *On Board computer*, (computador a bordo), el CPU está basado en un Atmel AT91M40800 32 bits del procesador RISC con frecuencia de bus de 16 MHz, cuenta con 2 MB de flash, 1 MB de RAM, ROM y 16 MB para el software de la radiación y la falta de arranque resistente. El software programado en la memoria flash se puede reconfigurar desde la estación terrena. La ROM contiene un modo de arranque a prueba de fallos y el software programado directamente en lenguaje C es capaz de salvaguardar al nanosatélite en los modos de falla y recibir configuraciones de software a través del enlace de radio.



(a) (b)

Figura 3-5: (a) Nanosatélite DTUsat de Dinamarca. (b) On board computer del DTUsat.

3.2 Experiencias en el Perú

En el Perú no existen experiencias del diseño, construcción y prueba de un nanosatélite que use el estándar *CubeSat*, esto es una desventaja para el proyecto ya que implica iniciar el proyecto desde cero, además implica problemas de logística para los permisos y licencias de funcionamiento durante la ejecución del proyecto,

estos problemas son solucionados por el módulo de gestión del proyecto Chasqui, ver Subsección 4.4.10.

En América latina solo la universidad nacional autónoma de México tiene experiencia completa en el diseño, construcción y prueba de un *CubeSat*, llamado UNAMSAT [17], desarrollado con fines de investigación. Luego la Universidad Sergio Arboleda de Colombia adquirió un módulo *Cubesat-Kit* y basado en éste, desarrolló el nanosatélite llamado Libertad-1 [18]. Además en el Perú la Universidad Alas Peruanas también adquirió el módulo Cubesat-Kit y desarrolla el UAP-Sat [19].

El instituto de radioastronomía de la Pontificia universidad católica del Perú viene desarrollando el proyecto PUCPsat [20] y sus investigaciones se centran principalmente en el sistema de comunicaciones y no en el módulo de control central.

El proyecto Chasqui no adquirió el *CubeSat Kit*, ya que el proyecto incluye el diseño de todo el nanosatélite basándose únicamente en el estándar *CubeSat* y experiencias de otros proyectos nanosatelitales, tal como los mostrados en este capítulo.

Capítulo 4

4 El proyecto Chasqui

4.1 Introducción

El proyecto “Nanosatélite de Investigación Chasqui I” es el primer proyecto satelital de la Universidad Nacional de Ingeniería y tiene como objetivo principal el mejoramiento de las capacidades de la UNI en tecnología satelital a través del diseño, análisis, ensamblaje, integración, prueba y operación de un nanosatélite de pequeñas dimensiones. Además el nanosatélite pretende tomar fotos de la Tierra y transmitirlo a la estación terrestre.

Las dimensiones del nanosatélite son 10x10x10 cm3 y de una masa aproximado de 1 kg. Posee dos cámaras, una en el rango visible y otra en el rango infrarrojo. Asimismo cuenta con mecanismos propios de control de actitud, comunicación en la banda de radioaficionados, de energía y control térmico basado en celdas solares de alta eficiencia así como un sistema embebido de control y gestión de la información de todos los componentes del nanosatélite.

El “Chasqui I” constituye un esfuerzo sin precedentes en nuestro país por lograr por primera vez acceder al espacio y nos da la oportunidad de abrir nuevos campos de aplicación específicos a nuestra propia realidad geográfica y social. Es además desde un punto de vista académico una herramienta que facilita la colaboración entre las diversas facultades y centros de investigación de la universidad, entrena a los estudiantes y docentes con experiencias del mundo real en nanosatélites. Además genera oportunidades de trabajo conjunto con diversas universidades del mundo y permite avances tecnológicos en la industria aeroespacial de nuestro país.

Desarrollar nanosatélites de la escala del Chasqui I abre camino a las diversas posibilidades de acceso al espacio con menores costos y tiempos de desarrollo. Por esta razón, diversas universidades, compañías y organizaciones gubernamentales en el mundo demuestran su interés en desarrollar pequeños nanosatélites que permitan llevar a cabo experimentos y misiones científicas que no son fácilmente accesibles desde nanosatélites convencionales. Los beneficios educacionales y tecnológicos del proyecto se pueden enfatizar en el entrenamiento en campo de ingenieros y científicos y en un futuro cercano la maduración de esta tecnología unida a la implementación de una red de este tipo de nanosatélites permitirá que se generen nuevas aplicaciones de bajo costo en el campo de las telecomunicaciones, observación terrestre y la exploración de recursos naturales, tan importantes para el desarrollo de nuestro país.

4.2 Objetivos del proyecto

4.2.1 Objetivo general del proyecto Chasqui

Mejoramiento de las capacidades de la UNI en tecnología satelital a través del diseño, análisis, ensamblaje, integración, prueba, lanzamiento, y operación de un nanosatélite con tecnología *Cubesat*. El nanosatélite Chasqui I permitirá tomar fotos de la Tierra y transmitirlo a la estación terrestre.

4.2.2 Objetivos específicos del proyecto Chasqui

- Establecer contactos y apoyo a otras Universidades y/o Instituciones involucradas en este tipo de proyectos.
- Profundizar el conocimiento en tecnologías de información y comunicaciones emergentes.
- Liderar este tipo de proyectos satelitales en América Latina.
- Demostrar y validar nuevas tecnologías.

4.3 Organización general del proyecto Chasqui

El proyecto nanosatélite Chasqui se inicia en las instalaciones del centro de tecnologías de información y comunicaciones CTIC-UNI, allí se dan las primeras ideas sobre el desarrollo de dicho proyecto, su factibilidad, costo, tiempo de ejecución y requerimientos en recursos humanos. Luego de varias coordinaciones con el instituto nacional de investigación y capacitación de comunicaciones INICTEL-UNI, se da inicio al proyecto Chasqui, siendo ésteel principal objetivo de ambos centros de investigación pertenecientes a la UNI. Luego el CTIC-UNI

convoca a estudiantes y egresados para formar parte del equipo de investigación, siendo éstos principalmente de las facultades de Mecánica, Electrónica, Ciencias, Sistemas, Química y Economía. A su vez estos investigadores son agrupados estratégicamente, según sus conocimientos, y asignados a cada uno de los módulos que conforman el Chasqui, cada módulo a su vez es asesorado por ingenieros, licenciados, magísteres y doctores especialistas de la UNI. La Figura 4-1 muestra la Organización del proyecto mediante un diagrama de bloques. También se puede observar que existen módulos están dentro del módulo de estructura mecánica, y existen otros módulos externos al nanosatélite que también cumplen funciones importantes en el proyecto satelital.

Figura 4-1. Organización del proyecto nanosatélite Chasqui-I

4.4 Módulos que conforman el proyecto Chasqui

El nanosatélite Chasqui-I consta de diez módulos de los cuales cinco son internos a la estructura del nanosatélite, cada una de éstos es una tarjeta electrónica

encargada de una función específica, además están diseñadas para soportar las condiciones espaciales. Los demás módulos del nanosatélite son los externos, éstos están encargados de que los módulos internos puedan cumplir sus funciones y atender sus requerimientos.

La Figura 4-2 muestra gráficamente la distribución de los módulos internos y externos a la estructura del nanosatélite.

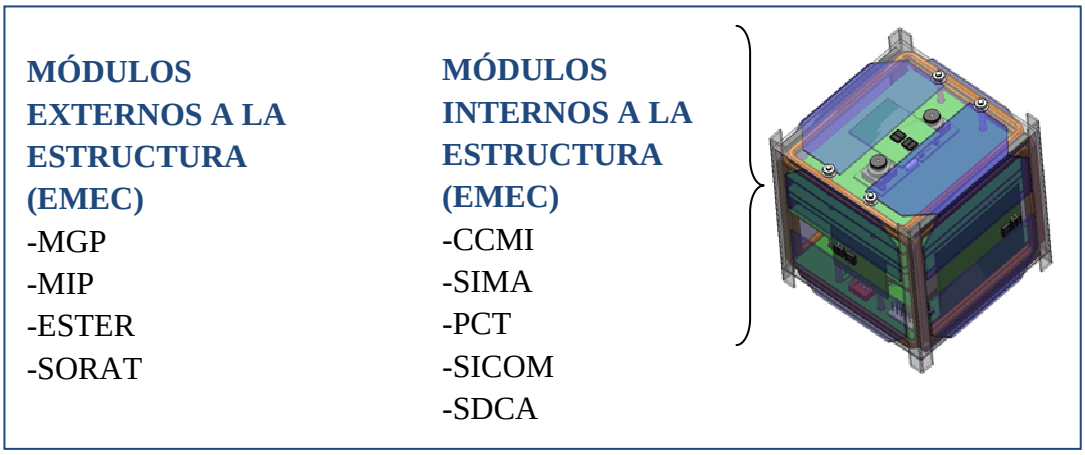


Figura 4-2: Módulos internos y externos del nanosatélite Chasqui.

4.4.1 Estructura mecánica – EMEC

El grupo de investigación del módulo de estructura mecánica EMEC [4], es el responsable de la revisión del estado de arte, análisis comparativo de los casos existentes para el diseño del pico-satélite y la fabricación de un modelo propio basado en el estándar *Cubesat*.

Dentro del pico-satélite se ensamblarán los siguientes módulos: Control Central y Manejo de Información (CCMI), Sistema de Determinación y Control de Actitud (SDCA), Sistema de Manejo de Imágenes (SIMA), Unidad de Potencia y Control Térmico (PCT) y el Sistema de Comunicaciones (SICOM).

La Figura 4-3 muestra uno de los modelos diseñados por el módulo EMEC, se puede apreciar gráficamente el análisis de esfuerzos realizado mediante software CAE.

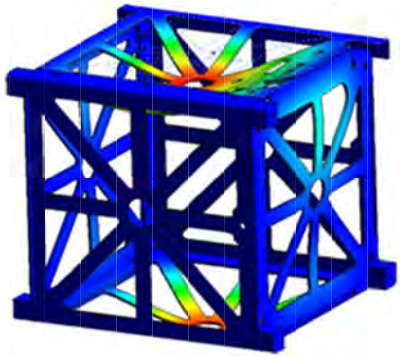


Figura 4-3: Diseño virtual de una estructura mecánica diseñada por el módulo EMEC

4.4.2 Control central y manejo de información – CCMI

Este módulo es el desarrollado en esta tesis, y la descripción del mismo fue presentada en el capítulo 2.

La Figura 4-4 muestra la tarjeta CCMI cuyo diseño, ensamble y prueba se muestra en esta tesis.

Figura 4-4: Tarjeta Electrónica del módulo CCMI

4.4.3 Sistema de adquisición de imágenes – SIMA

El objetivo principal de este módulo es obtener fotografías de la Tierra desde el Chasqui I. El módulo SIMA [4] está compuesto por dos cámaras, una de rango visible y la otra en el rango infrarrojo cercano. La información digital es recopilada por el módulo de Control Central y Manejo de la Información (CCMI), tal como se muestra en la Figura 4-5, y posteriormente se envía a la Estación Terrena (ESTER).

Además, el grupo se encargará del tratamiento digital de las imágenes obtenidas por el Chasqui-I.

Características:

- Resolución: 640 x 480 pixeles.
- Voltaje: 3.3 V.
- Comunicación: Se comunica serialmente con el módulo CCMI.

Figura 4-5: Módulo de sistema de adquisición de imágenes SIMA.

4.4.4 Potencia y control térmico – PCT

El primer subsistema de este módulo es el de Potencia y se encarga de recibir, transformar, almacenar y distribuir la energía a los otros subsistemas que componen

el Chasqui I. El objetivo de éste subsistema es el de asegurar el abastecimiento de energía eléctrica para el Chasqui I, dándole la energía necesaria en el tiempo adecuado. Para ello hace uso de celdas solares que sirven como única fuente de energía en el espacio [11].

El segundo subsistema es el de Control Térmico y éste se encarga de mantener la temperatura de las baterías y demás componentes del nanosatélite en su rango de operación, con la finalidad de asegurar el funcionamiento del Chasqui I. La tarea más crítica de éste subsistema es la de mantener las baterías funcionando dentro de su límite de operación (0 °C. a 20 °C). Para esto usa calentadores especialmente diseñados y construidos para este proyecto satelital.

La Figura 4-6 muestra las funciones del sistema de potencia, es decir que es lo que hace el sistema de potencia en el nanosatélite Chasqui-I, éstas principalmente son dos:

- Abastecer de energía eléctrica regulada (3.3V y 5.0V) a los demás sistemas que componen el nanosatélite
- Enviar datos de corriente, voltaje y estado de carga de la batería a CCMI para ser trasmitidos a Tierra con lo cual se podría monitorear el estado de consumo del nanosatélite.

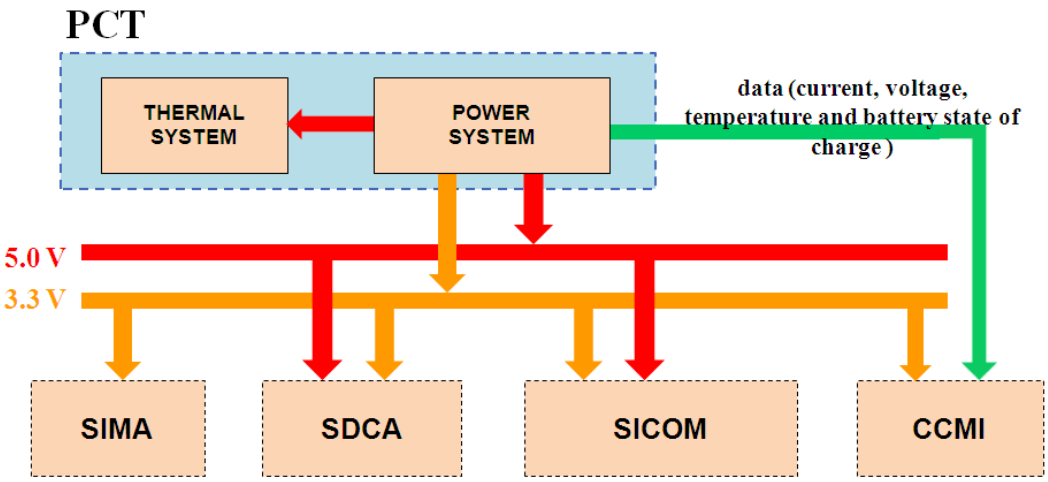


Figura 4-6: Esquema general del módulo PCT y su interacción con los demás módulos [11]

4.4.5 Sistema de determinación y control de actitud – SDCA

El SDCA [22] mantiene la estabilización del nanosatélite y lo orienta a una dirección deseada cuando sea necesario. Específicamente, podemos decir que el SDCA es el responsable de:

- Estabilizar el nanosatélite después de que sale del desplegador mediante reducción (a menos de 0.1rad/s) y control de sus velocidades angulares.
- Mantener una exactitud de apuntamiento de 3 grados durante la toma de fotos del Perú y, de ser técnicamente posible, contar con una amplia cobertura de América del Sur mediante realización de maniobras de 30 grados en el alabeo (roll) y 30 grados en el cabeceo (pitch).
- Mantener una exactitud de apuntamiento menos exigente (por ejemplo, 20 grados) que permita la subida/bajada de datos entre el nanosatélite y la estación terrena.

El SDCA permite que el nanosatélite determine su actitud usando sensores, calcule la corrección requerida para alcanzar la orientación deseada y ejecute las maniobras necesarias usando los actuadores. El sistema de determinación de actitud empleará magnetómetros, sensores solares y algoritmos de determinación de actitud para estimar posiciones y velocidades angulares. El uso de un GPS y giroscopios como sensores de determinación de actitud también será evaluado. El sistema control de actitud usará bobinas electromagnéticas e imanes permanentes como actuadores. Las bobinas electromagnéticas son especialmente importantes para la estabilización del nanosatélite una vez que éste sale del desplegador. La inclusión del imán permanente permite contar con un sistema de control activo-pasivo. Más de una ley de control será estudiada para una posible implementación. El uso de materiales magnéticos e histeréticos también será evaluado.

La Figura 4-7 muestra el diagrama de bloques general del módulo SDCA.

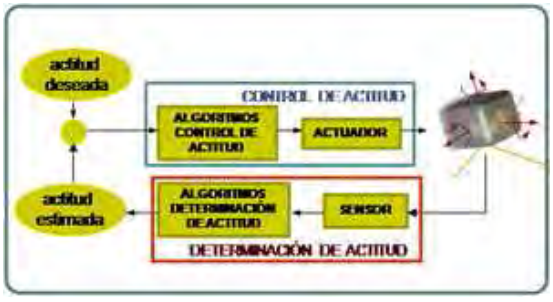


Figura 4-7: Esquema general de funcionamiento del módulo de SDCA.

4.4.6 Sistema de comunicaciones – SICOM

El módulo Sistema de Comunicación SICOM [23] es el encargado de recibir los comandos desde la estación terrena y de enviar la información solicitada, ya sean datos de sensores o una captura de imagen. Este módulo se puede subdividir en dos componentes: una unidad de control y otra de radio. La unidad de control permite encapsular o desencapsular del protocolo AX.25 la información que sea solicitada o los comandos enviados de Tierra. La unidad de radio es el medio que convierte o interpreta las señales de radio en señales digitales o viceversa, esto se resume en la Figura 4-8.

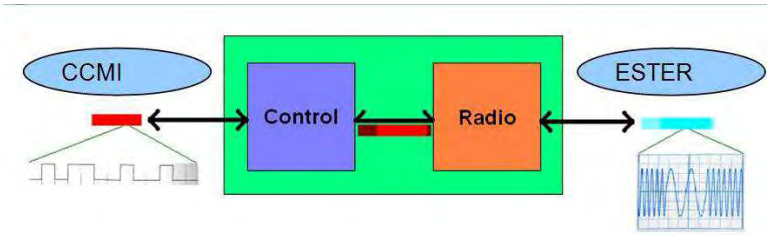


Figura 4-8: Esquema general de funcionamiento del módulo de SICOM.

4.4.7 Estación terrena – ESTER

Este subsistema no es parte del nanosatélite en sí, pero su existencia y operación es necesaria para lograr los objetivos del Chasqui I. Es el conjunto de instalaciones y equipos de comunicación inalámbrica (radio) necesarios para comunicarse con el Chasqui I, ver Figura 4-9, así como con cualquier [24].

Las principales funciones de éste módulo son:

- Seguimiento: Se escucha el *beacon* o *radioforo* del nanosatélite para conocer su posición.
- Telemetría: Pedir variables de estado (temperatura, voltaje, etc.) para monitorear al nanosatélite y validar el cálculo de la órbita.
- Comando: Ordenar al nanosatélite que extienda la antena; ordenar *reset* al sistema; ordenar la toma y envío de las fotos.

Características

- Bandas de frecuencia: VHF (144 - 146 MHz.) y UHF (435 - 438 MHz.)
- Modo de operación: Half duplex
- Potencia de transmisión: 50 W.
- Modulación: AFSK
- Velocidad del enlace de bajada: Entre 1200 bps. y 9600 bps.
- Velocidad del enlace de subida: Entre 1200 bps. y 9600 bps.

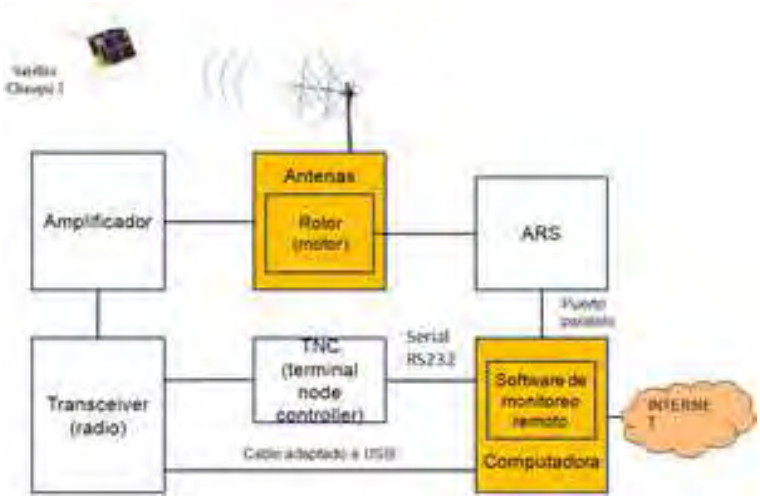


Figura 4-9: Diagrama de bloques del módulo ESTER [24].

4.4.8 Sistema de orbitas y atmósferas – SORAT

El objetivo del módulo es simular las trayectorias del Chasqui I [4], ver Figura 4-10, que se hace calculando previamente las ecuaciones diferenciales de movimiento y luego resolviéndolas en forma paralela con dos programas: Delphi y Matlab.

La simulación se logra tomando en consideración las siguientes fases:

- Considerando la Tierra como un sistema de referencia no inercial, el término *cuadrupolar* del potencial gravitatorio y usando la segunda ley de *Newton* se obtienen las ecuaciones de movimiento que son ecuaciones no lineales.
- Usando el método de Runge-Kutta de orden 4 con el programa *Delphi* se resolvieron las ecuaciones de movimiento manteniéndose constante la energía.
- Se repitió la fase 2 con el programa *Matlab* y con este software se efectúan las simulaciones de la trayectoria del Chasqui I.

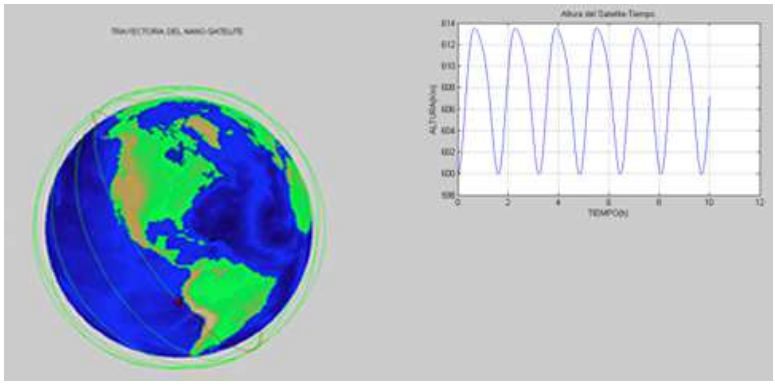


Figura 4-10: Ejemplo de simulación en MATLAB del módulo de sistema de órbitas y atmósferas SORAT

4.4.9 Integración y pruebas – MIP

El objetivo del módulo es lograr la integración de los componentes desarrollados por los diferentes módulos del proyecto como tarjetas de circuitos electrónicos, cámaras fotográficas, baterías, antenas, sensores, y bobinas electromagnéticas [3].

Los objetivos del módulo se pueden lograr:

- Optimizando superficies, volúmenes, masas, encontrando centro de gravedad, centro de masas.
- Planificando y realizando las pruebas de exigencias estandarizadas.
- Realizando las pruebas de campo planificadas en el proyecto.

En la Figura 4-11 se muestra el diseño virtual del Chasqui integrado realizado por el módulo MIP.

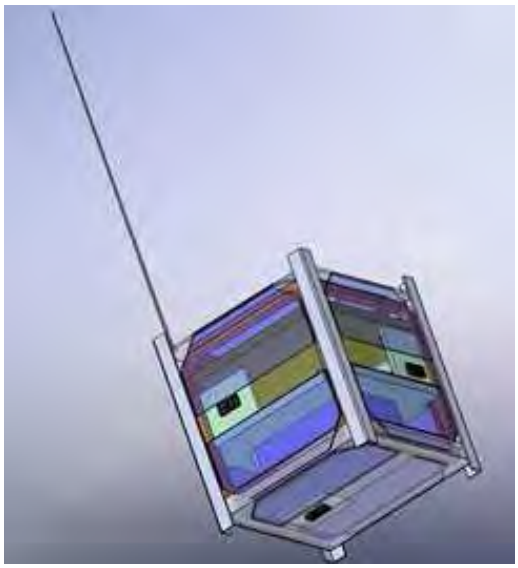


Figura 4-11: Diseño virtual del Chasqui Integrado, desarrollado por el módulo de integración y pruebas MIP.

4.4.10 Gestión del proyecto - MGP

El módulo MGP está encargado de gestionar la obtención los recursos necesarios para todo el proyecto, además de coordinar actividades necesarias como reuniones de trabajo del equipo, difusión de los resultados obtenidos dentro y fuera de la universidad organizando ferias y eventos de difusión de tecnologías satelitales, traer capacitadores especializados en temas satelitales y gestionar los permisos para pruebas de campo en instituciones externas a la universidad [4].

Capítulo 5

5 Marco Teórico

5.1 Introducción

En este capítulo se muestra los conceptos necesarios para la comprensión de esta tesis, se asume que son conocidos los temas básicos de la especialidad de ingeniería mecatrónica, como son: electrónica digital, programación en lenguaje C, microcontroladores, sistemas operativos, diseño e implementación de tarjetas electrónicas y manejo de software CAD.

5.2 Introducción a los sistemas embebidos

Los sistemas embebidos son dispositivos que por sí solos cumplen una determinada función, es decir no es necesario de algún complemento adicional para que pueda cumplir sus funciones.

Los sistemas embebidos están en constante desarrollo y evolución debido a la gran cantidad de aplicaciones posibles, es usado en múltiples proyectos de la ingeniería de desarrollo, como por ejemplo: instrumentación y control para monitorear variables o parámetros de un proceso, implementación de dispositivos

como relojes, alarmas de seguridad, control de motores eléctricos o de combustión, sistemas de comunicación como RS232, USB, Ethernet, etc.

Los sistemas embebidos son una parte fundamental de sistemas complejos como el nanosatélite Chasqui, se han desarrollado muchas técnicas que permiten un funcionamiento óptimo del sistema embebido, haciéndolos cada vez más rápidos, con mayor capacidad de almacenamiento en memoria de datos y programa. Adicionalmente se busca incluir los periféricos necesarios para facilitar la interfaz con el usuario del producto, además la portabilidad es una característica importante de los sistemas actuales ya que al estar en constante evolución es necesario que su diseño pueda adaptarse fácilmente a dichos cambios.

5.3 Proceso de desarrollo de un sistema embebido

En el desarrollo de sistemas embebidos se debe seguir un orden establecido para poder obtener finalmente un producto, idealmente este producto no debería requerir de modificaciones posteriores, ya que éstas implican mayores gastos y tiempo. El proceso de diseño se inicia con el diagrama de flujos, ya que permite ordenar nuestras ideas sobre los procesos y algoritmos que ejecutará nuestro sistema, luego sigue el desarrollo en software de los algoritmos diseñados, para ello es usual algún entorno de desarrollo integrado (IDE), dónde haremos uso de algún lenguaje de programación, el más común en aplicaciones microcontroladas es el lenguaje ANSI-C. La programación debe ser depurada, es decir deben tenerse en cuenta todos los posibles errores y poder detectarlos para su corrección, para ello los IDEs permiten la ejecución del código instrucción por instrucción, A continuación sigue el diseño e implementación de nuestro hardware, para ello nos ayudamos de un

software CAD (Diseño asistido por computadora), para aplicaciones industriales, militares o espaciales. Es usual someter dicho diseño a un análisis del comportamiento de la tarjeta sometida a pruebas de temperatura, campo magnético o radiación, para ello se usan software CAT (Testeo asistido por computadora). Un paso siguiente es el proceso de fabricación de la tarjeta para luego poder emular el sistema embebido, es decir hacer la depuración paso a paso del software pero cuando nuestro código de programación es ejecutado por el CPU de nuestro sistema embebido. La tarjeta implementada debe ser testeada, es decir analizar su comportamiento sometido a las condiciones de operación, por ejemplo si el ambiente de trabajo es de altas temperaturas entonces la tarjeta debe ser probada a dichas condiciones aplicando un factor de seguridad en el diseño. Finalmente se desarrollan los acabados del producto, esto implica darle una buena presentación y facilitar la operación a los usuarios, para quienes nuestro sistema embebido debe ser una especia de caja negra en la cual solo vean entradas y salidas, esto se resume en la Figura 5-1.

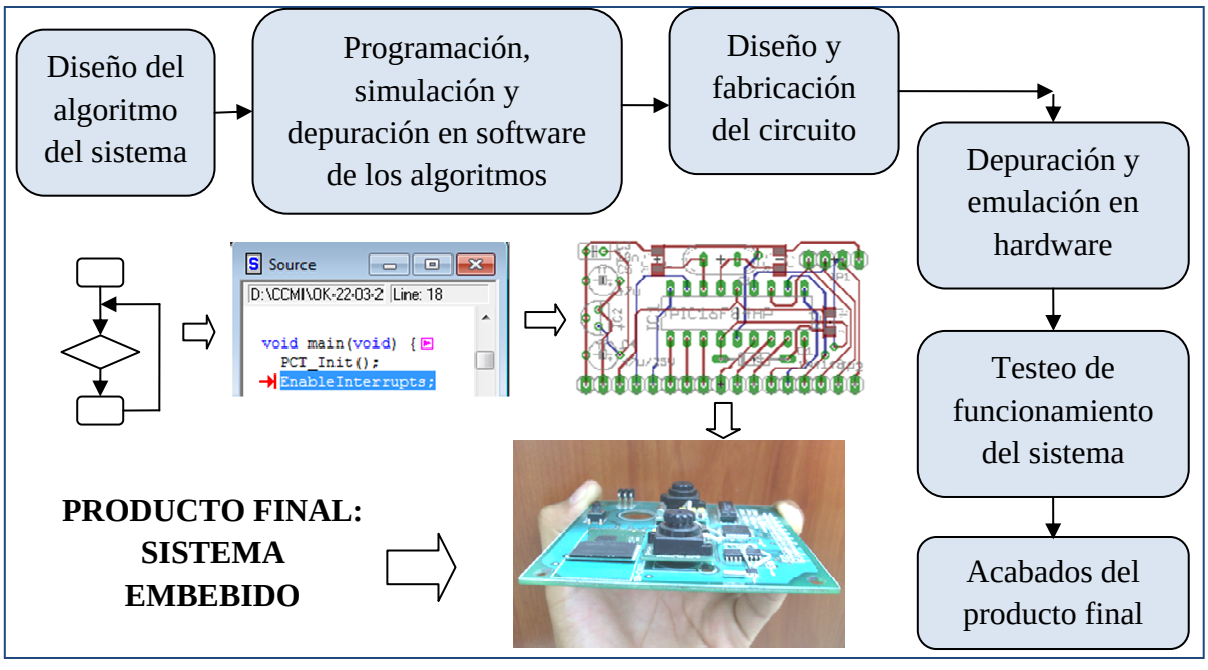


Figura 5-1: Proceso de desarrollo de un sistema embebido

En la práctica es usual que el producto final si necesite modificaciones debido a factores sin importancia inicialmente o debido a nuevos requerimientos de los usuarios, se recomienda que el sistema embebido esté preparado para que los posibles cambios solo requieran modificar el software. Esto no quiere decir que el producto esté mal sino que debe se modificar para adaptarse a los nuevos factores externos, es por ello que podemos ver en nuestros sistemas embebidos varias versiones. Es por lo expuesto en este punto que en el caso del módulo CCMI se diseñaron 4 versiones de hardware, denominadas CCMIv1, CCMIv2, CCMIv3 y CCMIv4, cada versión funcionó correctamente en su momento. De todo esto se puede concluir que el proceso de desarrollo de un sistema embebido es un ciclo continuo del proceso mostrado en la Figura 5-1 .

5.4 Principales dispositivos de los sistemas embebidos

A continuación se muestran las partes más comunes e importantes de los sistemas embebidos, los mencionados son los más empleados en aplicaciones como la desarrollada en esta tesis acerca del diseño de la tarjeta del módulo de Control Central, variando básicamente en el tipo y/o fabricante del dispositivo.

5.4.1 El microcontrolador

Físicamente un microcontrolador es un chip, el cual contiene internamente toda la arquitectura de computador necesaria para ser programada y para manejar señales de entrada y salida, para ello cuenta externamente con pines, entre ellos los que sirven de alimentación de potencia eléctrica, pines de grabación, puertos de entrada y salida, entrada de señal de oscilador para control del reloj interno, y para manejar sus diversos periféricos incluidos en su arquitectura. La forma externa de los

microcontroladores es denominada empaquetado, los cuales son estándares, es decir las dimensiones del chip pertenecen a un grupo establecido, en base al cual se diseñan las tarjetas impresas para que puedan ser compatibles dimensionalmente.

La Figura 5-2 muestra el empaquetado LQFP de 24 pines, para el caso de un chip microcontrolador de *Freescale*.

Los microcontroladores son programados en un lenguaje de bajo nivel llamado ensamblador y sus instrucciones son denominadas nemóticos y corresponden a una tarea básica del CPU, este lenguaje es propio de cada arquitectura por lo que hace que la portabilidad de un microcontrolador a otro sea un proceso complejo, más si es de otro fabricante, ya que implica cambiar cada línea de instrucción por los nemóticos equivalentes en el nuevo uC, si es que lo tuviera. Es por esto que el lenguaje C para microcontroladores es bastante útil, ya que permite dicha portabilidad, en este lenguaje cada instrucción forma parte del estándar ANSI-C, y cada IDE permite compilar la codificación para transformarla en su equivalente en ensamblador.

Figura 5-2: Microcontrolador como parte fundamental en un sistema embebido [4]

5.4.2 Principales Fabricantes de Microcontroladores

Cada fabricante tiene además varias familias de microcontroladores que se ajustan a las necesidades de los diseñadores, estas pueden ser la arquitectura interna,

tipos de instrucciones, cantidad de bits de sus registros, periféricos disponibles, etc. En los últimos años los microcontroladores más usados a nivel universitario son los de la marca microchip, debido a la gratuidad de muchos materiales de aprendizaje brindadas por la misma empresa, además del bajo precio de sus productos, lo que ocasionó una alta disponibilidad de estos microcontroladores en el mercado nacional y mundial.

En la Figura 5-3 se muestra a los principales fabricantes de microcontroladores en el mercado mundial.

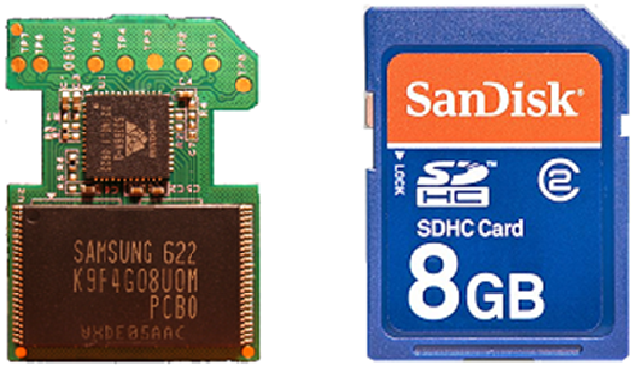
Figura 5-3: Logotipos de los principales fabricantes de microcontroladores.

5.4.3 Memorias de almacenamiento digital

Muchas veces la memoria incluida en los microcontroladores es insuficiente por lo que es necesario usar externas, las memorias se dividen principalmente en dos grandes grupos, las volátiles que pierden su información cuando dejan de ser energizadas y las no volátiles que a pesar de ello mantienen dicha información.

5.4.3.1 Tarjetas de memoria *Secure Digital* SD-Card

Son tarjetas con memorias del tipo no volátiles, ver Figura 5-4, generalmente del tipo NAND Flash, desarrolladas inicialmente por Panasonic [25], contienen internamente un uC que hace de interfaz para la transmisión de datos usando el protocolo SPI, la ventaja de estas tarjetas es la alta capacidad de almacenamiento que tienen, su bajo costo y facilidad de manipulación, es por ello que actualmente es ampliamente utilizado en equipos como celulares, PDAs, cámaras digitales que son capaces de almacenar cientos de fotos en alta resolución. Las SD-Card utilizan para la generación de archivos el estándar del FAT16 y FAT32.



(a) (b)

Figura 5-4: Tarjeta de memoria SD. (a) Composición interna, memoria NAND Flash y controlador. (b) Estructura externa de la SD-Card.

5.4.4 Fuente de energía eléctrica

Las fuentes de energía de los sistemas digitales son de valor constante, los más usuales para aplicaciones embebidas son de 3.3V y de 5V, éstos pueden provenir de una batería o de una fuente AC-DC. En los sistemas digitales se exige cierto grado de precisión de estos valores de voltaje de alimentación, ya que los valores que son considerados como uno o cero son relativos a estos voltajes, también para proteger a los mismos circuitos integrados, además la fuente de alimentación no debe generar

picos de corriente o voltaje, ya que esto provocaría que el uC se reinicie o se dañe. Para proteger el sistema se emplean reguladores de voltaje, que son circuitos integrados que mantienen un valor de voltaje fijo a su salida, además se emplean condensadores para disminuir el riso de voltaje e inductores para reducir los cambios bruscos de corriente, la ubicación de dichos condensadores se recomienda que sea lo más cercano posible a los circuitos integrados.

5.4.5 Periféricos de control para el usuario

Estos periféricos permiten al usuario manipular el sistema embebido sin necesidad que sepa qué tiene internamente, ya que un sistema embebido debe ser una especie de caja negra para el usuario, pues solo necesita saber cómo ingresar las entradas al sistema dependiendo de las salidas que desea obtener. Los periféricos de control más usuales son pulsadores, switchs, teclados matriciales, etc.

5.4.6 Indicadores de estado y/o proceso

Estos dispositivos nos permiten saber el estado actual del sistema, son programados para ello y son muy útiles para el usuario y para el mismo desarrollador del sistema ya que así puede saber cómo se comporta y detectar las posibles fallas. Estos dispositivos son por lo general LEDs, Displays, pantallas LCDs, buzzers, etc.

5.4.7 Interfaz visual para el usuario

Esta interfaz hace de periférico de control e indicador de estado, ver Figura 5-5, generalmente se desarrolla en PC usando un programa visual, que se comunica mediante RS232 o USB, este quizás sea el ambiente más amigable para el usuario ya

que usa las librerías visuales del sistema operativo para el control y monitorio del sistema embebido. Ejemplos de programas capaces de generar dicha interfaz son el Visual C#, Visual Java, MatLab, LabVIEW, etc.

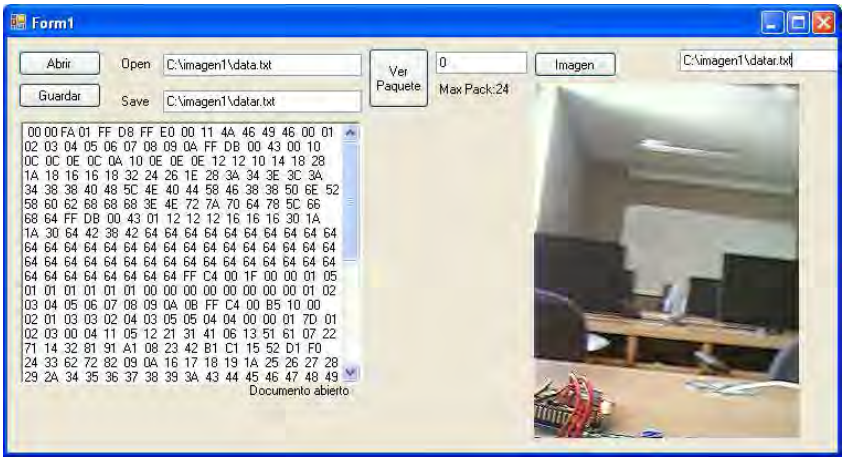


Figura 5-5: Interfaz visual para el usuario.

5.4.8 Conectores para comunicación con otros dispositivos

Estos conectores permiten al sistema embebido comunicarse con otros equipos, estos conectores deben permitir un buen contacto entre los pines, más en el caso en el que se transmite información a altas frecuencias, ya que un pequeño tiempo de falso contacto, llamemos así al estado en el que los pines solo parecen estar en contacto, puede significar pérdida o mal interpretación de los datos transmitidos. Además la ubicación de estos conectores por lo general es en los bordes de la tarjeta ya que facilita así su conexión. La Figura 5-6 muestra un ejemplo de conector denominado PC104 de 40 pines, este tipo de conectores también es denominado BUS PC104 y es especial para transmisión de datos, sus pines están recubiertos con oro para mejorar la conductividad.

Figura 5-6: Conector PC104

5.5 Protocolos de comunicación en sistemas embebidos

Muchas veces es necesario que los sistemas embebidos deban compartir información entre ellos para poder ejecutar sus acciones programadas, por ejemplo un equipo de *robot soccer*, para que sea realmente eficiente cada robot debe saber lo que los demás compañeros de equipo y los del otro equipo están haciendo, es decir deben existir protocolos de comunicación establecidos entre ellos para poder compartir información y actuar según sea ello. Estos protocolos deben cumplir características que permitan la seguridad de información, además de ser estándares para permitir que equipos de distintos fabricantes puedan comunicarse, siempre y cuando usen los mismos protocolos de comunicación.

Este punto es resaltado debido a que para el caso de la presente tesis, se debe implementar todos los protocolos de comunicación necesarios para el correcto funcionamiento de todo el sistema y su interacción con el módulo CCMI, incluso se mostrará que la creación de protocolos internos entre módulos, no tanto a niveles eléctricos, como son el TTL y RS232, sino a niveles más altos de software, es decir deberá establecerse los comandos a transmitir y recibir, la velocidad de la misma, su reacción ante una posible pérdida de comunicación, la detección de errores, etc.

La Figura 5-7 muestra un ejemplo en el que un MCU se comunica con diversos dispositivos haciendo uso de varios protocolos de comunicación.

Figura 5-7: Protocolos de comunicación, entre dispositivos electrónicos [26]

5.5.1 Protocolo UART (Universal Asynchronous Receiver-Transmitter)

Este protocolo de comunicación asíncrona, consiste en transmitir datos binarios de forma serial, para ello hace uso de dos líneas de comunicación y una tierra común entre los dispositivos a comunicar, estas líneas son simbolizadas generalmente por TX y RX, representando los pines de transmisión y recepción respectivamente. Para reemplazar la señal de *clock*, muy necesario cuando se trata de comunicación serial pues permite evitar la desincronización, se ha estandarizado velocidades de transmisión por defecto, las cuales deben configurarse en los microcontroladores o equipos que usen este protocolo, es decir, ambos dispositivos a comunicar deben coincidir en esta característica para que el dato no sea mal interpretado por errores de desincronización en la línea.

Es usual medir la velocidad de transmisión en *baudios*, bits por segundo (*bps*). Siendo alguno de los valores más comunes los que se muestran en la Tabla 5-1, en donde también se muestra un dispositivo de ejemplo para cada tipo de velocidad en baudios.

Tabla 5-1: Ejemplo de velocidades de transmisión UART y algunos dispositivos relacionados a velocidades estándar.

Baudios (bps)	**Dispositivos**
4800	GPS-Tyco
9600	Tarjeta USB-Weather
14400	Cámara C328R
19200	Módulo GSM

No se debe confundir el protocolo UART con el protocolo RS232, éste último especifica las características que debe cumplir la señal para permitir la comunicación UART con una computadora, es decir tiene en cuenta niveles eléctricos de voltaje, similar al protocolo o estándar TTL, en ambos casos se puede implementar una comunicación UART, pero para que dispositivos con niveles de voltaje TTL, puedan comunicarse con una PC, se debe hacer uso de convertidores, uno de los más usados es el MAX3232, de *MAXIM innovation delivered* [26]. La Figura 5-8 muestra un ejemplo del uso de este integrado, el cual permite cambiar los niveles de voltaje de TTL a RS232.

Figura 5-8: Aplicación del circuito integrado MAX3232

5.5.2 Protocolo I2C

Este protocolo I2C (*Inter-integrated circuit*) fue diseñado por Philips [27] y está orientado al intercambio de información a velocidades relativamente altas y eficientes, se trata de un protocolo síncrono, es decir hace uso de una señal de *clock*, la misma que controla la velocidad de la transferencia de información.

El bus IIC usa las líneas SDA Y SCL para el control y la transferencia de datos. Una nota importante sobre estas líneas es que deben ser de drenaje abierto, además, se necesitan resistencias *pull-up* para estos pines, Se ejecuta una función AND lógica entre la línea y las resistencias *pull-up*. El valor de la resistencia depende del sistema, para cuando se diseñan sistemas con muchos IC-esclavos, además, para evitar que todos incluyan en sus diseños resistencia *pull-up*, se recomienda que solo el IC-maestro sea el que incluya en su circuito estas resistencias.

En una transferencia IIC normal, se genera una señal START. Una señal START se consigue al mantener la línea SDA en bajo o cero, ver Figura 5-9 tomada de la hoja técnica del MCU HCS08 [28].

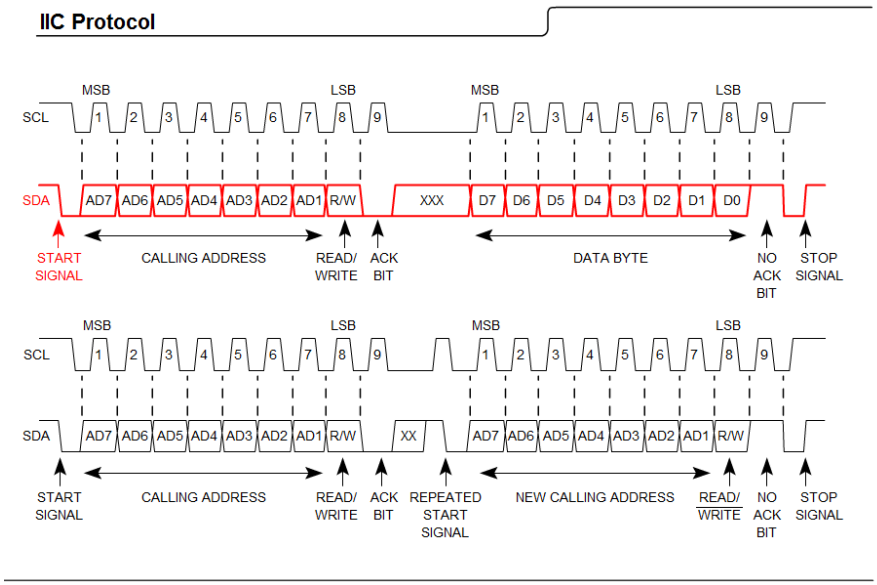


Figura 5-9: Señal de START del Protocolo I2C

En cada transmisión IIC, cada bit se reconoce en el flanco de subida de la señal de reloj. El bit LSB del registro de dirección estable si la operación es de lectura o escritura (bit R/W), ver Figura 5-10.

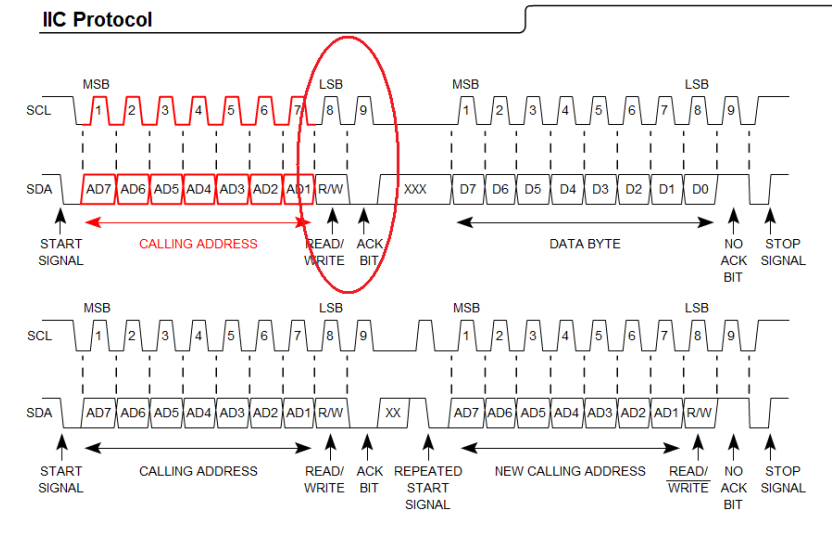


Figura 5-10: Luego de transmitir la dirección del esclavo, se envía un bit de escritura o lectura.

Una transferencia de datos puede continuar con tantos bytes como sean necesarios. Siempre que se reciba la señal ACK (viene del esclavo), la transferencia

continuará. Una señal STOP se envía (¿lo envía el maestro?) como un no ACK o simplemente NACK, ver Figura 5-11. Esto detiene la comunicación IIC. Debemos notar que una señal STOP no es lo mismo que una instrucción STOP del CPU. Una señal de stop se envía como un NACK.

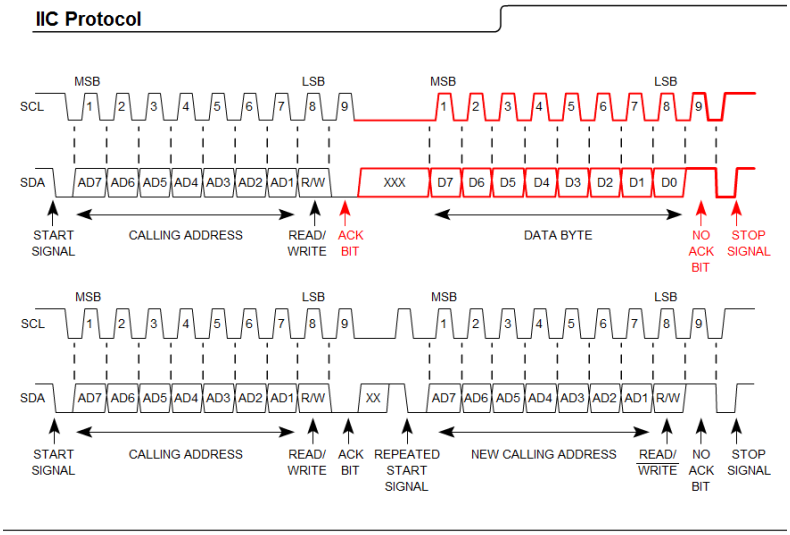


Figura 5-11: Luego de transmitir la dirección del esclavo, se envía un bit de escritura o lectura.

5.5.3 Protocolo SPI (Serial Periferical Interface)

Este protocolo, al igual que el I2C es de tipo síncrono, pero cuenta con tres líneas para su comunicación, SPSCK para la señal de clock, MOSI (*Máster Output, Slave Input*), MISO (*Máster Input, Slave Output*), una de las ventajas de este protocolo es que evita controlar lectura o escritura por el maestro, incluso ambas pueden darse en un mismo instante, ver Figura 5-12.

Para el caso en que se tengan múltiples esclavos, será necesario habilitar alguno de ellos, para ello los módulos SPI cuenta con un pin $\overline{SS}$ por lo que deberá haber uno para cada esclavo, la desventaja de este protocolo, comparado con el I2C, es que es necesario un mayo cableado en el sistema.

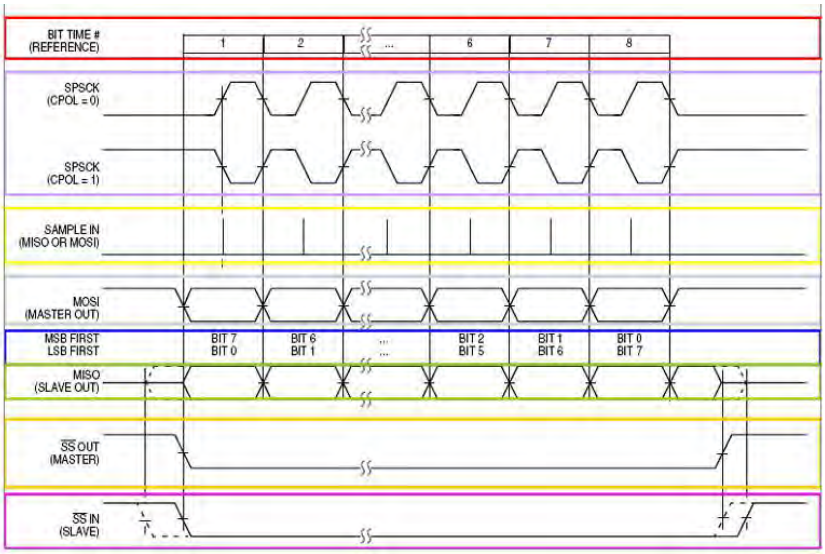


Figura 5-12: Señales que intervienen en la comunicación SPI.

5.6 Diseño del software – el estándar ANSI-C.

El diseño del software consiste en desarrollar los algoritmos que nuestro sistema debe ejecutar para luego implementarlo en algún IDE haciendo uso de un lenguaje de programación, uno de los más comunes en los microcontroladores es el ANSI-C.

La programación de los microcontroladores se da en su lenguaje básico de bajo nivel denominado ensamblador, propio de cada arquitectura, además los IDEs permiten programar en lenguajes de alto nivel como el ANSI-C, dichos IDEs convierten el lenguaje C al lenguaje ensamblador para facilitar la codificación y la comprensión del mismo, además para facilitar así la portabilidad entre distintos microcontroladores, dicho código compilado a ensamblador es transmitido a la memoria de programa del microcontrolador mediante un dispositivo denominado grabador o quemador, vía RS232 o USB.

5.6.1 Principales instrucciones del estándar ANSI-C

A continuación se muestra a manera de ejemplo algunas sentencias o instrucciones comunes del estándar ANSI-C, estas instrucciones son condicionales que dependen del valor de alguna variable.

Para diseñar un algoritmo de programación, primero se debe hacer el diagrama de flujo del algoritmo deseado, éste diagrama nos permite comprender mejor un proceso debido a la naturaleza gráfica del mismo, éstos consisten en figuras que dependiendo de su forman indican el proceso desarrollado por nuestro algoritmo, es necesario conocer la simbología usada en los diagramas de flujos, que consiste en tres elementos básicos, el terminal, bloques de proceso y decisión, además cuando el algoritmo es extenso y es preferible separarlas en partes y ponerlas en hojas separadas, entonces es necesario los notos para enlazar dichas partes, éstos elementos se muestran a continuación en la Figura 5-13.

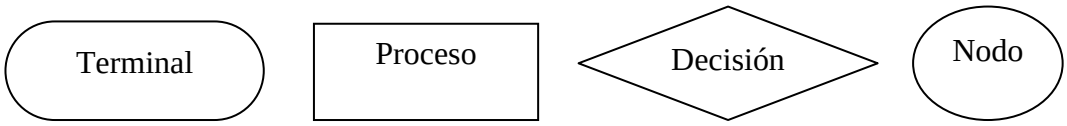


Figura 5-13: Símbolos de los diagramas de flujo

A continuación, en la Figura 5-14, se muestran algunas instrucciones comúnmente usadas en el desarrollo de algoritmos para microcontroladores, estas sentencias son universales para cualquier lenguaje de programación de alto nivel.

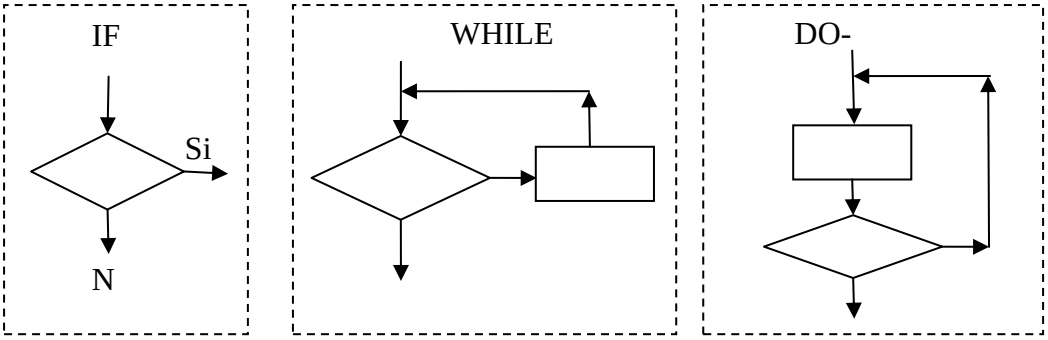


Figura 5-14: Símbolos de los diagramas de flujo

5.6.2 Directivas del compilador

Estas instrucciones son órdenes para el IDE y le indican la forma en la que debe compilar las instrucciones, entonces estas no se compilan a código ensamblador para el uC, son bastante útiles para facilitar la programación, modificación y comprensión del código [1].

5.7 Sistemas Operativos en Tiempo Real (RTOS) para sistemas embebidos

Un sistema operativo en tiempo real RTOS para sistemas embebidos, es un programa que coordina el funcionamiento de otros programas llamados tareas, al cual se le exige corrección en sus respuestas bajo ciertas restricciones de tiempo, la principal característica que distingue a los sistemas en tiempo real de otros tipos es el tiempo de interacción de sus tareas [1] [29]. A continuación se describen algunas definiciones importantes sobre estos sistemas.

- ***Scheduler*:** Su función consiste en repartir el tiempo disponible de un microprocesador entre todos los procesos que están disponibles para su ejecución, permite control multitarea y multiproceso.

- **Tareas *(Task)*:** Las tareas son procedimientos formados por códigos de programa que asumen que la CPU está disponible solo para operarla su ejecución. Las tareas trabajan con base en estados y eventos.

Una tarea es encargada de realizar una labor específica en la aplicación embebida; para su avance, el sistema operativo le entrega pequeñas porciones de tiempo invocando la tarea de forma consecutiva en el punto en el cual la suspendió la vez anterior, dando la impresión de continuidad.

Las tareas presentan cierto nivel prioridad para su ejecución, este nivel puede configurarse al momento de su creación, o puede ser cambiado por alguna otra tarea o en alguna interrupción.

Existen muchas maneras de controlar la prioridad que se asigna a cada tarea, para nuestro caso haremos uso de un orden de prioridad cuantitativo, teniendo en cuenta que mientras menor sea dicho orden la terea será de mayor prioridad.

1	**2**	**3**	**4**	**. . . .**	**61**	**62**
Mayor importancia importancia						**Menor**

La prioridad puede ser fija, definida al principio de la ejecución del sistema, o bien manejarse de forma dinámica en el transcurso del programa, por ejemplo dependiendo del estado del Chasqui-I, si este está en modo normal la batería estará con una prioridad alta pues el nanosatélite necesitará toda la energía para su correcto funcionamiento, en cambio si pasa a modo de emergencia, la batería tendrá una prioridad menor ya que tendremos que ahorrar energía hasta que volvamos al modo normal.

- **Suspendida (Dormant):** Indica que una tarea está creada o terminada y no está siendo ejecutada por el RTOS ni programada su ejecución por el Scheduler.
- **Ejecución (Run):** Una tarea se encuentra en este estado cuando el Scheduler del RTOS ha pasado el control a su código y está siendo ejecutada en el presente por la CPU.
- **Disponible (Ready):** Una tarea que está esperando turno para ser ejecutada por el Scheduler. Su prioridad es más baja que la que está en modo Run.
- **Espera (Wait):** Estado en el que una tarea está a la espera de algún evento para continuar su ejecución, hasta que no se presente dicho evento no estará en la lista de tareas que son ejecutadas por el Scheduler. El evento que espera la tarea puede ser un dato de temperatura, que un recurso esté disponible, etc.
- **Interrumpida (Interrupted):** Cuando una tarea está en modo Run, es suspendida por la aparición de un ISR. En este caso el RTOS no tiene participación en la administración de suspender y retornar el control a la tarea.

La Figura 5-15 muestra en resumen los estados de las tareas organizados como una máquina de estados.

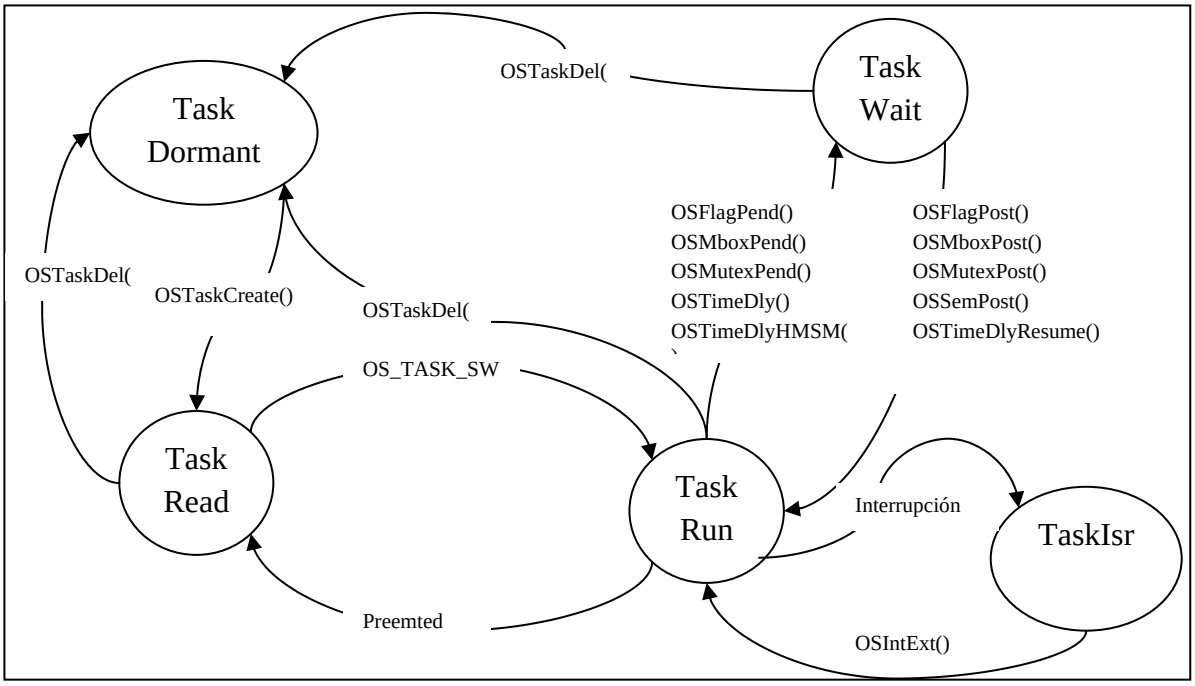


Figura 5-15: Máquina de estados de una Tarea del RTOS

- **Recursos (Resources):** Los recursos son entidades usadas por una tarea. Un recurso puede ser una función no reentrante, una estructura, una variable, un periférico interno del microcontrolador, un dispositivo externo como una memoria serial, un sistema de almacenamiento, etc.

Se dice que un recurso es compartido *(shared resource)*, cuando es usado por más de una tarea, debido a que cada tarea puede tener acceso exclusivo al recurso en un determinado espacio de tiempo, y prevenir que otra tarea lo haga, con el objetivo de prevenir daños en los datos o en los dispositivos, este proceso es denominado exclusión mutua *(mutual exclusión).*

- **Eventos (Events):** Son elementos de notificación hacia una tarea para indicar la ocurrencia de un hecho o estímulo que interesa a una o varias tareas que se están ejecutando. Los eventos pueden llegar ya sea del hardware interno o externo al

procesador, o bien pueden ser eventos de software: *timeouts*, interrupciones o un resultado de la operación de una función.

- **Semáforos (Semaphores):** Son mecanismos de control que proveen al RTOS una forma de prevenir que múltiples tareas realicen acceso al mismo recurso al mismo tiempo. Se usan además para indicar la ocurrencia de un evento, y permiten sincronizar varias actividades dentro del sistema. Una vez una tarea va a iniciar un proceso con un recurso, pregunta por el estado del semáforo; si el semáforo está activo la tarea quedará en el estado de ***wait,*** si no está activo, lo toma y lo bloquea, realiza su operación en el recurso y una vez termine, libera el semáforo, permitiendo que otras tareas a partir de ese momento puedan hacer uso del recurso.

Los problemas con los diseños de semáforos son bien conocidos: inversión de prioridades y puntos muertos.

En la inversión de prioridades, una tarea de mucha prioridad espera porque otra tarea de baja prioridad tiene un semáforo. Si una tarea de prioridad intermedia impide la ejecución de la tarea de menor prioridad, la de más alta prioridad nunca llega a ejecutarse. Una solución típica sería tener a la tarea que tiene el semáforo ejecutada a la prioridad de la tarea que lleva más tiempo esperando.

En un punto muerto, dos tareas tienen dos semáforos pero en el orden inverso. Esto se resuelve normalmente mediante un diseño cuidadoso, realizando colas o quitando semáforos, que pasan el control de un semáforo a la tarea de más alta prioridad en determinadas condiciones.

- **Bloques de memoria (buffers):** Son elementos de almacenamiento continuos para la capa de aplicación, permite crear listas circulares, FIFO o LIFO o de cualquier tamaño. Las listas pueden crearse y borrarse de forma dinámica, permitiendo de esta forma un manejo eficiente de la memoria. Funcionan de forma similar a las funciones malloc() y free() para el manejo dinámico de memoria.

El concepto del Buffer es similar al de caché. Pero en el caso del buffer, los datos que se introducen siempre van a ser utilizados. En la caché sin embargo, no hay seguridad, sino una mayor probabilidad de utilización.

- **Reloj y Timers (Clock Tick & Timers):** El *"clock tick"* o *tick* es una interrupción que ocurre de forma periódica, le permite al RTOS contar el tiempo que entrega a cada tarea y manejo de *timeouts*.

El valor del *tick* de frecuencia más alta genera mayor preámbulo (overead) en el sistema y tiene un mayor consumo de energía, debido a que debe tomar decisiones más veces por segundo, lo cual puede ser crítico para el desempeño del microcontrolador (mayor gasto de energía). Sin embargo un tiempo muy largo de *tick* ocasiona que las tareas tengan tiempos de respuesta más lentos y que la sensación de "tiempo real" se vea afectada.

Es responsabilidad del diseñador la selección adecuada del valor del *tick* dependiendo de los requerimientos de velocidad, consumo y tiempo de respuesta requerido.

- **Kernel:** Es el responsable de la distribución del tiempo del procesador para el manejo de las tareas, además de la comunicación entre ellas. El servicio fundamental del *kernel* es el cambio de contexto (context switching).

El uso de un *kernel* de tiempo real generalmente simplifica el diseño de sistemas permitiendo que la aplicación pueda ser dividida en múltiples tareas que el *kernel* administra una a una.

La parte del *kernel* que se encarga de determinar cual tarea debe ejecutarse es el *scheduler*, el que basado en la prioridad, pasa el control de la CPU.

5.8 Herramientas de simulación, depuración y emulación.

Los principales fabricantes de microcontroladores están en una competencia continua por mejorar sus productos e introducirlos en las industrias, por ello debe ofrecer herramientas como tutoriales, módulos electrónicos de aprendizaje, software para la compilación, depuración, simulación y emulación de las aplicaciones de los usuarios. Generalmente los fabricantes centran su interés en producir sus microcontroladores, por lo que prefieren que otras empresas afines sean las que desarrollen las herramientas de desarrollo mencionadas, por ejemplo para los microcontroladores de microchip tenemos a la empresa *Custom Computer Services* CCS [30] y para los MCU de *Freescale* tenemos a *PEmicro* [31].

Para el caso del proyecto de esta tesis, se tomarán en cuenta los microcontroladores de *Texas Instrument* y *Freescale*, debido a que ambos poseen características que permiten un muy bajo consumo de potencia, los módulos de desarrollo de los fabricantes mencionados se muestran en la Figura 5-16, éstos cuentan con entradas y salidas digitales, entrada analógica, interfaz para RS232,

programador incluido y demás periféricos útiles para aprender a diseñar con estos microcontroladores.

Figura 5-16: Herramientas de desarrollo para sistemas embebidos

5.9 Sistemas en modo ultra bajo consumo

El consumo energético es importante en todos los sistemas electrónicos, por ello los fabricantes desarrollan dispositivos de consumo cada vez menor. En el caso de los MCU la tecnología Flash de las memorias disminuye el consumo de éstos, también la frecuencia de funcionamiento influye en el consumo, a menor frecuencia el consumo será menor por ello se recomienda configurar los PLL en baja frecuencia, los MCU de bajo consumo permiten desconectar físicamente los periféricos deseados, por lo tanto dichos periféricos no consumirán energía. También incluyen instrucciones tipo WAIT o STOP que ponen en modos de espera o detienen la ejecución de los procesos a menos que sean interrumpidos [32].

Un factor importante en el consumo de un microcontrolador es el grueso del silicio. Las obleas de silicio con un grueso de 1.2 micras han pasado a 0.85, 0.6, 0.35 y actualmente hay con 1.8 micras. Así mismo que se baja el espesor disminuye la

superficie del silicio y baja el costo. Y al reducir el grueso del silicio también se ha podido bajar la tensión de alimentación, de 5V a 3.3V y ya se trabaja a 1.8V [32].

5.10 La cámara C328R de COMedia Ltd.

El módulo C328R, ver Figura 5-18, es una cámara tipo VGA que genera imágenes en formato comprimido JPEG, la configuración de las características de la imagen y la descarga de la misma se hace mediante comandos en protocolo UART. A continuación se muestra en la Figura 5-17 un diagrama de bloques de la cámara, obtenida de la hoja técnica del fabricante.

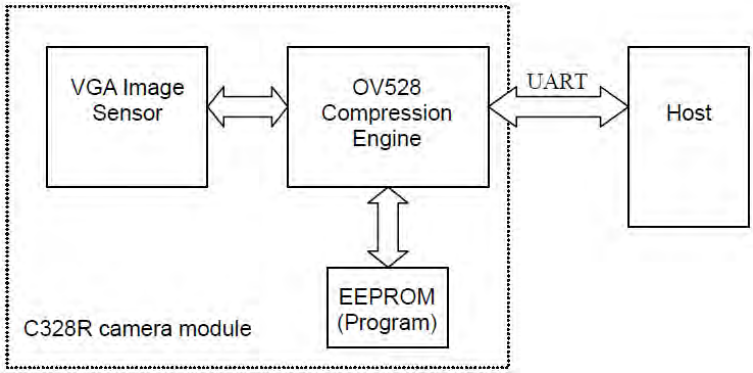


Figura 5-17: Diagrama de bloques de la cámara C328R

Características:

- Tamaño, 20x28mm
- Tipo VGA
- Bajo consumo de potencia 60mA
- Comandos de control sencillos para el usuario
- Interface UART de hasta 115200bps
- Auto detección de *baud rate* y conexión con el host
- Incluye modos de operación para bajo consumo.

- Varias configuraciones como tamaño, resolución, etc.

Figura 5-18: Cámara UART C328R.

Set de comandos: La Figura 5-19, tomada de la hoja técnica de la cámara, resume el set de comandos necesarios para el manejo de la toma de una foto, además en dicha hoja técnica también se muestran los algoritmos de configuración y la toma de foto en varios formatos [33].

Command	ID Number	Parameter1	Parameter2	Parameter3	Parameter4
Initial	AA01h	00h	Color Type	RAW Resolution (Still image only)	JPEG Resolution
Get Picture	AA04h	Picture Type	00h	00h	00h
Snapshot	AA05h	Snapshot Type	Skip Frame Low Byte	Skip Frame High Byte	00h
Set Package Size	AA06h	08h	Package Size Low Byte	Package Size High Byte	00h
Set Baudrate	AA07h	1st Divider	2nd Divider	00h	00h
Reset	AA08h	Reset Type	00h	00h	xxh*
Power Off	AA09h	00h	00h	00h	00h
Data	AA0Ah	Data Type	Length Byte 0	Length Byte 1	Length Byte 2
SYNC	AA0Dh	00h	00h	00h	00h
ACK	AA0Eh	Command ID	ACK counter	00h / Package ID Byte 0	00h / Package ID Byte 1
NAK	AA0Fh	00h	NAK counter	Error Number	00h
Light Frequency	AA13h	Frequency Type	00h	00h	00h

Figura 5-19: Set de comandos a la cámara C328R.

5.11 Diseño y fabricación de tarjetas impresas – PCB

A continuación describiremos los pasos necesarios para desarrollar un PCB, teniendo en cuenta ciertas consideraciones de diseño. Estos pasos consisten

básicamente en desarrollar el esquemático de nuestro circuito y luego en base a éste desarrollamos el PCB. Para esto se hace uso de software CAD, los cuales permiten desarrollar tanto el esquemático como el PCB. Este proceso de desarrollo consiste básicamente en hacer el dibujo en software de nuestro circuito, primero de manera esquemática, dónde cada componentes es representado por algún símbolo estandarizado, en nuestro caso se usó la simbología del estándar IEEE. Luego este esquema es convertido por el software CAD a su equivalente en diagrama PCB, es decir cada componente es remplazado por su forma real en la tarjeta impresa. De esta forma el desarrollador solo consiste en ubicar convenientemente los componentes y hacer el ruteo de las pistas. Algunos softwares permiten que estas labores también sean automáticas, pero es usual que uno prefiera hacerlas manualmente aplicando consideraciones de diseño no garantizadas por dichas herramientas automáticas.

Además, muchas veces es necesario hacer un diseño 3D del circuito para tener consideraciones dimensionales de la ubicación de los componentes para tener una buena distribución de los mismos, de esta manera también puede tenerse en cuenta la estética que tendrá nuestro circuito ensamblado. Luego del diseño del PCB viene la fabricación de la tarjeta y el montaje y soldadura de sus componentes, para lo cual existe también técnicas y recomendaciones a seguir, la Figura 5-20 muestra un resumen de estos pasos a seguir para desarrollar una tarjeta impresa.

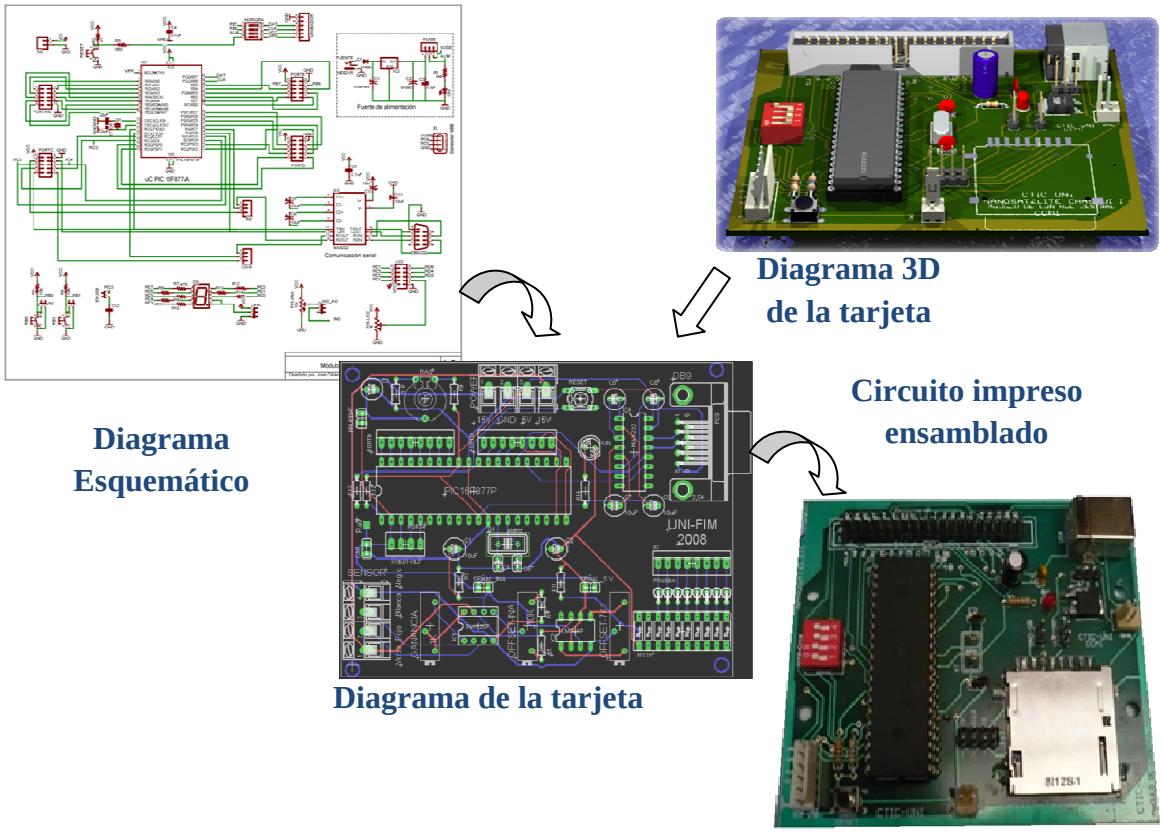


Figura 5-20: Proceso de diseño y fabricación de una tarjeta impresa.

5.11.1 Software CAD Easily Applicable Graphical Layout Editor - EAGLE

Este software CAD desarrollado por *CadSoft* [34] permite diseñar circuitos impresos fácilmente, es potente y contiene muchas herramientas útiles para el desarrollador, permite hacer la captura esquemática y conversión a equivalente PCB de sus componentes, contiene además una amplia cantidad de librerías de componentes listos para ser incluidos en el esquemático. Muchas empresas fabricantes de componentes han desarrollado librerías de sus productos para EAGLE. Además tiene un entorno para crear componentes que no se puedan encontrar en algunas de sus tantas librerías. La versión libre de EAGLE permite desarrollar circuitos de dimensiones hasta 100x80mm y solo dos capas para el ruteo de las señales.

5.11.2 Herramienta de visualización tridimensional EAGLE-3D

Esta herramienta permite visualizar en forma tridimensional, ver Figura 5-21, las tarjetas impresas, esto es útil para determinar la mejor ubicación de los componentes y su distribución en la tarjeta impresa, además para tener consideraciones estéticas del mismo y mejorar nuestras presentaciones. Básicamente EAGLE-3D consiste en una interfaz ULP que importa los parámetros de los componentes y sus coordenadas en el PCB [35], a su equivalente en diagramas 3D, para esto se usa otro software libre denominado POVRAY [36], en éste se desarrolló una librería de cientos de componentes renderizados, luego en la interfaz de POVRAY podemos ejecutar el código generado por EAGLE-3D y generar así nuestra visualización tridimensional.

Figura 5-21: Dibujo 3D del circuito electrónico desarrollado en EAGLE 3D [35].

5.11.3 Consideraciones de diseño para tarjetas impresas

Para tener un buen diseño se debe hacer un ruteo sencillo, con las pistas lo más cortas posibles, especialmente cuando se transmiten señales de datos, a continuación listaremos algunas de estas consideraciones.

- Placa de tierra especialmente donde haya zonas de señales de alta frecuencia.
- Ángulos de pistas deben ser mayores a 45°

- Las dimensiones de las pistas dependen de la cantidad de corriente eléctrica que pasará por ella.

5.12 Proceso de testeo de funcionamiento

Este proceso consiste en comprobar el correcto funcionamiento de nuestra tarjeta electrónica, es decir para una entrada debemos obtener la salida deseada, este análisis debe hacerse para todas las posibles combinaciones de entradas, además de cumplir este requisito, el sistema debe someterse a perturbaciones como son el ruido eléctrico, campo magnético y la radiación y el resultado debe continuar siendo el correcto.

5.13 Pruebas para certificación aeroespacial

Las pruebas mencionadas en este punto son exigidas por los proveedores del lanzamiento para garantizar la seguridad de éstos, la de los *CubeSat* y la del P-POD.

Si los requerimientos del vehículo lanzador son desconocidas, entonces puede usarse el documento de la NASA GSFC-STD-7000 [37] para determinar los requerimientos de las pruebas, ésta es una referencia útil pero no necesariamente coincide con las exigencias de los vehículos lanzadores. Los planes de prueba que no sean considerados por los lanzadores pueden ser tomados del GSFC-STD-7000.

Las pruebas que todo lanzador exige son las que se muestran a continuación [38]:

- **Prueba de vibración randómica:** El *CubeSat* completamente integrado es sometido vibración randómica relacionadas al vehículo lanzador.

- **Prueba de vacío térmico:** Se simula el medio ambiente de operación espacial.
- **Inspección visual:** Verificación de las características dimensionales del *CubeSat* antes de integrarse al P-POD.
- **Protoflight:** Se usa un prototipo de vuelo que simula el comportamiento del vehículo lanzador, esta prueba se realiza en conjunto con los proveedores del lanzamiento y con sus equipos.
- **Aceptación:** Luego de la inspección visual del *CubeSat*, sigue la integración de este en el P-POD, y las pruebas mencionadas anteriormente se realizan nuevamente para conseguir la aceptación final para el lanzamiento del nanosatélite.

Capítulo 6

6 Diseño e implementación del Software

El desarrollo del módulo CCMI implica el diseño e implementación del software y del hardware, no en ese orden, ya que ambos son dependientes de su evolución, generalmente un cambio en el hardware implica un cambio también en el código programado en el microcontrolador, mucho más si el cambio es de microcontrolador, por ello se debe tener cuidado al momento de elegir los nuevos componentes para nuevas versiones de la tarjeta CCMI, en lo posible conseguir dispositivos compatibles con el algoritmo programado. En la edición de esta tesis se decidió separar el desarrollo del software y del hardware para no hacer un capítulo demasiado extenso, al dividir el desarrollo del módulo CCMI en dos capítulos independientes podemos también ordenar mejor nuestras ideas para la comprensión del diseño e implementación global del sistema.

6.1 Herramientas para depuración e implementación del software

En esta sección describiremos las principales herramientas de software que se emplearon para el diseño e implementación de los algoritmos que debe ejecutar nuestro uC. Además se muestra sobre el IDE, este debe permitirnos depurar el código, simular el funcionamiento independiente del hardware, monitorear el estado de las variables y los registros internos al uC. Además debe contener herramientas que permitan ejecutar el código paso a paso, tanto en nivel del lenguaje C como del *assembler*. Otro requisito útil del IDE es que pueda actualizar los valores de sus registros en tiempo de ejecución, es decir sin necesidad de detener el CPU del uC para hacer dicha visualización, éstas características nos permitirán a la larga ahorrar tiempo cuando hagamos depuración, ya que nos permite ubicar los errores con facilidad. La importancia de estas características que debe poseer el IDE influye directamente en la selección del uC a emplear, esta selección se muestra en la Subsección 7.3, dónde se muestra el estudio realizado para elegir como mejor opción al microcontrolador de *Freescale* MC9S08QE128, cuyo IDE, denominado *Freescale CodeWarrior*, cuenta con todas las características mencionadas anteriormente.

6.1.1 Entorno de desarrollo integrado Freescale CodeWarrior IDE

A continuación mostramos en la Figura 6-1 los dos entornos de trabajo, en el primero se digita el código del programa y en el segundo se depura y emula nuestros algoritmos.

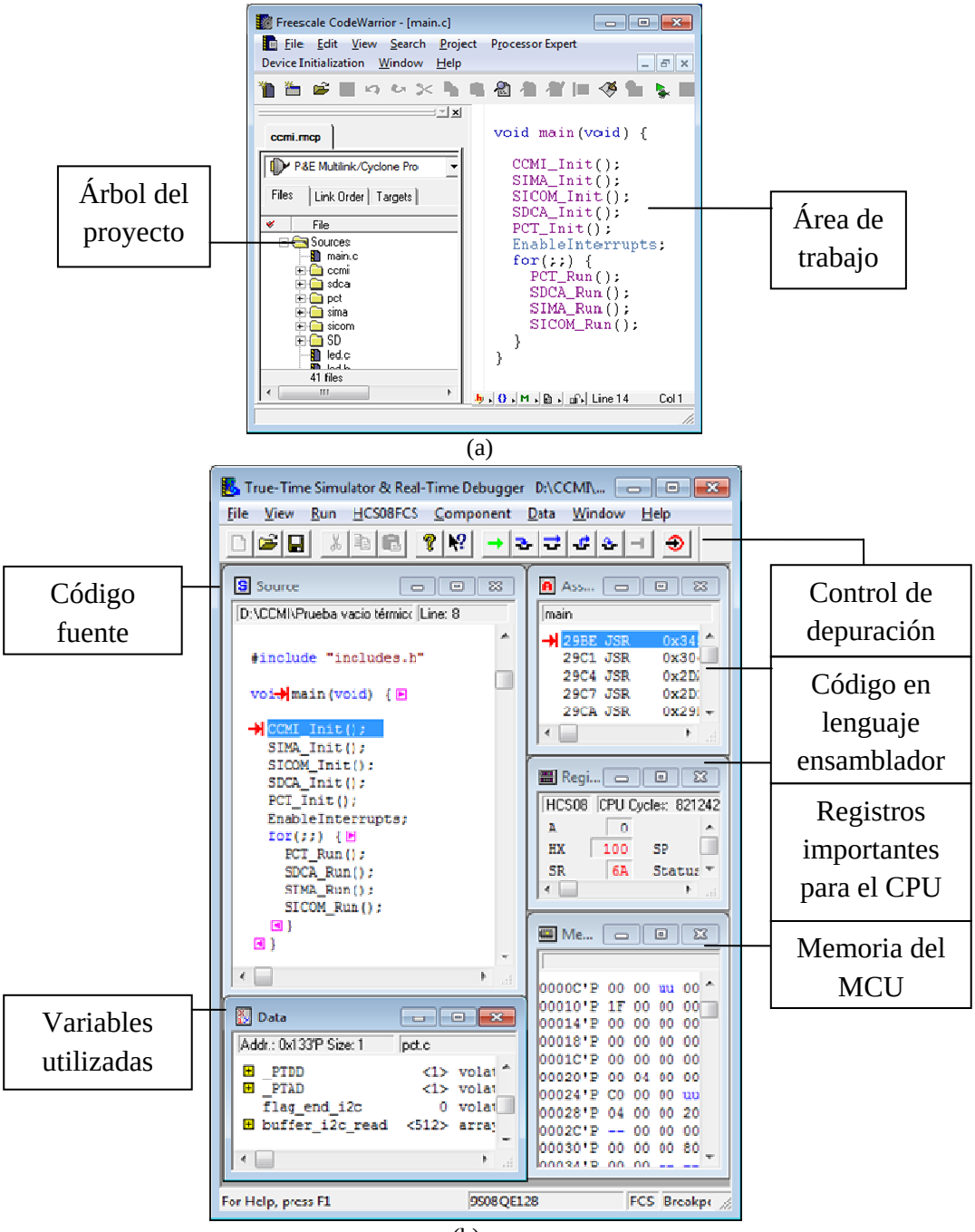


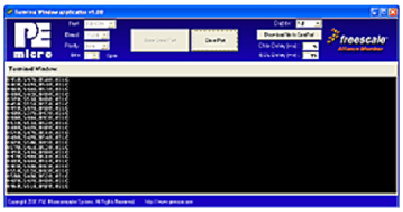
Figura 6-1: Entorno Integrado de desarrollo CodeWarrior. (a) Freescale CodeWarrior IDE. (b) True-Time Simulator & Real-Time debugger.

6.1.2 Herramienta P&E Embedded Multilink Toolkit

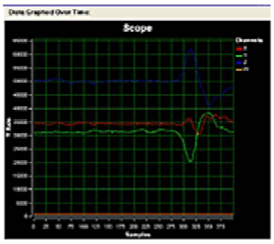
Estas herramientas están incluidas en el paquete de *Freescale* y vienen con el módulo de desarrollo DEMOQE [31], son bastante útiles y prácticas ya que se

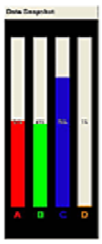
ajustan a aplicaciones embebidas como la nuestra, la Figura 6-2 muestra las 3 principales herramientas de *multilink*.

LogicAnalyzer
Usado para visualizar, en tiempo real, el estado lógico de dos pines incluidos en el módulo de pruebas DEMOQE de *Freescale*.



Terminal
Esta herramienta es una simple interfaz del *Virtual Terminal* para leer y escribir desde el puerto serial mediante protocolo UART, la visualización de los datos se limita al formato ASCII.





Serial Grapher
Esta herramienta grafica el valor leído del *Virtual Terminal*, para ello el dato transmitido debe cumplir ciertas características para que pueda ser interpretado por esta interfaz visual.

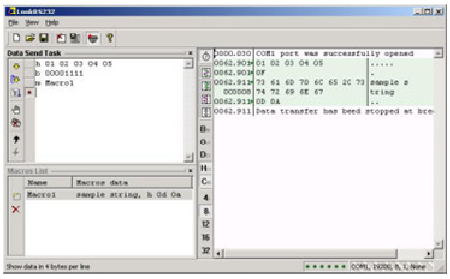
Figura 6-2: Herramientas P&E Embedded Multilink Toolkit

6.1.3 Interfaz de monitoreo de memoria SD

Además del compilador y las herramientas de simulación del uC HCS08 de *Freescale semicondutor*, fue necesario otras herramientas de software como el *Look-RS232* en su versión de prueba descargada de la página de los creadores [39], además del software *ICY Hexplorer* [40]que nos permite visualizar los datos grabados en la memoria SD.

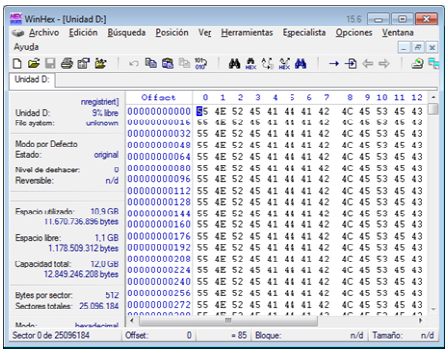
La principal diferencia entre estos programas es que el ICY *Hexplorer* nos permite visualizar mayor cantidad de datos debido a que el formato del software

muestra su información con un tamaño de letra muy pequeño comparado con el software *WinHex* [41]. Las interfaces de dichos programas se muestran en la Figura 6-3.



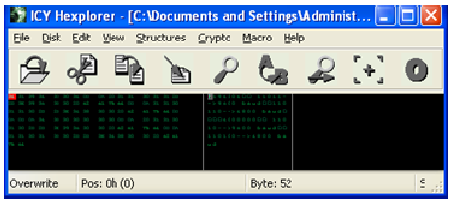
LookRS232

Es una interfaz para visualizar el flujo de datos del puerto serial, la ventaja que tiene este Virtual Terminal es que permite visualizar los datos en distintos formatos, como son el ASCII, hexadecimal, decimal, además del tiempo del evento de llegada o transmisión.



WinHex

Permite visualizar los bancos de memoria, será usado para monitorear la información almacenada en la tarjeta de memoria SD, se debe aclarar que con este software no nos limitamos a ver archivos creados en dicha memoria, sino todo el contenido real en los bloques de 512 de la tarjeta SD.



ICY Hexplorer

Cumple la misma función del *WinHex*, su ventaja es que al tener un tamaño de letra mucho menor para mostrar los valores, permite visualizar un mayor número de datos, esto es útil cuando solo se necesita saber la existencia de datos almacenados, sin necesidad de ver el valor de los mismos, de ser este el caso se debe usar el *WinHex*.

Figura 6-3: Software empleados para visualizar el puerto serie en distintos formatos y la información almacenada en la memoria de la tarjeta SD.

6.2 Diseño de los modos de operación y diagramas de flujo

Los modos de operación son los estados en los que se encontrará el nanosatélite Chasqui-I desde su lanzamiento del P-POD. En el primer modo, llamado

“modo lanzamiento” se realizarán ciertas rutinas para inicializar cada subsistema, terminadas estas rutinas se llega al estado de “modo espera”. El nanosatélite se mantendrá en dicho modo hasta que se encuentre una señal desde la estación terrena, momento en el cual se pasará al “modo normal” donde ejecutará rutinas en función de lo recibido de Tierra. Cuando termine la comunicación con Tierra, el sistema debe regresar al “modo espera”. Si ocurre algún problema en el “modo normal” o en el “modo espera” el sistema debe entrar al modo emergencia del cual saldrá cuando se solucione el problema. Al finalizar dicho modo se resetean nuevamente todos sus subsistemas e inicializan sus variables y módulos de los MCUs de todo el satélite.

La Figura 6-4 muestra gráficamente esta secuencia de estados o modos de operación del nanosatélite, podemos notar que la secuencia se puede representar como una máquina de estados, por ello se hizo el diseño de una máquina que represente de forma básica esta secuencia, la cual se muestra en el Anexo A1.

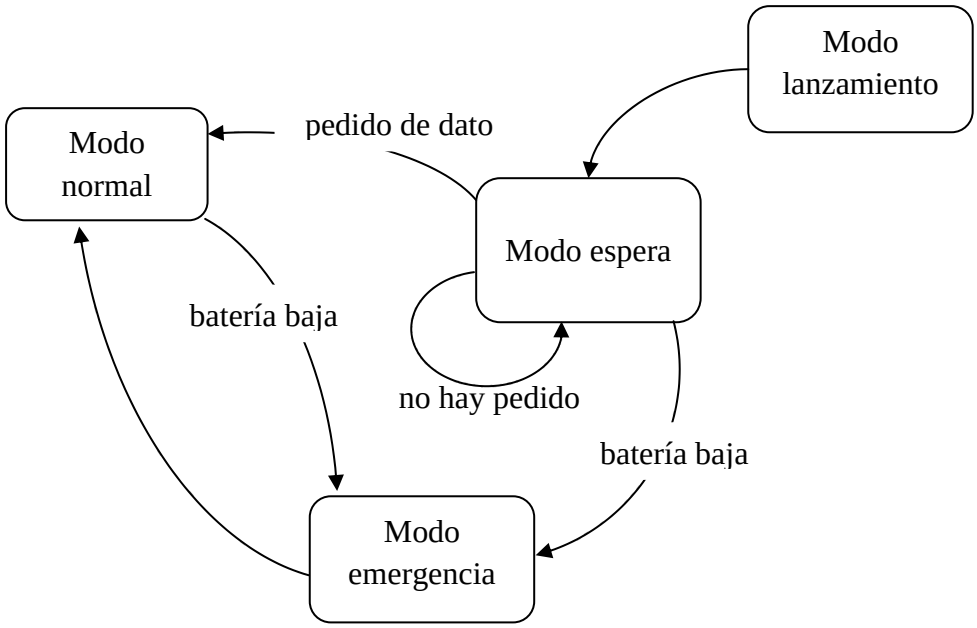


Figura 6-4: Modos de operación del nanosatélite Chasqui

6.2.1 Modo lanzamiento

Este modo se da una única vez, y se inicia en el instante en que el Chasqui es expulsado del P-POD, hasta que se hayan desplegado las antenas, y el nanosatélite se haya orientado por primera vez, terminado este proceso, el cual dura aproximadamente 30 min, se inicia el modo espera.

Se espera que en este modo las baterías se encuentren completamente cargadas, debido a que todo el sistema estará apagado hasta el momento en que el Chasqui es expulsado del P-POD, es en ese instante en que automáticamente se activa un *switch* normalmente cerrado, que encenderá al nanosatélite, si por algún motivo las baterías inician descargadas deberá esperarse un tiempo de recarga, por medio de las celdas solares pegadas en las caras del cubo, para que así pueda iniciarse las rutinas del modo lanzamiento con normalidad.

A continuación se muestra la secuencia de estados del modo lanzamiento:

1. PCT energiza a CCMI para que pueda inicializar.
2. Una vez iniciado CCMI, este debe pasar a bajo consumo, y debe esperarar a que el temporizador de CCMI expire. (Tiempo de inactividad después del lanzamiento: 15 minutos)
3. El módulo SICOM debe ser energizado que despliegue las antenas.
4. El paso al modo control requiere una espera adicional de 15 minutos. Durante ese tiempo, se podría tener al SICOM en LPTM (“*Low power transmission mode*”, definido en las especificaciones del *CubeSat* como señales de “beacon” cortas y periódicas)

5. El modulo CCMI pasa a bajo consumo una vez autorizado el suministro de energía a los otros módulos, los cuales también deben pasar a modo bajo consumo.

En la Figura 6-5 se muestra un diagrama del flujo de señales descritos en este punto.

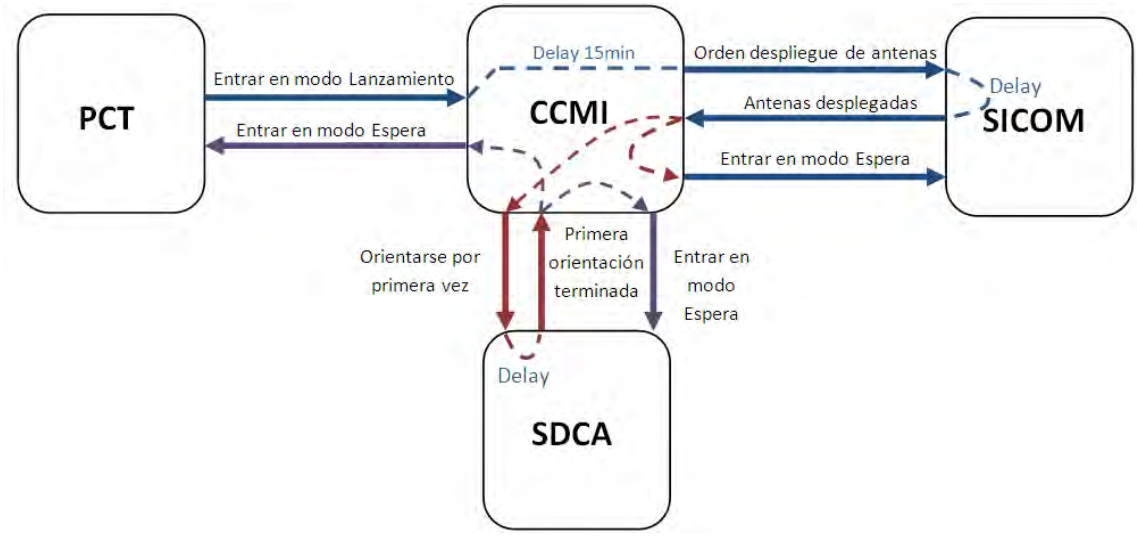


Figura 6-5: Diagrama de flujo de señales en el modo lanzamiento.

Luego de definir el diagrama de flujo de señales anterior, ahora sigue el diagrama de flujos para la programación del MCU, el cual se muestra en la Figura 6-6.

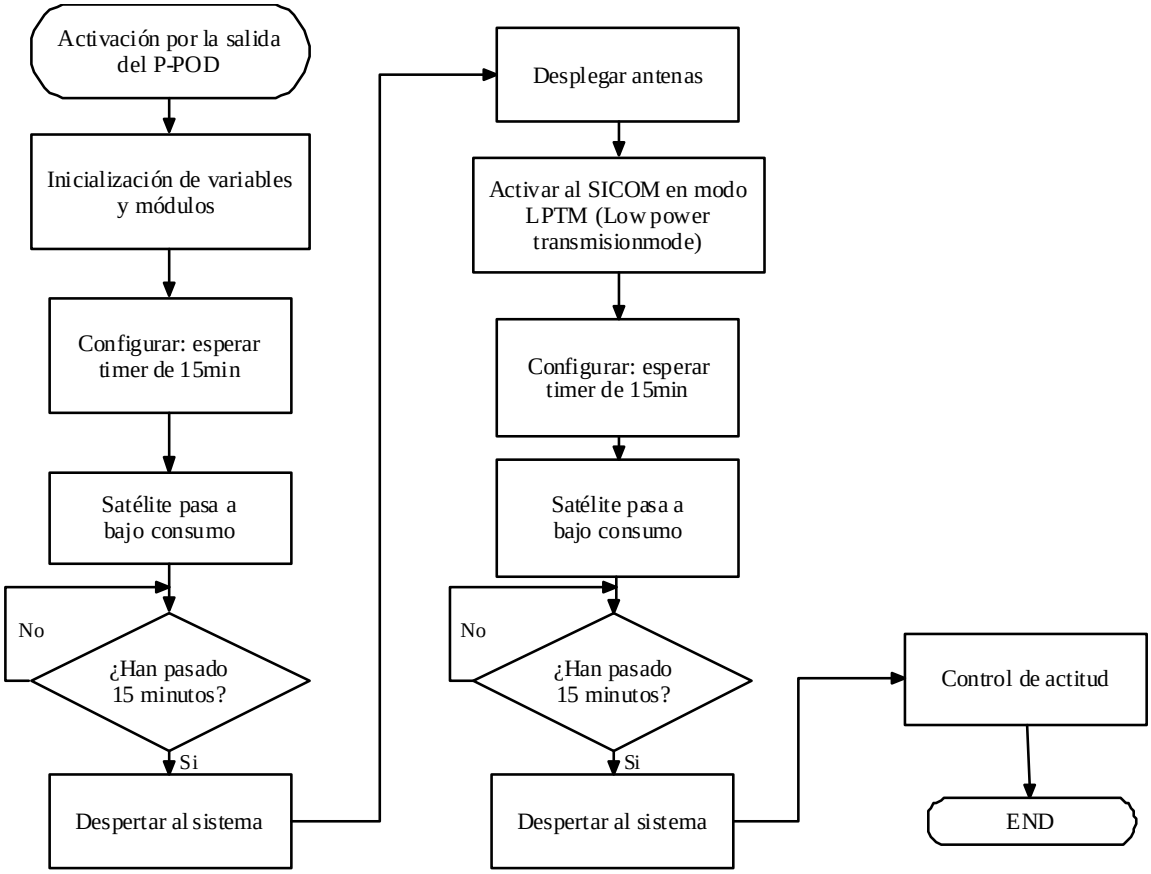


Figura 6-6: Diagrama de flujo del modo lanzamiento

6.2.2 Modo espera

La característica más importante de este modo es que todos los módulos están en modo de bajo consumo. El nanosatélite Chasqui pasará la mayor parte de su funcionamiento en este módulo, ya que en general estaremos en este modo cuando no haya comunicación con tierra, lo cual se da dos veces al día en un tiempo no mayor a 15 minutos.

1. Primero, el CCMI ordena pasar a este modo cuando el SICOM le indica que se perdió o terminó la comunicación con Tierra.
2. Luego de esto, el CCMI ordenará a los demás módulos pasar a modo bajo consumo.

3. El SICOM intentará contactar con Tierra nuevamente.
4. Al lograr contactar con Tierra, se informa al CCMI para pasar nuevamente al modo normal.

En todo este proceso solo requerirá energía el módulo SICOM.

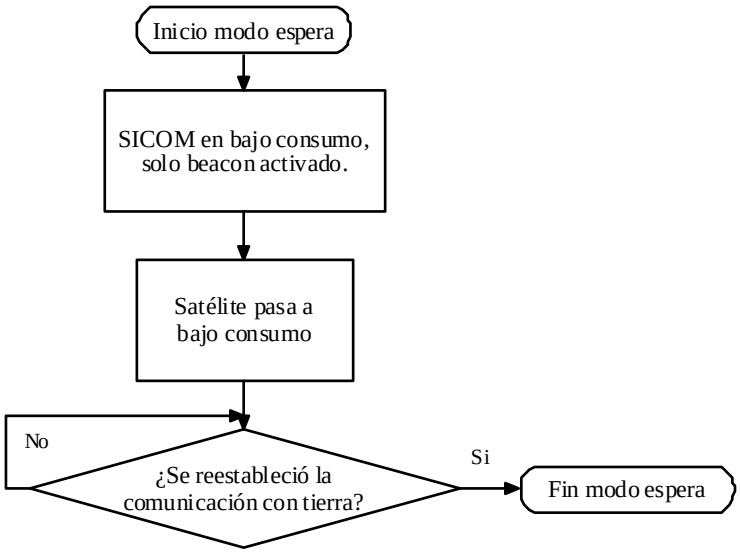


Figura 6-7: Diagrama de flujo del modo espera

6.2.3 Modo normal

Este modo se da cuando hay comunicación con Tierra desde ESTER, en general esto es para que CCMI pueda entregar al SICOM la información solicitada por ESTER.

Esta secuencia de estados o flujo de procesos por los que debe pasar el nanosatélite Chasqui se muestra en la Figura 6-8.

1. Primeramente el SICOM debe estar activado sin ninguna restricción de energía por parte de PCT.
2. El SICOM recibe las tramas de Tierra y se las transmite a CCMI.
3. CCMI procesa comandos de Tierra.

4. Cualquier módulo puede ser activado y requerir energía momentáneamente bajo petición del CCMI, para pedir estado y reportarlo a Tierra, o para funciones particulares como la toma de foto.
5. CCMI debe obtener los datos solicitados y enviarlos al SICOM para que pueda transmitirlos a Tierra.

Para este proceso requerirán energía los módulos, CCMI, SICOM, SDCA y PCT, la energización de estos dos últimos será en función de los comandos recibidos.

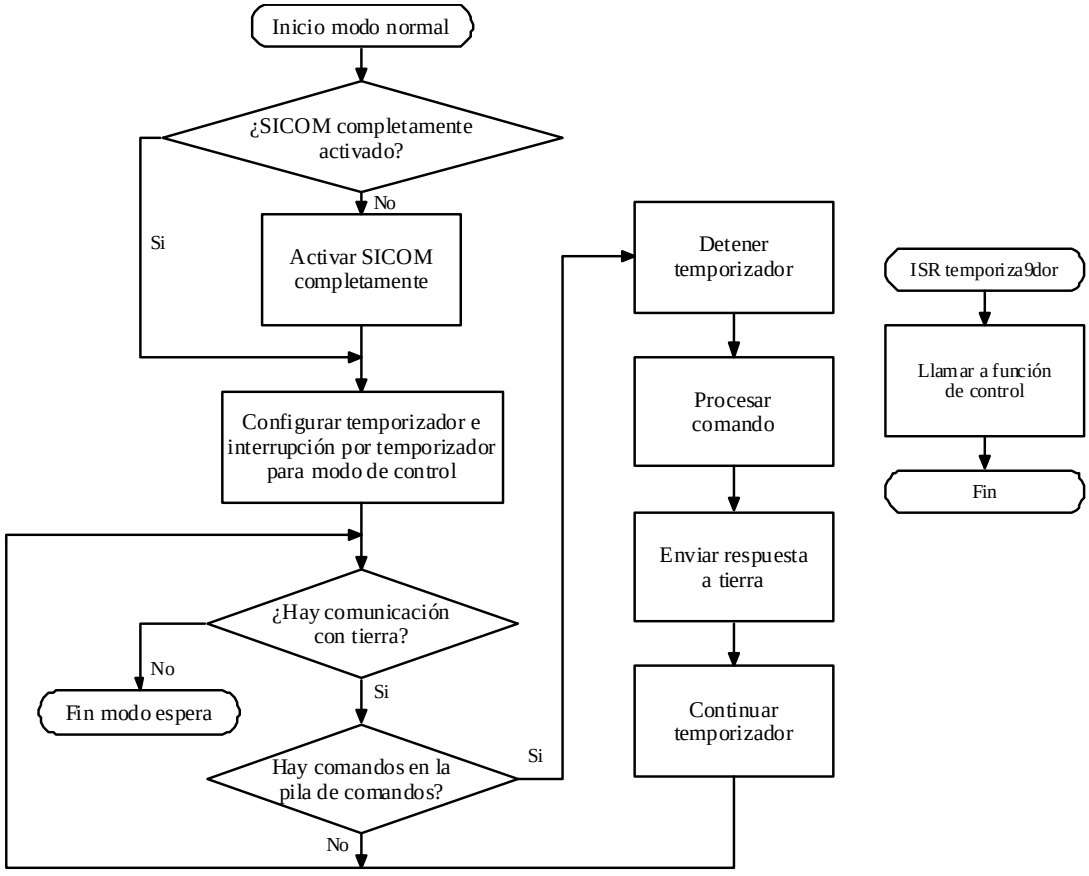


Figura 6-8: Diagrama de flujo del modo normal

6.2.4 Modo emergencia

Este modo se da cuando el sistema cuenta con la energía insuficiente para cumplir sus funciones con normalidad, es decir saldrá de este estado hasta que las

baterías se vuelvan a cargan usando para ello los paneles solares, el diagrama de flujo de este modo se muestra en la Figura 6-9.

1. Se plantea que cualquier módulo pueda solicitar pasar a este modo (por lo menos hasta determinar exactamente todas las condiciones “de emergencia”).
2. El CCMI debe indicar al PCT que pase a este modo, y éste debe cortar la alimentación a los módulos correspondientes. Los módulos que no se des-energicen, sino que pasen a bajo consumo, deben hacerlo por orden del CCMI. Al terminar, el CCMI pasará a bajo consumo.
3. El SICOM enviará una señal (*beacon*) que indique que está en modo de emergencia. El SICOM, al igual que PCT, es el único módulo que no podrá apagarse completamente.
4. Cuando el problema se resuelva, el PCT debe informar al CCMI, y regresar al modo normal.

El SICOM debe estar inicializado solo para envío de *beacon.*

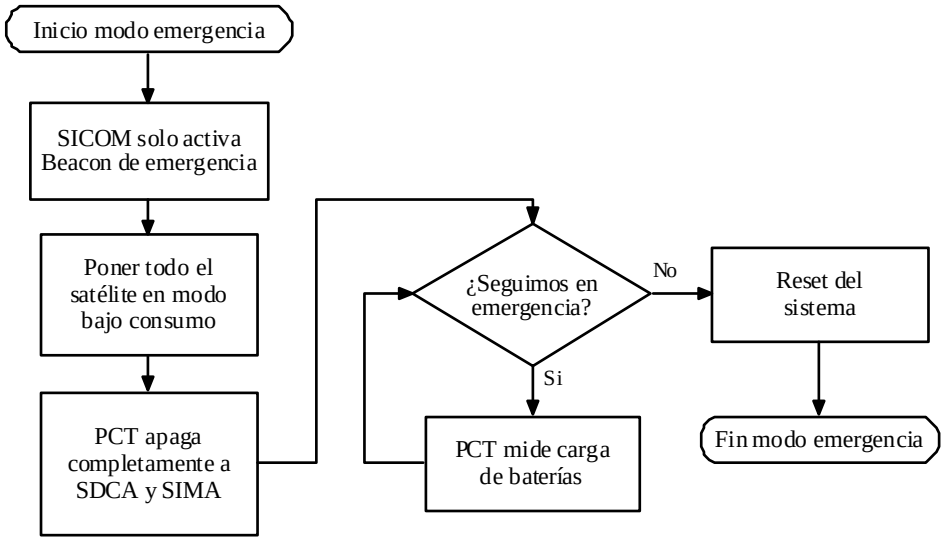


Figura 6-9: Diagrama de flujo del modo emergencia

6.2.5 Función de control

Esta función describe características de funcionamiento para el módulo SDCA, indica el momento en el que debe pasar su sistema a bajo consumo, además de los momentos en los que se debe activar y desactivar, este flojo se detalla en el diagrama de flujo mostrado en la Figura 6-10.

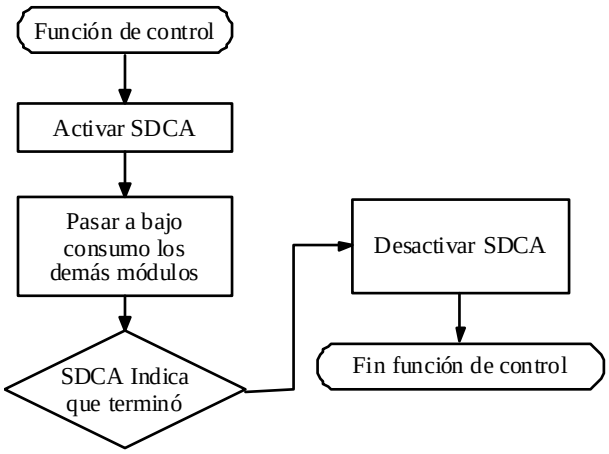


Figura 6-10: Función de control

a) Modo de control para estabilización

Este modo es para el módulo SDCA, y sirva para indicar determinar su comportamiento para el modo lanzamiento, es decir, será la primera función de este módulo.

- SDCA en modo "*Detumbling*"
- CCMI en espera.
- Al terminar se avisa al CCMI

Requerirán energía los módulos: SDCA

b) Modo de control para orientación

Este modo particular del módulo SDCA se da cada vez que se requiera una orientación del nanosatélite, ya sea para tomar una fotografía o para transmitir los datos, este modo no es el que se presenta la primera vez.

- El CCMI ordena entrar en este modo cada cierto tiempo, cuando la actitud esté muy lejos del *setpoint*, o a petición de Tierra.
- SDCA elige su sub-modo dependiendo de la discrepancia en actitud y las circunstancias.
- SICOM debe estar recibiendo comandos, y el CCMI debe estar acumulándolos en pila.

Requerirán energía los módulos:

- SDCA
- CCMI (posiblemente en bajo consumo, depende de las características de la función “DMA” en el MCU)
- SICOM en modo recepción (quizá transmisiones de confirmaciones)

6.2.6 Características de comunicación del módulo CCMI con PCT, SDCA, SICOM y SIMA.

El proyecto satelital Chasqui-I, cuanta con diferentes módulos cada uno encargado de tareas específicas, tal como se mostró en el Capítulo 4, donde vemos que estos módulos se dividen en dos grupos, aquellos que serán incluidas directamente dentro de la estructura del nanosatélite y aquellos cuya aplicación se da

fuera del nanosatélite, los primeros se materializan en una tarjeta electrónica la cual debe cumplir su función establecida. Los módulos incluidos internamente deberán comunicarse con el módulo CCMI, el cual se encargará en todo momento de gestionar la información entre ellos.

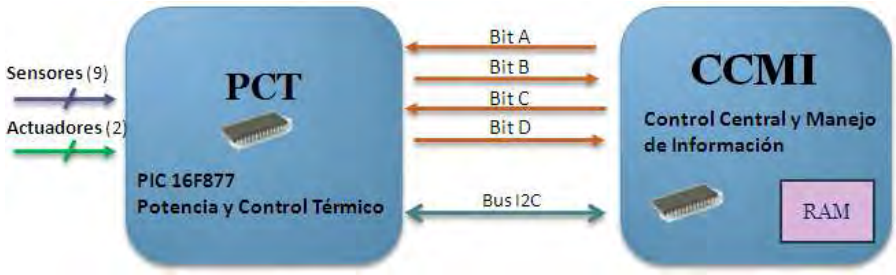
- **CCMI → PCT:** La función del módulo PCT es controlar las condiciones de potencia y temperatura del nanosatélite, además debe indicarle al módulo central si es que nos encontramos en modo emergencia, la Tabla 6-1 muestra las características de comunicación del módulo PCT.

Tabla 6-1: Características de comunicación del módulo PCT

Alimentación	
+3,3V; Tierra	PCT
Entradas binarias (BIT)	
(2) bit de disponibilidad	CCMI
Entradas analógicas	
(de cada sensor)	
Salidas digitales	
(2) bit de disponibilidad	CCMI
(3) Pin de salida de un bit	Actuadores
Sensores	
(2) sensor temperatura LM95075 (baterías)	
(5) Sensor temperatura (celdas solares)	
(1) Sensor de voltaje para las baterías	
(1) Sensor de corriente para las baterías	
Se comunica con	
CCMI (I2C)	
Actuador	
(2) Calentadores térmicos para las baterías	
Bus	
I2C	CCMI

En la Figura 6-11el MCU del módulo PCT se encarga de determinar el modo de operación basándose en la información de la lectura de sus sensores, además

podrá informar al módulo central la lectura dicha lectura usando el bus I2C. El módulo CCMI almacenará los datos de PCT en la tarjeta de memoria SD.



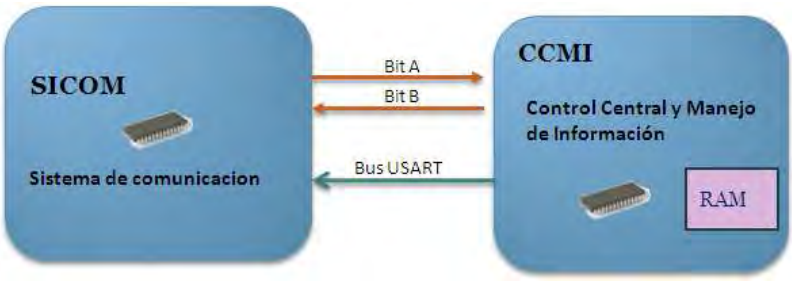
Bit A: Solicitud de datos de sensores
Bit B: Envío de datos de sensores
Bit C: Envío de bit indicador de operación
Bit D: Modo Emergencia

Figura 6-11: Sensores al uC del módulo PCT

- **CCMI → SICOM:** El módulo central gestiona la comunicación con la estación terrena a través del módulo SICOM quien se encarga del envío de los datos y comandos entre el nanosatélite y la estación Terrena, la Tabla 6-2 muestra las características de comunicación del módulo SICOM.

Tabla 6-2: Características de comunicación del módulo SICOM

Alimentación
+3.3V, +5V, Tierra
Entradas binarias (BIT)
(1) Bit información de emergencia(PCT)
Entradas analógicas
Vcc(3,3V)
Ondas de radio desde ESTER
Entradas digitales
Datos desde CCMI (UART)
Salidas digitales
Datos hacia CCMI (parámetros para SDCA)
Trama de comandos de ESTER a CCMI
Se comunica con:
CCMI
PCT
ESTER
Bus
UART a 9600bps



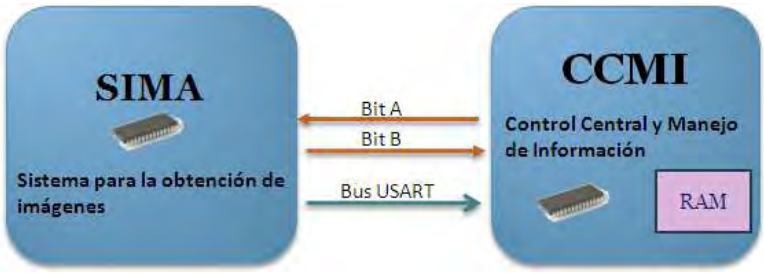
Bit A: Confirmación de envío Beacon
Bit B: Orden de envío Beacon
UART: Órdenes y data para ESTER

Figura 6-12: Comunicación SICOM-CCMI

- **CCMI → SIMA:** La función del módulo central en este caso es ordenar la captura de imágenes satelitales y su almacenamiento en una memoria al microcontrolador, la Tabla 6-3muestra las características de comunicación del módulo SIMA.

Tabla 6-3: Características de comunicación del módulo SIMA

Alimentación	
+3,3V; Tierra	PCT
Entradas binarias (BIT)	
(2) bit para reset de cámaras	CCMI
Se comunica con	
CCMI (UART)	



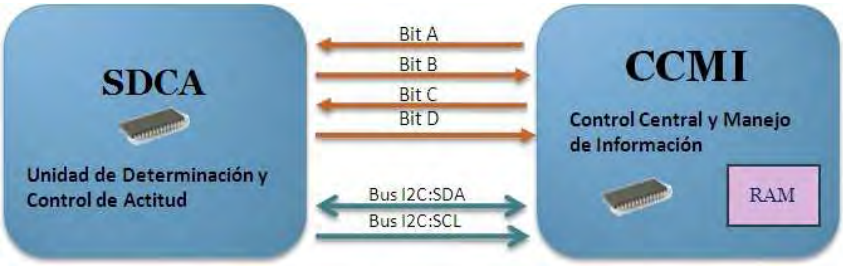
Bit A: Selector de cámara
Bit B: Sensor de Luminosidad (opcional)

Figura 6-13: Comunicación SIMA-CCMI

- **CCMI→SDCA**: El módulo central en este caso ordena y confirma solicitud de estabilización y orientación espacial del nanosatélite, la Tabla 6-4 muestra las características de comunicación del módulo SDCA

Tabla 6-4: Características de comunicación del módulo SDCA

Alimentación	
+3,3V; 5V;-5V; Tierra	PCT
Entradas binarias (BIT)	
(2) bit de disponibilidad	CCMI
Entradas analógicas	
(De cada sensor...)	
Salidas digitales	
(2) bit de disponibilidad	CCMI
(3) PWM	Actuadores
Sensores	
(10) sensor solar S65S60	
(3) Giroscopio 1-eje MLX90609	
(1) Magnetómetros 3-ejes HCM5843	
(1) GPS-Mini	
Se comunica con	
CCMI (I2C)	
Actuador	
(4) Imán Permanente	
(6) Magneto-torques	
Bus	
I2C (2)	CCMI



Bit A: Solicitud de orientación
Bit B: Respuesta de orientación
Bit C: Solicitud de auto-orientación
Bit D: Respuesta de auto-orientación

Figura 6-14: Comunicación SDCA-CCMI

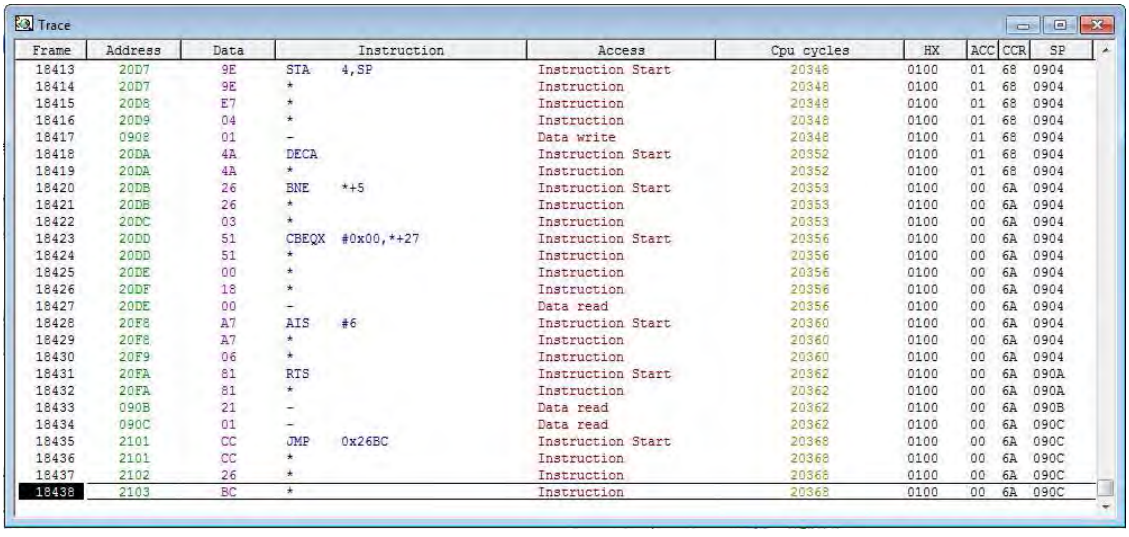
6.3 Organización de la memoria:

En esta parte mostraremos la evaluación de la necesidad de memoria de programa, de datos y de la memoria externa de almacenamiento masivo del tipo no volátil.

6.3.1 Evaluación de la necesidad de memoria del programa

La medición de este parámetro se hizo viendo el reporte de la compilación del IDE *CodeWarrior*, ver Figura 6-15, ya que nos permite visualizar las instrucciones compiladas a lenguaje ensamblador, reportándose un total de 17823 instrucciones, teniendo en cuenta que cada instrucción ocupa 2 bytes, entonces la cantidad de bytes que ocupa nuestro programa en total será de 35646 bytes (36KB), esta cantidad es mucho menor que la cantidad de memoria flash disponible en el MCU de CCMI, la cual equivale a 130KB.

En la Figura 6-15 del IDE de *CodeWarrior* podemos ver todo el código en lenguaje ensamblador, además notamos que se generó un total de 18438 instrucciones de 2 bytes cada una.


Trace

Frame	Address	Data	Instruction	Access	Cpu cycles	HX	ACC	CCR	SP
18413	20D7	9E	STA 4,SP	Instruction Start	20348	0100	01	68	0904
18414	20D7	9E	*	Instruction	20348	0100	01	68	0904
18415	20D8	E7	*	Instruction	20348	0100	01	68	0904
18416	20D9	04	*	Instruction	20348	0100	01	68	0904
18417	0908	01	-	Data write	20348	0100	01	68	0904
18418	20DA	4A	DECA	Instruction Start	20352	0100	01	68	0904
18419	20DA	4A	*	Instruction	20352	0100	01	68	0904
18420	20DB	26	BNE *+5	Instruction Start	20353	0100	00	6A	0904
18421	20DB	26	*	Instruction	20353	0100	00	6A	0904
18422	20DC	03	*	Instruction	20353	0100	00	6A	0904
18423	20DD	51	CBEQX #0x00,*+27	Instruction Start	20356	0100	00	6A	0904
18424	20DD	51	*	Instruction	20356	0100	00	6A	0904
18425	20DE	00	*	Instruction	20356	0100	00	6A	0904
18426	20DF	18	*	Instruction	20356	0100	00	6A	0904
18427	20DE	00	-	Data read	20356	0100	00	6A	0904
18428	20F8	A7	AIS #6	Instruction Start	20360	0100	00	6A	0904
18429	20F8	A7	*	Instruction	20360	0100	00	6A	0904
18430	20F9	06	*	Instruction	20360	0100	00	6A	0904
18431	20FA	81	RTS	Instruction Start	20362	0100	00	6A	090A
18432	20FA	81	*	Instruction	20362	0100	00	6A	090A
18433	090B	21	-	Data read	20362	0100	00	6A	090B
18434	090C	01	-	Data read	20362	0100	00	6A	090C
18435	2101	CC	JMP 0x26BC	Instruction Start	20368	0100	00	6A	090C
18436	2101	CC	*	Instruction	20368	0100	00	6A	090C
18437	2102	26	*	Instruction	20368	0100	00	6A	090C
18438	2103	BC	*	Instruction	20368	0100	00	6A	090C

Figura 6-15: Herramienta del IDE CodeWarrior para ver el código compilador

6.3.2 Evaluación de la necesidad de memoria de datos

A continuación se mostrará la cantidad de memoria de datos referida a cada módulo, veremos que las variables que mayor porcentaje ocupan son los buffers que almacenan datos de imágenes y sensores, éstos en general ocupan bloques de 512 bytes de memoria RAM.

Tabla 6-5: Memoria de datos de CCMI referente al módulo SIMA

SIMA			
	Tamaño (bytes)	**Cantidad**	**Total (bytes)**
Paquete para recibir datos de imagen	1	512	512
Número de bloques por fotos	1	100	100
Dirección en memoria de cada foto	2	200	200
Otros tipo CHAR	1	4	4
Otros tipo UINT	2	10	20
		TOTAL	**836**

Tabla 6-6: Memoria de datos de CCMI referente al módulo PCT

PCT			
	Tamaño (bytes)	**Cantidad**	**Total (bytes)**
datos tipo CHAR	1	1	1
Número de bloques por fotos	2	2	4
		TOTAL	**5**

Tabla 6-7: Memoria de datos de CCMI referente al módulo SDCA

SDCA			
	Tamaño (bytes)	**Cantidad**	**Total (bytes)**
datos tipo CHAR	1	1	1
Número de bloques por fotos	2	2	4
		TOTAL	**5**

Tabla 6-8: Memoria de datos de CCMI referente al módulo SICOM

SICOM			
	Tamaño (bytes)	**Cantidad**	**Total (bytes)**
datos tipo CHAR	1	11	11
Número de bloques por fotos	2	6	12
		TOTAL	**23**

Tabla 6-9: memoria de datos del fichero CCMI

CCMI			
	Tamaño (bytes)	**Cantidad**	**Total (bytes)**
Protocolo I2C - Recepción	1	512	512
Protocolo I2C - Transmisión	1	30	30
Protocolo SPI - Tarjeta SD	1	20	20
		TOTAL	**542**

Entonces la cantidad total de variables en RAM =1411 bytes. Según el fabricante, el uC que es empleado en el módulo CCMI (MC9S08QE128) tiene 4KB de memoria RAM, esto es mucho mayor que la cantidad usada actualmente (1.42KB).

6.3.3 Evaluación de la necesidad de memoria en la tarjeta SD

Tal como se mencionó en capítulos anteriores, el módulo CCMI contiene dentro de sí una tarjeta de memoria SD. En esta sección mostraremos los criterios de selección de la capacidad de almacenamiento, así como la distribución de espacio para la información que cada módulo necesita almacenar. Se debe tener en cuenta que esta cantidad depende directamente de la capacidad que tiene el nanosatélite de transmitir información, es decir, de nada sirve almacenar abundante información cuando solo es posible transmitir una pequeña parte de dicha información. Es por esto que se debe tener algún criterio para seleccionar la cantidad necesaria requerida por cada módulo. En el caso de aquellos módulos que muestrean datos de sensores también influye la resolución de sus muestreos, ya que mientras ésta sea menor, mayor podrá ser la cantidad de información transmitida.

Debe aclararse que el módulo SICOM no necesita almacenar información en la memoria SD ya que su función es pedir a CCMI dicha información para luego transmitirla a la estación terrena.

a) Memoria requerida por el módulo SIMA

Cada fotografía, de dimensiones 640x480 pixeles en formato JPEG, se agrupa en bloques de 512 bytes, cuya cantidad de bytes totales no es conocida debido a que el formato mencionado comprime la imagen dependiendo de sus características, cuando la foto contiene más información la compresión será de menor calidad, por lo tanto el tamaño de la foto será mayor, caso contrario la compresión funcionará mejor y el tamaño de la foto será menor. Debido a experiencias de nanosatélites anteriores, las fotos tomadas presentan baja calidad, incluso parecen manchas luminosas con fondo oscuro [43], tal como se muestran en la Figura 6-16, por lo tanto para determinar un tamaño de imagen máximo, tomaremos una fotografía a un escenario bastante recargado de imágenes para que la compresión sea de mala calidad, de esta experiencia obtenemos 30 bloques de 512 bytes equivalentes en total a 15 kBytes, también se revisó el historial que se tiene de todas las fotos que se han llegado a tomar con dicha cámara y se observa que el máximo número de bloques alcanzados es también de 15 kBytes, y se presenta solo bajo las condiciones mencionadas, para la determinación de la capacidad requerida por el módulo SIMA, por seguridad elegiremos un máximo de 40 bloques para cada foto el cual equivaldría a una foto de 20 kBytes.

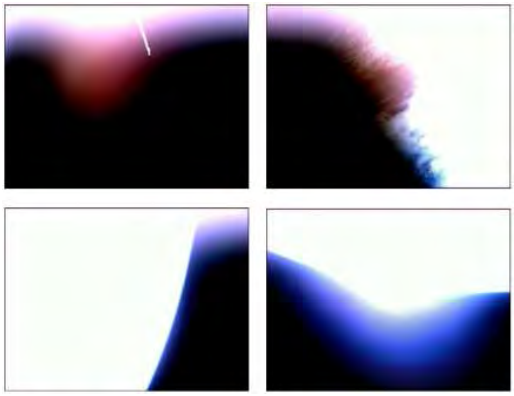


Figura 6-16: Imágenes capturadas por el nanosatélite Compass-1 [43]

Figura 6-17: Imagen tomada por la cámara del nanosatélite Chasqui en formato JPEG [21].

Memoria requerida: De acuerdo a los estudios realizados por los módulos SORAT, SICOM y ESTER, en el mejor de los casos, el nanosatélite tendrá la posibilidad de descargar una sola foto por día. Además teniendo en cuenta que una vez que se descarga una foto ya no hay necesidad de que ese espacio de memoria siga siendo ocupada en el uC, podemos simplemente tener una foto siempre disponible en memoria, la cual se actualizará solo cuando la anterior sea transmitida a la estación terrena, pero debemos recordar que se tienen dos cámaras, una visible y otra infrarroja, por lo que es útil para el módulo de cámaras tener las dos fotos tomadas en un mismo momento para luego poder contrastar los resultados de ambas, para ello se piensa que en una oportunidad de enlace pueda descargarse la primera foto, y en la siguiente

pueda descargarse la otra, entonces debemos tener disponible la memoria necesaria para dos fotos como mínimo. El problema de utilizar la cantidad mínima de dos fotos disponibles está en que si por algún motivo las cámaras fallan, entonces no se tendrá información que transmitir, por lo que es conveniente tener siempre cierta cantidad de fotos disponibles en memoria, por acuerdo de todo el equipo de trabajo se llegó a la conclusión que se necesitan como mínimo 20 fotos siempre disponibles en memoria, esto porque se calcula que podrían transmitirse en un mes aproximadamente, en cuyo caso la misión hace mucho que se consideraría haber sido exitosa, además la memoria debe almacenar de una imagen pregrabada en la memoria, esto para el caso en el que la cámara nunca funciona y podamos al menos hacer prueba de transmisión de imagen. La posibilidad de que las cámaras fallen es debido a que es un sistema embebido el cual adquirimos sin conocer las características de sus componentes internos, pero de todas formas éstas han sido probadas en Tierra para que soporten en cierto grado las condiciones espaciales.

De esto concluimos que la cantidad de memoria necesaria para el módulo SIMA es de 20 fotos de 20 kBytes, haciendo un total de:

$$size_{SIMA} = ((20 + 1)\, fotos)(20 KBytes)$$

$$size_{SIMA} = 420\, KBytes$$

b) Memoria requerida por el módulo PCT

De acuerdo a los algoritmos del módulo PCT se llena un buffer de 512 bytes cada media hora, y requieren información de tres órbitas para poder realizar sus estudios en Tierra, teniendo en cuenta que cada órbita demora 90 min, entonces en este tiempo se deben almacenar 6 bloques de 512 bytes, entonces un paquete útil contiene un total de:

$$data_{PCT} = (3\ órbitas)(6bloques)(512Bytes)$$

$$data_{PCT} = 9KBytes$$

Según los resultados obtenidos en la toma de fotos, esta cantidad ocupa lo mismo que una fotografía promedio, por lo tanto también podrá ser descargada solo un archivo de datos de PCT por día, correspondientes a tres órbitas. Siguiente el mismo criterio tomado en CCMI, también se tendrá 20 conjuntos de estos datos útiles los cuales se irán actualizando cada vez que los mismos se llenen, la actualización será de forma cíclica, asegurando que siempre la información más antigua sea remplazada por la más reciente.

$$size_{PCT} = (20\ paquetes)(9\ KBytes)$$

$$size_{PCT} = 180\ KBytes$$

c) Memoria requerida por el módulo SDCA

Según los requerimientos del módulo SDCA, se necesita un bloque de 512 bytes en memoria de CCMI para guardar la información correspondiente

a dato TLE enviada desde ESTER, además necesitan 10 bloques de 512 para almacenar datos de sus sensores. Todo esto hace un total de:

$$size_{SDCA} = 5\ KBytes$$

Con esto determinamos que la memoria tipo Flash mínima necesaria es de 605 kBytes. La Tabla 2-1 muestra el resumen de los valores de memoria calculados.

$$size_{\text{minimo}\quad SD} = size_{SIMA} + size_{PCT} + size_{SDCA}$$

$$size_{minimo\ SD} = 605KBytes$$

Actualmente se encuentran disponibles en el mercado memorias SD, valores comerciales son 512 kBytes, 1GB, 2GB, 8GB y 16GB, que superan por mucho la capacidad mínima requerida, la desventaja de usar una memoria SD de mayor capacidad es el consumo eléctrico, por ello se eligió el valor mínimo más próximo a 605 kBytes, es decir 1GB.

$$\boldsymbol{size_{SD} = 1GB}$$

Tabla 6-10: Memoria requerida por cada módulo en tarjeta SD

Módulo	Memoria requerida (KBytes)
SIMA	420
PCT	180
SDCA	5
Total	**605**

6.4 Diseño de los protocolos de comunicación

El uC del módulo CCMI se comunica con otros componentes internos y externos a dicho módulo, entre los componentes internos tenemos principalmente a las memorias I2C, memoria SD y RTC, el protocolo de comunicación con estos dispositivos están especificados en la hoja del fabricante de cada uno, solo hace falta que CCMI implemente en su software el algoritmo establecido para cada dispositivo.

Externamente debe comunicarse con los uC de los demás módulos del nanosatélite Chasqui, con estos si es necesario diseñar e implementar un protocolo de comunicación propio.

6.4.1 Protocolos de comunicación internos

La tarjeta de CCMI contiene dentro de si las cámaras y la SD-Card, cuyo protocolo de comunicación ya está establecido por los fabricantes de estos productos, a continuación mostraremos las características principales de estos protocolos que han sido implementados en CCMI según la documentación de los fabricantes de la cámara y la SD-Card.

a) Interfaz con cámara C328R

La cámara C328R se comunica con el MCU de CCMI a través del protocolo UART de 3.3V, la cámara está configurada por defecto a 14400bps, pero existe la opción de configurar a otras velocidades 7200, 9600 y 19200 baudios.

La Figura 6-18 muestra el diagrama de bloques representa la composición interna de la cámara:

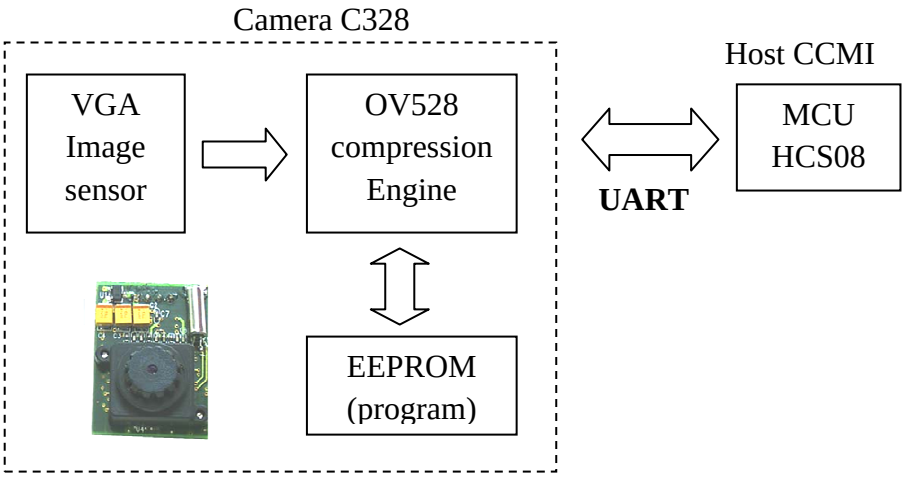


Figura 6-18:Interfaz de comunicación con la cámara.

A continuación se resume los comandos de control de la cámara:

Tabla 6-11: Resumen de comandos para el control de la cámara.

1	Initial	To configure the image size, color type
2	Get Picture	Get Picture type
3	Snapshot	Set snap shot image type
4	Set PackageSize	Set the package size to transmit data from module to Host
5	Set Baudrate	Changethebaudrate
6	Reset	Reset the whole system or reset the state machine
7	Power Off	Toentersleepmode
8	Data	Set the data type and length for transmitting data to host
9	SYNC	Sync signal to connect between host and module
10	ACK	Command to indicate the communication success
11	NAK	Command to indicate the communication fail with error code

b) Interfaz con tarjeta SD

Para controlar la tarjeta de memoria SD se hace a través del protocolo SPI, los comandos para el control de dicha tarjeta pertenecen al estándar SD, los cuales nos permiten formatear la tarjeta, escribir y leer en memoria y además generar y configurar archivos, ver Figura 6-19.

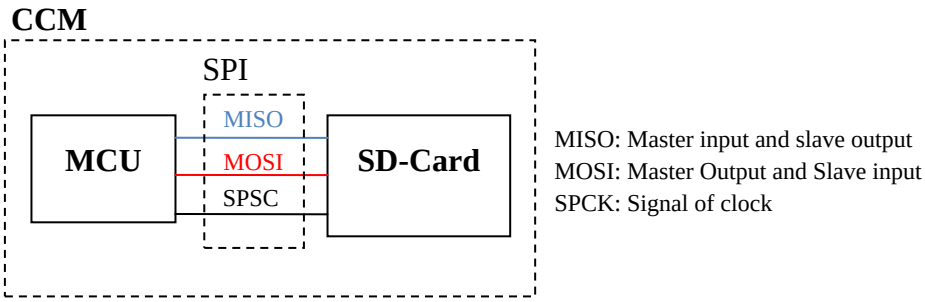


Figura 6-19:Interfaz de comunicación I2C con tarjeta SD.

6.4.2 Diseño de los protocolos de comunicación con módulos externos

En este punto se describen los protocolos de comunicación establecidos entre los módulos que se conectan físicamente a la tarjeta de CCMI, ver Figura 6-20, en cada uno de estos se especifica el protocolo eléctrico empleado, por ejemplo para el caso del módulo SICOM se emplea el protocolo UART a una velocidad de 9600 baudios.

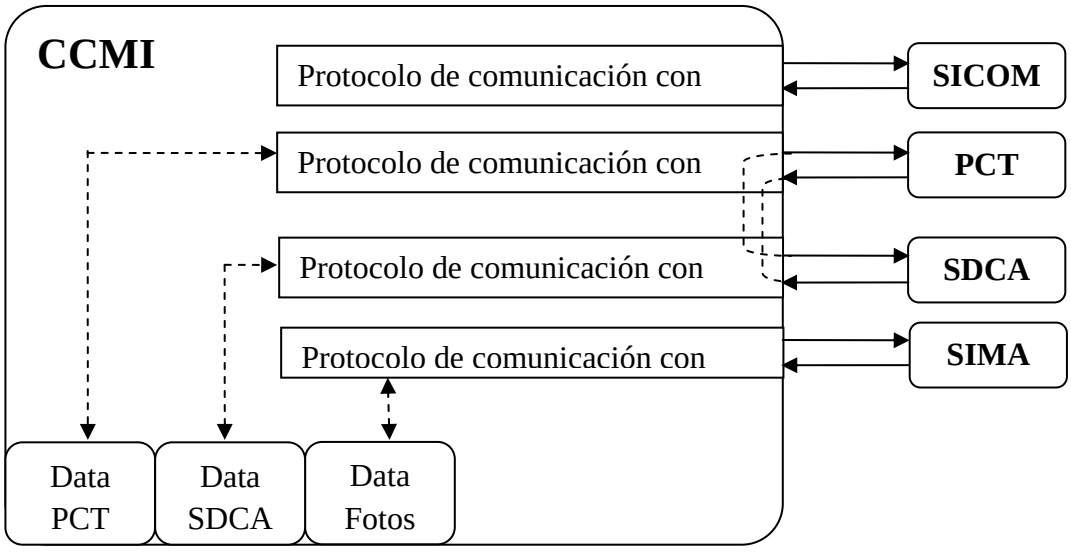


Figura 6-20: Diagrama de conexión de los flujos de datos internos y externos a CCMI

a) Interfaz con SICOM

El protocolo elegido para la comunicación entre CCMI y SICOM fue UART a 9600bps, ver Figura 6-21, esta interfaz debe ser para comunicación exclusiva con

CCMI, es por ello que no se usará el bus I2C que es compartido por los módulos CCMI, SDCA y PCT.

El siguiente diagrama muestra las características de la comunicación entre CCMI y SICOM.

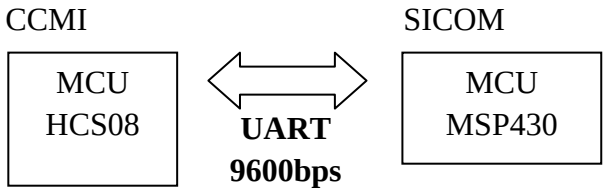


Figura 6-21: Interfaz de comunicación con el módulo SICOM

La Tabla 6-12resume los comandos utilizados para la comunicación con el módulo SICOM, esto consiste principalmente en transmitir los datos almacenados por los otros módulos en la memoria SD de CCMI, además SICOM requiere algunos datos de CCMI para transmitir la señal de beacon.

Protocolo de comunicación entre los módulos CCMI y SICOM

Tabla 6-12: Éspecificación de los comandos UART entre los módulos CCMI y SICOM

Command	INICIO	IDNumber	Parameter1	Parameter2	Parameter3	Parameter4	FIN
Cantidad de archivos	'#'	0x31	-	-	-	-	'*'
Canitdad de Bloques	'#'	0x32	Tipo	1-Num de Archivo	2-Num Archivo	-	'*'
Obtener Bloque	'#'	0x33	Tipo	1-Num de Archivo	2-Num de Archivo	Num de Bloque	'*'
Reset CCMI	'#'	0x34	-	-	-	-	'*'
Actualizar Fecha	'#'	0x35	Minuto	Hora	día	mes	'*'
Programar a CCMI para captura de datos	'#'	0x36	Tipo	Periodo[1]	Minuto	Hora	'*'
ACK	'#'	0x0E	Command	-	-	-	'*'
NAK	'#'	0x0F	Command	-	-	-	'*'

[1] Si el parámetro es {0xFF} solo debe tomar una foto.

Obs.: El número de bytes por comando es variable, cada guión significa que no se envía dicho parámetro.

Ejemplos de comandos entre los módulos CCMI y SICOM

Tabla 6-13: Ejemplos de comandos entre módulos CCMI y SICOM.

Command	Ejemplo	Obs
Cantidad de archivos	> {'#'} {'1'} {'*'}	
	< {byte1} {byte2} {byte3} {byte4}	
Canitdad de Bloques	> {'#'} {'2'} {‘F’} {0x02} {0x00}{'*'}	Del archivo 00 02
	< {byte1}	
Obtener Bloque	> {'#'} {'3'} {‘F’} {0x01} {0x00} {0x03}{'*'}	Bloque 3 del archivo 00 01
	< {byte1} {byte2} {byte3} {byte4} … {byte512}	
Reset CCMI	> {'#'} {'4'} {'*'}	Si CCMI no responde ACK, SICOM toma el control
	< {'#'} {0x0E} {'*'}	
Actualizar Fecha	> {'#'} {'5'} {30} {12}	Actualiza a 12:30
	< {'#'} {0x0E} {'*'}	
Programar a CCMI para captura de datos	> {'#'} {'6'} {'F'} {05} {20} {15}	Se tomarán fotografías cada 5 min a partir de las 15:20
	< {'#'} {0x0E} {'*'}	
ACK		Indica que la comunicación es correcta
	< {'#'} {0x0E} {'*'}	
NAK		Indica que la comunicación a fallado
	< {'#'} {0x0F} {'*'}	

Descripción de Comandos

Cantidad de Archivos (‘1’)

SICOM envía este comando a CCMI para saber cuántos archivos disponibles existen de fotos (2 bytes) y de datos de sensores (2 bytes). Enviándose un total de 4 bytes de respuesta, siempre se envía primero el byte menos significativo. Si se tienen 350 (0x015E) archivos de fotos y 950 (0x03B6) archivos de sensores, entonces:

SICOM > {'#'} {'1'} {'*'}

CCMI < {0x5E} {0x01} {B6} {03}

Cantidad de Bloques (‘2’)

Este comando permite saber cuántos bloques de memoria ocupa un archivo específico, este dato será útil a SICOM para saber cuántos bloques debe solicitar a CCMI usando el comando 3. Si la foto 125(0x007D) ocupa 14(0x0E) bloques de memoria, entonces:

SICOM > {'#'} {'2'} {‘F’} {0x7D} {0x00} {'*'}

CCMI < {0x0E}

Obtener Bloque (‘3’)

Solicita a SICOM la transmisión de un paquete de 512bytes a SICOM, debe especificarse en número de bloque requerido, además del número de archivo y el tipo (foto o sensores). Si se quiere el bloque 5 (0x05) de la foto 125 (0x007D), entonces CCMI debe responder con un paquete de 512 bytes, tal como se muestra a continuación:

SICOM > {'#'} {'3'} {‘F’} {0x7D} {0x00} {0x05} {'*'}

CCMI < {0x05} {0x00} {0xFA} {0x01} {0x14}… {0x00}

Reset CCMI (‘4’)

Comando de reset a CCMI, debe responder un ACK con su identificador de comando (0x04).

SICOM > {'#'} **{'4'}** {'*'}

CCMI < {'#'} {0x0E} **{'4'}** {'*'}

Actualizar Fecha ('5')

Permite setear la hora del reloj interno del módulo CCMI, El rango de seteo de hora será de 0h a 24h. No hay necesidad de indicar AM o PM.

Además la fecha indicando el día y mes respectivamente.

El siguiente ejemplo setea CCMI a las 12:30 horas del día 2 del mes de marzo.

> {'#'} **{'5'}** {30} {12} {02} {03}

< {'#'} {0x0E} **{'5'}** {'*'}

Programar a CCMI para capturar datos ('6')

Con este comando podemos programar a CCMI para tomar captura de datos de imágenes o sensores en un determinado momento, para ello se debe actualizar primero la fecha y hora.Por ejemplo si se desea tomar fotografías cada 5min a partir de las 15:20 se deberá enviar lo siguiente:

> {'#'} **{'6'}** {'F'} {05} {20} {15}

< {'#'} {0x0E} **{'6'}** {'*'}

Obs.: Para tomar solo una fotografía, entonces el parámetro que indica el periodo será 0xFF (en el ejemplo sería:

> {'#'} {'6'} {'F'} **{0xFF}** {20} {15})

ACK

Este comando solo lo envía CCMI, e indica a SICOM que el comando enviado se ejecutó correctamente, siempre se indica además el identificador del comando correspondiente.

< {'#'} {0x0E} **{'6'}** {'*'}

NAK

Este comando solo lo envía CCMI, e indica a SICOM que el comando enviado no se ejecutó correctamente, siempre se indica además el identificador del comando correspondiente.

< {'#'} {0x0F} **{'6'}** {'*'}

b) Interfaz con PCT

El protocolo de comunicación en este caso se basa en el bus I2C, este bus consiste de una línea de CLOCK generada por un MCU denominado Master, y otra para transmisión de DATA. En nuestro caso el MCU Master corresponde a CCMI y los MCU Slaves son PCT y SDCA, ver Figura 6-22.

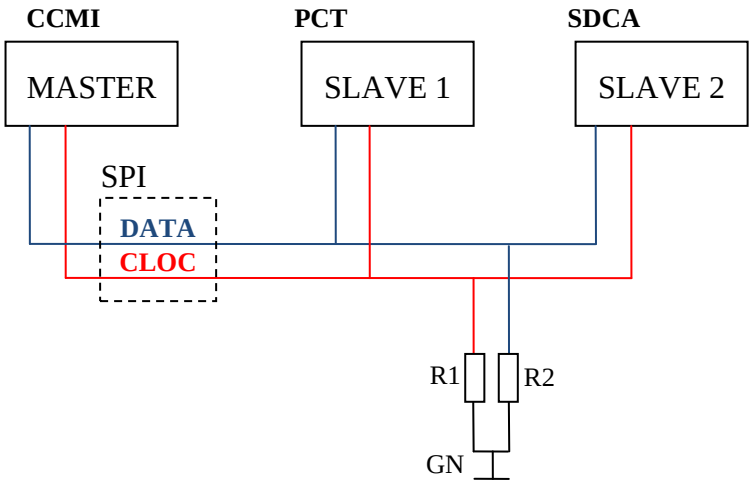


Figura 6-22:Interfaz de comunicación I2C entre CCMI con PCT y SDCA

La comunicación entre CCMI y PCT no es compleja, CCMI debe esperar que PCT llene su buffer de datos de 512 bytes, esto sucede aproximadamente cada 30 minutos, PCT debe enviar un flanco de subida por el pin BUFFER_PCT [diagrama de pines] y cuando CCMI detecte esto hará una lectura por I2C de los 512 bytes de datos, este algoritmo se muestra en el diagrama de flujo de la Figura 6-23.

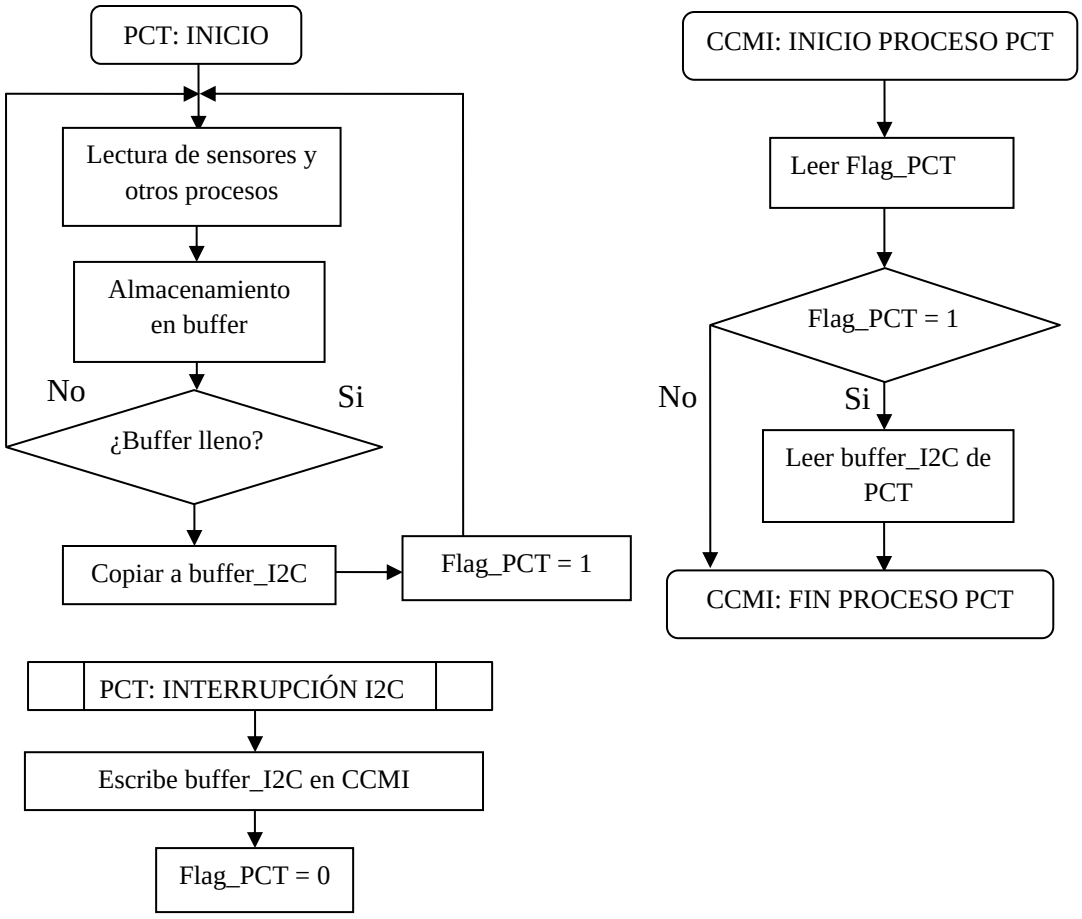


Figura 6-23: Diagrama de flujo de CCMI para atender al módulo PCT.

c) Interfaz con SDCA

La interfaz con el módulo SDCA es a través del bus I2C, tal como se muestra en la Figura 6-22, para este caso el protocolo es más complejo que el de PCT, pues no solo consiste en hacer la lectura periódica de un buffer de datos de sensores

de 512 bytes, sino que además existen ciertas órdenes a SDCA para gestionar acciones como transmisión de dato de TLE y consultar estado de orientación y estabilidad antes de tomar una fotografía.

La Figura 6-24 muestra el diagrama de flujo del protocolo de comunicación entre los módulos CCMI y PCT. Podemos notar que este proceso lo primero que hace es verificar el funcionamiento de SDCA, para esto simplemente se espera que responda a un estímulo de CCMI, caso contrario asumirá que falla y se dará la orden a PCT para reiniciar a SDCA, esto es importante ya que dicho módulo consume alta carga de las baterías y además existe alto riesgo de falla porque sus pruebas de funcionamiento no pueden ser realizadas en Tierra. Luego de esto, y dependiendo del estado de CCMI y el modo de operación del nanosatélite, se pueden enviar tres tipos de comando a SDCA, por ahora lo único que puede ser probado es la transmisión de datos, ya que para un test real de los demás comandos se debe poder simular un ambiente de ingravidez para que SDCA pueda orientarse o estabilizarse, actualmente no se cuenta con dicho ambiente, estas partes del software solo se muestran como plantilla en la implementación en código C.

Protocolo de comunicación entre los módulos CCMI y SDCA

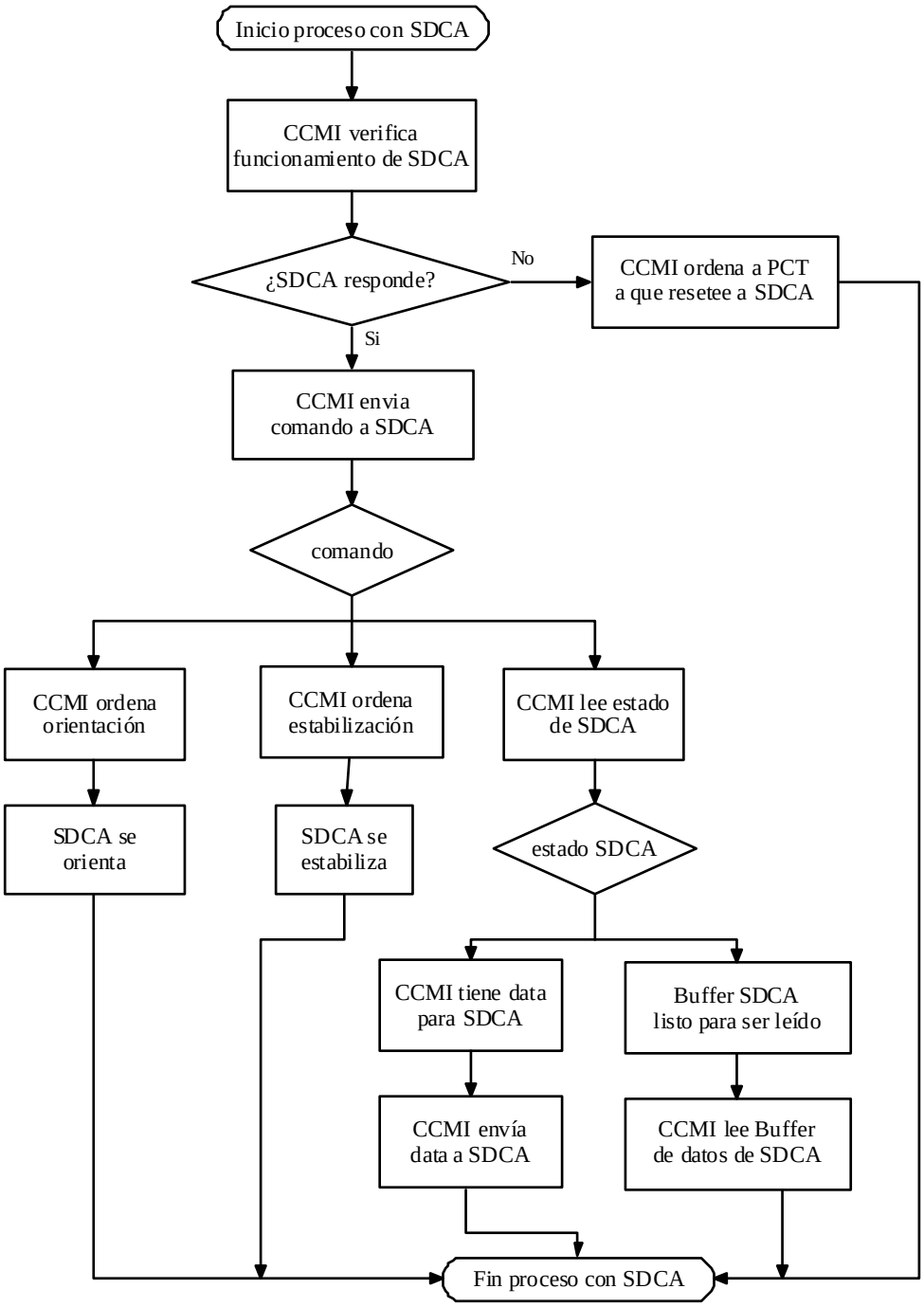


Figura 6-24: Diagrama de flujo del proceso de CCMI para atender al módulo SDCA.

No es necesario que SDCA implemente la función de enviar datos a CCMI, debido a que esa acción se realiza de manera automática usando interrupciones de SDCA, para ello se debe tener listo el buffer de recepción I2C [22].

6.5 Diseño y desarrollo de actividades

Para el diseño y desarrollo de actividades para el desarrollo del software, debemos empezar por el diseño general de la estructura del software, la organización de la memoria y el diseño del bus común, luego de terminar estos tres se deben diseñar los diagramas de flujos para luego desarrollar el código fuente basado en dichos diagramas, finalmente se programan los protocolos con los demás módulos y se depura el código, esto se resume en la Figura 6-25.

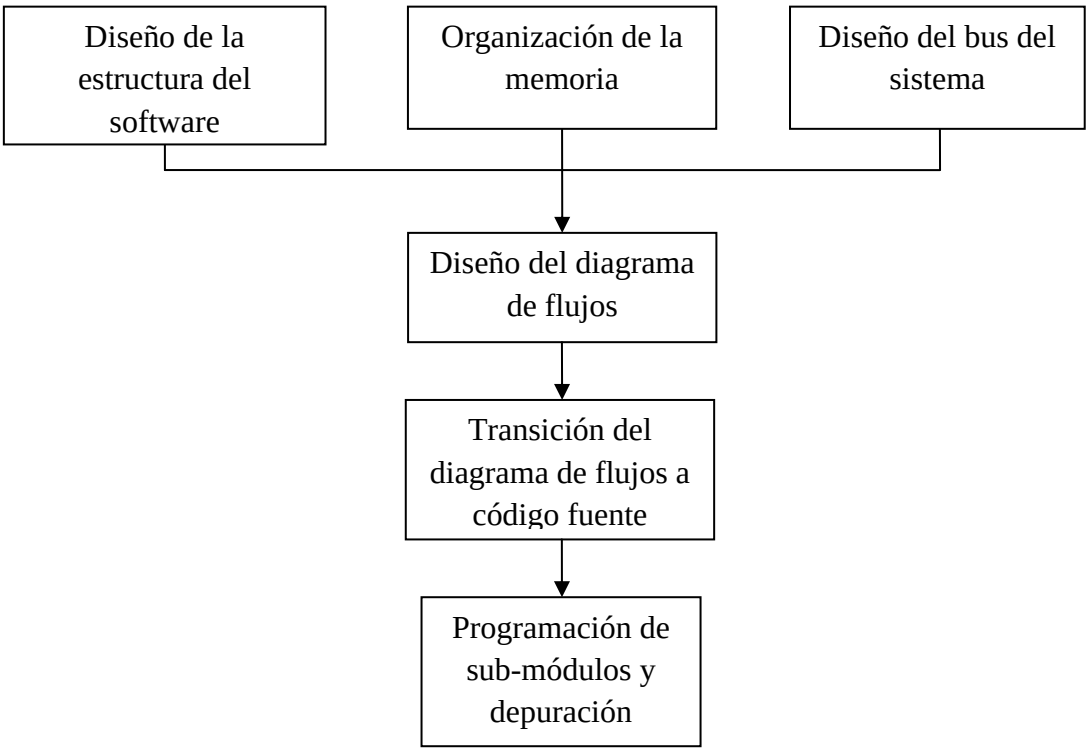


Figura 6-25: Diseño y desarrollo de actividades [42]

6.5.1 Estructura general del software

La estructura del software está compuesta de tres partes o niveles importantes, la del nivel más bajo trata del control de los periféricos internos del uC, es decir se comporta como el driver del hardware de nuestro uC, por ejemplo el SPI, I2C, SCI, Timers, etc. Cada uno de estos periféricos debe ser configurados tan como se indica en

la hoja técnica del uC, como resultado de esto se debe tener ficheros que se encarguen del control de cada periférico, los cuales una vez que estén bien depurados, deben ser una caja negra en la que solo nos interesará saber cómo funcionan las funciones que se han implementado en dicho fichero, es decir se debe orientar la programación de este nivel del software con funciones que no necesitan ser modificadas internamente para ser utilizadas. Este paso es necesario cuando recién se comienza a usar un uC, ya que la implementación de las funciones de control depende directamente de la arquitectura del uC, la idea es generar este nivel de modo que sea útil para múltiples proyectos ya que su funcionalidad es general para múltiples aplicaciones.

La segunda capa es denominada de interfaz de los dispositivos, hace uso de la capa de bajo nivel para implementar funciones que controlen dispositivos presentes en nuestra tarjeta, por ejemplo, para crear ficheros que permitan tomar una fotografía usando la cámara digital, debemos hacer uso de ficheros de bajo nivel ya que es necesario el control de los periféricos SCI y SPI, luego nuestro este fichero también será una especie de caja negra del cual solo nos interesa conocer las funciones que implementan.

La tercera capa es la denominada capa de aplicación, que al igual que los anteriores está codificado en lenguaje C, y es aquí donde implementamos nuestros diagramas de flujo haciendo uso de las dos capas inferiores. La Figura 6-26 muestra estas tres capas.

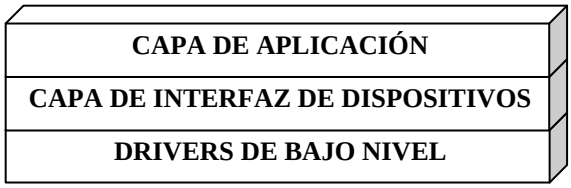


Figura 6-26: Estructura general del software [43]

6.6 Programación del uC MC9S08QE128

Para la programación del uC primeramente se crearon las librerías base de la capa de drivers de bajo nivel, ver Subsección 6.5.1, que controlan los periféricos internos al MCU como son: *SPI, SCI, I2C* y *Timer*, con estas se crearon otras librerías de la capa de interfaz de dispositivos que controlan periféricos externos al uC pero internos a la tarjeta CCMI, para almacenamiento y lectura de la tarjeta SD y toma de fotografía de cámara C328R. Finalmente se crean las librerías de la capa de aplicación que permiten implementar los protocolos de comunicación con las tarjetas externas a CCMI, los cuales se comunican con éste mediante el bus PC104.

6.6.1 Codificación del RTOS

Esta sección explica la codificación del RTOS para el uC del módulo CCMI, esta codificación se hará en el lenguaje ANSI C y se implementará en el uC MC9S08QE128 de *Freescale*. Usaremos el *IDE-CodeWarrior* para la compilación del código, esta IDE también permite la emulación paso a paso del software del CCMI.

Nuestro RTOS contiene 5 tareas, cada una corresponde a un módulo del Chasqui, el que tiene el control del RTOS, incluyendo el *Kernel* es la tarea que corresponde a CCMI, quién además se encargará de la administración de la memoria interna y externa, siendo la principal la tarjeta de memoria SD como memoria no volatil.

Cada tarea creada debe ser inicializada una única vez luego del reset del sistema, esta función incluye dentro de si las inicializaciones de los periféricos que empleará dicho módulo.

Cada tarea inicializada habilitará su ejecución en el bucle infinito del programa principal, esta ejecución deberá comportarse como una máquina de estados tal como se muestra en la Figura 6-27 y más detalladamente en el Anexo A2. El RTOS se codifica

en lenguaje C. Haciendo uso del comando *switch-case* para el control de a máquina de estados, obtenemos el código que se muestra en la Figura 6-28 y más detallado en el Anexo A3, donde se muestra de ejemplo la implementación de la plantilla general RTOS para el caso de la tarea de CCMI referente al módulo PCT. El *switch-case* controla la posición del contador del programa cuando una tarea sea ejecutada.

En el Anexo 0 se muestra todo el árbol de ficheros que conforman el proyecto en C implementado en IDE de CodeWarrior.

```
/*********************************************************
Proyecto: PICOSATÉLITE DE INVESTIGACIÓN CHASQUI
Módulo  : Control Central y Manejo de Información - CCMI
Tesista : Elvis Omar Jara Alegria
Fecha   : 2 Agosto 2010
*********************************************************/

#include "includes.h"

void main(void) {
  CCMI_Init();
  SIMA_Init();
  SICOM_Init();
  SDCA_Init();
  PCT_Init();
  EnableInterrupts;
  for(;;) {
    CCMI_Run();
    SICOM_Run();
    SDCA_Run();
    SIMA_Run();
    PCT_Run();
  }
}
```

Figura 6-27: Plantilla general de RTOS

Debe notarse que el estado principal de este código es denominado *Main_SDCA()*; debido a que es la parte del código que se asumirá como principal para dicha tarea.

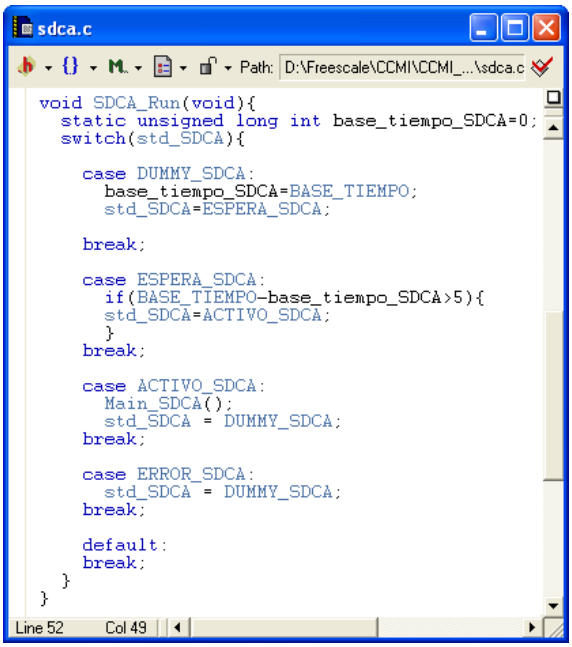


Figura 6-28: Plantilla general de RTOS

6.6.2 Análisis de tiempos del RTOS

En esta sección determinaremos unos parámetros muy importantes en todo RTOS. Con dichos parámetros se calibra el tiempo de ejecución de cada tarea. Una mala determinación de dicho tiempo puede causar que nuestro RTOS no sea *planificable*, es decir que varias tareas quieran ejecutarse al mismo tiempo, generándose una cola de procesos que complicaría nuestro sistema. Si bien un OS debe ser capaz de distribuir el tiempo de ejecución entre todas las tareas aunque todas deban ejecutarse al mismo tiempo, en un RTOS existen procesos cuyo tiempo de ejecución es crítico y es preferible evitar dicho caso y en su lugar se dará todo el CPU para dichas tareas consideradas críticas. Para nuestro análisis debemos saber el tiempo de ejecución de las tareas que consumen un tiempo considerable a nuestro sistema, estos tiempos se han determinado experimentalmente en la Subsección 8.1.1 y los valores de las mediciones se resumen en la Tabla 8-1.

A continuación en la Figura 6-29 se muestra el diagrama de tiempos usado para determinar los periodos de ejecución de cada tarea correspondiente a cada módulo, para ello se tomó como referencia la función que consume el máximo tiempo en cada tarea, el periodo de cada tarea debe ser mayor que dicho tiempo máximo. Los datos correspondientes a los periodos corresponden a los requerimientos de todos los módulos internos y su valor se representa como P_CCMI, P_PCT, P_SDCA, P_SIMA y P_SICOM. También se analizó el caso extremo en el que en un periodo del RTOS se ejecuten todas las tareas de forma continua, mientras mayor sea el periodo del RTOS menor será la probabilidad de que se forme una cola de funciones listas para ser ejecutadas, por ello el periodo P_RTOS debe ser mayor a 31 segundos. El software del programa permite configurar este parámetro y también los periodos correspondientes a cada tarea, no es objetivo de esta tesis la determinación de los periodos óptimos de tiempo, por ello se eligió un valor de periodo de un minuto, tiempo muy por encima del mínimo.

La tarea crítica del sistema es la correspondiente a SICOM, por ello la comunicación con la estación terrena no debe nunca ser interrumpida por otras tareas, para evitar esto su implementación se hizo dentro de una interrupción que deshabilita todas las demás interrupciones. Si bien la transmisión de una foto a ESTER demora 5 minutos, tiempo mayor al P_RTOS, esto se ejecutará a lo más una vez por día, por ello las posibles alteraciones causadas por la tarea SICOM no serán tomadas en cuenta ya que solo retrasará la ejecución del *scheduler* del RTOS más no las demás tareas ya que su periodo de ejecución es mucho mayor a 5 minutos.

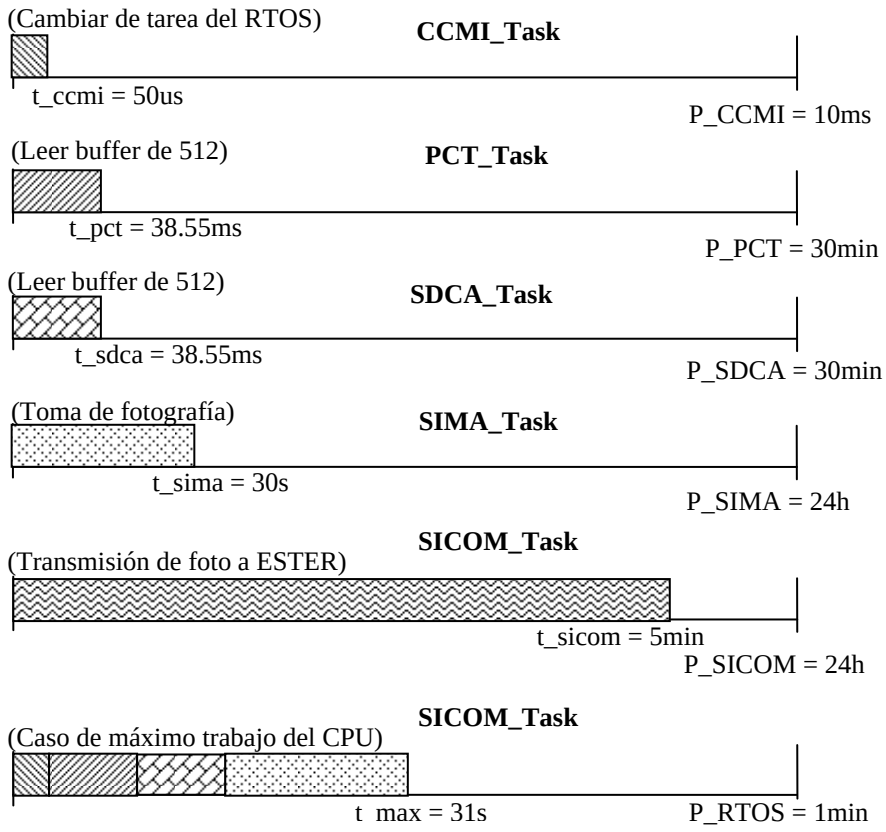


Figura 6-29: Análisis de tiempos del RTOS

6.7 Implementación del software

Este proceso consiste en digitar las instrucciones en lenguaje C basados en los diagramas de flujo del sistema. En este punto se mostrará la implementación base referente a los drivers de bajo nivel y capa de interfaz de dispositivos, ver 6.5.1, como son el almacenamiento de información en la tarjeta SD, toma de fotografía y almacenamiento en memoria, además se muestran las funciones en C encargadas de dichas tareas y los resultados de las pruebas, finalmente se muestran las funciones base para la implementación de los protocolos de comunicación con los módulos PCT y SDCA. La capa de aplicación, ver Subsección 6.5.1, está implementado en un modelo básico de RTOS para sistemas embebidos. Las características de esta capa se muestran en la Subsección 6.6.1, la plantilla codificada en lenguaje C del RTOS se muestra en la Figura 6-28.

6.7.1 Almacenamiento de información en tarjeta SD

La Figura 6-30 muestra que se logró escribir el mensaje: “HOLA CHASQUI UNI-CTIC” en la dirección de memoria 512 de la tarjeta SD.

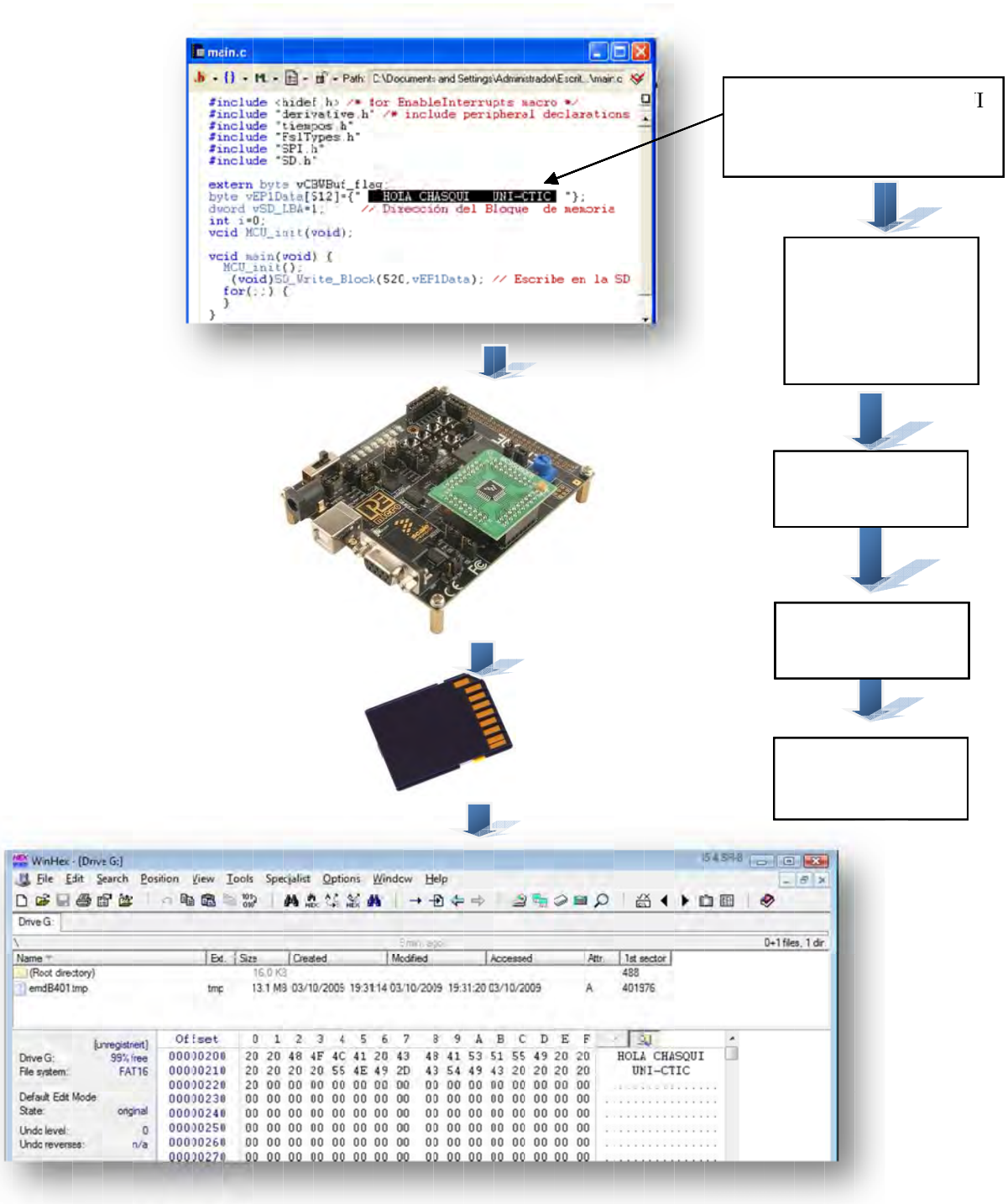


Figura 6-30: Proceso de escritura en una tarjeta de memoria SD.

6.7.2 Toma y almacenamiento de fotografía.

A continuación mostramos el proceso seguido, desde la codificación del programa hasta la lectura de la imagen por un lector de tarjeta SD de una PC.

Figura 6-31: Proceso para la toma de una fotografía, usando el DEMOQE, y su visualización desde una PC.

A continuación se muestra las funciones en lenguaje C que permiten tomar una fotografía y almacenarla en una tarjeta de memoria SD, para esto se hizo uso de los ficheros anteriormente mencionados:

SPI.H → Para comunicarnos con la tarjeta SD

SD.H → Permite leer y escribir en la tarjeta SD

SCI.H → Permite comunicarnos con la cámara tipo UART C328R

Dichas funciones se incluyen en el fichero CAM.H, que deberá ser incluido en el programa principal.

Fichero CAM.H: Permite controlar la captura de una imagen desde la cámara C328R y su almacenamiento en una memoria SD, para ello este fichero controla los periféricos SCI, para la comunicación con la cámara, y SPI para el control de la tarjeta de memoria SD.

Proceso de toma de fotografía: En el programa principal deben inicializarse los parámetros necesarios para la toma de una fotografía, primeramente se debe establecer una velocidad del bus UART de 14400 baud, ya que es el valor por defecto al que está configurada la cámara C328R.

Sci_Init(bRate14400);

Luego simplemente debe llamarse a la función *tomar_foto()*, esta se encargará de hacer todos los manejos de las rutinas encargadas del proceso de la toma y almacenamiento de imagen.

Las principales funciones incluidas son:

- **byte tomar_foto(void)→** Esta función primeramente envía los comandos de sincronización para luego enviar los comandos que permiten que la cámara C328R envíe los paquetes de la imagen que capturó según los parámetros que se definen por defecto en el fichero CAM.H, estos parámetros son:
    - Formato de imagen: JPEG
    - Tamaño de paquete: 512 Bytes
    - Tipo de Snapshot: Imagen comprimida
    - Baudrate: 14400 (valor por defecto)
    - Resolución de JPEG: 640x480

A continuación se muestra el algoritmo de la captura de la imagen desde la cámara C328R, usando un MCU *Freescale*.

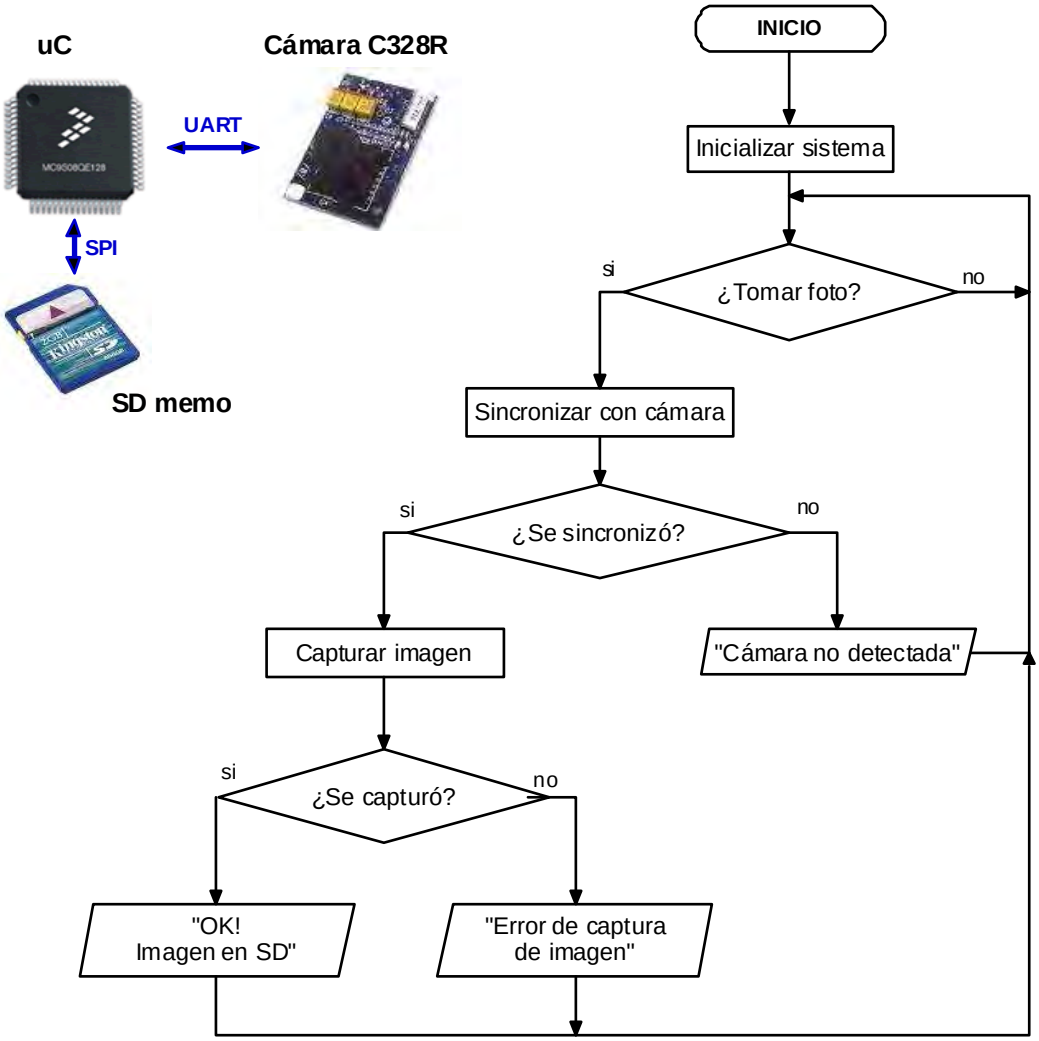


Figura 6-32: Algoritmo para toma de fotografía.

- **byte sync_C328_OK(void)→** Envía los comandos de sincronización, cuando recibe un ACK de la cámara esta función retorna un FALSE, si ocurrió algún problema de sincronización retorna TRUE.

  En la Figura 6-33 se muestra la secuencia comandos, en hexadecimal, transmitidos para la sincronización con la cámara, además de los

diagramas de flujo que representan el algoritmo que se programó para la sincronización con la cámara.

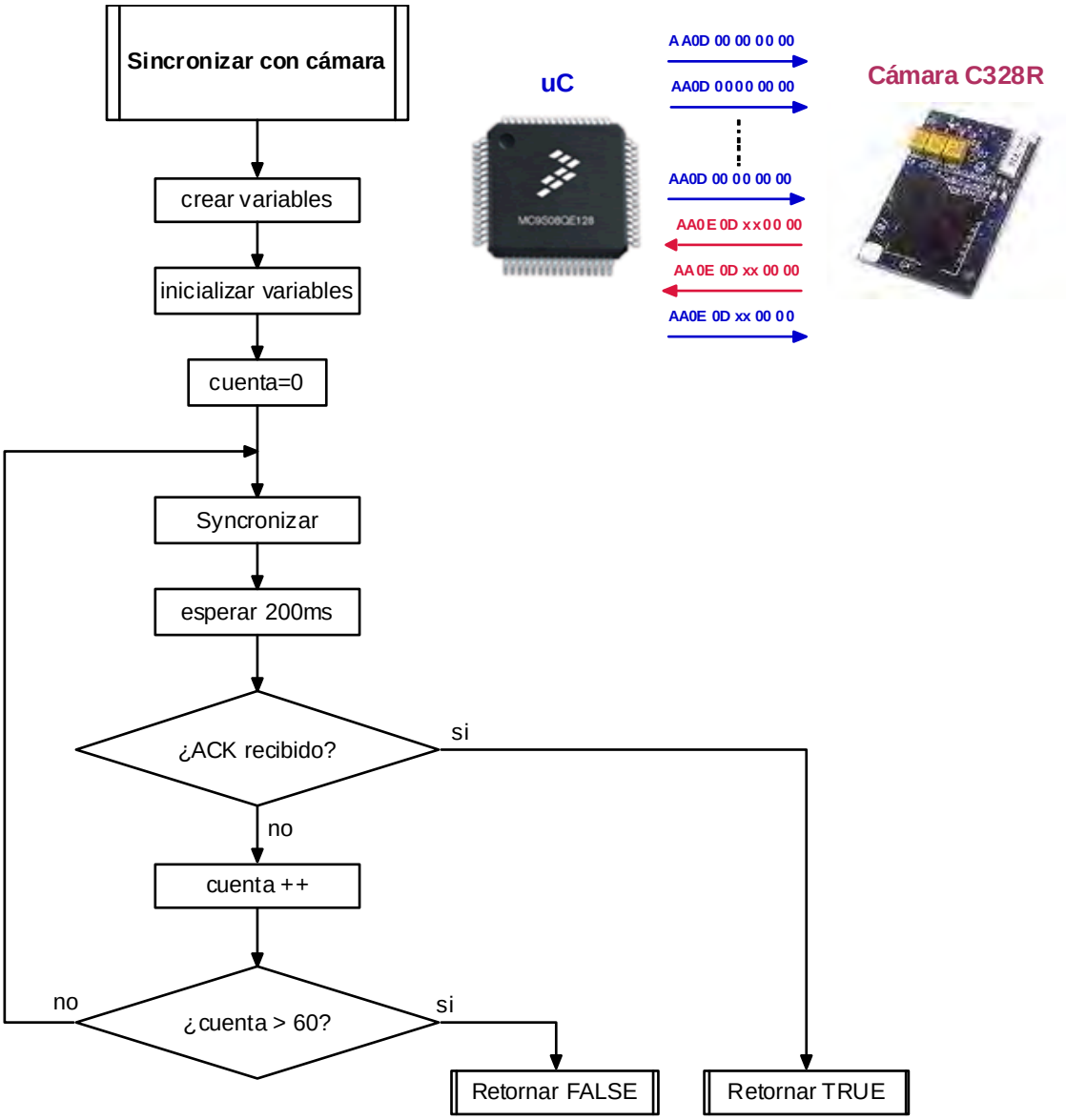


Figura 6-33: Algoritmo de la sincronización con la cámara C328R

- **byte espera_respuesta(byte cmd, byte num)** → Envía el comando '*cmd*' y espera que el buffer de recepción reciba '*num*' bytes desde la cámara.

Retorna TRUE si se completó el número exacto y FALSE si se perdió algún byte.

Cada comando representa 6 bytes enviados y dependiendo del tipo de comando pueden recibirse 6 o 12 bytes de respuesta desde la cámara C328R.

- **byte leer_paquete(void)** → Lee un byte del paquete enviado por la cámara y lo almacena en el buffer *'paquete_imagen[i]'*, retorna TRUE si el dato corresponde al último byte. Esta función es invocada desde la interrupción por recepción del módulo SCI.
- **byte capturar_data(void)** → Se encarga gestionar la recepción de paquetes de la imagen, cuando se completó toda la información, esta se almacena en la tarjeta SD, desde la dirección 520.
- **void limpiar_buffer(void)**→ Limpia el buffer de recepción de dato para poder tomar más fotografías.

6.7.3 Implementación y prueba de toma de foto y almacenamiento en memoria SD

Los siguientes resultados muestran el buen funcionamiento de los ficheros mostrados en el punto anterior. Se tuvieron algunos problemas al momento de la implementación, inicialmente la imagen se mostraba distorsionada, tal como se muestra en la Figura 6-34, para detectar el error se tuvo que estudiar un poco sobre el estándar JPEG y hacer un análisis del código hexadecimal que representa la imagen, resultado de este estudio se pudo notar la existencia de varios ceros (0x00) innecesarios, el problema se corrigió modificando algunos puntos en el código del programa del uC.

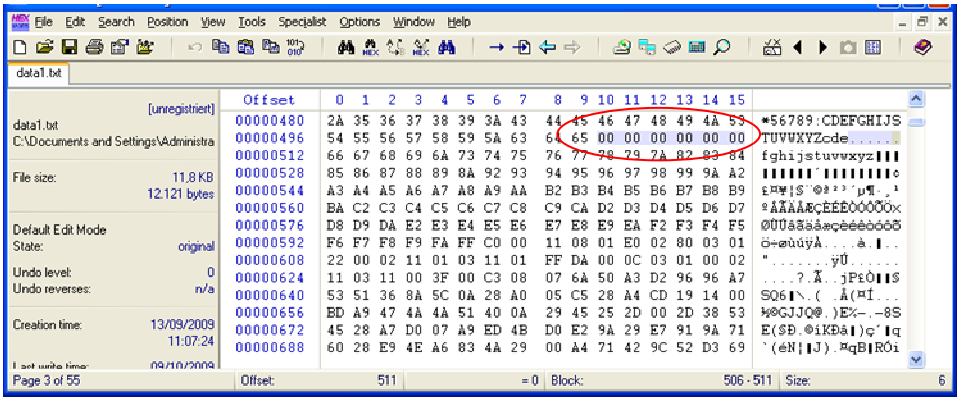


Figura 6-34: Revisión del estándar JPEG, encerrado en el círculo rojo se muestra código innecesario, esto se corrige en el programa del uC

En la imagen anterior se muestra el código de la data almacenada en la tarjeta de memoria SD. Notamos que cada 512 bytes de datos se inserta innecesariamente 6 bytes de ceros, luego de corregir el problema en el código del programa del MCU, se obtuvo los siguientes resultados:

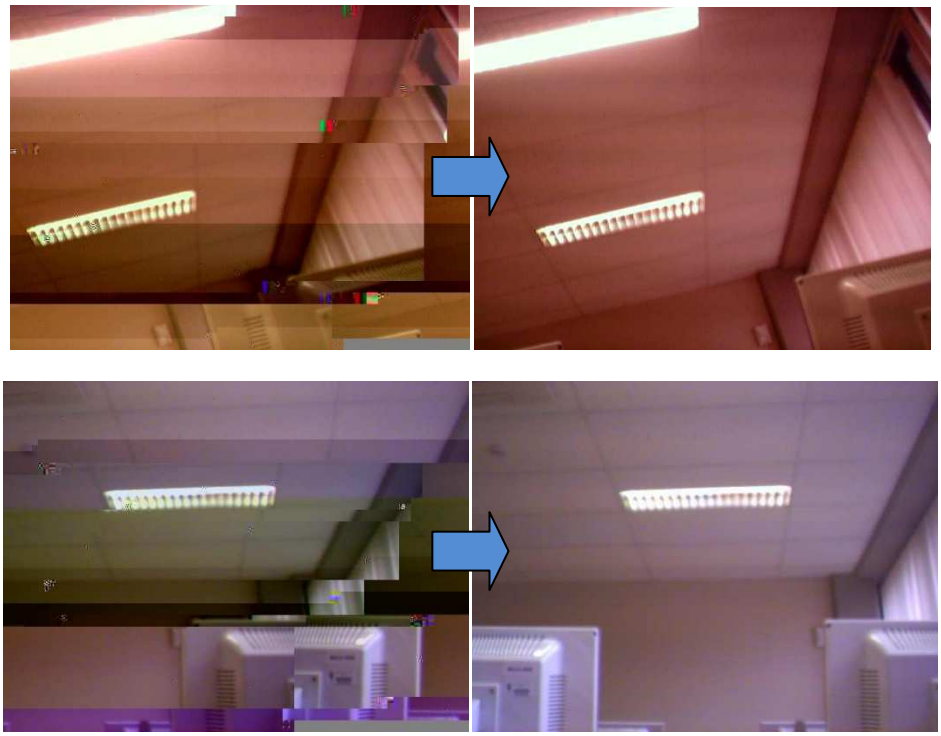
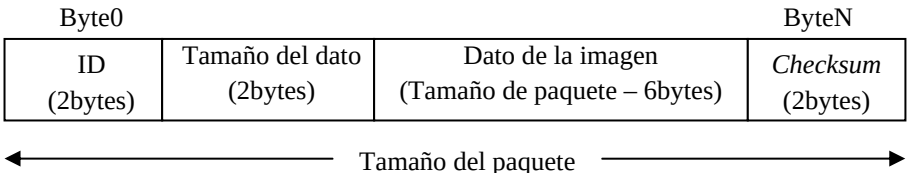


Figura 6-35: Corrección de la imagen.

6.7.4 Empaquetamiento e interfaz para visualización de fotos.

Para nuestra aplicación, será necesario almacenar los datos de la imagen en paquetes, los cuales serán enviados hacia el módulo SICOM para su posterior envío a la estación terrena. Esto permitirá enviar la información por partes, cada una identificada por el número que le corresponde, la cantidad de bytes que contiene y el *checksum* de verificación correspondiente. Se eligió usar el mismo formato de empaquetamiento usado por la cámara C328R, esto para simplificar el código. Este dato, almacenado por paquetes en la tarjeta SD, ya no cumple con el estándar JPEG, por lo que para poder visualizarlo se debe quitar los bytes de identificación de paquete, tamaño de paquete y *checksum*.



Para hacer nuestras pruebas se hizo un programa en Visual C# que permite leer el archivo de la imagen y le da el formato JPEG para luego mostrarla en la parte derecha, además permite visualizar la información de cada paquete.

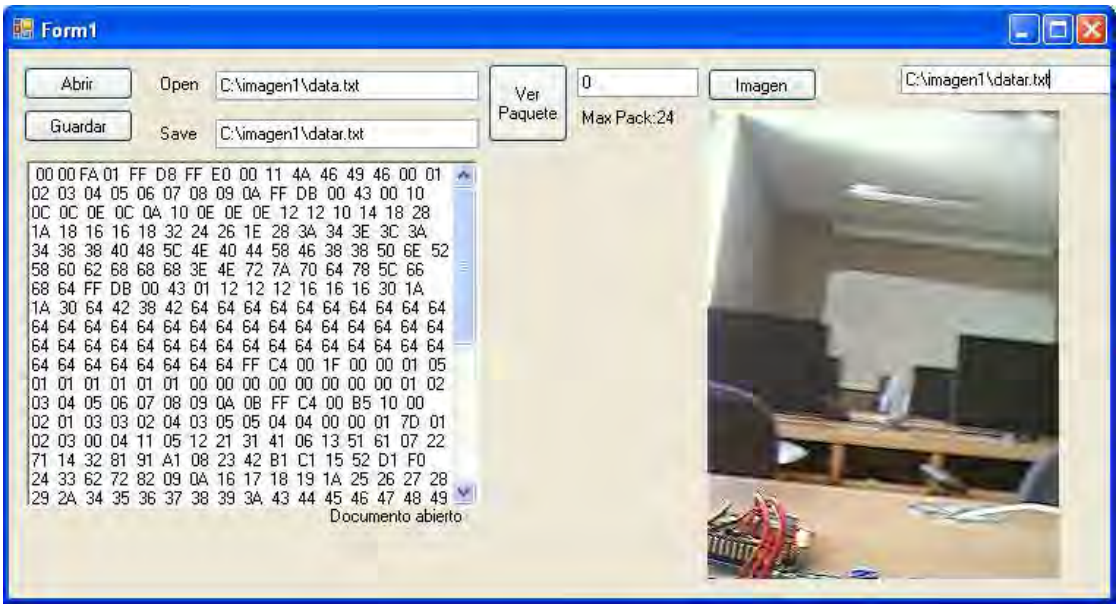
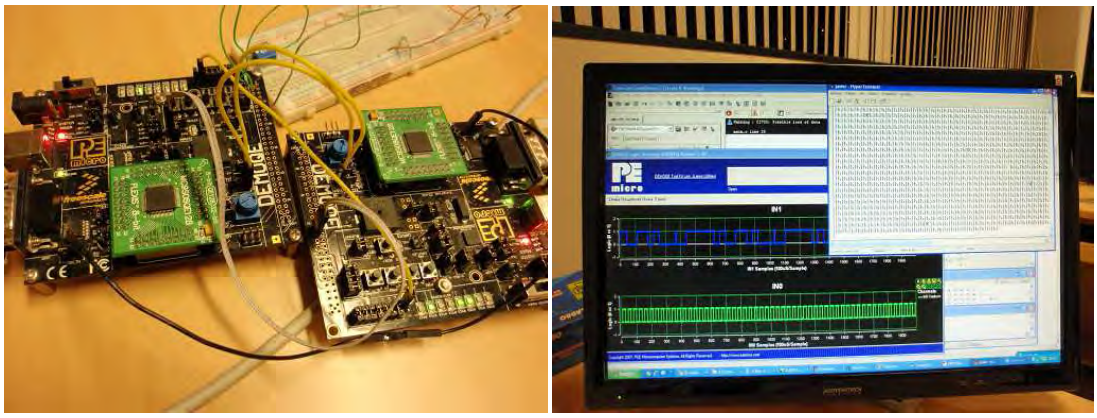


Figura 6-36: Software codificado en Visual C# para monitorear y mostrar la data de la imagen y sus paquetes.

6.7.5 Transmisión de datos I2C

A continuación se muestran las imágenes del circuito implementado para probar la comunicación I2C, para ello primeramente se usó dos módulos DEMOQE, ver Figura 6-37 (a), uno configurado como maestro y otro como esclavo, a modo de prueba el esclavo lee continuamente la señal analógica proveniente de un potenciómetro y almacena una lecturas cada 5ms en un buffer de 512 bytes, cuando este buffer se llena entonces un pin del esclavo se pone en uno y es detectado por el maestro y entonces se da la lectura de los 512 bytes mediante I2C, esto puede verse en la Figura 6-37 donde se muestra las señales I2C, la señal azul representa los datos y la señal verde representa la señal de clock.



(a) *(b)*

Figura 6-37: Transmisión de datos I2C entre dos módulos DEMOQE. (a)Conexión de dos DEMOQUE mediante el bus I2C. (b) Monitoreo de la transmisión de data y clock.

6.8 Implementación de los niveles de respaldo del software

Para asegurar el funcionamiento del sistema ante posibles fallos del software, se debe implementar un triple nivel de respaldo en partes estratégicas del software, para ello es importante saber identificar dichas partes en nuestro código, en nuestro caso estos han sido los posibles bucles infinitos, es decir toda sentencia del tipo *While*. A modo de ejemplo mostraremos una parte del software de CCMI encargado de esperar un sexto byte de la cámara C328R al momento de tomar una fotografía. La hoja técnica de la cámara indica que una respuesta tipo ACK de la cámara consta de 6 bytes, y nosotros en nuestro código esperamos esos 6 bytes dentro de una sentencia tipo *While*, pero debemos tomar precaución de una posible falla en la comunicación que haga que algún byte no llegue, el CPU de CCMI se mantendría esperando dicho byte y por lo tanto nuestro software quedaría colgado. Para evitar esto usamos como primer nivel de respaldo el uso de un *Timeout*, el cual hará salir el *programcounter* del bucle *While* luego de un plazo predefinido de tiempo, esto solo afecta una pequeña parte de código, como segundo nivel de respaldo se usa un WDT que interrumpe el proceso normal del sistema y resetea completamente el software de CCMI, como tercer nivel de respaldo, el módulo PCT es capaz de cortar la alimentación de CCMI si este no le responde por un tiempo aproximado de 30min. A continuación explicaremos el modo de funcionamiento

del *Timeout*, ya que los otros dos son bastante conocidos, en el diagrama de la izquierda mostramos el algoritmo sin Timeout, Flag se pondrá en 1 automáticamente cuando se reciba un sexto byte por SCI, pero si este nunca llega entonces el programa quedaría colgado. En laFigura 6-38 (derecha) notamos la presencia del *Timeout*, el cual es activado automáticamente cuando pasa un tiempo preestablecido, este tiempo vendría a ser el plazo máximo de tiempo en el que debería llegar dicho sexto byte, la implementación en lenguaje C de este algoritmo se muestra en el Anexo 0.

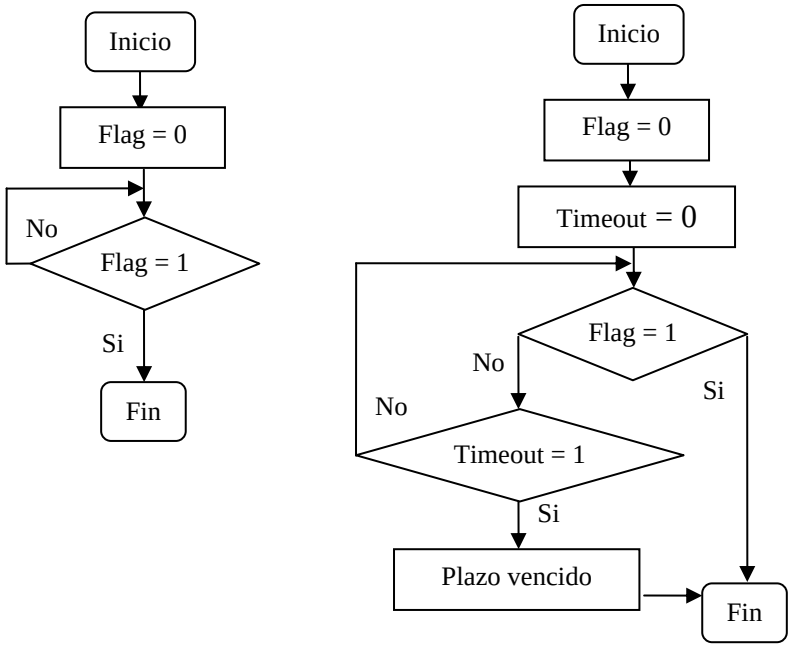


Figura 6-38: Función del Timeout en un algoritmo tipo While.

Capítulo 7

7 Diseño e implementación del Hardware

El diseño del hardware se inicia con la selección de nuestras herramientas de trabajo y los dispositivos que emplearemos en nuestro diseño, para esto se deben tener en cuenta los requerimientos del sistema mostrados en el punto 2.3.1, para esto deben tenerse en cuenta factores como el costo y disponibilidad de dichos componentes en e mercado local o la facilidad de importación de los mismos, además de las herramientas de diseño, en lo posible se trató de usar software gratuito para el diseño CAD.

7.1 Herramientas para la implementación del hardware

A continuación se muestran las herramientas necesarias para la implementación y testeo del hardware, siendo el más importante el software CAD para diseño de tarjetas impresas, en nuestro caso se eligió el CAD EAGLE. También se muestra, en la Figura 7-1, las herramientas e instrumentos importantes para la fabricación y testeo de la tarjeta y soldadura de los componentes.

	**Estación de soldadura** Esta estación debe permitir graduar la temperatura y ésta debe ser ajustada dependiendo de la información técnica de cada componente, además debe poseer pistola de aire caliente para soldaduras especiales de componentes tipo SMD.
	**Materiales útiles para soldadura** Estos materiales deben ser de aplicación aeroespacial, como la soldadura de estaño sin plomo, soldadura en pasta, flux, quita-flux y pasta para soldar, guantes de jebe, mascarilla y un cuarto limpio.
	**Osciloscopio** El osciloscopio elegido contiene funciones para analizar sistemas digitales de alta frecuencia, esto es necesario para depurar nuestro protocolo I2C.

Figura 7-1: Principales herramientas para la implementación del hardware CCMI

7.2 Requerimientos dimensionales

Antes de comenzar con el diseño del hardware de CCMI, debemos determinar cuáles son nuestras limitaciones dimensionales, estas dependerán de todos los demás módulos ya que no todos requieren el mismo espacio dentro del nanosatélite, una necesidad que se dio mientras se avanzaba con el proyecto fue que dos tarjetas, correspondientes a dos módulos, deberían unirse para formar una sola, evaluando las características dimensionales de todas las tarjetas, se concluyó que la tarjeta CCMI y SIMA podrían unirse sin mucha complicación, por lo que se decidió que el circuito electrónico de CCMI incluiría dentro de sí al circuito del módulo SIMA, además se debe tener en cuenta un agujero pequeño que estaba originalmente en la tarjeta SIMA y

servía para que el conector de la tarjeta SICOM pueda pasar hasta la superficie del nanosatélite, por lo que la nueva tarjeta también debería tener en cuenta esta consideración. A continuación se muestra el plano base de la tarjeta, luego se ubicaron los componentes mostrados en el esquema electrónico sobre esta tarjeta.

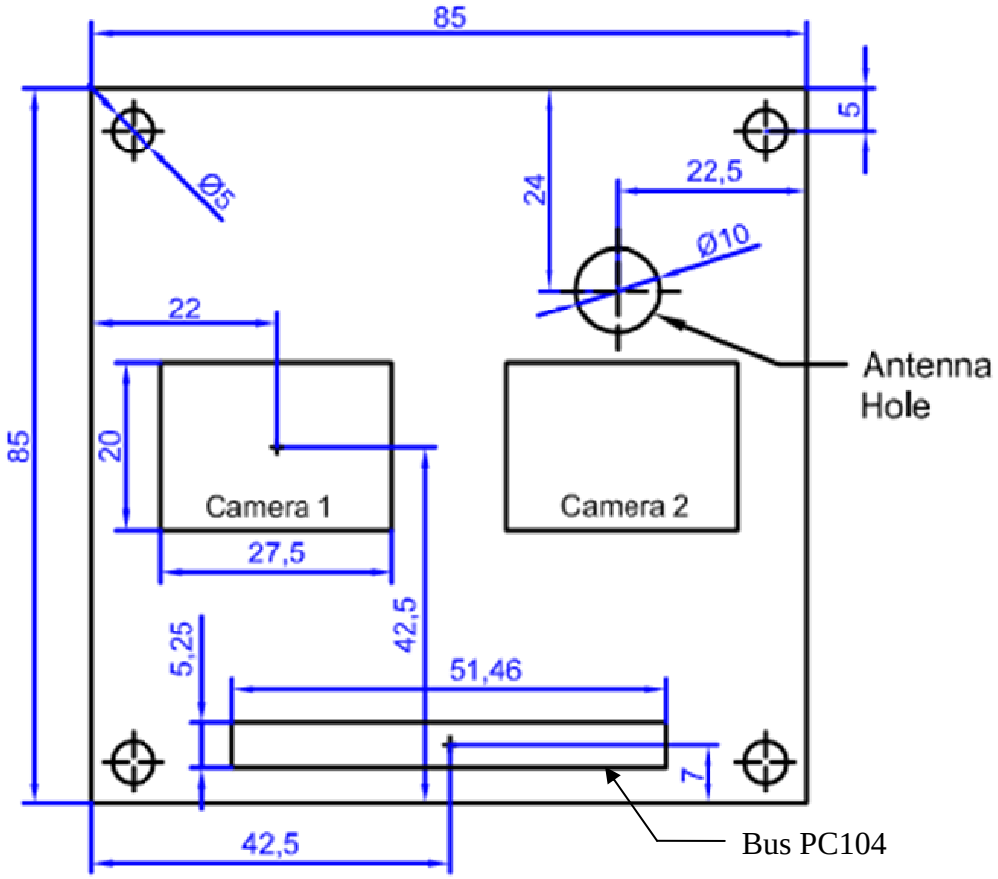


Figura 7-2: Plano base de la tarjeta CCMI

7.3 Selección de componentes

El criterio de selección de componentes electrónicos para nuestra aplicación debe basarse en los requerimientos del hardware, mostrados en la Subsección 2.3.1.1. Resumiendo dicho punto, los principales requisitos que deben cumplir los componentes son soportar vacío térmico de 0.02 mbar, rango de temperaturas de operación entre -20°C a 80°C y ser de bajo consumo eléctrico.

7.3.1 Selección del microcontrolador para el módulo CCMI

El componente más importante del CCMI es el microcontrolador. Este es el primer dispositivo que debe elegirse ya que en base a este se buscarán otros periféricos que se adapten a nuestro controlador. Los principales requisitos que debe cumplir nuestro MCU, además de los ya mencionados, es que microprocesador interno cuente con los periféricos necesarios para comunicarse mediante los puertos UART(2), SPI y I2C(2). Además el microcontrolador debe cumplir con los requerimientos mínimos de memoria de datos y de programa, ver Sección 6.3. Otro factor importante es la facilidad de obtención del chip, documentación, herramientas de entrenamiento y depuración que permitan agilizar nuestro proceso de implementación. También importa la familiaridad del diseñador para programar en el IDE, ver Subsección 6.1.1, ya que esto puede ahorrar tiempo de capacitación. El uso en aplicaciones satelitales aumenta la confiabilidad del MCU elegido. Además se da gran importancia al bajo consumo eléctrico. Para esto muchos microcontroladores pueden configurarse en modos de bajo consumo. Basados en todos estos puntos, se eligieron dos posibles opciones de MCU para CCMI, el MSP340 de *Texas Instrument* y el MC9S08QE128 de *Freescale*. El cuadro comparativo de estos se muestran en el Anexo A6. De este cuadro podemos notar que el MCU de *Freescale* tiene en general mejores recursos que el MSP340 de TI, como son el tamaño de la memoria Flash, la velocidad del CPU y el hardware multiplicador de 32 bits. Además el módulo de *Freescale* cuenta con 2 módulos I2C, lo cual es requisito para los protocolos de comunicación de CCMI. Por estas ventajas se decidió trabajar con el MCU de MC9S08QE128.

La Tabla 7-1 resume los componentes seleccionados para la implementación del módulo CCMI. Esta lista se muestra de forma más detallada en el Anexo A1, donde podemos ver los detalles técnicos de cada componente. En el Anexo 0 también se muestra los costos de cada componente.

Todos estos componentes se eligieron basados en información mostrada en sus hojas técnicas. Además éstas fueron comprobadas en las pruebas de rendimiento del hardware de CCMI, mostradas en la Sección 8.2.

Tabla 7-1: Principales componentes de la tarjeta del módulo CCMI.

Principales Componentes del módulo CCMI			
Descripción	**Valor/Tipo**	**Codigo DIGIKEY**	**Empaquetado**
uCFreescaleLowPower	MC9S08QE128CLH	MC9S08QE128CLH-ND	LQFT-64
RTC	DS1338-33	DS1338Z-33+-ND	SOIC-08
Memoria EEPROM I2C	24AA1025	24LC1025-I/SM-ND	SOIJ-08
SD-Connector	SG5	670-1526-ND	SMD
Cristal para RTC	32,768Khz	XC1371-ND	Throughhole
Cristal para MCU	4MHz		SMD
Bateria + Socket	BATTERY LITHIUM 3V RECHARGE	728-1040-1-ND	HB414-II06E
Multiplexor	IC MULTIPLEXER QUAD 2CHAN 16SOIC	497-7843-1-ND	SOIC-16
BuzzerMagnetico	BUZZER MAGNETIC 3.0V 9MM SMD	668-1079-1-ND	SMD
Cámara UART	C328R	C328R-2820BW (electronics123)	Throughhole

7.4 Desarrollo del esquema electrónico

En este punto mostraremos a detalle la función de los pines con los que CCMI interactúa con los demás módulos.

7.4.1 Pines de interconexión con los demás módulos

En el proceso de desarrollo de todos los módulos internos del Chasqui es importante definir desde un inicio cómo se van a comunicar todos estos. En base a esta información cada módulo podrá diseñar sus tarjetas electrónicas respetando siempre los acuerdos mostrados en este punto. Se ha definido previa coordinación con los otros

módulos conectados a CCMI, es decir SICOM, SIMA, UDCA y PCT, los pines de conexión física, así como los tipos de protocolos y características de comunicación a usar (I2C, SPI o UART). Se definen primeramente los siguientes pines que representan estados lógicos:

- BME: Bit modo de emergencia
- BPEST: Bit, SDCA pregunta a CCMI, ¿Puedo estabilizarme?
- BCPET: Bit Confirmación, desde CCMI, de petición de estabilización.
- BOEST: Bit Orden a SDCA, desde CCMI, de estabilización.
- BCEST: Bit Confirmación de estabilización.
- BSCAM: Bit de selección de cámara.
- BST: Bit de señal de Tierra para entrar en modo de emergencia.
- BRE: Bit de respuesta, de SICOM para PCT, a solicitud de entrar en modo de emergencia.
- BSE: Bit de solicitud de entrada a modo de emergencia.

El tipo de conectores elegido para interconectar los módulos es del tipo PC104. Basados en lo mostrado anteriormente en la Figura 2-1, el equipo del proyecto acordó un bus común para todos, este cuenta con 40 pines e incluye todas las señales que sirven para comunicación entre módulos, incluyendo fuente de alimentación, buses I2C, UART, y pines de entrada y salida. La Figura 7-3 resume esta distribución, se debe resaltar la independencia de tierras para cada módulo, esto para evitar que el alto consumo de corriente, dados en determinados momentos de operación de módulos como SICOM o SDCA, puedan afectar a los demás.

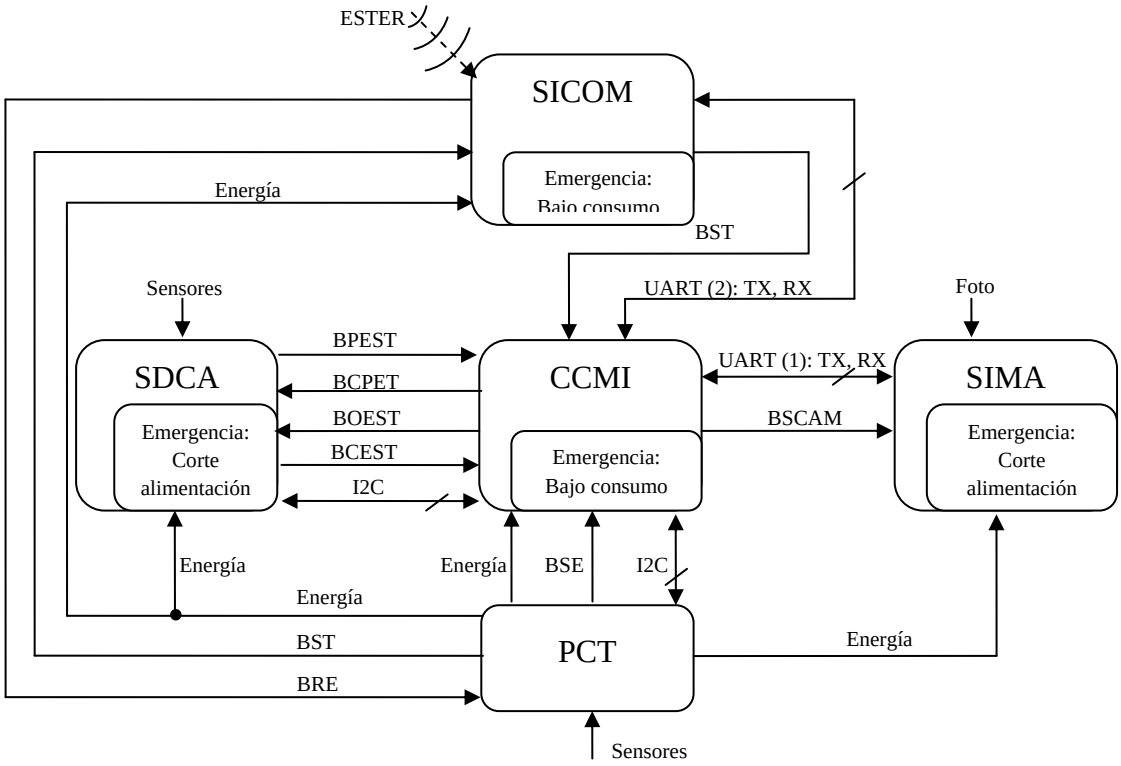
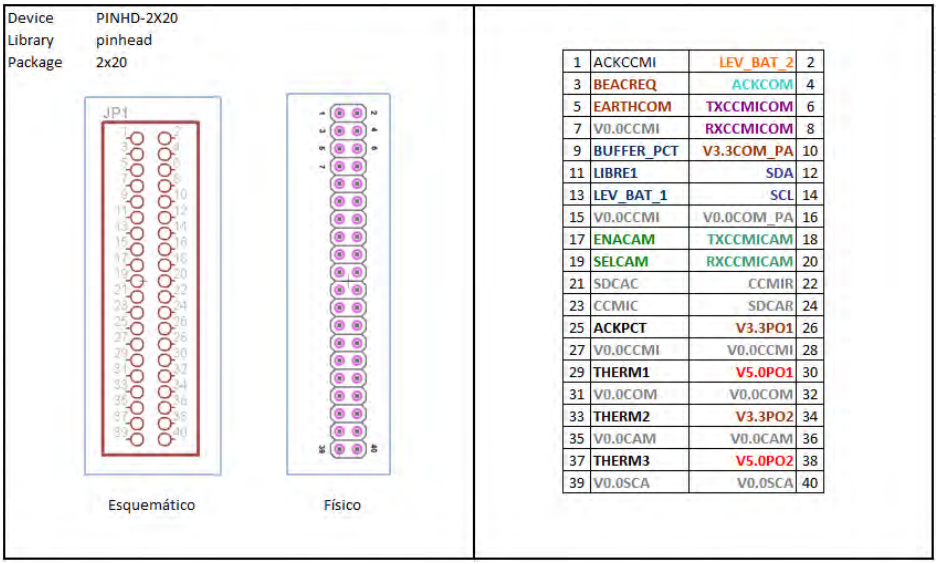


Figura 7-3: Diagrama de bloques eléctrico del sistema.

En base a este diagrama circuital se desarrolló un diagrama de distribución de pines completo, en el cual se tiene en cuenta todas las consideraciones requeridas por los demás módulos, la Figura 7-4 muestra el esquema del conector y el nombre de designado a los pines del bus, la descripción de cada uno de estos pines se detalla claramente en el Anexo 0, donde también se muestra las características de cada pin como son la dirección del mismo, tipo de señal a transmitir, niveles de voltaje, etc.



1	ACKCCMI	LEV_BAT_2	2
3	BEACREQ	ACKCOM	4
5	EARTHCOM	TXCCMICOM	6
7	V0.0CCMI	RXCCMICOM	8
9	BUFFER_PCT	V3.3COM_PA	10
11	LIBRE1	SDA	12
13	LEV_BAT_1	SCL	14
15	V0.0CCMI	V0.0COM_PA	16
17	ENACAM	TXCCMICAM	18
19	SELCAM	RXCCMICAM	20
21	SDCAC	CCMIR	22
23	CCMIC	SDCAR	24
25	ACKPCT	V3.3PO1	26
27	V0.0CCMI	V0.0CCMI	28
29	THERM1	V5.0PO1	30
31	V0.0COM	V0.0COM	32
33	THERM2	V3.3PO2	34
35	V0.0CAM	V0.0CAM	36
37	THERM3	V5.0PO2	38
39	V0.0SCA	V0.0SCA	40

(a) (b)

Figura 7-4: Diagrama de distribución de pines del bus común PC104, (a) esquemática y forma física del componente, (b) descripción de los pines.

7.5 Diseño del esquemático de la tarjeta

A continuación se muestra las características del diseño de la tarjeta CCMI que incluye al módulo SIMA, esta tarjeta fue desarrollada de tal forma que permita facilitar las pruebas de funcionamiento del módulo CCMI, ya que cuenta con periféricos que permiten la comunicación de dos buses RS232. Además de un *buzzer* que nos permita monitorear el estado del uC de acuerdo al sonido que este dispositivo emita, esto con el objetivo de identificar el buen funcionamiento del sistema cuando esté integrado con el cubo. Esta fue una limitación de los indicadores de LEDs de los prototipos anteriores ya que dichos LEDs no pueden ser visualizados dentro del cubo, este prototipo también incluye 5 LEDs a la salida del uC y otros 4 LEDs para identificar la transmisión y recepción de los dos puertos UART.

Esta tarjeta incluye además el termistor en la posición central del PCB, a solicitud del módulo PCT.

7.5.1 Microcontrolador HCS08QE128 con buzzer magnético.

En la Figura 7-5 notamos que el MCU está conectado a un cristal que funciona como señal de reloj para nuestro CPU, este cristal es de 4 MHz ya que así podemos configurar nuestro SCI a 9600 bps, para la comunicación con SICOM, con un mínimo error. El buzzer es incluido en el circuito ya que permite monitorear el funcionamiento del sistema cuando CCMI esté dentro de la estructura del cubo y no podamos visualizar los LEDs indicadores de estado.

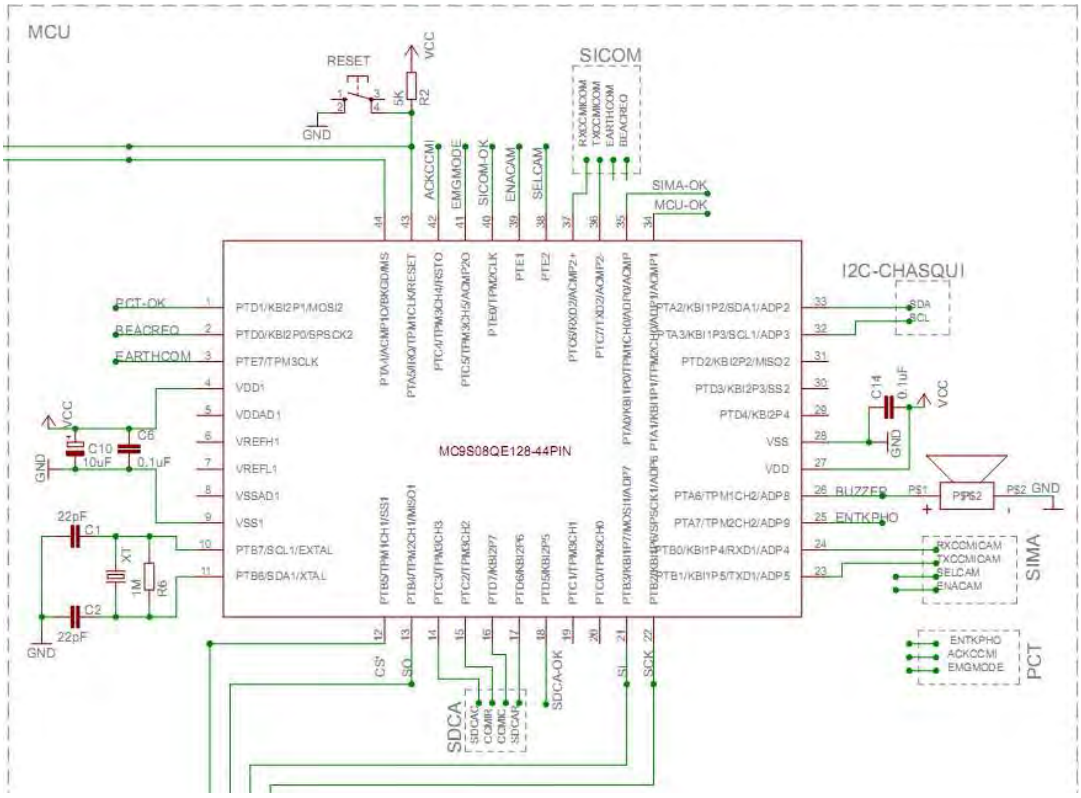


Figura 7-5: Esquema electrónico del MCU

7.5.2 Regulador para la alimentación externa independiente del módulo PCT.

Para hacer pruebas de CCMI se incluye en la tarjeta el circuito que regula el voltaje de alimentación, esto es para no depender del módulo PCT cuando se hagan pruebas de funcionamiento.

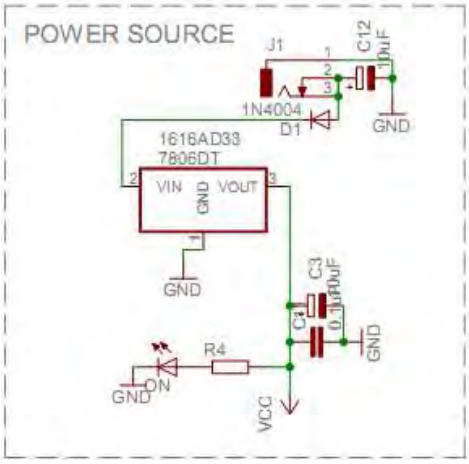


Figura 7-6: Esquema electrónico de la alimentación externa.

7.5.3 LEDs indicadores de estado.

Al igual que el Buzzer, los LEDs permiten monitorear el estado de funcionamiento de CCMI, esto siempre que CCMI no esté dentro de la estructura cerrada del Chasqui. Los cátodos de los LEDs son enviados a VCC ya que el MCU puede enviar menos corriente de la que puede recibir, de esta forma ahorramos energía.

Figura 7-7: Esquema electrónico para la alimentación externa

7.5.4 Conector de programación del microcontrolador

Todo circuito microcontrolado en proceso de desarrollo debe tener los pines de grabación disponibles, la ubicación de este conector debe ser en los bordes de la tarjeta

para facilitar su conexión, este conector debe ser tipo L para evitar que impida la conexión de una tarjeta superior.

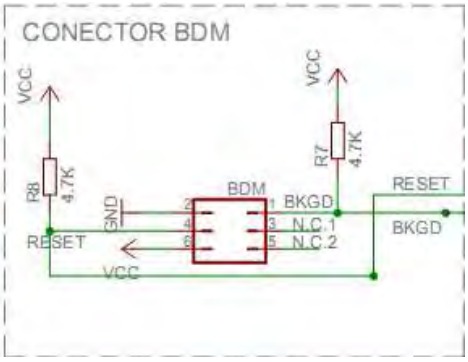


Figura 7-8: Esquema electrónico del conector BDM

7.5.5 Memorias EEPROM

Las memorias EEPROM han sido incluidas desde un comienzo como una alternativa para guardar información de los datos. En este caso ya no son utilizados pues la memoria SD funciona correctamente.

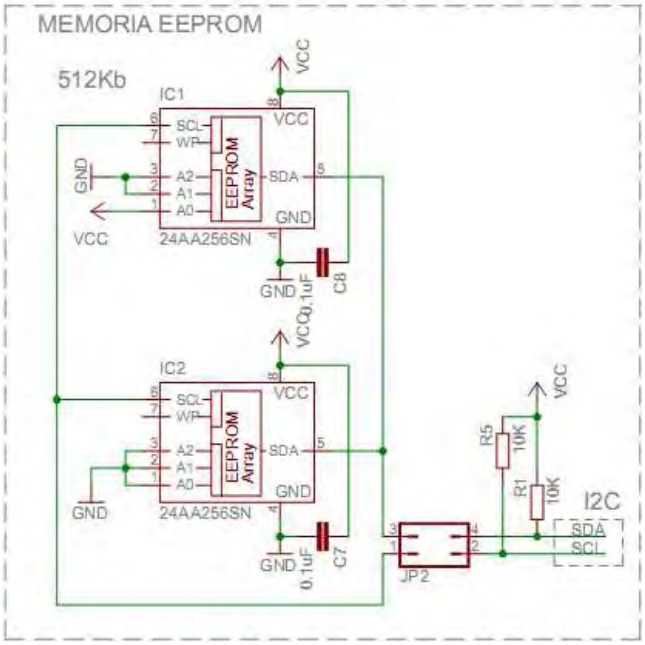


Figura 7-9: Esquema electrónico de las memorias EEPROM

7.5.6 Tarjeta de memoria SD-Card

Existe un problema con la memoria SD, ya que están cubiertos por un material tipo plástico, para solucionar esto dicho plástico debe ser retirado con cuidado y el circuito interior debe ser soldado directamente a la tarjeta.

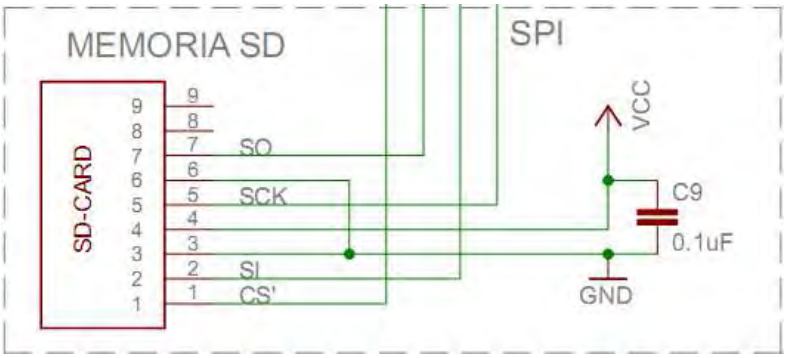


Figura 7-10: Esquema electrónico de la SD-Card

7.5.7 Bus común PC104 y termistor de PCT

El bus común conecta en forma horizontal las 4 tarjetas que conforman el Chasqui, en el Anexo 0 podemos visualizar la distribución de pines detallado.

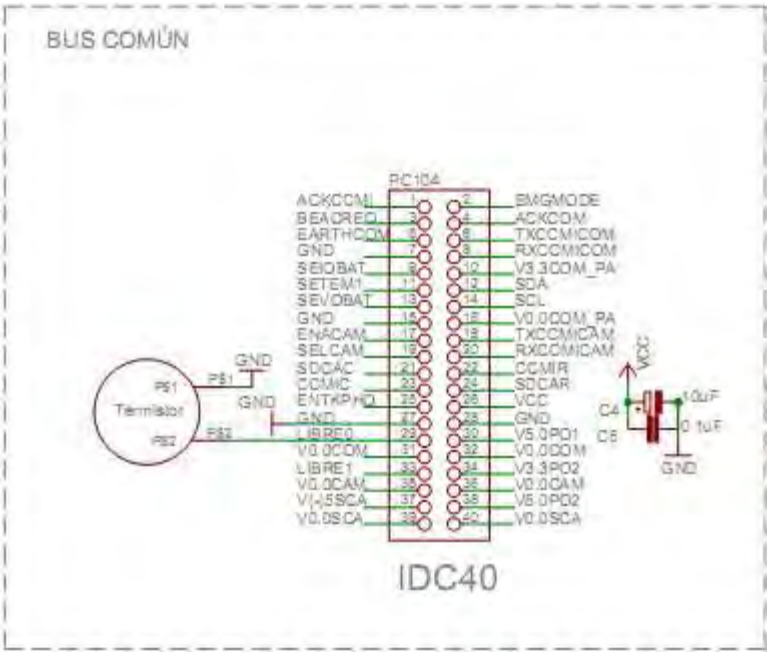


Figura 7-11: Esquema electrónico del bus común PC104

7.5.8 Módulo CCMI que incluye dos cámaras: Visible e IR

Debido a que solo contamos con dos módulos SCI, además uno de ellos está empleado únicamente para la comunicación con SICOM, entonces se decidió incluir un circuito multiplexor que permita tomar fotografías con ambas cámaras, tal como se muestra en el circuito de la Figura 7.4.8.

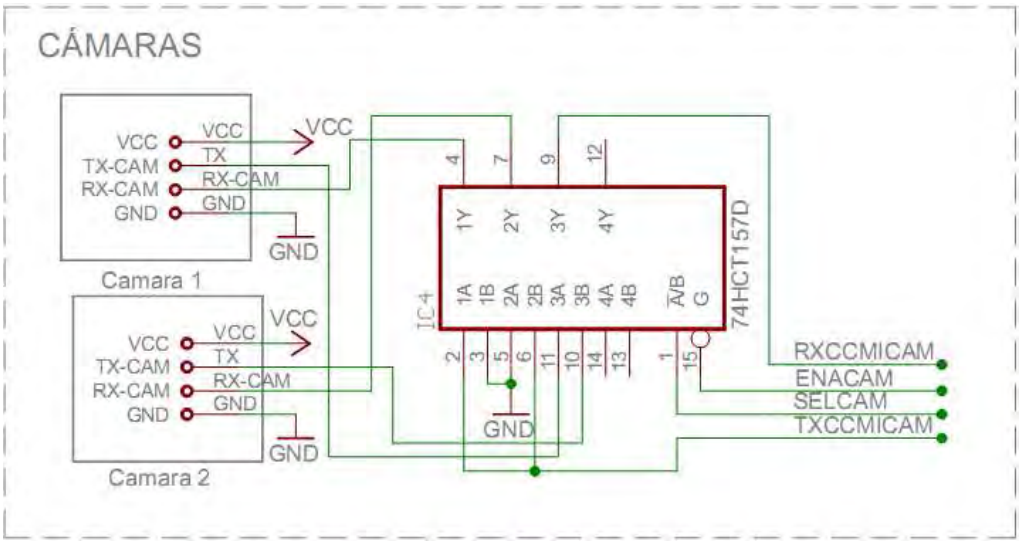


Figura 7-12: Esquema electrónico de las cámaras

Para prevenir que un mal funcionamiento de las cámaras pueda afectar nuestros procesos, es necesario que el módulo CCMI pueda acceder a los pines de reset de los MCUs que controlan las cámaras, este detalle no se muestra en este esquema, pero si se puede visualizar en la imagen Figura 7-13.

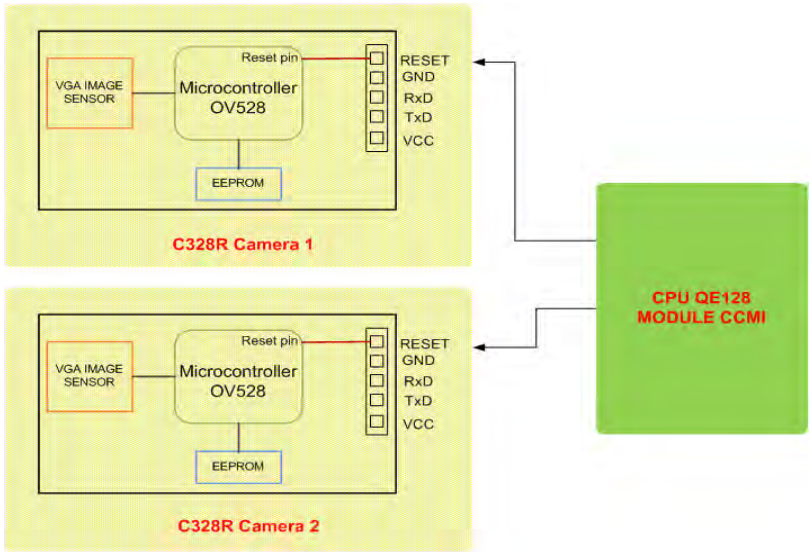


Figura 7-13: Conexión entre el MCU y los pines de reset de las cámaras.

7.5.9 Interfaz RS232 para dos puertos UART

Para poder monitorear más detalladamente el estado de funcionamiento de CCMI se incluyó un circuito driver para RS232, esto nos permitirá enviar reportes de funcionamiento a una computadora, esto fue bastante útil al momento de hacer la prueba de vacío térmico y detectar en qué momento falla alguno de los dispositivos de CCMI, ver Subsección 8.2.1.

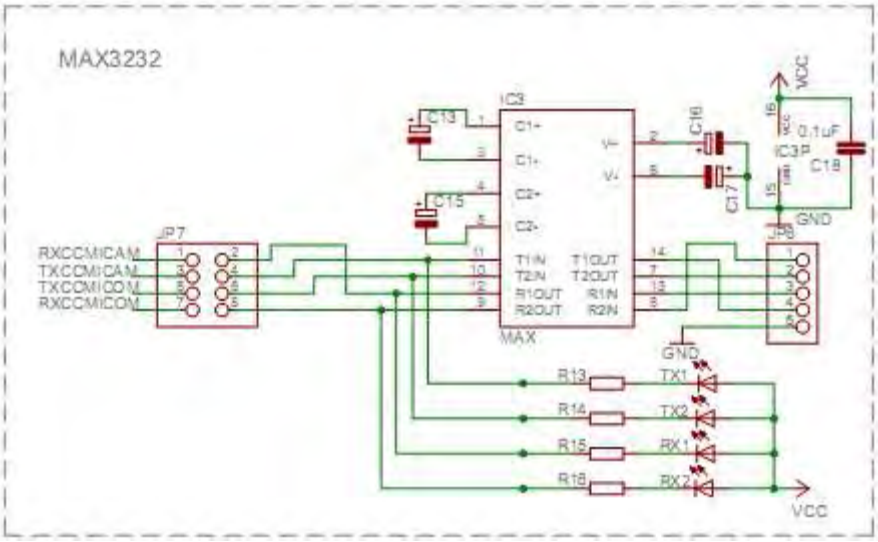


Figura 7-14: Esquema electrónico del driver RS232

7.6 Diseño CAD del PCB

El diseño CAD del PCB fue realizado en el software EAGLE, cuyo diagrama electrónico completo está mostrado en el Anexo A10. La Figura 7-15 muestra el diseño PCB en EAGLE, en los siguientes puntos mostramos a detalle cada parte de que conforma el diseño.

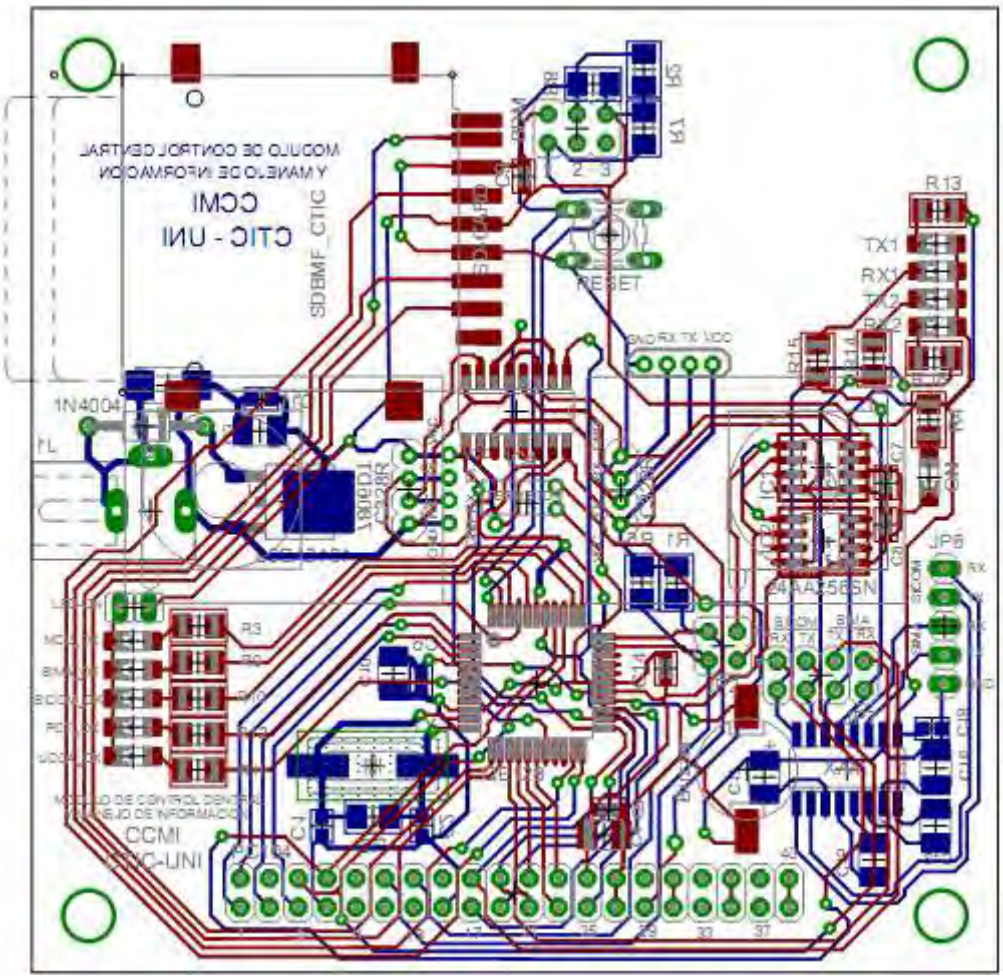
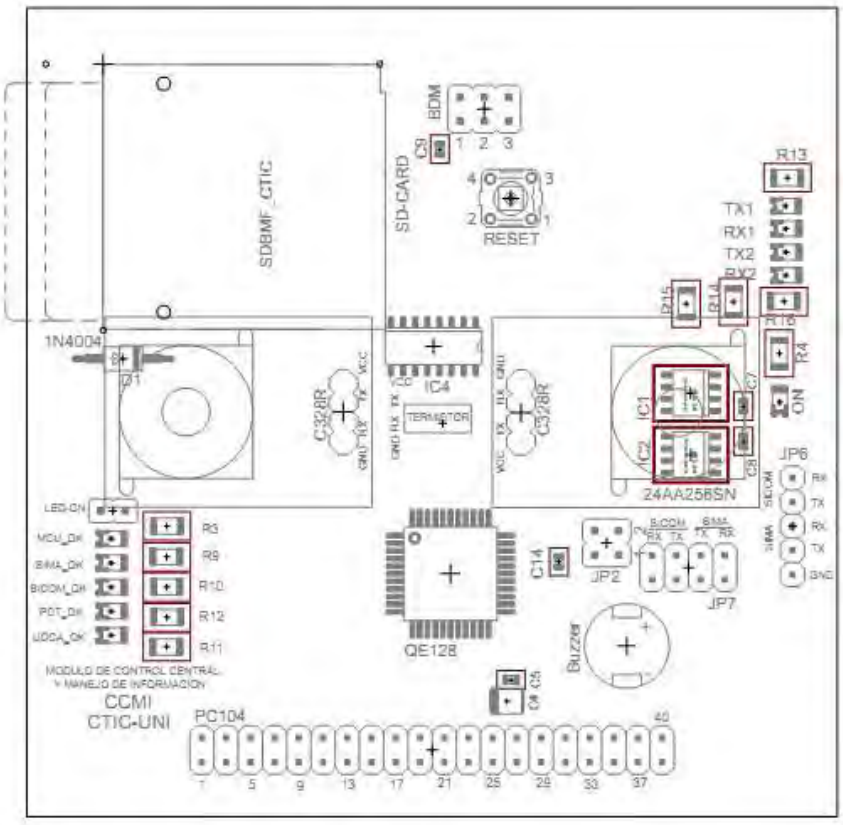


Figura 7-15: Diseño CAD del PCB, se muestra en azul las rutas de la capa inferior y en rojo las rutas de la capa superior.

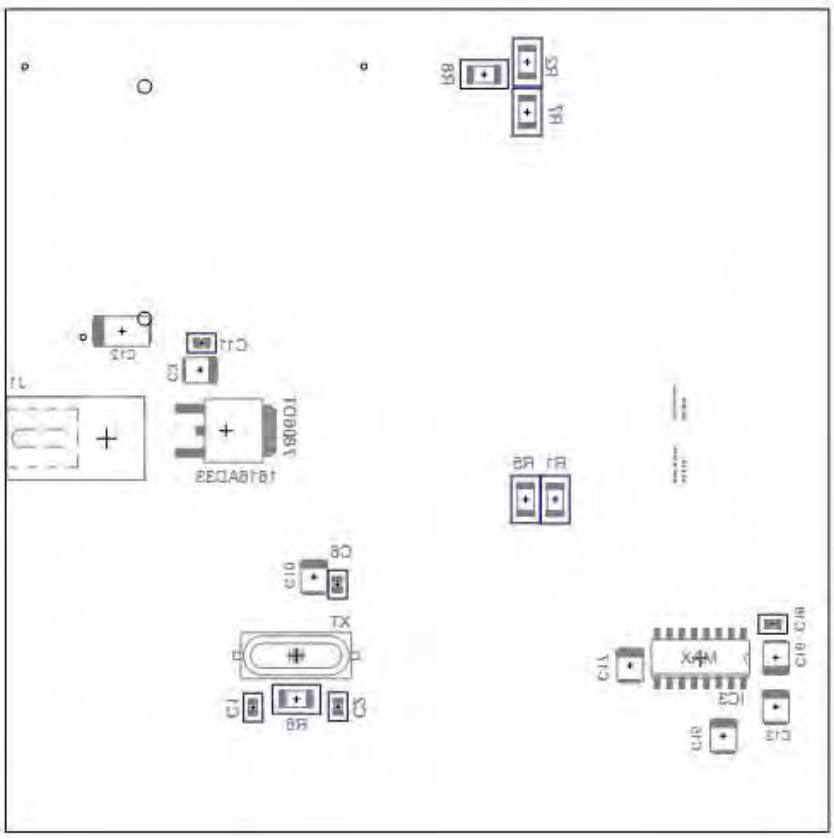
7.6.1 Ubicación de los componentes

Debido a que el MCU tiene muchas conexiones con el bus PC104, ver Figura 7-4, fue conveniente ubicar dicho MCU en una posición central cercana al PC104, la posición de las cámaras está predefinida por la estructura mecánica, debido a que el

circuito de dicha cámara está ubicado en una capa superior a CCMI entonces podemos ubicar componentes debajo de éstas. Los LEDs fueron posicionados en el borde de las tarjetas para poder visualizarlos con facilidad. Los pines de grabación también están ubicados en el borde para facilitar su conexión. Por requerimiento del módulo PCT se ubicó un termistor en el centro de la tarjeta CCMI, ver Figura 7-16.



(a)

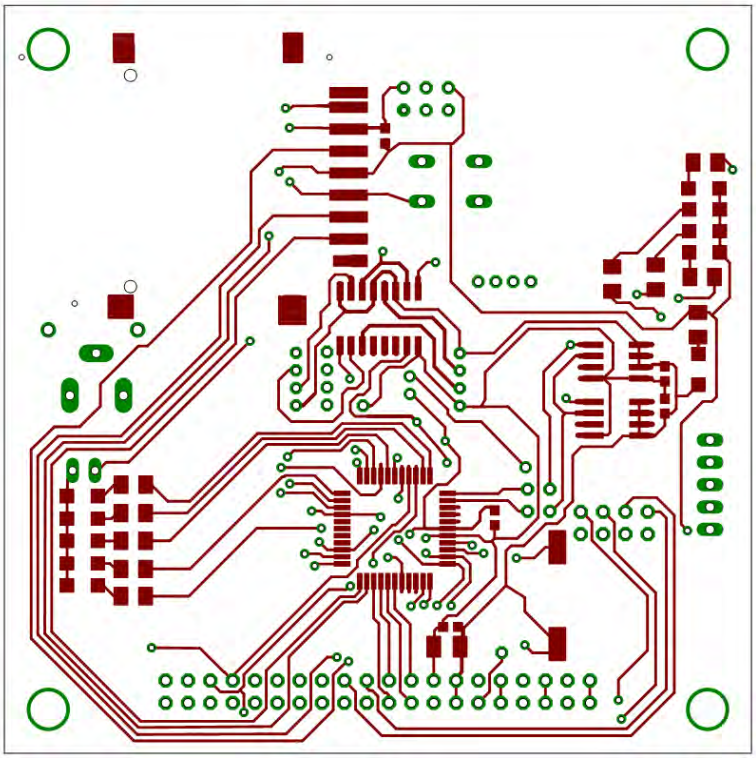


(b)

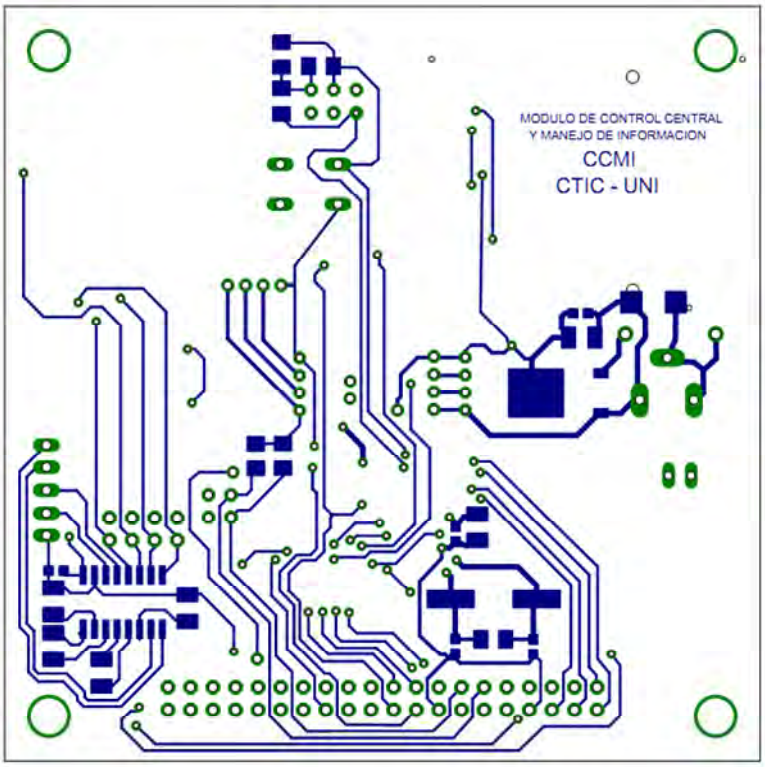
Figura 7-16: Ubicación de los componentes en la tarjeta CCMI. (a) capa TOP, (b) capa BOTTON.

7.6.2 Ruteo de las conexiones del circuito

El ruteo de la tarjeta consta de dos capas de cobre, superior (TOP) e inferior (BOTTON), ver Figura 7-18, el espesor de las líneas de cobre está en función del nivel de corriente que circula por él, en nuestro caso CCMI solo trabaja con señales digitales cuyas corrientes están en el orden de los microvoltios y le corresponde un espesor de 0.3 mm. Solo las rutas de la fuente de alimentación fueron diseñadas con un espesor de 0.6 mm, los *pads* de los componentes fueron basados en la información de las hojas técnicas de cada dispositivo. Las rutas en ningún caso forman ángulos agudos ni rectos.



(a)



(b)

Figura 7-17: Ruteo de las conexiones en la tarjeta CCMI, (a) capa TOP, (b) capa BOTTON.

7.6.3 Verificación del diseño del esquema eléctrico

Para verificar el buen diseño, primero se hace uso de las herramientas de EAGLE para verificar la continuidad de las líneas, además de atender las alertas del programa, ver Figura 7-18. Para verificar que el diseño cumpla con los requerimientos dimensionales, se imprime el PCB en hojas simples con cuidado de no escalar la imagen. Luego se ubican los componentes sobre la hoja para comprobar que los *pads* tienen dimensiones correctas. Después de verificar esto, podemos fabricar la tarjeta electrónica y antes de soldar los componentes verificamos si existe algún error de fabricación en la tarjeta, para esto se comprueba la continuidad en todas las rutas. El primer dispositivo a soldar debe ser el regulador y debe comprobarse que los voltajes son correctos y deben corresponderse con los pines correspondientes. Al cumplirse estas revisiones se sueldan todos los demás componentes electrónicos.

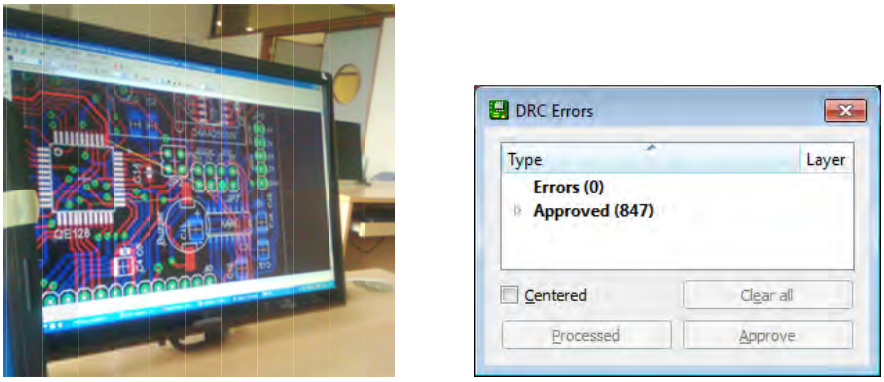


Figura 7-18: Un total de 847 observaciones de error detectados por el software EAGLE, los cuales fueron corregidos en el proceso de verificación del diseño CAD.

7.7 Implementación de la tarjeta CCMI

Para esto soldamos todos los componentes usando soldadura de estaño sin plomo, verificamos las conexiones con un multímetro.

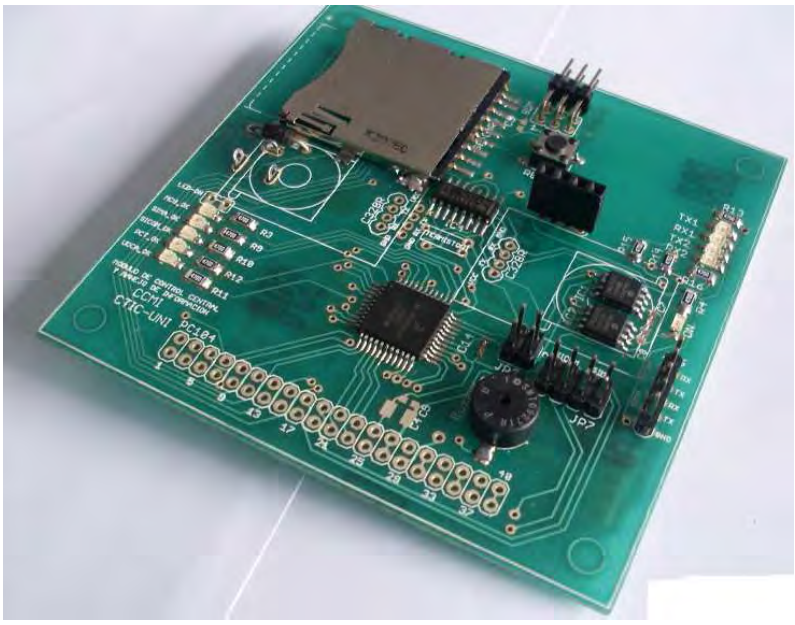


Figura 7-19: Implementación de tarjetas sin cámaras integradas.

Luego de esto y antes de soldar las cámaras, debemos soldar un cable desde el pin de reset de ésta hasta dos pines libres del MCU, esto para poder corregir algún problema del módulo de cámaras. La Figura 7.8 muestra a detalle esta operación.

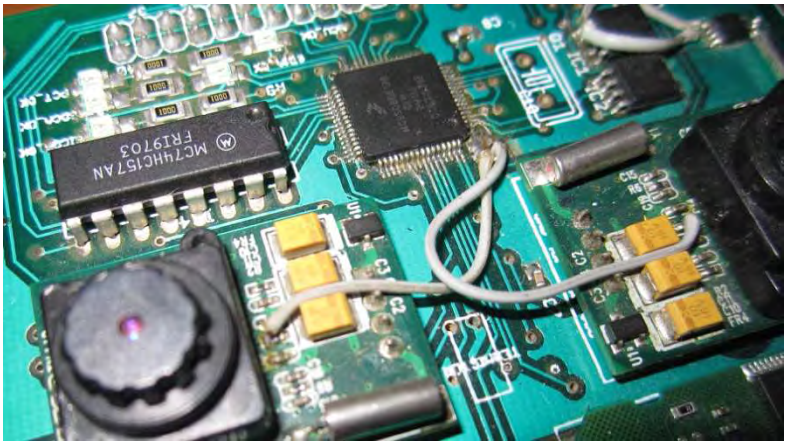
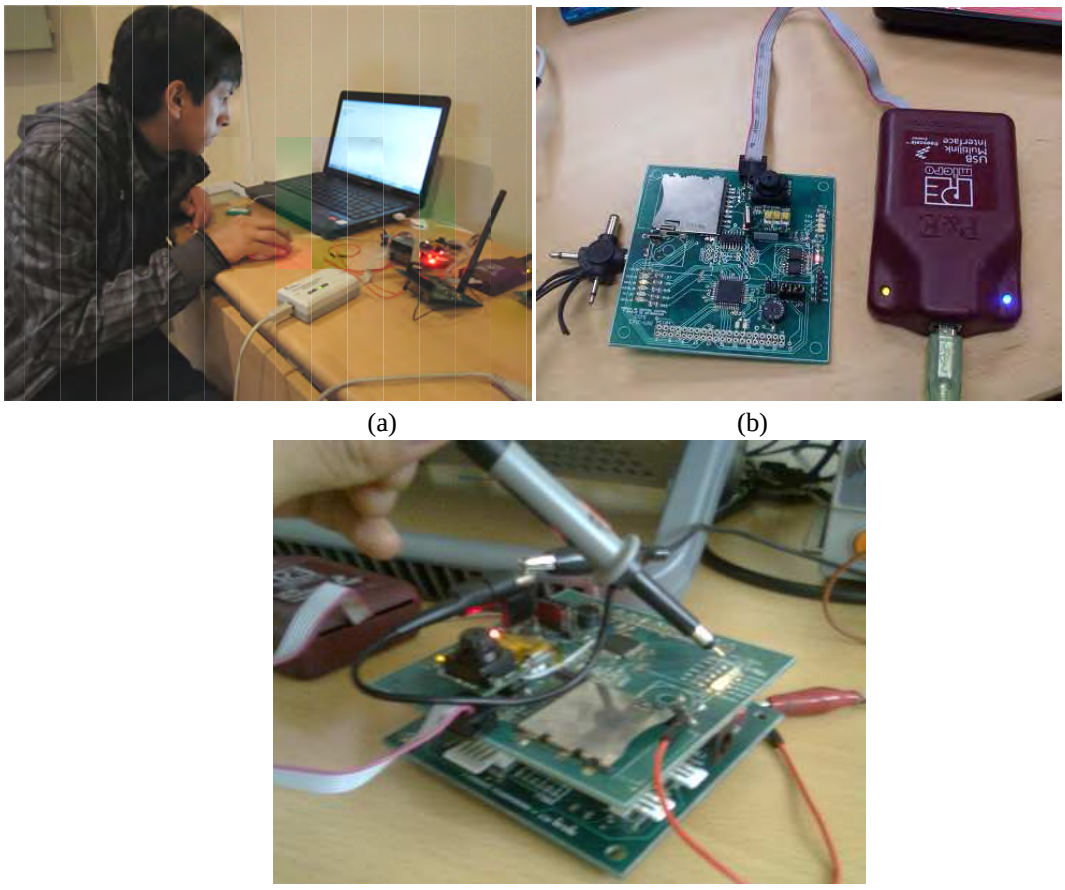


Figura 7-20: Integración de las cámaras y conexión de los pines de reset al MCU.

7.8 Testeo de funcionamiento de la tarjeta CCMI

La Figura 7-21 (a) muestran las primeras pruebas de testeo del circuito en laboratorio, para asegurarnos que todos los periféricos internos y externos al MCU de CCMI son correctos, se probaron distintos programas básicos de pruebas encargados únicamente del dispositivo cuyo funcionamiento se desea comprobar, para esto nos

ayudamos del depurador del IDE CodeWarrior, ver Figura 7-21(b), con el cual ejecutaremos paso a paso el código para detectar algún posible error, de darse este caso procedemos a emplear cuidadosamente instrumentos de medición para detectar y corregir el error en la tarjeta, ver Figura 7-21(c).



(a) (b)

(c)

Figura 7-21: Testeo de la tarjeta CCMI. (a) Monitoreo, usando una PC, de la ejecución del código del programa. (a) Emulación del programa de CCMI usando USB Multilink. (b) Testeo del funcionamiento del hardware y software haciendo uso de instrumentos de medición.

7.9 Modelo de vuelo de tarjeta CCMI

El circuito descrito hasta ahora cuenta con partes que son útiles para monitorear el funcionamiento y estado de los procesos que ejecuta la tarjeta CCMI, pero estos no nos servirán cuando esté en funcionamiento en el espacio, por ello, como producto final se desarrolla un modelo de vuelo que no necesita de los periféricos como LEDs o Buzzer, además en esta versión la SD debe estar soldada directamente a la tarjeta y su

cubierta debe ser retirada. En esta tesis se detalló el modelo completo a pesar de no ser el que finalmente será puesto en operación ya que éste incluye dentro de sí al módulo de vuelo. La Figura 7-22 muestra la imagen de la tarjeta de vuelo con las características descritas y el esquema electrónico puede verse en el Anexo A11.

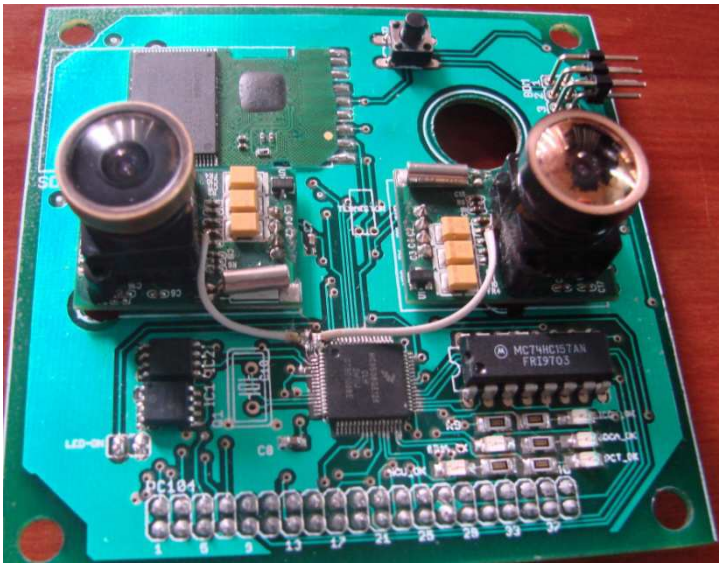
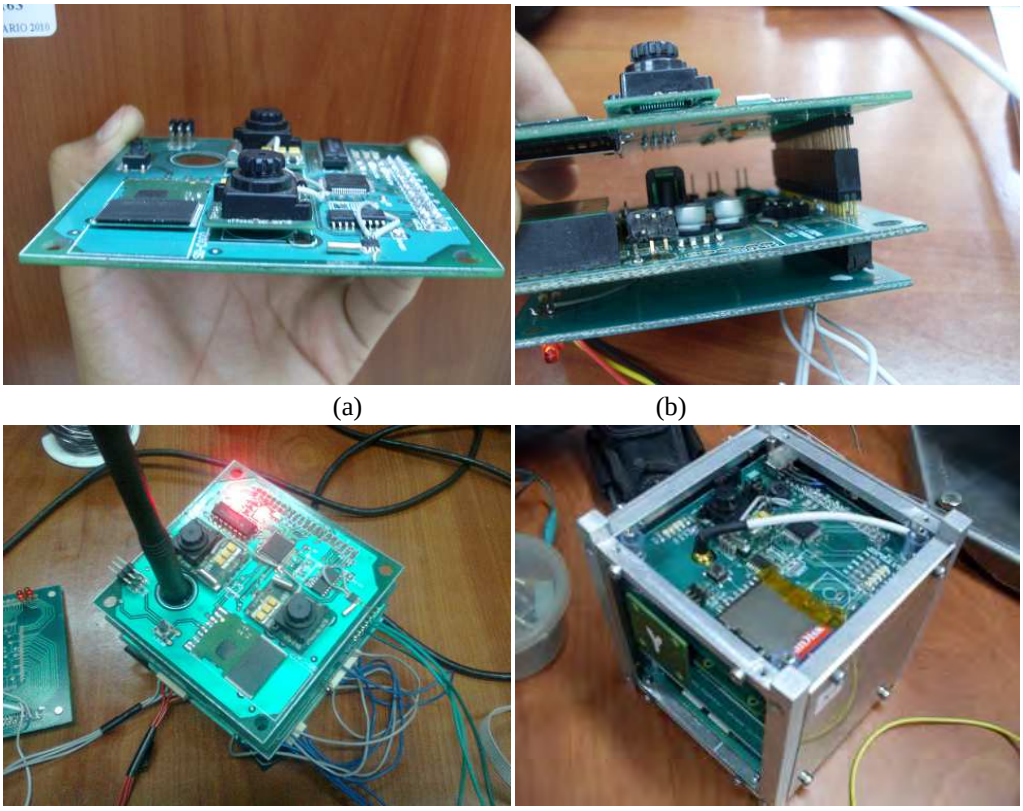


Figura 7-22: Modelo de vuelo de la tarjeta CCMI.

7.10 Integración de la tarjeta CCMI

Debido a que la tarjeta CCMI estará ubicado en la parte superior, éste no necesita un conector PC104 tipo hembra, ver Figura 7-23(a), por ello también debe tenerse mucho cuidado para evitar contacto con la cara superior de la estructura de aluminio, el cual es conductor, el módulo MIP se encarga de prevenir dicho caso. La tarjeta CCMI debe ser conectada con el bus PC104 hembra de la tarjeta SICOM, ver Figura 7-23(b), la antena de SICOM debe pasar por el agujero de CCMI y deben conectarse además el módulo SDCA y PCT debajo de SICOM, cuando esto esté terminado se comprueba el funcionamiento global con todos los módulos conectados, ver Figura 7-23(c). Luego de esto se repite este proceso pero usando la estructura mecánica que sujetará firmemente todas las tarjetas para evitar problemas de conexión, de esta última labor mencionada se

encarga el módulo MIP. La Figura 7-23(d) muestra el resultado de la integración de las tarjetas internas con la estructura del nanosatélite Chasqui.



(a) (b)

(c) (d)

Figura 7-23: Integración de la tarjeta CCMI. (a) Vista de perfil de la tarjeta con las cámaras y tarjeta SD. (b) Conexión de la tarjeta a las demás mediante el conector PC104. (c) Conexión y primera prueba de todas las tarjetas conectadas. (d) Integración final de todas las tarjetas y la estructura del Chasqui.

Capítulo 8

8 Pruebas de funcionamiento

8.1 Pruebas de funcionamiento del software de CCMI

A continuación mostraremos dos pruebas realizadas al software de CCMI, el primero con la intensión de medir los tiempos de ejecución de cada proceso. Estos parámetros sirven para comprobar que nuestro RTOS es *planificable,* ver Subsección 6.6.2. La segunda prueba mostrada sirve para comprobar el funcionamiento del algoritmo de toma de fotografía. Para lograr esto se debe programar a CCMI para que repita continuamente esta acción y almacene en memorias fotografías y que reporte continuamente su estado a ESTER por intermedio del SICOM.

8.1.1 Medición del tiempo de ejecución de los procesos de CCMI

Para la evaluación del tiempo se hizo uso de un osciloscopio. Al iniciar a ejecutarse una función de CCMI se pone un pin en nivel alto, y luego al terminar de ejecutarse la función dicho pin se pone en nivel bajo, entonces el tiempo que demora una función será el tiempo en que la señal de dicho pin estuvo en nivel alto. Para facilitar el proceso de medición se programó un bucle infinito solo con la función a analizar, la cual se está ejecutando continuamente. En este caso cada vez que termina de ejecutarse la función hacemos que un pin cambie de estado, así lo que veremos en el osciloscopio al monitorear dicho pin será una onda cuadrada, cuyo periodo será dos

veces el tiempo de ejecución de dicha función. La Figura 8-1 muestra el diagrama de flujo que representa el algoritmo utilizado para medir el tiempo de ejecución de cada función de forma independiente.

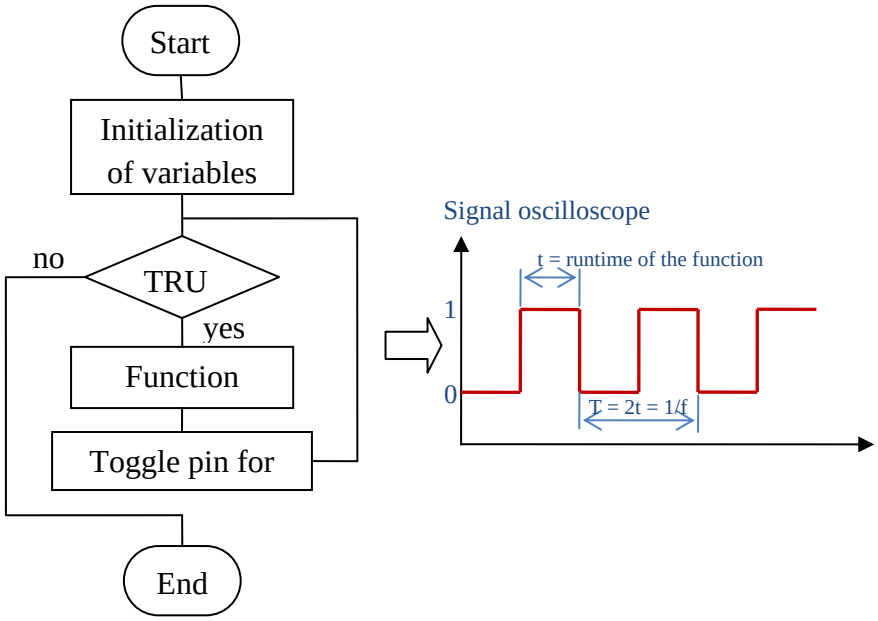


Figura 8-1: Diagrama de flujos del proceso de determinación del tiempo de ejecución de los procesos de CCMI.

Para esta prueba se usaron tres funciones importantes para la toma de una fotografía, la primera se encarga de la transmisión básica de un comando de 6 bytes, usando protocolo UART a 14400 bps, hacia la cámara C328R, la segunda función se encarga de sincronizar con la cámara, para ello envía continuamente el comando de sincronización hacia la cámara, luego de un máximo de 60 veces el programa abandona la rutina ya que no se logró dicha sincronización, la tercera función se encarga de toda la rutina de toma de una fotografía, desde que se sincroniza hasta que almacena todos los paquetes en la tarjeta de memoria SD.

La Figura 8-2 muestra el resultado de esta experiencia realizada en laboratorio, haciendo uso de un osciloscopio se analiza únicamente el proceso de sincronización.

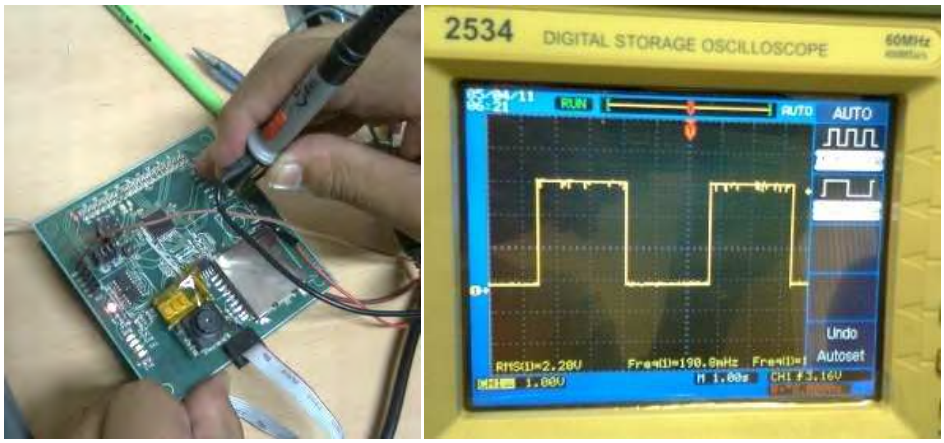


Figura 8-2: Imágenes de las pruebas realizadas en laboratorio.

a) Función de envío de comando

Estas función transmite, mediante UART a 14400 bps, 6 bytes que conforman cualquier comando desde el MCU de CCMI hasta la cámara C328R, así mismo ésta cámara también responde 6 bytes, a excepción de los paquetes de información de imagen, las cuales para nuestro caso están configuradas para que sean de 512 bytes.

- **Según cálculo**

  14400 bits en un segundo (1Hz), entonces un comando de 6 bytes (6x9 = 54 bits, se está tomando en cuenta el bit de parada) le corresponde una frecuencia de:

$$f_{enviar\ comando} = \frac{14400bps}{54b} = 266.67Hz$$

$$t_{enviar\ comando} = \frac{1}{266.67Hz} = 3.75ms$$

- **Según experiencia**

Para calcular el tiempo real de ejecución de la función es más confiable el uso de un osciloscopio donde podemos ver directamente dicho tiempo, ya que en el caso anterior no se han tomado algunos factores, por ejemplo el hecho de que en la transmisión UART, la transmisión entre byte y byte tiene un pequeño tiempo de demora que corresponden al tiempo generado luego del bit de parada. A continuación se muestra el resultado medido en laboratorio según el algoritmo mostrado en la Figura 8-1.

$$f_{medida} = 116Hz$$

La frecuencia con la que se ejecuta una función en el bucle infinito del programa es el doble de la frecuencia medida según el método de la Figura 8-1, por lo tanto:

$$f_{real\ de\ envío\ de\ comando} = 232Hz$$

$$t_{real\ de\ envío\ de\ comando} = \frac{1}{232Hz} = 4.3ms$$

Este es resultado que utilizaremos como tiempo de ejecución de la función de transmisión de comando, es lógico que demore más que el tiempo teórico calculado, ya que no se tuvieron en cuenta algunos factores que pudieron retrasar a esta función.

b) Función de envío de comando

Según la hoja técnica del fabricante de la cámara C328R, esta función consiste en enviar periódicamente el comando de sincronización, hasta llegar a un máximo de 60 veces, normalmente la cámara responde en un promedio de 25 veces, debido a esta incertidumbre del tiempo de ejecución de esta función, trabajaremos con el tiempo máximo posible, equivalente a las 60 veces, por lo que para asegurar esto desconectamos la cámara del circuito.

- **Según cálculo**

La función de sincronizar, transmite el comando de sincronización y luego espera un tiempo para que la cámara pueda procesar la información, éste tiempo no es especificado por el fabricante, por lo que al hacer pruebas se detectó que la cámara nunca sincroniza si dicho tiempo es inferior a 30ms, por lo que para evitar esto se incluyó un retardo equivalente a 40ms aproximadamente, estos 40ms representa un gran porcentaje del tiempo que el CPU dedica a la función de sincronización por lo que será empleado para hacer un cálculo aproximado del tiempo de ejecución de esta función:

$$t_{calculado\ de\ sincronización} = 60x40\ ms$$

$$t_{calculado\ de\ sincronización} = 2.4\ s$$

- **Según experiencia**

Según experiencia directa se muestran los siguientes resultados:

$$f_{medida} = 190mHz$$

$$f_{real\ de\ sincronización} = 380mHz$$

$$t_{real\ de\ sincronización} = 2.6\ s$$

Al igual que el caso anterior, el caso real es ligeramente mayor al calculado, y es el que debe ser tomado en cuenta para nuestras consideraciones de diseño.

c) Función de toma de fotografía

Para este caso no es muy útil hacer uso de un osciloscopio ya que el tiempo no requiere de tanta precisión, además es muy variable ya que depende de la cantidad de veces que intenta sincronizar con la cámara, además depende de la imagen tomada ya que la cámara envía su información en formato JPEG, la cual será más pesada si es que comprimió menos, es decir si se fotografía una imagen con muchos contrastes de colores el proceso interno de compresión será malo, por lo que la cámara tendrá más paquetes de información.

El tiempo máximo aproximado de la toma de una fotografía, según las experiencias obtenidas en las distintas pruebas está entre 10 y 25 segundos, por lo que fijaremos el valor de 30 segundos para tener cierto grado de seguridad para hacer nuestro diseño, esta aproximación no será problema debido a experiencias anteriores cuyas fotografías no muestran muchos contrastes de colores y la compresión es buena por lo que las fotografías son de bajo peso.

d) Función de lectura de un paquete de 512 bytes mediante I2C

Esta función se realiza periódicamente y es empleada para guardar la información de los módulos PCT y SDCA, éstos indicarán a CCMI cuando sus buffers de información estén llenos y puedan ser leídos.

- **Según cálculo**

El tiempo de ejecución de esta función será calculado teniendo en cuenta la frecuencia del clock de la señal, para ello se debe tener en cuenta que CCMI controla dicha señal de clock, según el manual del fabricante del uC de CCMI, la máxima velocidad de transmisión es de 100kbps, elegiremos una velocidad de trabajo de 80kbps, y para calcular el tiempo de ejecución debemos tener en cuenta que cada byte transmitido está acompañado de un bit de start y un bit de stop, haciendo un total de 10bits, entonces en total un paquete de 512 bytes tendrá 5120 bits, considerando los 10 bits para la dirección serán entonces 1530 bits. Con un simple regla de tres podemos aproximar el tiempo en el que se transmitirá este paquete:

$$t_{transmisión\ 512\ bytes} = \frac{1530bits}{80kbps} = 19.12ms$$

- **Según experiencia**

Para este caso debemos hacer una modificación al algoritmo mostrado en la Figura 8-1, ya que nuestra función de lectura de 512 datos se hace mediante interrupciones, por lo que para obtener el tiempo máximo que el CPU dedicará a esta función, teniendo en cuenta que

atenderá únicamente a esta función, se esperará a que sea leído el último byte para luego recién hacer el cambio de estado del pin y leer su frecuencia en el osciloscopio, el algoritmo tal como muestra la Figura 8-3.

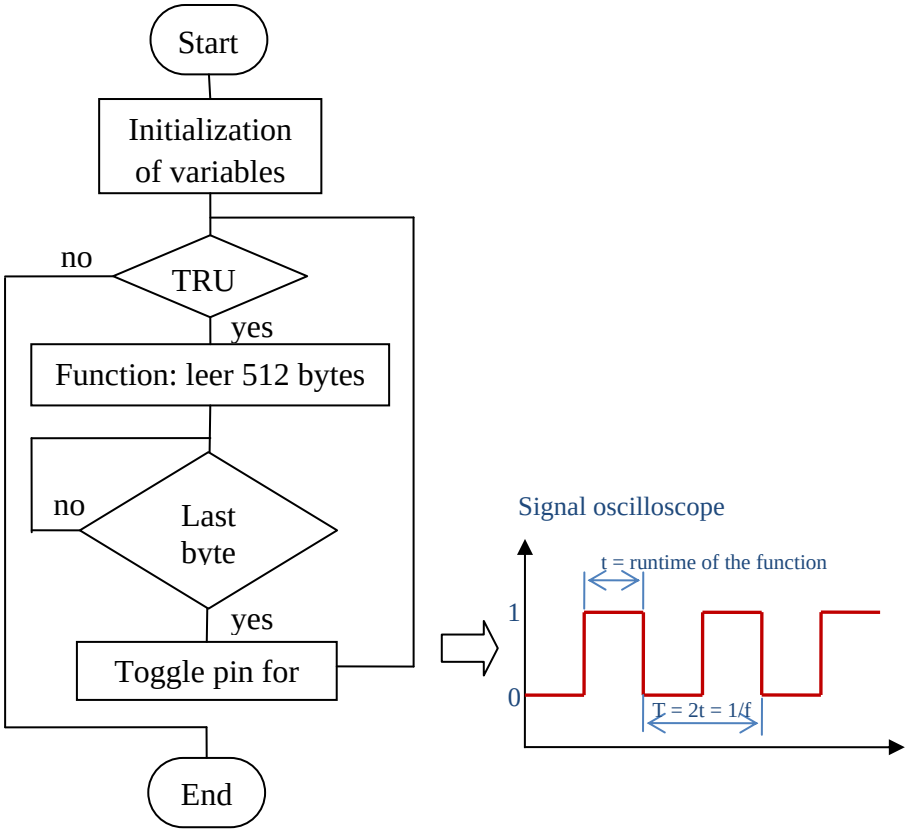


Figura 8-3: Algoritmo programado para medir el tiempo de ejecución de una función haciendo uso de un osciloscopio, para el caso en que la función se ejecuta por interrupción.

El resultado de esta medida se muestra a continuación:

$$f_{medida} = 12.97Hz$$

$$f_{real\ de\ sincronización} = 25.94Hz$$

$$t_{real\ de\ sincronización} = 38.55ms$$

A continuación se resume el resultado de esta experiencia en la Tabla 8-1, notamos que la función que demora más es el proceso de toma de fotografía, además teniendo en cuenta que el tiempo de toma de foto incluye el tiempo de sincronización, entonces los demás procesos representan un tiempo despreciable respecto al tiempo de toma de fotografía. Estas mediciones fueron usadas para diseñar el RTOS en la Subsección 6.6.2 y sirven para probar que nuestro RTOS es *planificable*.

Tabla 8-1: Resumen de tiempos de ejecución de las principales funciones de CCMI.

	Tiempo de ejecución
Envío de comando	4.3ms
Sincronización	2.5s
Toma de fotografía	30s
Lectura de un paquete de 512 bytes mediante I2C	38.55ms

8.1.2 Prueba con globo cautivo

Para esta prueba el prototipo Chasqui fue sujetado como carga a un globo con helio el cual llevó al Chasqui a más de 200m de altura, deteniéndose cada 5m para tomar datos.

Esta parte de la tesis muestra el estado del funcionamiento del módulo de CCMI y su interacción con los módulos de Potencia y control térmico (PCT), Sistema de adquisición de imágenes (SIMA) y Sistema de comunicaciones (SICOM) se mostrará las características de las comunicaciones implementadas haciendo uso de protocolos como UART, SPI, I2C, etc.

Los objetivos de esta prueba es que CCMI tome de fotografías y almacene en la tarjeta SD y además transmitir el estado del proceso a SICOM, quien a su vez debe transmitirlas a una radio receptora. Además hacer un análisis del funcionamiento y de

los errores obtenidos en la prueba. La siguiente figura muestra dos de las 301 fotografías correctamente tomadas.

(a) (b)

Figura 8-4: Fotografías transmitidas por el Chasqui en la prueba de globo cautivo en rango (a) visible y (b) Infrarroja.

- **Características de comunicación entre los módulos CCMI y PCT**

El módulo PCT entregó la alimentación necesaria de 3.3V para el módulo CCMI, SIMA y SICOM usando estos tres módulos la salida de un mismo regulador de voltaje, en el nuevo diseño de la tarjeta electrónica del módulo de SIMA se está contemplando su alimentación del regulador de SDCA, debido a que se espera que el módulo de comunicaciones consuma gran cantidad de potencia que puede saturar al regulador.

Para la presente prueba no se implementó la transmisión de las lecturas de los sensores del módulo PCT debido a incompatibilidad de niveles de voltaje para la comunicación mediante I2C, debido a que al tratarse de un protocolo full-duplex se debe usar un circuito integrado especial el cual quedó descartado por inestabilidad de funcionamiento. Actualmente el módulo PCT está haciendo un nuevo diseño usando un microcontrolador de *Freescale* MC9S08QE128

- **Características de comunicación entre los módulos CCMI ySIMA**

Con el módulo de cámaras la comunicación se da a través del protocolo UART a una velocidad por defecto de 14400 baud, dicho módulo cuenta con dos cámaras, visible e infrarroja, las cuales son seleccionadas usando un multiplexor hacer la toma de una de las dos cámaras de tal forma que las tomas conmuten de cámara, en caso de fallar la sincronización con alguna de ellas volverá a intentar pero con la otra.

A continuación se detalla el control para la selección de cámara.

Tabla 8-2: Bit select *y* enable *para habilitar cámaras*

Select(bit)	***Enable(bit)***	**Visible**	**Cámara**
x	1	0	0
1	0	Visible	0
0	0	0	Infrarroja

- **Características de comunicación entre los módulos CCMI y SICOM**

El módulo CCMI registra en todo momento el número de fotos y las fallas que se pueden haber presentado en el proceso de toma de fotografía y almacenado en la tarjeta de memoria SD, para tener conocimiento de estos datos, se envían dos bytes mediante protocolo UART, a 9600baud, al módulo SICOM quién a su vez se encarga de transmitirla mediante señales de FM a una radio receptora. Para la prueba en el observatorio de Jicamarca se espera el módulo SICOM implemente el protocolo de transmisión inalámbrica AX25 el cual nos permitirá descargar las imágenes cuando el Chasqui este aún en vuelo, además de mantener todas las imágenes en la tarjeta de memoria SD. A continuación se muestra en la Figura 8-5 el esquema global del sistema.

Figura 8-5: Esquema global del sistema en la prueba de Globo cautivo.

Resultados obtenidos en la prueba

Durante todo el tiempo que duró la prueba, el módulo CCMI estuvo capturando imágenes y almacenándolas en la tarjeta de memoria SD, además luego de cada toma se envió por UART al módulo SICOM el valor del contador de fotos y el estado de posibles fallas detectadas en la cámara y en la tarjeta SD. El módulo SICOM transmite estos datos de forma inalámbrica, usando codificación *morse*, a una radio receptora cuya señal se pudo apreciar desde un software que decodifica dicha señal en datos ASCII.

Esta transmisión del estado del módulo CCMI, no implementada en la prueba realizada en el observatorio del IGP, permitió monitorear su buen estado durante la prueba, ya que de producirse alguna falla, esta podría detectarse en tierra usando la radio receptora.

- **Resultados de la prueba para el módulo CCMI**

Tabla 8-3: Resultados de la prueba para el módulo CCMI

Identificador	Significado	Resultados de la prueba
N	Ningún error detectado	**306**
S	Falló sincronización con cámara	**1**
I	Falló inicialización con cámara	**0**
T	Falló configuración del tamaño de paquete	**0**
M	Fallo configuración del modo SNAPSHOT	**0**
C	Falló captura de imagen	**0**
F	Paquete recibido incompleto o fallado	**2**
W	Falló escritura en tarjeta SD	**0**

- **Principales errores detectados en el módulo CCMI:**

Se detectaron dos fotos que contienen tramas incompleta, para solucionar este problema debe programarse en el microcontrolador de CCMI alguna forma de detección de errores, en el caso de la cámara C328R podemos comprobar el *checksum* de cada paquete de imagen y en caso de no coincidir con el que se recibe en la trama, debería de volver a pedir dicho paquete errado. A continuación, en la Figura 8-6, se muestra el resultado de una fotografía con pérdida de datos en alguna de las tramas enviadas.

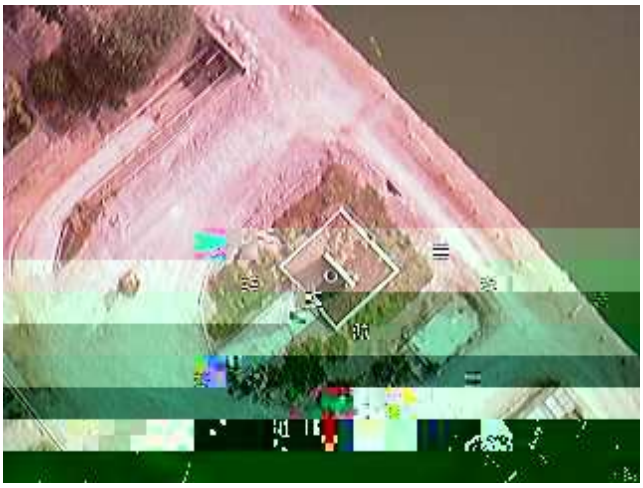


Figura 8-6: Falla detectada por trama de imagen incompleta

La falla debido a sincronización con cámara no impide que se sigan tomando otras fotos, ya que tal como está el programa el microcontrolador del módulo CCMI, en cuando se presente esta falla primeramente enviará a SICOM el motivo del mismo y luego volverá a intentar la sincronización pero con la segunda cámara y luego continuará conmutando continuamente para la toma de fotografías.

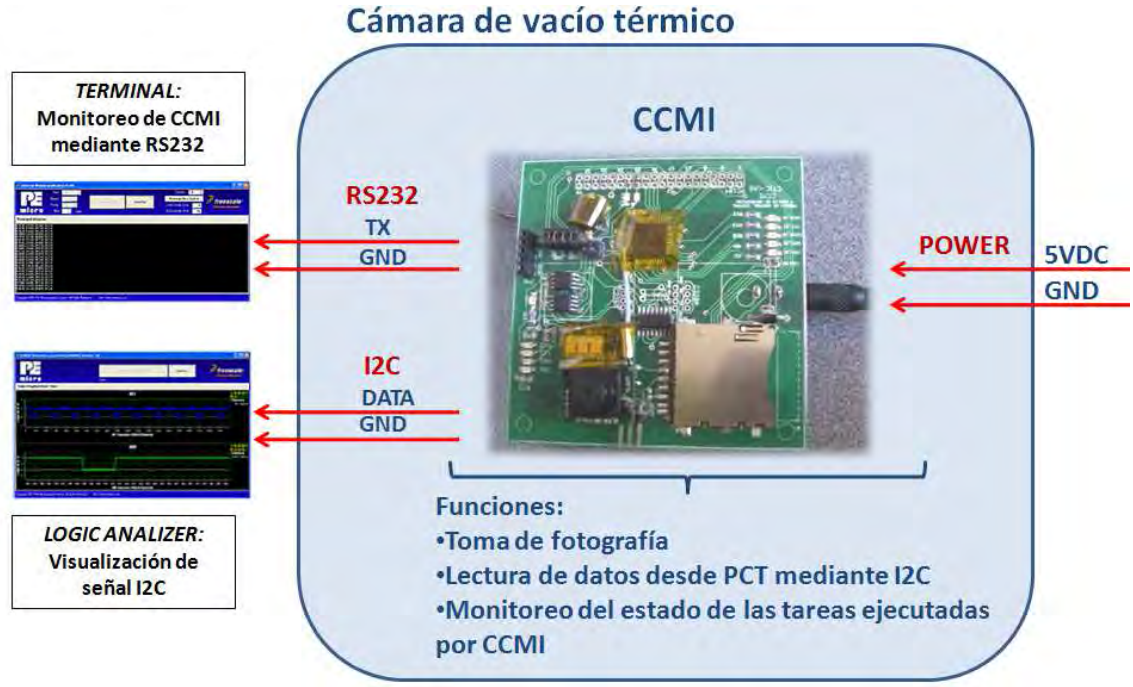
8.2 Pruebas de rendimiento del hardware de CCMI

Ya hemos visto el correcto funcionamiento de CCMI a condiciones normales. Ahora debemos comprobar que la toda la tarjeta CCMI pueda funcionar correctamente cuando sea sometido a pruebas del módulo MIP. Existen tres pruebas principales realizadas al hardware del Chasqui, éstas son: vibración mecánica, radiación y vacío térmico, para la primera es necesario tener todas las tarjetas integradas pues son pruebas globales de resistencia de la estructura, en caso de detectarse error en esta prueba deberá hacerse modificaciones a la integración mas no al diseño del software o hardware de CCMI, por tal motivo no será tomado en cuenta en esta tesis. Para la segunda prueba es necesario contar con un laboratorio que permita irradiar al chasqui integrado completamente, en este caso se puede solucionar mejorando la estructura o mejorando el software de cada tarjeta para que pueda ser capaz de solucionar posibles errores generados por la radiación, debido a que no se cuenta en la universidad con un laboratorio adecuado para irradiar al Chasqui, ésta prueba tampoco será tomada en cuenta en esta tesis. La tercera prueba es considerada la de mayor importancia por el módulo MIP y será realizada gracias a una cámara de vacío térmico diseñada e implementada por dicho módulo.

8.2.1 Prueba de vacío térmico

La prueba de vacío térmico de la tarjeta CCMI fue realizada en conjunto con el módulo MIP, y en coordinación con la de los demás módulos, PCT SICOM y SDCA, el tiempo aproximado de cada prueba fue de 2 horas, para esta prueba es necesario indicar el número de pines necesarios para monitorear el funcionamiento de cada módulo, en el caso de CCMI se utilizaron en total 5 pines, dos para la alimentación, dos para la transmisión serial y uno para el monitoreo de la señal de transmisión de datos mediante I2C, son estos 5 pines las únicas salidas de la cámara de vacío térmico, ver Figura 8-7 (a). La función de la tarjeta de CCMI en esta prueba fue la de tomar fotografías y leer el buffer de 512 bytes de la tarjeta PCT mediante el protocolo I2C, esto se realiza cada vez que dicho módulo le indique a CCMI que su buffer está listo para ser leído. Se debe aclarar que el módulo PCT fue usado únicamente para probar comunicación I2C, es decir no se usaron sus reguladores debido a que la fuente de alimentación fue desde una fuente externa a la cámara de vacío, tampoco se incluyeron en la tarjeta PCT sus sensores de temperatura, corriente ni voltaje. La Figura 8-7 (b) muestra el sistema de la prueba de vacío térmico de CCMI.

(a)



(b)

Figura 8-7: (a) cámara de vacío térmico. (b) Diagrama de bloques de la prueba realizada al hardware CCMI.

Objetivo de la prueba

- Probar el funcionamiento de los periféricos internos de la tarjeta CCMI hasta -20°C y a una presión de 0.02mbar.
- Probar la desincronización del bus-clock generado desde el oscilador RC interno al uC a una temperatura de -16°C.

Procedimiento realizado

1. Primero se debe asegurar que las tarjetas CCMI y PCT estén listas para ser probadas, es decir que previamente se haya probado el funcionamiento y que se tengan los cables conectados a los pines necesarios para la prueba. Estos cables salen por unos conectores especiales ubicados en la estructura de la cámara de vacío térmico.

2. Luego se debe aislar todo aquello que pueda generar un corto circuito cuando la tarjeta CCMI haga contacto con la base de acero inoxidable. Para evitar esto se utilizó cinta de Kapton como aislante por recomendación del módulo PCT.

3. Se ingresa el módulo CCMI y PCT a la cámara de vacío y se hace las conexiones necesarias, como se mencionó estos consisten de 5 cables, para la alimentación y visualización de datos serial e I2C, este procedimiento se debe hacer con supervisión y apoyo del módulo MIP, quienes luego de este proceso se aseguraron de que las condiciones para iniciar la prueba estén dadas.

4. Antes de iniciar a variar las condiciones de presión y temperatura interna, se hace una prueba de funcionamiento, para descartar algún problema ocurrido al momento de hacer la instalación de nuestro dispositivo en la cámara de vacío térmico. Lo que sigue es labor del módulo MIP, quienes primero se encargan de disminuir la presión para generar el vacío necesario para luego disminuir la temperatura gradualmente.

5. Se debe registrar toda la información transmitida por el módulo CCMI, en este caso vía RS232. Dicha información consiste de un reporte que indica el estado actual de operación, los cuales son principalmente la acción de tomar fotografía, donde también se reporta el estado las funciones más importantes del proceso de la toma de fotografía. Esto es para verificar el comportamiento de la cámara.

6. Una vez que se ha disminuido hasta la temperatura de -10°C, indicada por el módulo de control térmico, se continuó disminuyendo para determinar hasta qué temperatura funciona correctamente, llegándose al límite inferior de la cámara de vacío térmico (-20°C).

7. Lo que sigue es volver nuevamente a las condiciones de presión y vacío iniciales, para esto se apaga el circuito de pruebas, tanto CCMI como PCT, para evitar que la posible formación de condensado al interior de la cámara pueda generar un corto circuito.

8. Se debe esperar la indicación del módulo MIP para poder sacar la tarjeta y revisar que se encuentre en buen estado, desconectamos todos los cables.

9. La tarjeta al continuar en temperaturas bajas, formará condensado hasta superar los 0°C, luego se debe secar con cuidado la tarjeta.

10. Finalmente se debe guardar y analizar los datos registrados de la prueba.

Resultados de esta prueba:

- La tarjeta CCMI funcionó correctamente durante toda la prueba, en la cual se llegó hasta -16°C, se probó toma de fotografía y comunicación I2C con el uC de la tarjeta PCT, la cual estaba conectada con el módulo CCMI por el bus PC104.

- La frecuencia de oscilación del *bus-clock*, generada por el oscilador interno tipo RC, no se notó afectada ya que en todo momento la PC estuve leyendo correctamente las tramas por RS232.

- Se tomaron 160 fotos durante la prueba, todas completamente oscuras.

- Se almacenaron 82 paquetes de 512 bytes de PCT, no se perdió ningún paquete.

A continuación se muestra, en la Figura 8-8, la primera parte de la información enviada por el MCU de CCMI usando RS232, revisando este archivo pudimos visualizar en tiempo real el comportamiento correcto del módulo CCMI.

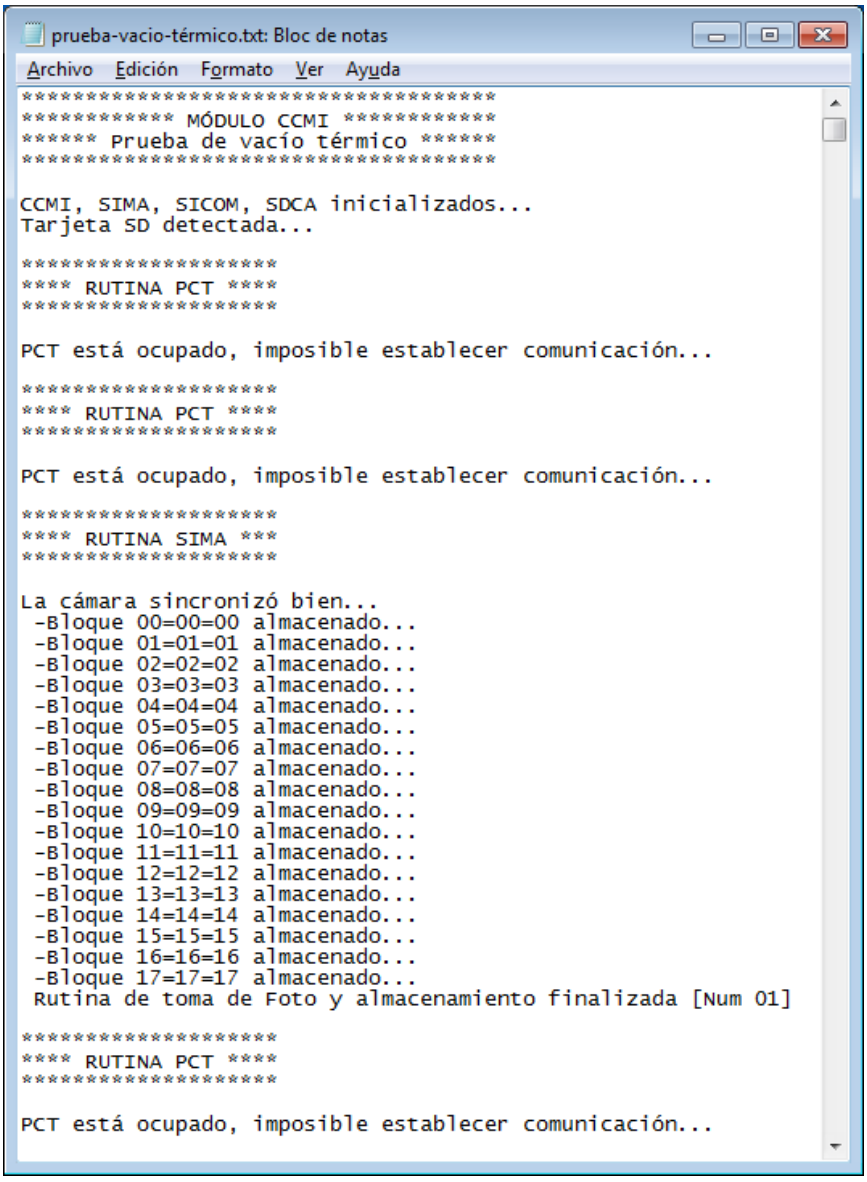


Figura 8-8: Reporte de los resultados registrados por CCMI durante todo el proceso de funcionamiento en la prueba de vacío térmico.

8.2.2 Prueba de consumo de energía

Para esta prueba se midió el voltaje y corriente a la salida del regulador de voltaje de 3.3V que alimenta toda la tarjeta CCMI, incluyendo las dos cámaras, los procesos realizados corresponden a una toma de foto de aproximadamente 30 segundos, cada 4 minutos durante 15 minutos, en todo este tiempo se ejecutan los demás procesos de CCMI. En la Figura 8-9 se muestra la respuesta del regulador que prácticamente se mantiene en 3.3V con una pequeña caída cada vez que se activa las cámaras para la

toma de fotografía, dicha caída no afecta el funcionamiento de nuestros dispositivos, los cuales en general funcionan hasta los 3V. En la Figura 8-10 podemos visualizar el consumo de corriente de CCMI y vemos que los puntos más altos se dan cuando se activa las cámaras para la toma de fotografía, llegando a consumir hasta 85mA, en el tiempo de la prueba el consumo promedio fue de 50.4mW, este valor está por debajo de lo impuesto por el módulo PCT desde un inicio, ver 2.3.1.1, En realidad el consumo promedio será mucho menor ya que el nanosatélite Chasqui tomará solo dos fotos al día ya que será imposible descargar más a la estación terrena, por ello la mayor parte del tiempo CCMI tendrá un consumo de 15mA.

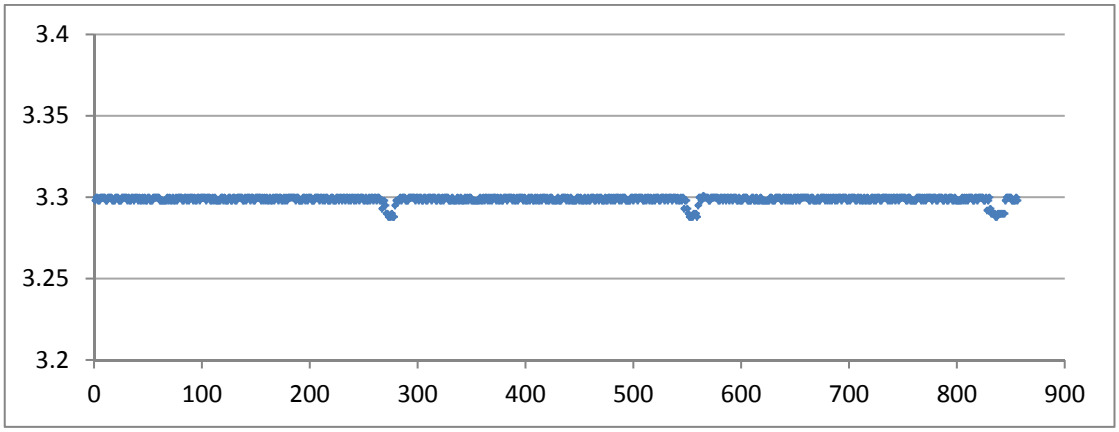


Figura 8-9: Consumo de potencia del módulo CCMI. Tiempo (s)-Voltaje (V).

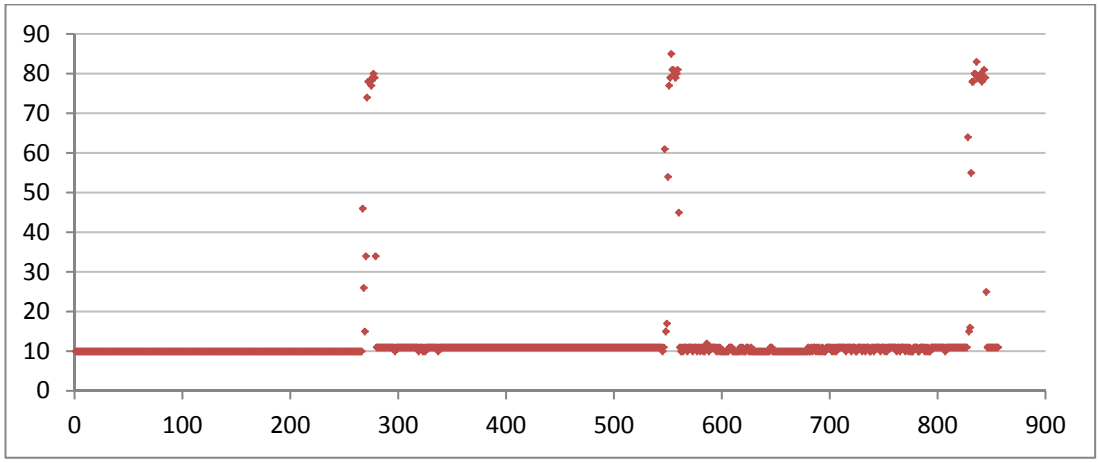


Figura 8-10: Consumo de potencia CCMI. Tiempo (s)-Corriente (mA).

Conclusiones

Un sistema embebido, incluye hardware y software, fue diseñado e implementado para solucionar el problema de la necesidad de un módulo de control central y monitoreo de información del nanosatélite Chasqui-I. La tarjeta ha sido sometida a importantes pruebas de funcionamiento, entre ellas la toma y almacenamiento continuo de fotografías, transmisión de datos entre módulos, prueba de vacío térmico y globo cautivo, en todos los casos el resultado ha sido satisfactorio. El módulo CCMI funciona correctamente en los laboratorios de test de la UNI, por lo que la tarjeta está lista para someterse a pruebas de certificación internacional para dispositivos espaciales para luego ser puesto en operación en el espacio exterior. Pese a estas consideraciones siempre es posible la existencia de problemas que no se tomaron en cuenta o no se probaron por falta de laboratorios especializados de test en la UNI, por ello en el futuro es importante someter la tarjeta a intensas pruebas, esto es esencial para cualquier dispositivo de aplicación satelital.

A continuación se muestra detalladamente las conclusiones referentes a cada objetivo específico mencionado en el capítulo 1 de esta tesis.

- Se determinaron los principales parámetros de diseño, como son la temperatura de operación, características dimensionales de la tarjeta y los agujeros y conectores comunes a todas las tarjetas del Chasqui, nivel de protección del sistema y de la información importante ante disminución o corte de energía eléctrica (modo emergencia), inclusión del *Watchdog* y *TimeOuts* para evitar

bucles infinitos indeseados, distribución de tareas asignadas al MCU previa determinación de tiempos de ejecución de sus funciones.

- Para la comunicación entre el módulo CCMI y los otros módulos se emplearon protocolos de comunicación tipo I2C y UART, ambos a velocidades bajas para reducir el consumo eléctrico, a su vez se diseñaron los protocolos de nivel alto con los comandos necesarios para la comunicación entre módulos.
- Se determinó e implementó los modos de operación del nanosatélite y la secuencia en la que deben ejecutarse estos modos, esto se implementa en el software de CCMI.
- Se seleccionó los componentes electrónicos capaces de trabajar bien bajo condiciones de vacío térmico, componentes de aplicación industrial o militar, el uso de dispositivos tipo SMD para reducir el consumo eléctrico,
- La tarjeta CCMI terminada fue sometida a las pruebas de vacío térmico y globo cautivo, en ambas el resultado fue satisfactorio. En todo momento hubo un trabajo conjunto con los demás módulos y esto permitió una fácil integración de todo el sistema.

Recomendaciones

La tarjeta CCMI ha sido sometida a importantes pruebas pero no son suficientes, es necesario probar muchas veces la tarjeta en laboratorios especializados en test de dispositivos espaciales, la prueba más importante que no se realizó a la tarjeta CCMI es la de radiación, esto debido a limitaciones de equipos necesarios para su realización. Otra prueba que se debe hacer es la de dejar el sistema integrado funcionando por varias semanas, ya que para efectos prácticos el tiempo de ejecución del software fue escalado para no tener que esperar los tiempos reales de funcionamiento.

No se ha optimizado el consumo de energía de la tarjeta CCMI, esto debido a que su consumo es muy poco comparado con los módulos SICOM y SDCA, pero un correcto diseño debe ahorrar todo lo posible la energía. Para esto debe modificarse principalmente el software haciendo uso de los modos de bajo consumo presentes en el MCU.

En todo proyecto que debe integra varias partes, debe tenerse en cuenta que muchas de estas pueden fallar debido a que están en proceso de desarrollo. Por eso es importante fijar bien los puntos comunes a todos los módulos, especialmente los pines que comunican entre ellos. Este problema se presentó varias veces debido principalmente a que las señales que comunican todas las tarjetas, a través del bus PC104, no eran las que se habían definido en un inicio, estos errores implicaron la

fabricación de nuevas tarjeta de remplazo que afectan económicamente y retrasan al proyecto.

Si bien el módulo CCMI puede tener muchas mejoras, especialmente en el software ya que es conocido que los niveles de radiación espacial son capaces de afectar las memorias y ello provoca que el software programado no funcione según lo indicado, el trabajo mostrado en esta tesis puede servir de base para otros módulos de control central y monitoreo de información de cualquier nanosatélite independiente de sus dimensiones.

Referencias

[1] Gustavo Galeano, *Programación de sistemas embebidos en C*. Colombia: Alfaomega, 2009.

[2] Sitio Web Stándar Secure Digital. [En línea]. www.sdcard.org/home/

[3] "Pruebas de vacío térmico y vibraciones mecánicas del nanosatélite Chasqui", CTIC-UNI, Lima, Informe final 2012.

[4] (2011, Diciembre) Web oficial del proyecto Chasqui. [En línea]. http://www.chasqui.uni.edu.pe/

[5] (2011) Sitio Web de la Universidad Nacional de Ingeniería. [En línea]. www.uni.edu.pe

[6] (2011, diciembre) Web oficial Cubesat. [En línea]. http://www.cubesat.org/

[7] (2011) Web del Centro de tecnologías en información y comunicaciones. [En línea]. http://www.ctic.uni.edu.pe/

[8] (2011, Diciembre) Web del Instituto Nacional de Telecomunicaciones. [En línea]. http://www.inictel-uni.edu.pe/

[9] (2011, December) Standford University. [En línea]. http://www.stanford.edu/

[10] (2011) Web ESA. [En línea]. http://www.esa.int/esaCP/index.html

[11] "Informe final del módulo potencia y control térmico", CTIC-UNI, Informe Final 2012.

[12] (2011) Cansat Competition. [En línea]. http://www.cansatcompetition.com/Main.html

[13] (2011, Diciembre) Web oficial Cubesat-Kit. [En línea]. http://www.cubesatkit.com/

[14] (2010) Web oficial del proyecto compass-1. [En línea]. http://www.raumfahrt.fh-aachen.de/compass-1/home.htm

[15] (2011, Diciembre) Proyecto Kysat. [En línea]. http://ssl.engineering.uky.edu/missions/orbital/kysat1/

[16] (2011) Web oficial proyecto DTUsat. [En línea]. http://dtusat.dtu.dk/

[17] (2008) Blog del proyecto UNAMSAT. [En línea]. http://unamsat.blogspot.com/

[18] (2010) Proyecto cubesat Libertad-1. [En línea]. http://www.usergioarboleda.edu.co/proyecto_espacial/index.htm

[19] Web proyecto UAPSat. [En línea]. http://www.uapsat.info/

[20] Instituto de radioastronomía de la PUCP. [En línea]. http://inras.pucp.edu.pe/index-en.html

[21] "Informe final del módulo de control central y manejo de información del Chasqui-1", CTIC-UNI, Lima, Informe Final 2012.

[22] "Informe final del módulo de sistema de control y determinación de actitud SDCA-", CTIC-UNI, Lima, Informe Final 2012.

[23] "Informe final del módulo de comunicaciones del nanosatélite Chasqui-1", CTIC-UNI, Lima, Informe Final 2012.

[24] "Informe final del módulo de estación terrena", CTIC-UNI, Lima, Informe Final 2012.

[25] (2011, diciembre) Nand Flash de Samsung. [En línea]. http://www.samsung.com/global/business/semiconductor/products/flash/Products_NANDFlash.html

[26] (2011) MAXIM innovation delivered. [En línea]. http://www.maxim-ic.com

[27] (2011) Web I2C Philips. [En línea]. http://www.i2c-bus.org/

[28] "MC9S08QE128 Datasheet, HCS08 Microcontrollers, Rev. 2", Freescale Semiconductor,.

[29] Fabio Pereira, *Designer's Guide to the HCS08 Microcontrollers, Booksurge*. Brasil: Booksurge, 2008.

[30] (2011) sitio web de custom computer services CCS. [En línea]. http://www.ccsinfo.com/

[31] (2011) sitio web PEmicro for freescale micros. [En línea]. http://www.pemicro.com/

[32] Jordi Mayné, "Nuevas tendencias de los microcontroladores", SILICA an avnet division,.

[33] "C328R User Manual", Junio, 2007.

[34] (2010) Sitio web del software EAGLE. [En línea]. http://www.cadsoft.de/

[35] (2010) Sitio web EAGLE3D. [En línea]. http://www.matwei.de

[36] (2008) Sitio web POV-Ray. [En línea]. http://www.povray.org/

[37] center, NASA Goddard space flight. (2005, April) General environmental verification standard (GEVS) for GSFC Flight programs and projects. [En línea]. http://standards.gsfc.nasa.gov/gsfc-std/gsfc-std-7000.pdf

[38] The CubeSat Program, Cal Poly SLO. (2009, August) CubeSat Design Specification Rev. 12. [En línea]. http://www.cubesat.org/images/developers/cds_rev12.pdf

[39] (2007) Sitio web del virtual términal LookRS232. [En línea]. http://www.lookrs232.com/

[40] (2011, Junio) Sitio web ICY Hexplorer. [En línea]. http://artemis.wszib.edu.pl

[41] Sitio Web X-Ways software technology AG. [En línea]. http://www.winhex.com/winhex

[42] Herbert. Schildt, *Turbo C/C++: Manual de referencia*. México: Mc Graw ill, 1992.

[43] Artur Scholz, "Design and development of a CDHS for a pico satellite", Fachhochschule Aachen, Diploma Tesis Septiembre 2004.

[44] Freescale semiconductor, "DRM104: SD Card reader using the M9S08JM60 Series", Tutorial HCS08.

[45] David Stracker, *C-Style, Standards and guidelines*.: Prentice Hall, ISBN 0-13-116898-3.

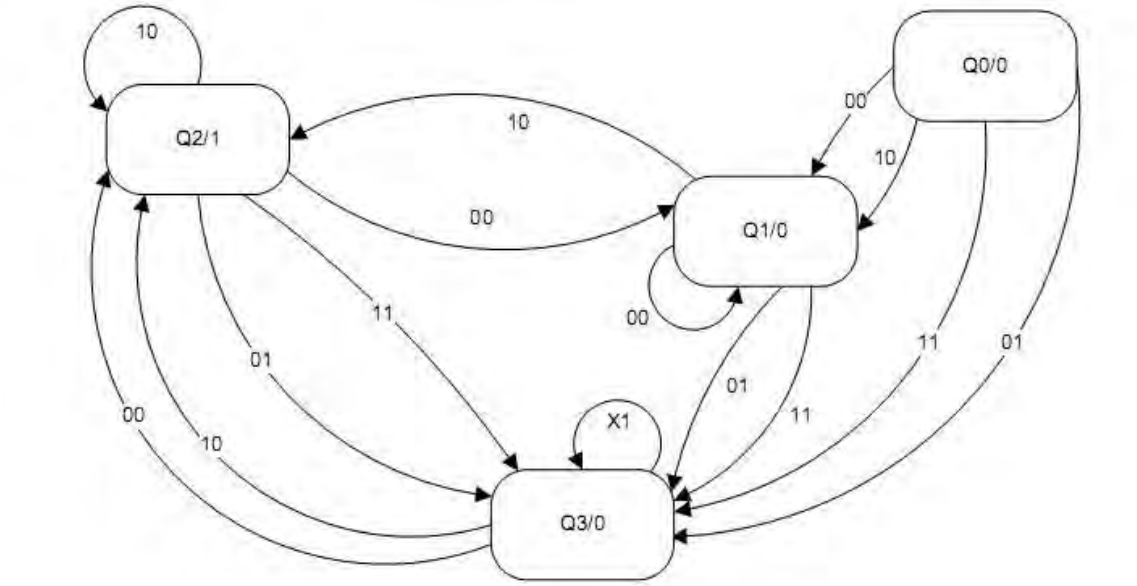
Anexos

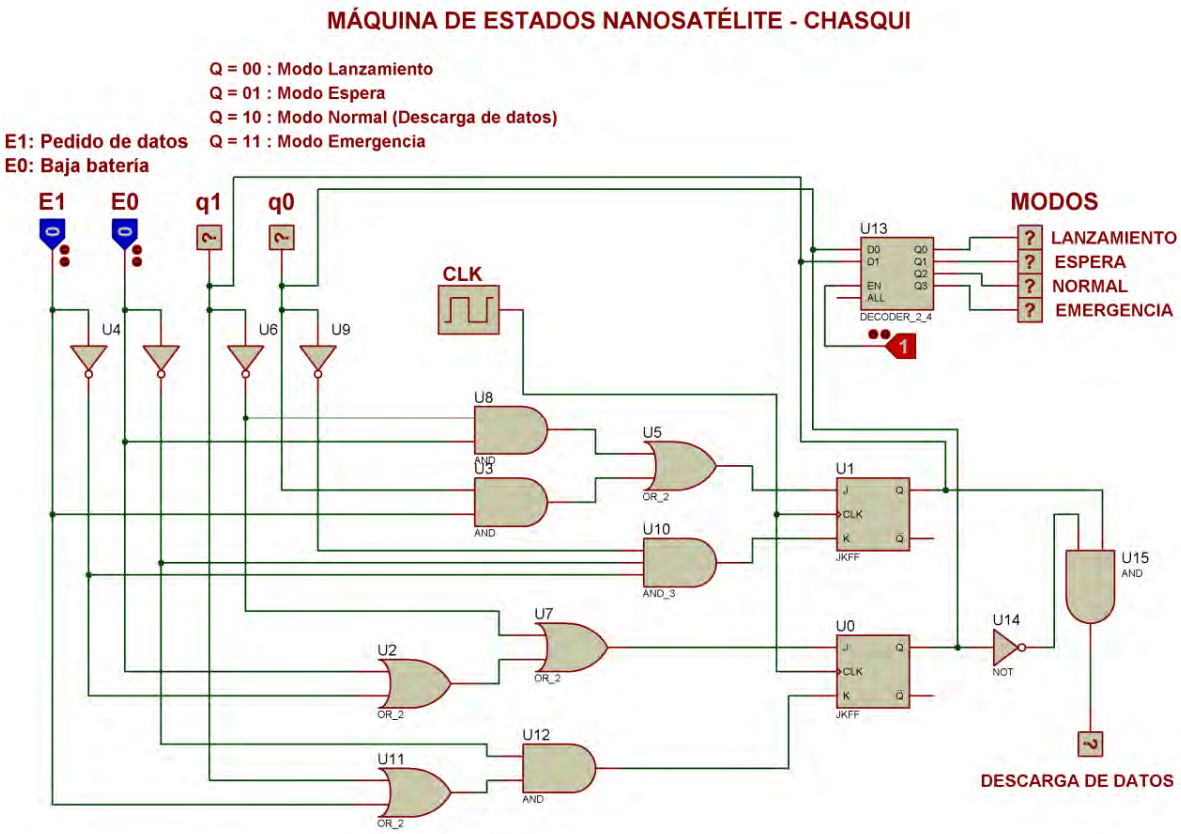
A1. Modos de operación como máquina de estados

Estados:
Q0: Modo lanzamiento
Q1: Modo espera
Q2: Modo normal
Q3: Modo emergencia

Entradas:
E0: Baja batería
E1: Pedido de datos

Salida:
S: Descarga de datos





A2. Código C de la plantilla CCMI

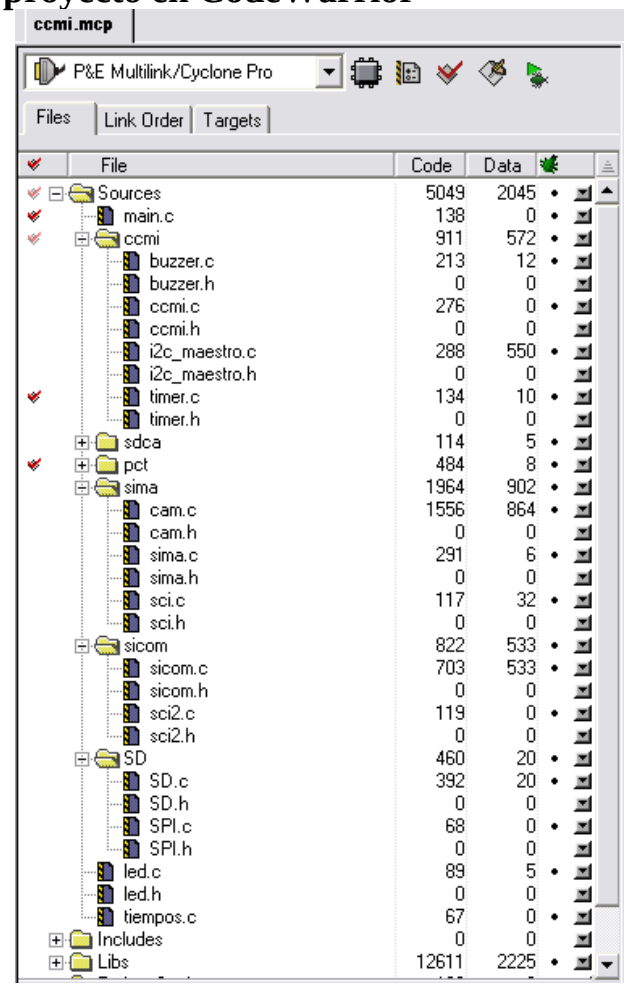
```
/*************************************************************
Proyecto: PICOSATÉLITE DE INVESTIGACIÓN CHASQUI
Módulo  : Control Central y Manejo de Información - CCMI
Tesista : Elvis Omar Jara Alegria
Fecha   : 01 Noviembre 2011
*************************************************************/

#include "includes.h"

void main(void) {
    CCMI_Init();
    PCT_Init();
    SDCA_Init();
    SIMA_Init();
    SICOM_Init();

    EnableInterrupts;

    for( ; ; ) {
        PCT_Run();
        SDCA_Run();
        SIMA_Run();
        SICOM_Run();
    }
}
```

A3. Plantilla interna a CCMI para la tarea PCT

```
// pct.c
#include "includes.h"

#define DIR_SLAVE_PCT   0x05

#define DUMMY_PCT       0
#define ESPERA_PCT      1
#define ACTIVO_PCT      2
#define ERROR_PCT       3

#define CMD_1           10    //número de datos

char    std_PCT = DUMMY_PCT;   // Inicia dormido
extern  unsigned long int BASE_TIEMPO;
extern  char flag_i2c_OK;
extern  char buffer_i2c[30];
extern  char buffer_i2c_read[512];
extern  volatile char flag_end_i2c;
static  uint cuenta_SD = 0;

void PCT_Init(void){
  _PIN_ENTKPHO = PIN_IN;
  _PIN_EMGMODE = PIN_IN;
  _PIN_ACKCCMI = PIN_OUT;
}
void PCT_Run(void){
  static unsigned long int base_tiempo_PCT=0;
  switch(std_PCT){

    case DUMMY_PCT:
      base_tiempo_PCT = BASE_TIEMPO;
      std_PCT = ESPERA_PCT;

    break;

    case ESPERA_PCT:
      if(BASE_TIEMPO - base_tiempo_PCT > PERIODO_PCT){
      std_PCT = ACTIVO_PCT;
      }
    break;

    case ACTIVO_PCT:
      Main_PCT();
      std_PCT = DUMMY_PCT;
    break;

    case ERROR_PCT:
      std_PCT = DUMMY_PCT;
    break;

    default:
    break;
  }
}
```

```
void Main_PCT(void){
  static uchar cnt_paquete = 0;

  #ifdef ENABLE_DEPURATION
  LED_PCT = ~LED_PCT;
  Buzzer_Beep(NOTA_SOL,TONO_SOL,1);
  Buzzer_Beep(NOTA_SI,TONO_SI,1);
  #endif

  #ifdef ENABLE_SHOW_STATE
    sciPrintf2("\n\r********************\n\r");
    sciPrintf2("**** RUTINA PCT ****\n\r");
    sciPrintf2("********************\n\r\n\r");
  #endif

  PIN_ENTKPHO = 0;
  if(PIN_ENTKPHO == 1){
    #ifdef ENABLE_SHOW_STATE
      sciPrintf2("--> Leyendo a PCT...\n\r ");
    #endif
    i2c_read(0x00,512);
    flag_end_i2c = 0;
    while(flag_end_i2c == 0);

    msDelay(10);
    SD_Write_Block(MEMO_SD_PCT + cuenta_SD, buffer_i2c_read); // offset +135
    cuenta_SD++;
    #ifdef ENABLE_SHOW_STATE
      if(PIN_ENTKPHO == 0) {
        sciPrintf2("data [Num ");
        cnt_paquete ++;
        Write2((cnt_paquete/10) + 48);
        Write2((cnt_paquete%10) + 48);

        sciPrintf2("] PCT almacenada en memoria...\n\r ");
      }
      else sciPrintf2("falló comunicación I2C...\n\r ");
    #endif
  } else{
    #ifdef ENABLE_SHOW_STATE
      sciPrintf2("PCT está ocupado, imposible establecer comunicación...\n\r ");
    #endif
  }
}
```

A4. Árbol del proyecto en CodeWarrior



A5. Implementación en C del *Timeout*

```
// timer.c

#include "timer.h"

volatile  char  flag_timer=0;
unsigned  long  int BASE_TIEMPO = 0;
static    char  std_time_out=0;

void TPM4_Init(uchar tim){
  TPM2SC = (byte)0x00;
  TPM2MOD = (word)0x00;
  (void)TPM2SC;
  TPM2SC = (byte)tim;
}

void Timer_Stop(void){
  SCGC1_TPM2 = 0;
}
void Timer_Start(void){
  SCGC1_TPM2 = 1;
}

void interrupt VectorNumber_Vtpm2ovf interrupt_TPM2(void){
  BASE_TIEMPO++;
  TPM2SC_TOF  = 0;
}
char Time_Out(int time_out){
    static unsigned long int base_time_out=0;
    switch(std_time_out){
      case 0:
        base_time_out=BASE_TIEMPO;
        std_time_out=1;
      break;
      case 1:
        if(BASE_TIEMPO-base_time_out>time_out){
          std_time_out=0;
          return 0;
        }
        else {
          std_time_out=1;
          return 1;
        }
      break;
      default:
        std_time_out=0;
      break;
    }
 }

 void Clear_Time_Out(void){
    std_time_out=0;
 }

void TPM4_Run(void){

}
```

A6. Diferencias entre el MSP430 de TI y MC9S08QE128 de Freescale

Features	Cross Reference	MSP430FG4619	QE128
Supply voltage range	n/a	1.8 V to 3.6 V	1.8 V to 3.6 V
Flash size	Section 3.10.2	120K	128K
Flash programming range	Section 3.10.4	2.7 V to 3.6 V	1.8 V to 3.6 V
RAM size	n/a	Up to 8K	Up to 8K
Pin quantity	n/a	100	Up to 80
Analog comparator (ACMP)	Section 3.2	1	2
Max CPU speed	n/a	8 MHz	50.233 MHz
Analog to digital convertor (ADC) channels (12-bit)	Section 3.1	12	Up to 24
Clock generation module	Section 4	FLL with up to two external or one internal source	FLL with one internal or one external source
Digital to analog convertor (DAC)	n/a	2-ch 12-bit	None
Debugger	Section 3.12	JTAG	BDC and DBG
Direct memory access (DMA)	n/a	Yes	None
Hardware multiplier	Section 3.5	Yes, 16-bit (peripheral)	Yes, up to 32-bit (core)
Inter IC (IIC)	Section 3.11.1	Up to 1	2
Interrupt request (IRQ)	Section 3.8	Yes	Yes
Liquid crystal display (LCD)	n/a	160 segments	None
Low voltage detect	Section 3.6	Yes, with 14 trigger levels	Yes, with 2 trigger levels
Keyboard interrupt (KBI)	Section 3.4.3	16 pins	16 pins
Port I/O	Section 3.4.1	80 (100-pin package)	70 (80-pin package)
Op Amp	n/a	3	None
RTC / Calendar	Section 3.3	Yes / Yes	Yes / No
Serial communication interface (SCI)	Section 3.11.2	Up to 2	2
Serial peripheral interface (SPI)	Section 3.11.3	Up to 3	2
Timer	Section 3.7	Up to 3	3
Watchdog	Section 3.9	Yes	Yes

A7. Lista de materiales de la tarjeta CCMI

COMPONENTES CCMI				
Descripción	**Detalles técnicos**	**Código Digikey**	**Etiqueta en esquemático**	**Cantidad**
Multiplexor	IC MULTIPLEXER QUAD 2CHAN 16SOIC	497-7843-1-ND	MUX	1
BuzzerMagnetico	BUZZER MAGNETIC 3.0V 9MM SMD	668-1079-1-ND	BUZZER	1
Cámara UART*	C328R	www.electronics123.com	Cam1, Cam2	2
Memoria EEPROM I2C	24AA1025	24LC1025-I/SM-ND	IC1, IC2	2
uCFreescale HCS089	MC9S08QE128	MC9S08QE128CLD-ND	QE128	1
MAX 232	MAX3232CSE+	MAX3232CSE+-ND	MAX	2
Regulador 3,3V	1616AD33	FAN1616AD33X-ND	V-REG	1
Cristal	Cristal 4MHZ	535-9857-1-ND	XT	1
LEDs Amarillos	LED 1206 YELLOW SMT	350-1567-1-ND	MCU_OK, PCT_OK, SIMA_OK, SICOM_OK, SDCA_OK, ON, TX1, TX2, RX1, RX2	6
Resistencia 4,7K	RES 4.7K OHM 1/4W 5% 1206 SMD	RHM4.7KECT-ND	R7, R8	2
Resistencia 10K	RES 10K OHM 1/4W 5% 1206 SMD	541-10KECT-ND	R1, R5	2
Resistencia 470	RES 470 OHM 1/4W 5% 1206 SMD	541-470ECT-ND	R3,R4,R9,R10,R11,R12, R13,R14,R15,R16	10
Resistencia 5,1K	RES 5,1K OHM 1/4W 5% 1206 SMD	541-5.1KECT-ND	R2	1
Capacitor 0,1uF	CAP CER .10UF 25V 1206 LOW DIST	587-1148-1-ND	C5,C6,C7,C8,C9,C11,C14,C18,	8
Capacitor 22pF	CAP CER 22PF 25V 1206 LOW DIST	478-1472-1-ND	C1,C2	2
Capacitor 10uF	CAP TANTALUM 10UF 10V 10% SMD	478-1654-1-ND	C3,C4,C10,C12,C13,C15,C16,C17	8

A8. Costos de los componentes de la tarjeta CCMI

COMPONENTES CCMI				
Descripcion	**Etiqueta en esquemático**	**Cantidad**	**Precio unidad**	**Precio total**
Multiplexor	MUX	1	$ 1.02	$ 1.02
BuzzerMagnetico	BUZZER	1	$ 4.92	$ 4.92
Cámara UART*	Cam1, Cam2	2	$ 49.95	$ 99.90
Memoria EEPROM I2C	IC1, IC2	2	$ 4.32	$ 8.64
uCFreescale HCS089	QE128	1	$ 5.11	$ 5.11
MAX 232	MAX	2	$ 4.74	$ 9.48
Regulador 3,3V	V-REG	1	$ 0.43	$ 0.43
Cristal	XT	1	$ 0.64	$ 0.64
LEDs Amarillos	MCU_OK, PCT_OK, SIMA_OK, SICOM_OK, SDCA_OK, ON, TX1, TX2, RX1, RX2	6	$ 0.71	$ 4.26
Resistencia 4,7K	R7, R8	2	$ 0.10	$ 0.20
Resistencia 10K	R1, R5	2	$ 0.09	$ 0.19
Resistencia 470	R3,R4,R9,R10,R11,R12, R13,R14,R15,R16	10	$ 0.09	$ 0.94
Resistencia 5,1K	R2	1	$ 0.09	$ 0.09
Capacitor 0,1uF	C5,C6,C7,C8,C9,C11,C14,C18,	8	$ 0.65	$ 5.19
Capacitor 22pF	C1,C2	2	$ 0.37	$ 0.74
Capacitor 10uF	C3,C4,C10,C12,C13,C15,C16,C17	8	$ 0.43	$ 3.44
			Total en componentes	$ 145.20

Total en componentes	$ 145.20	S/. 935.10

Descripción	**Precio**
Dispositivos electrónicos	S/. 935.10
Fabricación de tarjeta	S/. 100.00
Soldadura de componentes	S/. 100.00
TOTAL	**S/. 1,135.10**

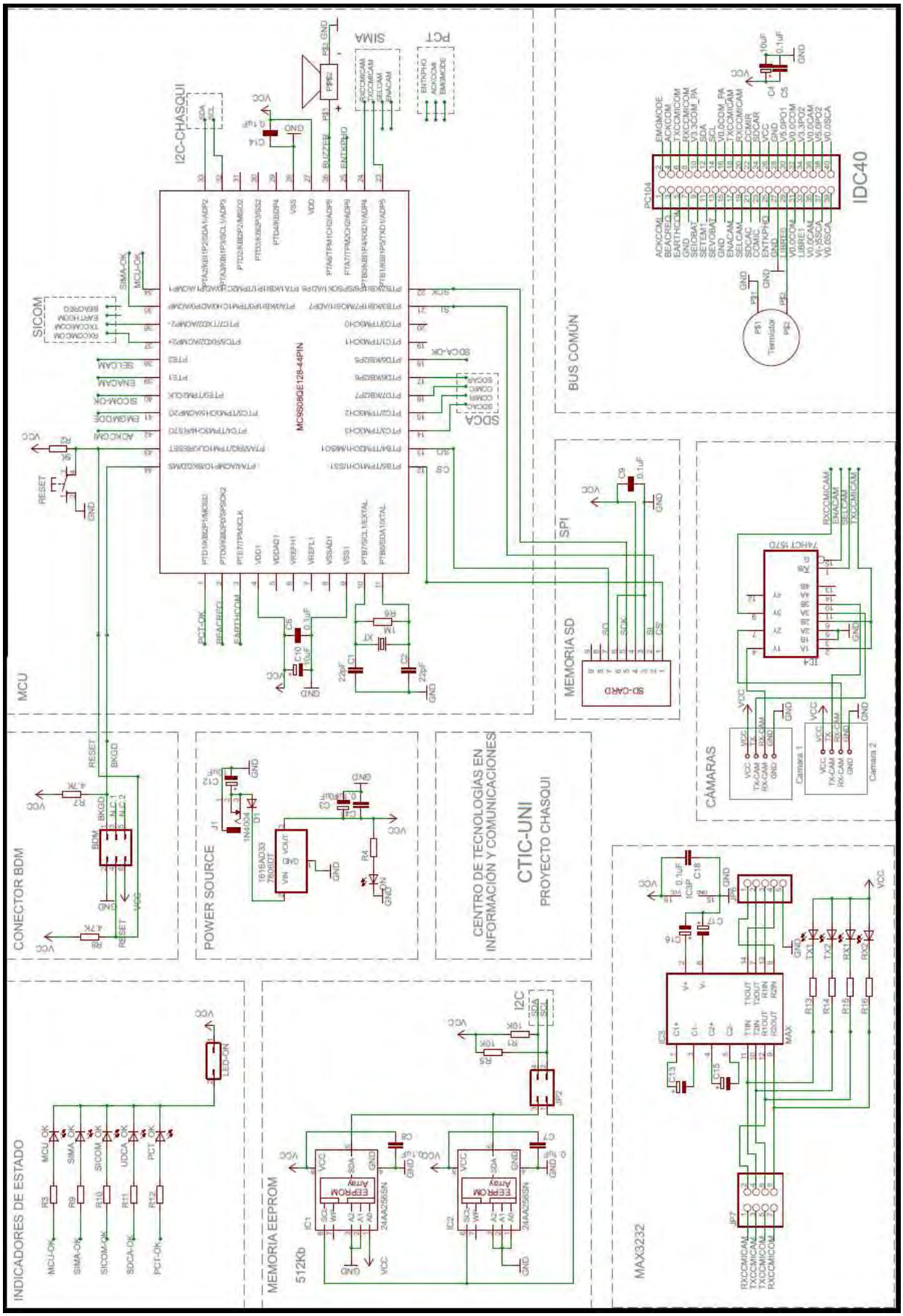
A9. Descripción detallada del bus común PC104

DESCRIPCIÓN DE PINES NANOSATÉLITE CHASQUI I

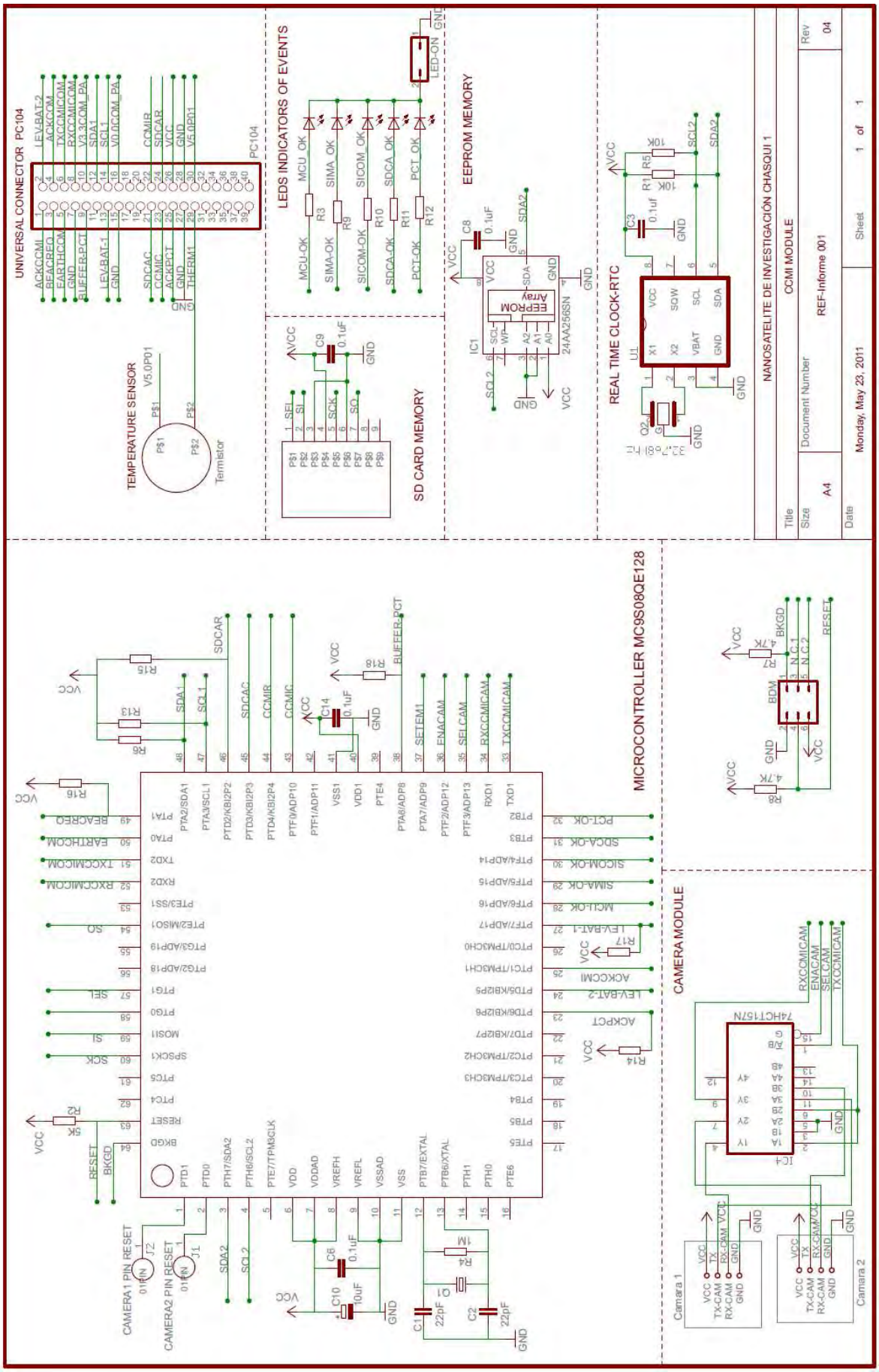
	GENERAL					CCMI-FREESCALE		SICONA-TEXAS		PCT-FREESCALE		SDCA-FREESCALE	
N	NOMBRE	RECORRIDO	Voltaje	MODO	DESCRIPCIÓN	PC104	NOMBRE	TIPO	NOMBRE	TIPO	NOMBRE	TIPO	NOMBRE	TIPO
1	LEV_BAT_2	PCT a CCMI y SICOM	3.3	DIGITAL	Estamos en modo emergencia (Nivel bajo de bateria)	2	LEV_BAT_2	IN	LEV_BAT_2	IN	LEV_BAT_2	OUT		
2	BUFFER_PCT	PCT a CCMI y CDMI	3.3	DIGITAL	Indica estado del buffer de PCT	9	BUFFER_PCT	IN			BUFFER_PCT	OUT		
3	LEV_BAT_1	PCT a CCMI y CDMI	3.3	DIGITAL	Junto con LEV_BAT_2 indican el nivel de la bateria	13	LEV_BAT_1	IN			LEV_BAT_1	OUT		
4	ACKPCT	PCT a CCMI	3.3	DIGITAL	Confirmación de PCT a CCMI	25	ACKPCT	IN			ACKPCT	OUT		
5		PCT a CCMI y COM	3.3	DIGITAL	Pin libre entre CCMI, PCT y SICOM	11	LIBRE1	IN/OUT			LIBRE1	IN/OUT		
6					Pin libre, actualmente conectado solo a PCT (GND-CAM)	35-36								
7	ACKCCMI	CCMI a PCT	3.3	DIGITAL	Confirmación de CCMI a PCT	1	ACKCCMI	OUT			ACKCCMI	IN		
8	CCMIR	CCMI a SDCA	3.3	DIGITAL	Pedido de orientación del CCMI a SDCA	22	CCMIR	OUT					CDMIR	IN
9	CCMIC	CCMI a SDCA	3.3	DIGITAL	Confirmación de permiso del CCMI al SDCA	23	CCMIC	OUT					CDMIC	IN
10	EARTHCOM	CCMI a SICOM	3.3	DIGITAL	Indica estado de CCMI	5	EARTHCOM	OUT	EARTHCOM	IN				
11	TXCCMICOM	CCMI SICOM	3.3	UART	Comunicación con SICOM UART	6	TXCCMICOM	OUT	TXCCMICOM	IN				
12	THERM1	CCMI SIMA A PCT		ANALO	voltaje del termistor1	29	THERM1	OUT			THERM1	IN		
13	RXCOMICOM	CCMI SICOM	3.3	UART	Comunicación con SICOM UART	8	RXCOMICOM	IN	RXCOMICOM	OUT				
14	ACKCOM	SICOM a PCT	3.3	DIGITAL	Confirmación de COM que está en Modo Emergencia	4	ACKCOM		ACKCOM	OUT	ACKCOM	IN		
15	BEACREQ	SICOM a CCMI	3.3	DIGITAL	Solicitud de Beacon	3	BEACREQ	IN	BEACREQ	OUT				
16	THERM2	SICOM A PCT		ANALO	voltaje del termistor2	33			THERM2	OUT	PERMA2	IN		
17	SDA	CCMI SICOM PCT SDCA	3.3	I2C	I2C para comunicación	12	SDA	DATA			SDA	DATA	SDA	DATA
18	SCL	CCMI SICOM PCT SDCA	3.3	I2C	I2C para comunicación	14	SCL	CLOCK			SCL	CLOCK	SCL	CLOCK
19	SDCAC	SDCA a CCMI	3.3	DIGITAL	Confirmación de orientación del SDCA	21	SDCAC	IN					SDCAC	OUT
20	SDCAR	SDCA a CCMI	3.3	DIGITAL	Permiso de orientación del SDCA	24	SDCAR	IN					SDCAR	OUT
21	THERM3	SDCA A PCT		ANALO	voltaje del termistor3	37					THERM3	IN	THERM3	OUT
22	V3.3PO1	PCT a CCMI y SICOM	3.3	POWER	Alimentación de 3.3V	26	V3.3PO1	IN	V3.3PO1	IN	V3.3PO1	OUT		
23	V0.0CCMI	PCT a CCMI	0	POWER	Tierra	7-27-28	V0.0CCMI	IN			V0.0CCMI	OUT		
24	V5.0PO1	PCT a CCMI y SICOM	5	POWER	Alimentación de 5.0V	30	V5.0PO1	IN	V5.0PO1	IN	V5.0PO1	OUT		
25	V0.0COM	PCT a SICOM	0	POWER	Tierra	31-32			V0.0COM	IN	V0.0COM	OUT		
26	V3.3PO2	PCT a SDCA y CAM	3.3	POWER	Alimentación de 3.3V	34					V3.3PO2	OUT	V3.3PO2	IN
27		PCT a SIMA	0	POWER	Tierra	35					V0.0CAM	OUT		
28	V5.0PO2	PCT a SDCA y CAM	5	POWER	Alimentación de 5.0V	38					V5.0PO2	OUT	V5.0PO2	IN
29	V0.0SCA	PCT a SDCA	0	POWER	Tierra	39-40					V0.0SCA	OUT	V0.0SCA	IN
30	V3.3COM_PA	PCT a SICOM	3.3	POWER	Alimentación de 3.3V	10			V3.3COM_PA	IN	V3.3COM_PA	OUT		
31	V0.0COM_PA	PCT a SICOM	0	POWER	Tierra	16			V0.0COM_PA	IN	V0.0COM_PA	OUT		
32					Pin libre	17								
33					Pin libre	18								
34					Pin libre	19								
35					Pin libre	20								

NOTAS
Las alimentaciones tendrán doble pin
No olvidar condensadores en la alimentación en los integrados
Todos los módulos sueldan pero no hacen agujeros de columna
Se mandará un modelo en EAGLE para su diseño final

A10. Plano CCMI – Modelo de laboratorio



A11. PlanoCCMI – Modelo de vuelo



A12. Abreviaciones

CAD	Diseño asistido en computadora
CCMI	Control central y monitoreo de información
COM	Sistema de comunicación
EMEC	Estructura mecánica
ESTER	Estación terrena
I2C	Entre circuitos integrados
IDE	Entorno integrado de desarrollo
IR	Infrarojo
MCU	Microcontrolador
MGP	Módulo de gestión del proyecto
MIP	Módulo de integración y pruebas
OS	Sistema operativo
PCB	Tarjeta de circuito impreso
PCT	Potencia y control térmico
RTOS	Sistema operativo en tiempo real
SCH	Esquema eléctrico
SD	Seguridad digital
SDCA	Sistema de determinación y control de actitud
SICOM	Sistema de comunicaciones
SIMA	Sistema de imágenes
SMD	Dispositivo de montura superficial
SORAT	Sistema de órbitas y atmósfera
SPI	Interfaz serial de periféricos
UART	Unidad de transmisión y recepción asíncrona
WDT	Watchdog timer