PRATHAM

IIT BOMBAY STUDENT SATELLITE

Preliminary Design Report

On Board Computer

By

Subsystem Leader Vishnu Sresht

All Subsystem Members

Department of Aerospace Engineering,

Indian Institute of Technology, Bombay

May, 2009

Contents

Tables

Figures

Introduction

Pratham is the first satellite designed and built by the IIT Bombay Student Satellite Project. It's defining features are as follows:

- Weight: under10 kilo grams
- Size: 26cm X 26cm X 26cm cube
- Solar Panels on 4 sides
- Orbit – 10:30 polar sun-synchronous
- 2 deployed monopoles
- Downlink at 2 frequencies (150MHz and 400MHz)
- No uplink
- Completely autonomous

The main aim of *Pratham* is to measure the Total Electron Count (TEC) over India. It is also recognized that the learning and experience gained from this endeavor is quite valuable by itself regardless of the data obtained from the TEC measurements. The mission statement has, therefore, been broken down into of various levels. The successful completion of each level adds to the overall success of the mission.

Description of each level of the mission success criterion	Success of Mission
Flight Model ready for Launch	55%
Beacon Signal received on Ground-station	65%
Communication link is established – coherent data is received	75%
TEC measurements at IITB completed	85%
Satellite is functional for 4 months	100%

Table 1: Mission Success

The On-Board Computer Subsystem comprises of almost all the computing power employed by *Pratham*. The mission statement of *Pratham* requires the satellite to stay in a stable orbit for a reasonably long period of time and transmit data to the ground-station in order to facilitate measurement of the Total Electron Count (TEC) in the atmosphere.

The design of *Pratham* has the various sub-systems like the Communication sub-system and Controls sub-system providing the equipment for communication and controls respectively, with the On-Board Computer subsystem interfaces with these elements and executes the algorithms to keep the satellite stable and transmit data to earth.

The System Engineer has decided on an operating sequence for the various functions of the satellite in orbit. This sequence orders tasks on the basis of orbital time/position and also orders the execution of select tasks during the response to certain contingencies. The design of the On-Board Computer is based on achieving this operating sequence with the maximum efficiency and minimum risk.

Design Methodology

The design of the On-Board Computer sub-system has two aspects – the design of the hardware that forms the physical basis, and the accompanying software. The hardware design has been motivated by the requirement to interface with the components provided by the other sub-systems on a reliable basis.

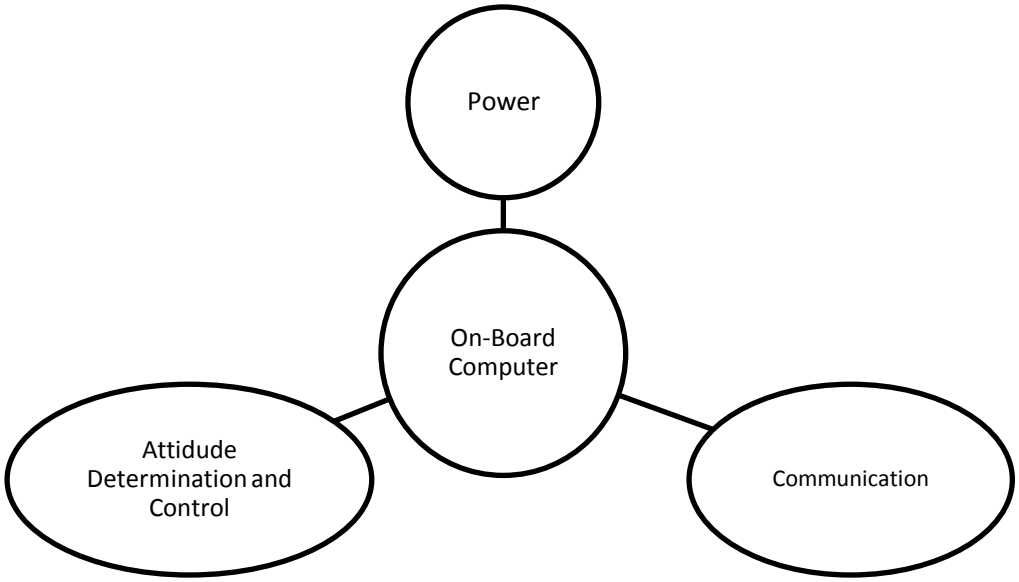

Figure 1: Overview

The On-Board Computer must interface with the Power sub-system to receive information about the health of the various components of the satellite. It must interface with the Attitude determination and Control sub-system to stabilize the satellite, and it must interface with the Communication sub-system to transmit data down to the ground-station. These design and implementation of these interfaces make up the major portion of hardware design.

The simplest satisfactory solution, in terms of components and layout has been chosen as this is the first time such a project has been attempted. Thus, along with reliability, the feasibility of the implementation is a major concern.

The satellite's operational sequence translates to a more detailed operational sequence for the On-Board Computer, where satellite tasks such as downlink, for example, correspond to a sequence of storage and retrieval from memory, interfacing with the antenna and transmission of suitably encoded data. The software of the On-Board Computer is concerned with the implementation of the desired operational sequence.

Due to the nature of the system – a satellite in a regular, periodic orbit, the nominal operation of the satellite involves the cyclic execution of a series of predefined tasks in a predetermined order. This allows us to design the software without the need for an operating system or a complicated scheduler. This should greatly simplify the implementation and testing of the On-Board Computer.

The nature of the system – a satellite far removed from manual control, also imposes certain limitations of the contingencies that can be handled and the methods by which the satellite can attempt to return to nominal operation. The failure of solar panels, for example, or of the actuators are exigencies that cannot be recovered from. On the other hand, the satellite can still function, to a degree, after the failure of the Global Position System (GPS) unit. The system engineer has identified the nature of the failures that can be handled and their appropriate responses. The implementation of these responses is part of the software design.

Fabrication and Testing

The idea behind a *cube-sat* is a satellite built by universities using commercially available off-the-shelf components. In keeping with this principle, the components used as part of the On-Board Computer will be those that have been used regularly by students- no special space grade components will be used. The procedures for fabrication, on the other hand, will be those that are prescribed by ISRO and the system – including the printed circuit board and the associated soldering in of components will be done at ISRO suggested facilities and will confirm to ISRO approved standards.

The testing of the On-Board Computer comprises of multiple levels. The software and hardware comprising of each interface has been tested individually with the associated components from the other sub-systems. This has been followed by the testing of the entire information pathway which links the power, on-board computer and communication sub-systems. This will be followed by On-board computer In Loop Simulations (OILS) which test the complete functioning of the On-Board Computer with its implementation of the Control algorithms, Communication tasks and Health Monitoring duties by simulating the actual conditions of space as experienced by the various sensors. These simulations will locate design flaws, if any, and also provide estimates on the reliability of the system as a whole.

Sub-System Requirements

The various tasks that the On-Board Computer sub-system is supposed to perform on the flight model are listed in this section. The efficient and reliable execution of these tasks is the primary motivation for the design of the On-Board Computer sub-system.

System requirements

The On-Board Computer sub-system is required to conduct pre-flight checks to confirm the operational status of select, important components like the GPS unit. These checks will be conducted before the satellite is integrated with the launch vehicle.

The health status of a number of components will be monitored and stored while the satellite is in orbit. This "Health – Monitoring" Data (or HM data) will consist of Attitude data, Position data, data regarding various loads as seen by the Power sub-system, and the associated timestamps.

Attitude Determination and Control

The Controls subsystem is primarily concerned with determining the satellite's position and maintaining a stable orbit. The On-Board Computer is required to interface with sensors and actuators and execute the control-law calculations that determine the required actuation from the current position.

Sensor Interfacing:

The On-Board Computer is required to interface with the following sensors:

1. Global Positioning System (GPS) Unit
2. Magnetometer
3. Sun-Sensors

Control Law Execution:

The On-Board Computer shall execute the control law as designed by the Controls sub-system. This includes performing the requisite numerical calculations with the desired accuracy as well as according to the predefined sequence.

Actuation:

The On-Board Computer will interface with and actuate the magnetorquers according to the control law. This implies provision of suitable pulse-width-modulated signals for suitable time intervals.

Power

The power subsystem is concerned with acquiring, regulating and distributing power to the various components of the satellite. The Power and On-Board Computer subsystem must interact to exchange data about satellite health.

Response to low-power situations

The Power subsystem will generate a signal indicating a predicted inability to supply adequate power to the On-Board Computer subsystem. The On-Board Computer subsystem is required to recognize this signal as a *hard interrupt* and initiate suitable shutdown measures.

The data exchanged regularly between the Power and On-Board Computer subsystem will indicate any misbehavior on the part of any component. The On-Board Computer will respond with suitable redundancy measures.

Communication and Ground-Station

For the purposes of monitoring the status of the mission, the satellite will transmit the health of the various components during downlink. In this regard, the On-Board Computer sub-system is required to send the health data in packets encoded using the AX-25 communication protocol, at a rate of 1.2 kbps whenever the satellite is over the Ground-Station.

Integration

To ease the integration of the On-Board Computer sub-system with the rest of the satellite, the final Printed Circuit Board (PCB) of the On-Board Computer must be 12 cm x 12 cm in dimensions, with a natural frequency above 120 Hz. The final PCB must confirm to the requirements of the harness to be designed by the integration team.

Quality

The Quality sub-system requires the On-Board Computer hardware to be tested extensively to reduce chances of infant mortality. While the exact testing procedures have not yet been fixed, they must provide an estimate of the probability of failure. For the satellite to have a life of 1 year, the estimate of the probability of failure should be below some threshold.

The quality of the software plays a big role in the success of the satellite as a whole. The software will be tested using OILS to ensure that the probability of failure is low. It is also recognized that formal verification of the software is not necessary.

In order to execute the above functions, the On-Board Computer sub-system places several constraints on other sub-systems.

Power

The Power sub-system is required to supply continuous power of up to 1Watt at all times. The Power sub-system will also respond to the On-Board Computer's requests to turn on or off certain components depending on their need. This information will be communicated to the Power sub-system in the form of one packet every 2 seconds. The Power sub-system will also send "Health Monitoring" data regarding the status of each of the major loads on the satellite to the On-Board Computer when polled for this data.

Thermals

The Thermals sub-system will be required to remove excess heat from the On-Board Computer PCB, in order to maintain the temperatures of all the components between -40 °C and 85 °C (the operating range of the components).

Hardware Design

Requirements of the Hardware Design

The Hardware of the Onboard Computer must satisfy the following requirements:

1. Interfaces: The Onboard Computer must be able to interface with the sensors employed by the ADCS subsystem. It must also interface with the CC1020 circuit employed by the communication subsystem. Finally, it should be able to drive the actuators (magnetorquers) and communicate with the microcontroller used by the power subsystem.
2. Robustness: The design must be robust enough to withstand the temperature and radiation conditions found in space. Ideally, space/industrial grade components should have been used, but due to the desire to use COTS components, the design should be robust enough to compensate for any internal failures and still provide some degree of functionality.
3. Simplicity: Pratham is the first project of its kind. Thus, emphasis was on designing a system using components we were familiar with. This included both the chips and the communication interfaces connecting them.

Basic Design

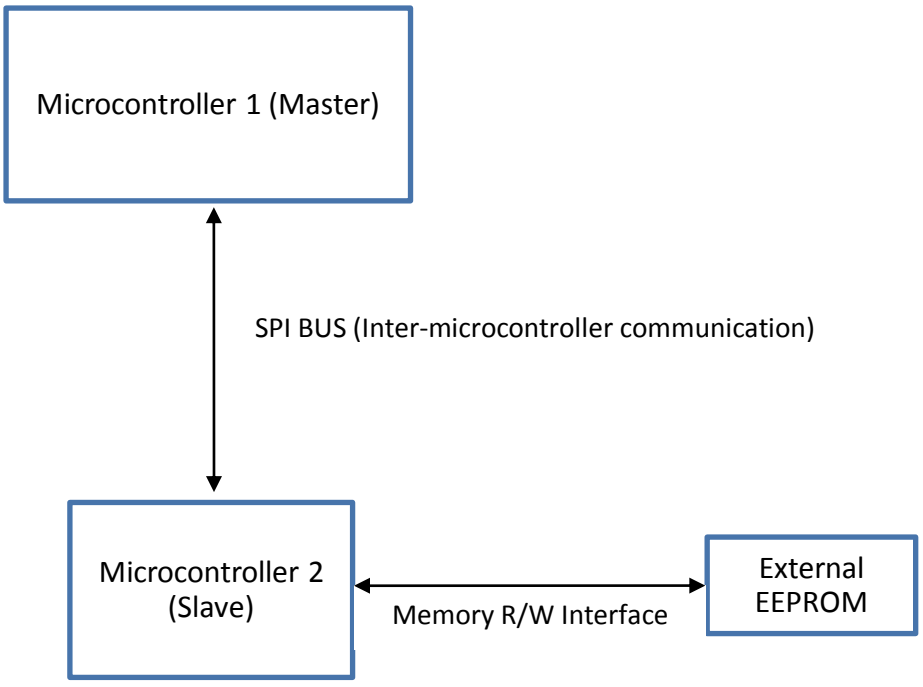

Figure 2: Basic Design

Microcontrollers

The tasks that have to be performed by the Onboard Computer are quite complex. Thus, we decided to use microcontrollers in place of other options like FPGAs. The main points that tipped the choice in favor of microcontrollers were:

1. Familiarity with their programming and use
2. Ability to change programming quickly, faster development and testing cycles
3. Versatile peripherals, no need for supporting circuitry
4. Easily available locally, cheap, ease of PCB design, fabrication,

The specific microcontroller chosen was the Atmel ATmega128. This microcontroller had been used for various other embedded systems projects in our institute. It also boasts of an incredible set of peripherals that would greatly simplify the rest of the design. Some of its features are:

- 16 MIPS throughput at 16 MHz operating frequency
- Low Power Consumption (~3mW/MHz)
- 4 KB on-chip SRAM, 128 KB on-chip Flash, 4 KB on-chip EEPROM

Some of the peripherals that are essential to the hardware design:

- 3 timers (2 8-bit and 1 16-bit) - for PWM generation and system clock
- 8 channel 10-bit Analog to Digital Converter – to sample voltages on sun-sensors
- Two-Wire Serial Interface (TWI or I^2C) – interface with power microcontroller and external memory bank
- Two independent UARTs - to interface with magnetometer, GPS and also for debugging
- Master/Slave SPI interface – to program CC1020 chip
- On-chip Watchdog timer – to restart microcontroller in case of system lockup

The microcontroller will be operated at the rated clock frequency of 16 MHz at 5 V.

The on-chip Flash memory will be used to store the program memory. While it is realized that the use of Flash increases the susceptibility of the system to radiation induced failures, it is felt that the use of an

external PROM to store the program memory would overly complicate the design. Precautions against radiation will be taken on a system level and will be described later.

The execution of the control law in a speed optimized manner necessitates the use of several tables for computational purposes. These tables will be stored in the on-chip EEPROM and the on-chip Flash memories.

To provide an accurate record of any failures or shut-downs and to keep a record of what the satellite was doing prior to these failures; a concise history will be maintained in the on-chip EEPROM. This will consist of flags signaling the state of the system before the most recent failure/shut-down.

Sample computations, of a similar complexity as that of the final control law, have been executed on these microcontrollers and it has been determined that the amount of on-chip SRAM is sufficient for these purposes.

Need for 2 separate microcontrollers:

Introducing 2 separate microcontrollers into the hardware design was necessary for a number of reasons:

- Logical separation of the ADCS and the Communication portions of the Onboard Computer. The Primary microcontroller handles the interfaces with the sensors and actuators of the ADCS and also runs the control law. The secondary microcontroller interfaces with the CC1020 chip provided by the communications subsystem and handles the various subroutines involved in the downlink of data to earth.
- Separation of data acquisition and data handling: The primary microcontroller is responsible for the collection of data from the sensors as well as receiving health monitoring data from the Power subsystem. The secondary microcontroller stores this data in the external memory bank, encodes the data using Error Detection and Correction codes (EDACs) and transmits this data to the CC1020 chip packed in AX25 frames.
- Simplification of the Operational sequence: When the satellite is over the ground-station, it must simultaneously execute 2 main functions – attitude determination and control, and downlink of accumulated data. In order to execute these tasks concurrently without the need

for multi-threading or context-switching, these tasks are executed on two different microcontrollers.

Inter-Microcontroller Communication Interface

The two microcontrollers will be interfaced using the on-chip SPI peripheral communication interfaces. The primary microcontroller will always initialize any communication (act as a Master) and the secondary microcontroller will be hardcoded to act as the Slave. The SPI interface is a byte-oriented serial interface that is essentially two shift registers (one on each microcontroller) connected together. The transfer of bytes between this shift register is synchronized by a clock generated by the Master. The data carried by this channel will be described later.

External Memory Bank

The mission statement requires the onboard computer to store and transmit two types of data.

1. Attitude data: In order to carry out atmospheric tomography, the satellite's position and attitude must be known to reasonable accuracy. The position of the satellite can be accurately tracked by ground-stations and with the aid of agencies such as NORAD. However, the attitude of the satellite can be best determined using the sensor readings transmitted from the satellite.
2. Health Status: As *Pratham* is the first student-satellite initiative at IIT Bombay, data regarding the progress of the mission will be immensely useful to subsequent missions. This data will consist mainly of the health of various components onboard, and the history of any component failures. This data will help us diagnose design faults and determine the gap, if any, between design conditions and operating conditions.

In order to store this data, an external EEPROM, 24FC1025 from Microchip, has been chosen as the third component of the onboard computer hardware. The main features of the EEPROM are:

- 128 KB serial EEPROM
- Two-Wire Interface (I^2C) interface (100 kHz or 400 kHz clock compatible)
- Low Power Consumption (2.25 mW max)

- High speed operation – 3ms page write
- 128 byte page (buffered write)

Memory Read/Write Interface

The memory is read from or written to using the on-chip I^2C interface of the secondary microcontroller. The interface consists of one line for the clock (generated by the secondary microcontroller) and one bi-directional line for the data. The EEPROM allows for sequential and random reads, and page or byte writes.

Interfaces

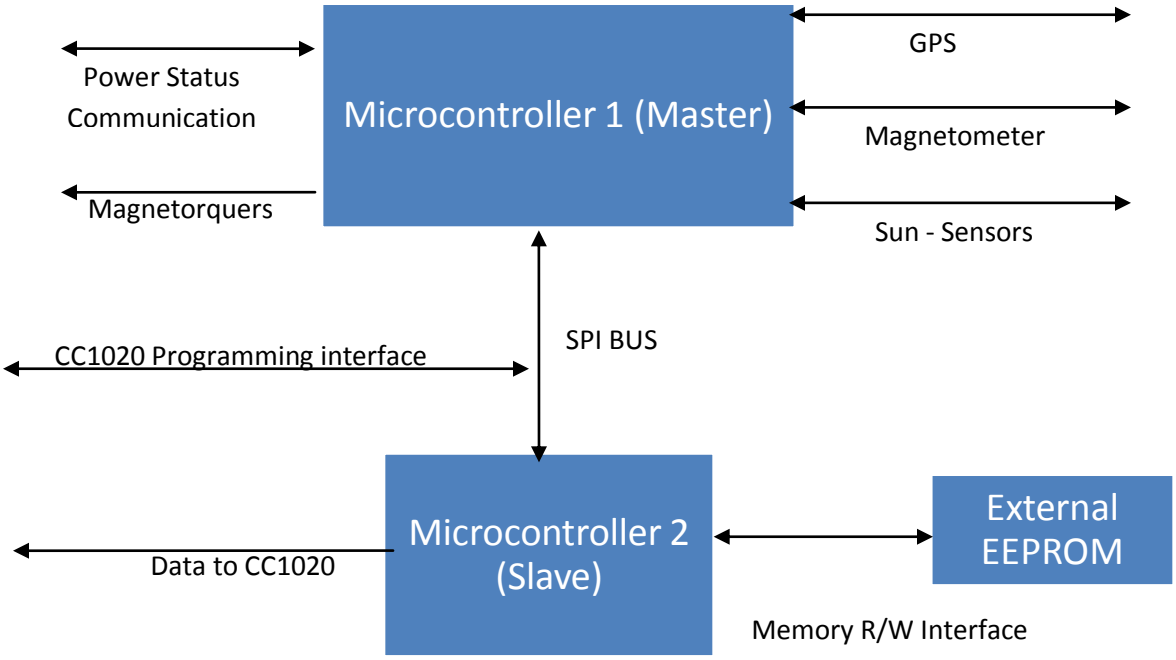

Figure 3: Block Diagram of On Board Computer

1. INTER MICROCONTROLLER SPI BUS:
   - Bidirectional (uses SPI protocol)
   - 1 permanent master (Microcontroller 1) and 2 Slaves (CC1020 and Microcontroller 2)
     - Used to program/initialize CC1020 after launch
     - Used to transmit housekeeping data to Microcontroller 2

2. POWER STATUS COMMUNICATION BUS
   - Bidirectional I^2C bus
   - Microcontroller 1 is always the master with the power microcontroller always the slave
     - Current status (on/off) of various loads on power bus ,sent from power microcontroller

    - Required/desired status (on/off) of various loads sent from Microcontroller 1

3. UART BUSSES
    - 2 independent bidirectional UART lines
    - GPS line
        - Used to configure GPS
        - Used to poll for and receive GPS data at regular intervals
    - Magnetometer line
        - Used to program magnetometer after launch
        - Used to poll for and receive magnetic field data at regular intervals

4. EEPROM R/W BUS
    - Bidirectional I^2C bus
    - Used to store/retrieve house-keeping data from EEPROM

5. CC1020 DATA BUS
    - Unidirectional line
    - Used to transmit housekeeping data (suitably encoded) to the CC1020 chip for down-linking

Auxiliary components

The Onboard Computer main board will also host a few other circuits.

- H-Bridges: The GPIO pins of the microcontrollers will not be able to drive the magnetorquers themselves. They will serve as the logic inputs to a H-Bridge which will then drive the torquers with the required currents. The microcontroller output pins will provide the PWM signal which will act as inputs to the enables of the magnetorquers.

- An Analog Multiplexer: There are a total of 12 sun-sensors on board the satellite. As our microcontroller has an 8 channel ADC, we will need to multiplex the sun-sensor inputs before A/D conversion. This multiplexer will also be placed on the Onboard Computer mainboard.

Software Design

Issues in software design

As part of the pre PDR phase, the Onboard Computer subsystem also completed a major portion of the software design. Software design comprises of formulating solutions to the following issues:

- To design algorithms that would perform the various tasks listed as part of the sub-system requirements. These algorithms must be robust and efficient.
- To write code to drive the various peripherals of the microcontrollers. These codes would form part of a primitive hardware abstraction layer and would provide methods that other developers could use to drive the peripherals without going into the details of microcontroller programming.

The entire software design process for the PDR was split up into two concurrent phases. These phases consisted of writing codes for hardware control and for individual task execution simultaneously by different members of the onboard computing team.

The task of software integration and testing is still to be done and will be taken care of before the DDR.

Hardware Control

In order to execute various tasks, it is necessary to use a number of the atmega128's peripherals. One part of software design is to write the C code required to drive these peripherals. This effort was further sub-divided into two portions –

Sensor interfacing:

The on-board computer is required to interface with the GPS unit, the magnetometer, and the sun-sensors. The GPS and Magnetometer are both compatible with the RS232 communication standard. As the atmega128 has 2 fully independent UART peripherals, both the magnetometer and the GPS will have a dedicated UART peripheral interfaced with them. The sun-sensor required the use of the on-chip analog-to-digital converter (ADC)

- In order to interface with these components, the UART (Universal Asynchronous Receiver Transmitter) peripheral needed to be configured. This peripheral allows for complete duplex reception and transmission.

  The configuration of this peripheral for asynchronous use involves the specification of the required baud rate, and the frame format (number of stop bits and the presence/absence of parity bits). Transmission is initiated by simply writing the byte to be transferred to a specific hardware register. The end of transmission or reception is indicated by the corresponding flag in a dedicated status register. The setting of these flags can also be used to trigger appropriate interrupts.

  - The magnetometer will be operated in polling mode. Every time a reading is required, the primary microcontroller will send a 4 byte command to the magnetometer via the UART peripheral. The microcontroller will then enable receive-complete interrupts and wait for a response (wait for the UART receive complete flag to be set and the interrupt to be triggered). The magnetometer will send a 7 byte string of the magnetic field values (2 bytes each for x, y and z directions and one 'end of message' byte). As soon as byte is received, the setting of the receive-complete flag will trigger an interrupt. The received byte will be buffered in each interrupt. Once 7 bytes are received, the receive-complete interrupts will be disabled and the received bytes will be processed.
  - The GPS requires no command to be sent. As soon as it is switched on (to be discussed later), it will take around 90 seconds (warm start) to acquire a fix and start transmitting GPS data. During this period, the receive-complete interrupt of that UART peripheral will be enabled. The GPS unit will send one packet of GPS data every second. Each data packet consists of 8 'messages' conveying position, velocity, time-date and other information. Each message consists of a number of bytes. The bytes making up relevant messages will be buffered in the receive-complete interrupt for future parsing. The GPS UART interface will be operated at 9600 baud.

- The sun-sensors are similar in operation to photodiodes. They provide an analog voltage as output. These voltages will be routed to an analog multiplexer. The microcontroller will use GPIO pins to control the select lines of the multiplexer and the multiplexer output will be connected to one of the pins associated with the ADC peripheral.

The Atmega128 carries an on-chip 10-bit successive approximation ADC and allows for up to 15kSPS at maximum resolution – with 13 clock cycles per conversion. Completion of conversions can be used to trigger interrupts.

In order to configure this peripheral, it is first enabled; the appropriate input channel is then selected. The 10-bit result is stored in 2 registers – the result may be suitably adjusted so that the 2 least significant bits may be ignored and only one of these registers needs to be read.

Each conversion is started by writing 1 to a particular bit in the configuration register. Once a conversion is complete, a flag is set - this will be used to trigger interrupts. In the interrupt service routine, the GPIO pins connected to the select inputs of the multiplexer will be changed so that a different sun-sensor output is selected and given as input to the ADC. After all 12 conversions, the ADC peripheral will be disabled until it is time for the next set of readings to be taken.

Data Communication

The Onboard Computer design involves the exchange of data between the microcontrollers and several other devices on board the satellite. These data transfers are handled by several on-chip peripherals and codes must be written to drive these peripherals.

The data pathways onboard the satellite are:

1. Power – Onboard Computer : Health monitoring data and load configuration (I^2C)
2. Primary Microcontroller – Secondary Microcontroller: Health monitoring and attitude data, commands for downlink (SPI)
3. Primary Microcontroller – CC1020 : configuration of data registers (SPI)
4. Secondary Microcontroller – CC1020 : downlink (GPIO)
5. Secondary Microcontroller – EEPROM : data storage before downlink (I^2C)

- I^2C communication: The I^2C or TWI standard consists of one data line and one clock line, with transmission speeds up to 400 kHz. There is a predefined master (the primary microcontroller in pathway 1 and the secondary microcontroller in pathway 5, in our design) and one slave (the power microcontroller in pathway 1 and the EEPROM in pathway 5, in our design). The master

may either be the transmitter or the receiver. In either case, it is the master that initiates the transmission and provides the clock.

TWI operation is quite involved and protocol details may be found in the datasheets of TWI compatible devices.

- SPI communication: The Serial Peripheral Interface is a very simple on-chip communication interface. The physical layer consists of 4 lines – MOSI (master output slave input), MISO (master input slave output), SS (slave select) and CLK (clock). Speeds of up to one fourth of microcontroller clock frequency can be attained in normal operation.

  The master (always the primary microcontroller) provides the clock pulse. The master and slave can be imagined to have linked shift registers. With each clock pulse one bit of data is shifted from one shift register via the MISO and MOSI lines to the other shift register. After 8 clock pulses, the master and slave have exchanged the contents of their respective registers – one byte has been transferred between them.

  Configuring this peripheral is as simple as selecting whether the device is to be the master or slave, fixing a clock frequency (in case of master) and enabling interrupts as desired. The master initiates transmission by pulling the SS line low (it's actually SS complement) and then starts sending clock pulses. Data stored in a (specific register on the Atmega128) is exchanged and once the exchange is complete, a flag is set (interrupts may be triggered by this).

- GPIO use for communication: The CC1020 communication circuit requires binary data as input. It then modulates this data and passes it on to an antenna circuit for transmission. In our design, this input signal is provided by a GPIO pin on the secondary microcontroller. For our purposes, the CC1020 input must be provided at a constant baud rate of 1200 kbps. A hardware timer is used to trigger interrupts at this frequency, and in each interrupt service routine, depending on the bit to be sent to the CC1020, the GPIO pin is set high or low.

  The configuration of the timer involves the selection of the base timer frequency (determined by the multiplication of a pre-scalar with the CPU frequency), and fixing the upper bound of the timer counter. The timer-counter counts from 0 to the upper bound and afterwards overflows. By enabling the corresponding interrupt, this overflow can be used to generate interrupts at a fixed frequency.

The data passed along these pathways is packaged into specific frames. These frames provide two distinct advantages,

- Data not in this format can be ignored as noise; any discrepancies in frame can be used to detect faults in either the microcontroller or the interface itself.
- Usually multiple bytes of data need to be passed along these channels. The peripherals however transmit on a byte by byte basis. Instead of calling a 'send_byte' function several times, it is better to create a function that takes a number of bytes as its argument and encapsulates the individual byte sends and the framing.

Further note that these communication channels are all operated on an interrupt basis (each of the corresponding peripherals features receive/transmission complete interrupts). This frees up the microcontrollers to take care of executing other tasks simultaneously.

1. Power-Onboard Computer Communication:

   The Power sub-system recognizes a total of 7 distinct loads on the satellite (Onboard computer, GPS, etc.). They will respond to our requests to turn off/on each of these loads (assuming nominal operation of the loads). As part of health monitoring, they will provide us with 10 bytes of data – detailing the voltage/power consumed by various loads on the satellite at a given instant.

   The onboard computer will send the power-microcontroller one byte of data every 2 seconds. The 7 most significant bits of this byte will signify whether one of the 7 loads should be on or off. (For example, one of these bits will correspond to the GPS unit – when this is to be switched on, the byte sent by the onboard computer will have that particular bit as 1). The last bit signifies whether health monitoring data is required or not.

   The power microcontroller will respond by turning on the loads as specified by the byte they receive. If the last bit is 1, they will send us 10 bytes of health monitoring data; otherwise they will send us back a single byte with the final status (on/off) of the loads. During nominal operation, it is expected that the most significant 7 bits of the byte sent both ways will match. In case there is a discrepancy, it will be read as the failure of that particular load.

2. Primary-Secondary Microcontroller Communication:

   This interface will be used for 2 purposes:

   - Initialization / Termination of Downlink – a single byte will be sent from the primary microcontroller to the secondary microcontroller indicating the start / end of the satellite's visible path over the ground-station. On receiving this message, the secondary microcontroller will begin/terminate the downlink routines.
   - Transmission of Health-Monitoring Data – A packet comprising of 25 bytes will be sent from the primary to the secondary microcontroller. These packets will be received and then stored in the external EEPROM.

   Each message will be followed by a confirmation by the secondary microcontroller. The primary microcontroller will not wait for this confirmation, but will handle it in an interrupt – setting a flag on successful confirmation.

3. CC1020 Programming:

   The programming/configuration of the CC1020 will be performed by the primary microcontroller via the SPI interface (SPI allows for multiple slave devices – in this case both the secondary microcontroller and the CC1020 chip are slaves to the primary microcontroller – the relevant device is selected by pulling the corresponding chip-select line low).

   To program each of the number of registers on the CC1020, the primary microcontroller will first transmit the address of the register to be programmed (1 byte) and then transmit the data to be written to that register.

4. Data transmission to CC1020:

   The GPIO line between the secondary microcontroller and the CC1020 will simply be toggled (at 1200 Baud). The bit signaled by the status of the line (high/low) will be the bit modulated and transmitted by the CC1020 chip.

   The data itself consists of health monitoring data packaged into the AX25 format. This format has been chosen in order to enable the reception of our down-linked data

   AX.25

The following is the frame structure for AX.25.

Flag	Address	Control	PID	Info	FCS	Flag
0x7E	112/224 Bits	0x3F	0xF0	N*8 Bits	16 Bits	0x7E

Flags

The flags are simply the hex value 7E (01111110 in binary) sent over and over again when no information is being transmitted. For example, when you set the TX delay on your TNC to some value, it sends flags (7E's) over and over again for that period. These flags provide the receiver with a clear indication of when one packet has ended and the next is beginning. Thus, there must be at least one flag between two adjacent packets.

Address

Consider the simple packet: “W2FS -4> CQ, RELAY: Test”

This is a simple frame where

- W2FS-4 is the source
- "CQ" is the destination
- "RELAY" is the only digipeater
- —Test‖ is the text to be transmitted

The address field contains the destination (CQ, in this case) the source (W2FS-4 in this case), and up to eight digipeaters (only RELAY, in this case). Each of "call-signs" in the address field must contain exactly 7 characters, six for the call-sign and 1 for the SSID. If a call-sign is less than 6 characters long, it must be padded with blanks. In addition, the receiving station must have some way to determine when the address field has ended (since it could have anywhere from 0 to 8 digipeaters in it). This is handled by shifting each of the bits one position to the left so that a 0 appears as the least significant bit. This bit is then set to a one for the SSID (seventh byte) or

the last call-sign in the address field. For example, the destination call-sign would be encoded as follows:

Character	Hex Value from ASCII Table		Shifted Hex Value	
C	43	(01000011)	86	(10000110)
Q	51	(01010001)	A2	(10100010)
Space	20	(00100000)	40	(01000000)
Space	20	(00100000)	40	(01000000)
Space	20	(00100000)	40	(01000000)
Space	20	(00100000)	40	(01000000)

The seventh byte, the SSID, is a bit trickier. Use the following bit pattern: 011SSSSx where SSSS is the SSID (in binary) and x is a 0 if this is not the last call-sign in the address field and a 1 if it is the last call-sign in the address field. Since the destination address in this case has no SSID (that is, the SSID =0) and it is not the last call-sign, the value should be: 01100000 or 60 in hex.

The next call-sign is the source call-sign, which in this case is W2FS-4. Using the rules established above we get the following string of bytes for this call-sign:

Character	R	E	L	A	Y	Space	SSID = 0
Shifted Hex Value	A4	8A	98	82	B2	40	61

There is only one digipeater so it is necessary to set the least significant bit of the SSID of that call-sign to 1:

Character	W	2	F	S	Space	Space	SSID = 4
Shifted Hex Value	AE	64	8C	A6	40	40	68

Control and PID

For our purpose, we need to use the hex value 3F for the control byte and the value F0 for the PID byte.

Information

In this case the information being conveyed is simply the word "Test". This consists of 4 hex bytes as follows: 0x54 0x65 0x73 0x74

The only thing to be careful about here is that these values are NOT shifted to the left as the address bytes are.

Frame Check Sequence:

FCS is nothing but CRC calculated as per CRC-16 as described in the error detection and correction section.

Putting it All Together

Aside from the flags and the FCS (which will be calculated as we go along), our test packet can now be implemented as an array of 27 hex bytes as follows:

C	Q	Sp	Sp	Sp	Sp	SSID = 0	W	2	F	S	Sp	Sp	SSID = 4
86	A2	40	40	40	40	60	AE	64	8C	A6	40	40	68

R	E	L	A	Y	Sp	SSID = 0	Control	PID	T	e	s	t
A4	8A	98	82	B2	40	61	3F	F0	54	65	73	74

Or, as an initialized C array:

SendData[27] = {0x86, 0xA2, 0x40, 0x40, 0x40, 0x40, 0x60, 0xAE, 0x64, 0x8C, 0xA6, 0x40, 0x40, 0x68, 0xA4, 0x8A, 0x98, 0x82, 0xB2, 0x40, 0x61, 0x3F, 0xF0, 0x54, 0x65, 0x73, 0x74 }

5. EEPROM read/write:

   As many bytes as required may be written to the EEPROM at a time as required (subject to the upper limit of the page size of the EEPROM – 128 bytes). The master transmits a start byte (address of device) followed by the high and low address of the address where the read/write is to occur. The device responds with the data stored in that location (1 byte).

The data stored in the EEPROM is encoded with Error Detection and Correction Codes (EDACs)

Since the amount of memory that was available was not very low and the data was not so crucial to the mission success, we could easily forgo the efficiency of codes and gain on time taken to implement them. As such the choice was:

- CRCs for error detection.
- Hamming Codes for error correction

Implementation of CRCs for error detection ensures that we don't have to check the complete frame for errors if the CRCs go through without any errors.

Cyclic Redundancy Check:

Cyclic redundancy check was implemented as part of the communication protocol (AX.25). CRC-16 was used in the protocol. The codes essentially implement polynomial/binary carry-less division and concatenate the remainder to the end of the raw frame, giving a string that is divisible by the CRC-16 polynomial.

The main idea behind the CRC algorithm is that the FCS is generated so that the reminder of T/P is zero, where T is the encoded message and P is the polynomial. Its clear that

(1) $T = M * x^n + F$

Where M is the raw frame and F is the frame check sequence. This is the result of the concatenation of the Frame Check Sequence (FCS) to the raw frame.

We want the transmitted frame, T, to be exactly divisible by the pre-defined polynomial P, so we would have to find a suitable Frame Check Sequence (F) for every raw frame (M).

Suppose we divided only $M*x^n$ by P, we would get:

(2) $M*x^n / P = Q + R/P$

There is a quotient and a reminder. We will use this reminder, R, as our FCS (F). Returning to Eq. 1:

(3) $T = M*x^n + R$

We will now show that this selection of the FCS makes the coded message (T) exactly divisible by P:

(4) $T/P = (M*x^n + R)/P = M*x^n / P + R/P = Q + R/P + R/P = Q + (R+R)/P$ but any binary number added to itself in a modulo-2 field yields zero so:

(5) $T/P = Q$, with no reminder.

Hence, the following process has to be implemented in software for CRC.

1. Get the raw frame
2. Left shift the raw frame by n bits and the divide it by P (CRC-16 in our case).
3. The reminder of the last action is the FCS.
4. Append the FCS to the raw frame. The result is the frame to transmit

And to implement a check for error in the transmitted CRC encoded message.

1. Receive the frame.
2. Divide it by P.
3. Check the reminder. If not zero then there is an error in the frame.

Hamming Codes:

The fundamental concept behind Hamming codes is Hamming distance, which is the number of bits that 2 messages differ in. Hamming codes limit the word-space for messages in turn increasing the minimum hamming distance between any 2 valid messages.

Due to the convenience of having an 8 bit word, we chose to go ahead with (8,4) Hamming codes. It can be shown that the minimum Hamming distance between two words that are distinct messages of (8, 4) Hamming codes is 4.

$$G = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

Here, G is the generator matrix, while H is the parity-check matrix for decoding the message. If the raw message is a = $a_1a_2a_3a_4$, then the encoded message is given by

$$m = aG$$

The decoded message is checked for errors by calculating:

$$e = mH$$

If the message is untampered then e is a zero matrix. If e is a non-zero matrix, then the message has been altered and is rectified by applying the corresponding correction the last 4 bits of the message, which should represent the original message for an unaltered one.

For example,

- Let $a = [0 \quad 0 \quad 0 \quad 0]$ then, $aG = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$
- If the message that is received at the decoder is $m = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1]$, then, $e = [1 \quad 1 \quad 1 \quad 0]$ and hence, the original message would be $(\text{m}\text{\^{}}0\text{x}01) \ \& \ 0\text{x}0\text{f}$ which is $[0 \quad 0 \quad 0 \quad 0]$.

Note: We discarded the 4 most significant bits and only retained the 4 bit original message.

The codes required for sensor interfacing were written in such a way that the actual hardware/register manipulation is encapsulated within a series of functions. These functions formed a primitive hardware abstraction layer – and these functions could be used directly for writing more complex code. This encapsulation also allowed us to test the sensors and communication interfaces without having to run the control law and other complex tasks.

Task Execution

The other portion of the software comprises of the code written to execute the various complex tasks assigned to the onboard computer. This again can be subdivided into two parts.

The major tasks (like control law execution and downlink) are independent of each other and can be coded as separate blocks (each of these tasks can themselves be broken down into smaller sub-blocks – i.e. the control law task includes parsing of the GPS data as a sub-task). The coding of these sub-blocks formed one part of writing codes for task execution.

The second part comprises of deciding on a certain execution sequence and putting together the various blocks/sub-blocks according to this operational sequence.

Coming up with the operational sequence and the nature of each sub-block were the tasks undertaken for the PDR. The coding of several of these sub-blocks and their integration will be covered as part of the DDR.

The list of tasks/sub-blocks that are part of the software design is listed below along with their frequency of execution.

The List of Tasks:

1. Control Law Implementation
2. Communication Routines
3. Health Monitoring

Control Law Implementation

Data Acquisition

READ FROM GPS

1. Read GPS – receive data via UART, store sentence in array (This buffering takes place in the UART receive-complete interrupts). Since the GPS has a warm start-up time of around 70 seconds, the GPS must be turned on about 70 seconds before readings are to be taken. (Once every 10 minutes)
2. Parse array and store Lat, Long, Altitude and Time in separate arrays/variables. (Once every 10 minutes)

READ FROM MAGNETOMETER

1. Read Magnetometer – receive data via UART, directly receive Bx, By, Bz readings in binary. Store in array/variables (This buffering takes place in the UART receive-complete interrupts). (Polled for readings once every 2 seconds)
2. Convert the stored binary values to decimal values for use in the control law. (Once every 2 seconds)

READ FROM SUN-SENSOR

1. Sample ADC readings – for 12 sun-sensor readings, store 12 voltages in array (This buffering takes place in the ADC conversion-complete interrupts). (Once every 2 seconds)

Data Processing

1. Use GPS/Orbit Propagator and magnetometer readings along with magnetic field model to get attitude.
2. Use GPS/Orbit Propagator and sun-sensor readings along with solar model to get sun vector.

The GPS will be used only once every 10 minutes – to correct the orbit estimator. At other times, the orbit estimator itself will be used to determine the satellite's position.

3. Insert quantities calculated above into an "Optimal to Observation" algorithm that calculates the attitude quaternions.
4. Pass above quaternions through an Extended Kalmann Filter to get corrected attitude quaternions.
5. Calculate angular velocities after passing through software filter.

6. Determine angular velocities to decide which control law to use (De-tumbling / nominal).

Actuation

1. Use attitude quaternions and velocities (obtained from GPS) and calculate necessary PWM signals to magnetorquers using control law.
2. Modify PWM duty cycle in accordance with above calculations.

2 second time frame for actuation comprises of:

During magnetometer read, no PWM (first 1 ms of frame)

– remove interference due to magnetic field of torquers.

For rest of cycle – previously calculated values of current

PWM duty cycles updated after control law is executed

– This is just writing the new values to a microcontroller register.

NOTE: Entire sequence of Data Processing and Actuation steps is done once every 2 seconds. All these routines occur on the primary microcontroller. Thus 2 seconds is our primary frame time, while the GPS read routine occurs once every 10 minutes. Downlink is the only asynchronous event in the system.

Communications Routines

Check Position

Use GPS readings/Orbit Propagator to determine if we are over India or over ground-station or neither; and act accordingly (Executed every 3 second cycle):

a. If we are over Ground Station / India Switch on second monopole and configure CC1020 chip (via SPI). Else put CC1020 in power-down mode.
b. If we are over ground-station – transmit Health Monitoring Data
c. If we are over India but not over ground-station – transmit attitude data

NOTE: Primary microcontroller detects position, and sends signals to both the power microcontroller as well as the secondary microcontroller. Once power and secondary microcontrollers are ready, Primary microcontroller then configures the CC chip.

Data Transmission

Read necessary data from external memory via I2C and transmit data to CC1020 chip.

(Data transmission happens continuously while satellite is over India/ground-station)

NOTE: Secondary microcontroller takes care of simultaneously reading memory and transmitting to CC chip.

Health Monitoring

Data Acquisition

1. Primary microcontroller requests and receives (stores in temporary buffer) load data packet from Power microcontroller (via I2C). (Once every 1.5 minutes).
2. Primary microcontroller creates a data packet containing the HM data. (Once every 1.5 minutes)
3. RTC time is attached to data packet as timestamp.

Data Analysis

1. Loads currently switched on are monitored.
2. If necessary, loads are turned on/off, as in the case when second monopole must be turned on when over ground-station.

Data Storage and retrieval

1. Primary microcontroller sends packets to secondary microcontroller (via SPI bus) as follows:
    a. In nominal operation: Attitude + HM packet – once every 1.5 minutes
    b. When over India (and not over GS): packets containing only attitude data once every 2 seconds, and packets containing both attitude + HM data once every 1.5 minutes
    c. Over Ground Station: Packets will be buffered in primary microcontroller until downlink is terminated. Then they will be sent one after the other after downlink is terminated.
2. Secondary microcontroller processes the packets as follows:
    a. Packets containing both HM and attitude data are encoded using EDACs and AX25 and stored in EEPROM

b. Packets containing only attitude data are encoded using EDACs and AX25 and then directly transmitted to the CC1020 for broadcast

3. When downlink is in progress, attitude and HM data packets are retrieved, from the EEPROM, and transmitted to CC1020.

NOTE: all memory management and packaging is done by the secondary microcontroller.

Integration and Scheduler Design

The final portion of software design consists of putting together all the individual blocks to form the execution sequence. The determination of the execution sequence itself depends only on the list of tasks, their nature and the requirements on onboard computer. Thus, the execution sequence was finished before the PDR, while completion of coding of all the sub-blocks will proceed after the PDR.

Issues in design of execution sequence

- The tasks must be executed in the exact order and at the exact frequency as determined from the sub-system requirements. Thus emphasis must be given not only to the intelligent ordering of tasks –but also to their scheduling in a manner that allows the execution of all the tasks within a specified frame time.
- The presence of asynchronous events – interrupts from various interfaces, the downlink process – complicates the scheduling of blocks into a neat sequence. However, it must be noted that that the presence of interrupts allows for the delays associated with communication and sensor interfacing tasks to be bypassed – removing idle periods.
- The requirement for a fixed frame time necessitates the implementation of a system timer. This timer must be used to trigger execution of tasks at the appropriate times.
- For every task in the task list, worst case execution times must be calculated. It is imperative that the sum of the execution times of all tasks in a frame be smaller than the frame time itself.

  Certain measures must be taken so that when a task does not achieve completion within a reasonable time window, the microcontroller does not “hang”.

Implementation

Two features greatly simply the integration:

- The separation of communication tasks and control tasks allows us to write separate schedulers for the primary and secondary microcontrollers. The scheduler for the primary microcontroller comprises of the control and health monitoring tasks; the scheduler for the secondary microcontroller consists of the routines necessary for downlink.
- The tasks themselves have been broken down in to smaller modules/sub-blocks. Each of these sub-blocks is simple enough for us to reject very complicated scheduling algorithms. Further, the tasks are intrinsically sequential – concurrency is not required and therefore we can safely reject multi-threading / context switching.

A simple round-robin scheduler is proposed for both microcontrollers – tasks are placed in a predetermined sequence in a cycle and in every frame time (2 seconds for the primary microcontroller) all the tasks in the cycle are executed one by one. This simple design is thought to be sufficient for our purposes due to the existence of very few event driven / hard-real-time tasks. Downlink is the only "real-time" task – and the handling of this event (the event: entering the field of vision of the ground-station; the handling: downlink initialization) is taken care of exclusively by the secondary microcontroller.

The start of each cycle of the primary microcontroller is triggered by the system clock (implemented using one of the 3 timers). This is the same timer that is used to trigger the switching on of the GPS unit.

The start of downlink will depend on the satellite's position. This is re-evaluated every frame (2 seconds – either by the GPS or by the orbit estimator). The primary microcontroller will command the secondary microcontroller to initiate/terminate downlink.

The secondary microcontroller's cycle is triggered by commands/data received from the primary microcontroller. If health monitoring data is received, a cycle consisting of the following is triggered:

- Receive and buffer all the data sent
- Once all the data has been received, encode data with EDACs and AX25
- Store data in EEPROM

If the command for downlink is received, a different cycle consisting of the following is triggered:

- Read Data from EEPROM and buffer
- Read buffer and set state of GPIO pin accordingly
- Repeat till a “Stop Downlink” command to be received

Thus, while the cycle of the primary microcontroller is timer/clock driven, the cycle of the secondary microcontroller is command (event) driven.

The primary concern in the design of the cyclic scheduler described above is that the total execution time be less than the frame time. Listed in the table below are the execution times of the most computationally intensive blocks of code:

Name of the program	Execution Time (in ms)	Comment
Magnetic Model	285.36	7878 bytes
Optimal to Observation	10	
Matrix Multiply	38	6x6 Matrices
Inverse Matrix	51	6x6 Matrices
Matrix Addition	2.5	6x6 Matrices
Sun model	6.29	
Kalman filter	328.42	

Table 2: Execution Times

The communication/sensor interfacing tasks are not listed above as they occur through on-chip peripherals operating at speeds of the order of 100 kbps.

The total execution time of the tasks comes out to be 1.2 seconds. This is 60 % of the total frame time of 2 seconds.

The other issue is that of preventing microcontroller “hang-ups”. In a typical embedded system application, these hang-ups may be caused by:

- I/O waits – the microcontroller waits for data that never arrives.
  In our system, all data inputs are handled in interrupts and stored in relevant memory locations. In-case a particular input never arrives, the relevant memory location is never updated (updates are done in interrupts) and the microcontroller proceeds with old data. This may lead to incorrect outputs, but the system will still be functional.
- Infinite Loops in code – faulty programming leads to smaller cycles of execution that are never exit.
- In our system, it is proposed to detect these errors during extensive testing/debugging performed on ground.
- Radiation induced errors that lead to latch ups
  It is proposed to shield the system using suitable metal coatings to reduce the probability of such errors. However, a corruption of the Flash memory (in which the program memory is contained) cannot be recovered from.

The atmega128 microcontroller boasts of an on-chip watchdog timer with an independent timer counter. This watchdog timer will be used to reset the microcontroller in case of “lock-ups”.

The watchdog will be “kicked” at the start of each frame of the primary microcontroller. The watchdog will also be loaded with slightly more than the frame time as the reset time. In case of the microcontroller hangs, the watchdog will not be kicked, and it will reset the microcontroller after the reset time.

stages of operation

The lifetime of the satellite will be divided into several stages of operation. In each stage, the onboard computer will be called upon to perform different sets of tasks. Each stage characterizes the state of the state of the satellite. Thus, for a preliminary software design, a state-space was assembled listing out the various states of the satellite and the inputs, outputs and processing tasks associated with each state. This design also includes the paths that the onboard computer will take from one state to another.

Outline of Stages of Operation:

- Stage -1 : Pre-Flight Checks

This stage consists of testing the satellite just before integration onto the launch vehicle. A specific access port will be designed and will provide connection to the TX line of one of the UART peripherals of the primary microcontroller (The GPS does not require commands to be sent – hence only the RX line of the peripheral dedicated to it is required for GPS interfacing, TX is free). This port will also connect to a GPIO pin of the primary microcontroller. During the preflight checks, a logic high will be provided on this line – signaling preflight checks. When the onboard computer boots up for the first time after launch, this line will not be high signaling that launch is over. The tasks performed in this stage are:

  - Check for logic high on a GPIO line on boot
  - Series of routines for testing of sensors and interfaces
  - Power microcontroller interface checked
  - Sensors polled and readings dumped onto UART
  - Communication with 2nd microcontroller and working of EEPROM verified

- Stage 0: Launch

During Launch, the onboard computer will be switched off.

- Stage 1: De-tumbling

  In this stage, a specialized de-tumbling control law is executed. All the sensor interfaces will become active.

  - OBC switched on by Power subsystem
  - Start up configuration of peripherals.
  - De-tumbling Control Law executed till angular velocity of satellite comes below a certain threshold
  - No attempt at down-linking; HM still performed
  - Once angular velocities are below a certain threshold, the GPS is switched on for the first time in order to determine the position of the satellite.

- Stage 2: Nominal Operation

  This stage corresponds to the expected operation of the satellite. All parameters are within normal values, all peripherals and sensors are functional. The satellite is expected to spend the maximum time in this stage.

  - Nominal control law executed
  - Communication check routine executed every 2 seconds
  - Health monitoring, data logging in progress

  This stage is left when either downlink is initiated, or there are multiple failed components

- Downlink Modes

  These modes correspond to when the satellite is either over India or over the ground-station. In either case, data must be broad cast from the satellite. This calls the secondary microcontroller into active use.

  - Commands sent to the power subsystem to turn on monopole (antennae for downlink)
  - Commands sent to secondary microcontroller to perform appropriate action
  - CC1020 configured from scratch – all the registers are programmed by the primary microcontroller.

  - Stage 3a : Attitude Broadcast – required for other universities to collect TEC data
    - Satellite over India but not over GS
    - Attitude data sent to secondary microcontroller every 2 seconds
    - Secondary microcontroller broadcasts attitude data immediately (no storage)
    - Health monitoring data sent to secondary microcontroller for storage as usual

  - Stage 3b: Downlink – required to recover health data from satellite for future reference/diagnostics purposes.

- Satellite over Ground Station
- Secondary microcontroller retrieves logs from EEPROM and transmits to GS
- HM data packets buffered at primary microcontroller till downlink is over; and then sent all at once.

- Disaster Management

  These modes entered when the confirmation bytes sent back from the power microcontroller indicate the failure of one or more components on the satellite.

  - Emergency mode:

    Is entered when the power sub-system is unable to provide sufficient power to supply the onboard computer

    - An Interrupt triggered by logic high on GPIO line from power
    - Indicates microcontroller has just enough time to store current mode of operation into memory (on-chip EEPROM) and prepare for shutdown
    - On reset, microcontroller examines the on-chip EEPROM to determine cause and appropriate mode to enter.

  - Safe Mode:

    This mode is entered when one or more auxiliary components (sensors etc.) fail or when there is low power on satellite in general

    - Triggered by message received from the power microcontroller
    - Components like the GPS may be switched off and OBC will rely on alternatives like the Orbit estimator

    Errors and their compensatory measures:

    1. Failure of GPS: compensated for by relying exclusively on the orbit estimator – this will induce inaccuracies, but the satellite will still be functional.

2. Failure of magnetometer or sun-sensors or magnetorquers: cannot be tolerated, the control law will be shut down and the satellite must rely on being statically stable. The primary microcontroller will only collect health monitoring data and initiate downlinks according to a yet to be decided algorithm.

Some special features:

Whenever a new stage is entered, data will be entered at a special location in the NVRAM. Thus, in case of a reset, the boot-loader can read that memory location and determine the stage of operation at the time of failure. This will prevent unnecessary repetition of certain tasks. It will also enable logging of the error and its circumstance.

Testing and validation

This is the first time that a system of such complexity is being designed by the members of this sub-system. Hence it is anticipated that a lot of time and effort will be spent on the testing and validation of the entire onboard computer system.

The purpose of testing and validation is the following:

1. To ensure that the design and implementation are capable of satisfying the requirements imposed on the sub-system.
2. To reduce the probability of infant mortality.
3. To detect faulty components or incorrect integration/assembly procedures.
4. To ensure that there are no 'bugs' in code that might lead to system crashes or lock-ups.

Both the software and hardware will be subjected to rigorous testing.

Formal Verification

Formal verification consists of the analysis of our system by considering it to be a timed automata and showing that the every required state of the system (consisting of task executed and memory status) is attained at the correct time through the correct paths.

Various facilities for the formal verification and validation of our software are available both inside IIT Bombay as well as elsewhere. However it has been decided that the complexity of, and the time required by such formal methods outweigh their benefits.

Testing methodologies

For the preliminary design review (PDR), the hardware portion of the design was completed and tested. The testing consisted of connecting each of the interacting hardware individually to the onboard computer hardware and checking that each and every interface performed as expected. In this manner,

the hardware and the codes written for the interfacing of the GPS, the magnetometer, the power microcontroller, the external EEPROM and the CC1020 chip were tested out.

It is realized that the integration of various individual components will take as much time as the interfacing of each individual component. The time and complexity of the integration were also considered and integration tests were performed.

To test the integration, a rig was assembled, consisting of the power microcontroller, the 2 onboard computer microcontrollers and the CC1020 chip. Mock Health Monitoring data was requested for then received and passed down to the secondary microcontroller. The storing and encoding of this data into the external EEPROM followed. Then, on receiving a signal from the primary microcontroller, the secondary microcontroller passed the stored HM data to the CC1020 chip whose output was then seen on a CRO.

In this manner, most of the hardware design and its integration was tested out.

Onboard computer In-Loop Simulation (OILS)

OILS (On-board computer In-Loop Simulation), as the name suggests is a setup for simulating the on-board computer with inputs similar to those that it may experience in the satellite. It is part of the second level testing and will help identify bugs and shortcomings of the hardware and software of the On-Board Computer. In short, the objective is as follows:

Objective

To setup an arrangement to supply input to the on-board computer from a computer and verify the outputs against expected response

Approach

The problem consists of the following parts:

1. Hardware interfacing with the computer.

2. Generation of sensor data to be fed in as input to the OBC and verification of the response of the system against the expected response.

Implementation

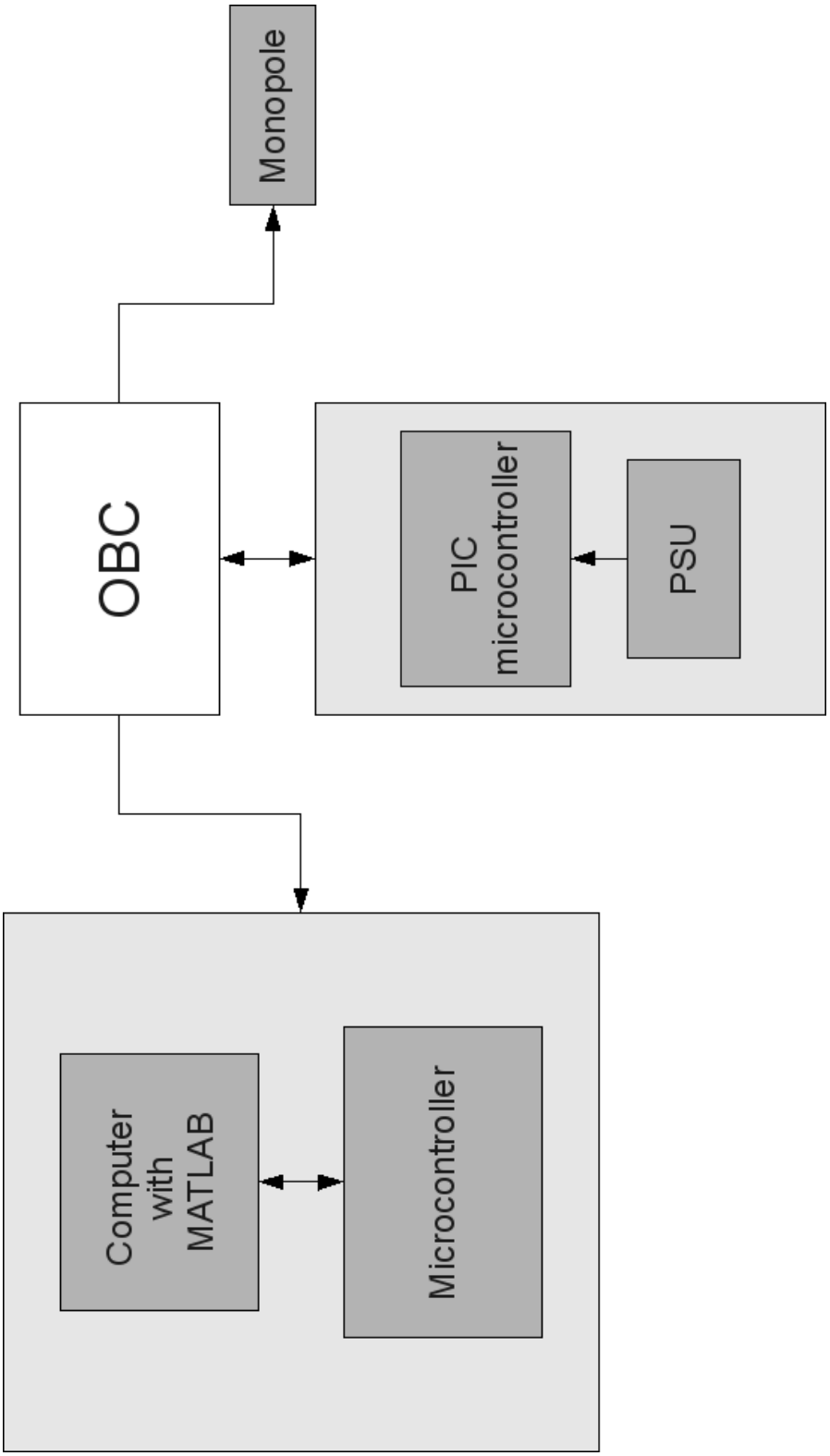

Figure 4: Block Diagram of OILS

As compared to the actual satellite, the following modifications were made for OILS

1. The solar panels were replaced by a Power Supply Unit (PSU).
2. The control components (magnetometer, GPS, magnetorquer and sun-sensors) were replaced by a computer with matlab and a microcontroller.
3. The power board has been replaced by a PIC microcontroller giving out a constant health monitoring message instead of sensor dependent health monitoring that the actual power board does.
4. The access port and the umbilical cord are not part of the OILS setup. However they may be added later.

Interfacing with the computer

- The interfacing with the computer is done through a data acquisition card. The following pin requirements are imposed on the data acquisition card:
    - 12 analog outputs for the sun-sensors.
    - 2 serial ports: 1 each for the magnetometer and the GPS.
    - 3 analog inputs: 1 for each of the magnetorquer.
- Hence, we want to buy a Data Acquisition card with at least 15 analog inputs, 5 analog outputs and 4 serial ports. Other than this, we'd like it to have a high frequency limit.
- MATLAB will be handling the data acquisition card's functioning and writing the system inputs and reading the system outputs.

MATLAB

Data Acquisition using MATLAB

MATLAB has a data acquisition toolbox which works in the following manner:

- Data is acquired and brought up through the particular hardware vendor's software
- The data is handed off to the Data Acquisition Toolbox engine.
- The data is made available in MATLAB.
- The data is run through whatever control algorithm the user designs in MATLAB.
- The data is then routed back to the engine and down through the hardware vendor's software into the board.

Working in real time on MATLAB

One of the major requirements of setting up a realistic environment such as OILS is to provide the inputs real-time in order to ensure that the responses are timely. With this also comes the requirement of testing the system response as fast. This requirement is fulfilled by using one of the following packages:

- Real-Time Windows Target 3.2
- Real-Time Workshop 7.2
- xPC Target 4.0

MATLAB Real-time toolbox

One of the options for a real-time toolbox in MATLAB is Humusoft Extended Real Time Toolbox 3.1. It has the following features:

- Real-time kernel with sampling frequencies up to 66 kHz with no external clock source required
- Data acquisition sampling rate up to the speed of your data acquisition board (even over 1 MHz)
- Simulink block library for creating real-time simulations and control loops

Challenges:

The major challenges to clear are:

Giving Magnetometer signal as a continuous wave and not as a discretely changing input: To fulfill this, we would want to change the magnetometer signal at a higher frequency (about 10 times) than the frequency at which the signal is read.

Secondly, GPS is read through an interrupt, hence we will need to figure out whether we can implement interrupts in the real time toolbox.

Tasks on OILS:

Since the most important task will be to test the OBC, the OBC will be taken through the Operation sequence. The requirement of operation over 320hrs to remove infant mortality will also be done using OILS. Finally the working of the Control Law and the control of the simulated Satellite using actual torques given by the magnetorquer will also be tested.