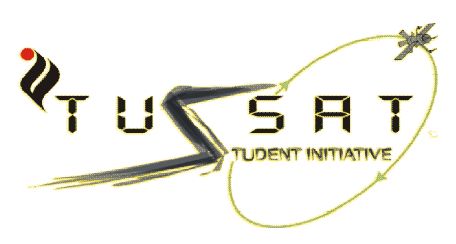
****

**TUSSAT**

**THAPAR UNIVERSITY STUDENT**

**SATELLITE INITIATIVE**

**GENERAL ANALYSIS REPORT**

**ON BOARD COMPUTER**

**OBC: On Board Computer Subsystem**

The On Board Computer Subsystem is the ‘brain’ of the whole autonomous TUSSAT. The

OBC controls the operation of the TUSSAT. The OBC has software installed that manages

the programs written to handle various tasks. The satellite’s operational sequence translates to

a more detailed operational sequence for the On-Board Computer.

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.

**1. The On-Board Computer sub-system places several constraints on**

**other sub-systems:-**

**1) Power: -** The Power sub-system is required to supply continuous power of up to

1Watt(ASSUMPTION) at all times. 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.

**2) 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).

**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

misbehaviour 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.

**2. Modes**

The different operational modes on the OBC are:

• **OFF:** In this mode, the whole subsystem shall be turned off.

• **STAND-BY**: In this mode, only the OBC shall be in low power.

• **INITIALISATION:** This mode is a transient mode after a power on and the OBC shall

initiate all functions.

• **NORMAL:** In this mode the OBC shall be fully operational

 **EMERGENCY:** In this mode, unexpected interrupts or operations are handled.

• **SHUTDOWN:** This mode is a transient mode which led to off mode.

**3. Determination of OBC Hardware Functions**

 The On-Board Computer must interface with the Power sub-system to receive

information about the health (power on/off) of the various components of the satellite.

 It must interface with the Attitude determination and Control sub-system to stabilize

the satellite.

 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.

**4. DESIGN DRIVERS**

**ENVIRONMENT**

During the preparation for launch and then during the flight, the spacecraft is exposed to a

variety of mechanical, thermal, and electromagnetic environments. This part of the report

provides a description of the environment that the spacecraft is intended to withstand.

**Radiation effects on electronics**

Radiation in space is generated by particles emitted from a variety of sources both within and

beyond our solar system. The nature of the radiation differs in place and time. In some places

TUSSAT might cross the so-called ‘solar wind’, which is a burst of particles (mostly protons

and electrons) emitted from the sun, and here radiation will be substantial. In other places

radiation will be very light. The single particles also vary in the amount of energy they

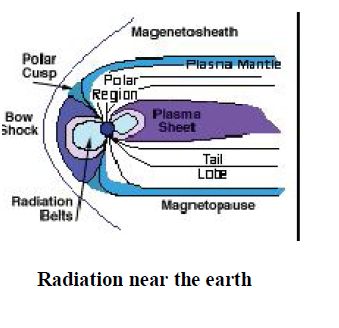
possess. High-energy particles are more dangerous to OBC circuitry than low-energy

particles (often referred to as background radiation), as they penetrate deeper into the

components and cause greater damage in case of collision. The radiation is not only

depending on the sun but also very much on the magnetic field of earth (magnetosphere).The

orbit of TUSSAT is so low that one will not touch the Van-Allen belts (turquoise zone).



**4.1. Single event effects**

A single-event effect results from, a single, energetic particle. If such a particle hits one of

OBC components, TUSSAT might encounter three different errors:

**a) Bit-flips (soft error)**

This phenomenon causes the value of a specific bit to change. This will lead to software

malfunction. In order to avoid this malfunction we plan to implement error detection and

correction circuitry (EDAC) between the processor and the memory chips. This will be able

to correct most errors. In case of substantial damage to the software, it might be necessary to

reboot or even upload new software from earth. The ROM has to be radiation-hard so that it

does not suffer from these bit-flips.

**b) Latch-up (hard error)**

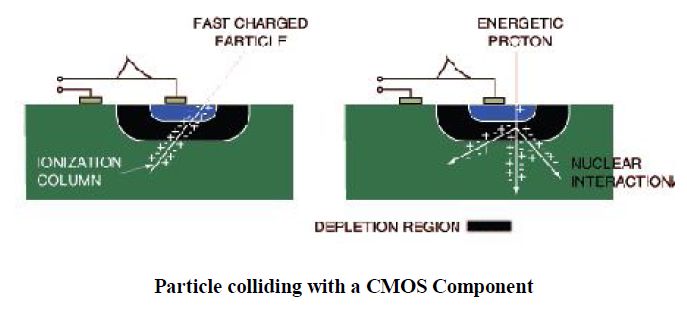
In this a part of a CMOS component is been hit by a high-energy particle. This particle can

either short-circuit the source or drain to ground. The short circuit arises because the particle

creates a conductive ‘tunnel’. A short circuit will draw a lot of current and the component is

very likely to take permanent damage. The issue is to include a latch-up protection circuitry

near the sensible component to turn off the power immediately.

**c) Burn-out (hard error)**

In case of the power not being turned off at a latch-up, a burn-out can occur. A burn-out can

sometimes be seen by the human eye, as a burn-out means that the chips is simply melted in

the area of the latch-up. After this, the chip is useless.

**4.2. Total ionizing dose (TID)**

The total ionizing dose, mostly due to electrons and protons, can result in device failure. TID

is measured in terms of the absorbed dose. The TID is calculated from the trapped protons

and electrons, photons, and solar flare protons. As TID increase, material degradation

increase. Long-term exposure can cause device threshold shifts, increased device leakage and

power consumption, timing changes, decreased functionality, etc.

In order to have an idea of how OBC components will react to these long-term effects, a

sample of component should be tested. The amount of radiations is measured in kRad. 5 to 10

kRad is the average amount of radiation we can expect in one year on the orbit used by

TUSSAT. The duration of the spatial mission is set as six months. The components should so

be able to work with a TID of 3 kRad. This factor should not play a major role for TUSSAT’s

short mission; nevertheless the component must be tested.

The only thing that can be done to limit these long-term effects is to place shielding material

around the components. This could for example be aluminium, which is both light and well

shielding. This kind of shielding will not stop any of the high-energy radiation that causes

latch-ups and bit-flips.

**MODULES**

On Board Computer subsystem has been divided into two modules:-

**Hardware**

**Software**

**Operating System:**

Operating system (OS) selection is a pivotal aspect of the satellite development and

operational environments. Embedded programming is split in two by the consideration of solution's implementation: with and without Real Time Operating System. When the application becomes very complex, when we have a lot of tasks that need to run very strictly and when we have a lot of events that may come asynchronously from outside, the logic is hard to accomplish. Here comes in scene the second category of embedded applications: using a real time operating system (RTOS).We considered the these operating systems:

**Linux:**

Linux is a flexible Unix–like operating system that can run on a host of platforms from small embedded computers (1–2 MB with no memory management unit) to super computers.

We considered linux for many reasons that any Embedded project benefits from open source code. Linux provided a platform that allowed the operational software to be built and tested on the specific test environment and any generic Linux-based workstation. This supported a distributed development environment that required a minimum overhead to set up and support. It also gave the developers a common set of development tools and a vast working, documented, and tested code base that could be harvested over the Internet for free.

Another advantage was that several key device drivers were already written for AX.25

(packet protocol) .

Linux also provides a host of tools that were utilized in the operational satellite without

burdening the software development team with coding, testing, and maintenance overhead.

These include free utilities that report memory use, disk space utilization, software status, and date/time manipulation. The utilities were easily integrated into a set of diagnostic

scripts that were used to report status to the contact software. Other useful utilities included

compression (bzip2), data integrity checking (md5sum), and shell utilities like cat, grep,

bash, etc.

The major disadvantages of using an OS like Linux are that

(a) The great flexibility requires much more testing (there can be a high number of corner cases), and

(b) There is limited time-tagging accuracy .

(c) The problem is that a kernel like Linux requires the support of many files that work together and must be present in a valid state. With such a small team never had the time or developed the code necessary to validate all of the Linux files necessary to operate the payload software. It should be pointed out that Linux tools exist to solve this problem; one only needs time.

**Salvo:**

Salvo is a proven, powerful, high-performance and royalty-free real-time operating system (RTOS) that requires very little program and data memory, and no task stacks. It is an easy-to-use software tool to help you quickly create powerful, reliable and sophisticated applications (programs) for embedded systems. Salvo was designed from the ground up for use in microprocessors and microcontrollers with severely limited resources, and will typically require from 5 to 100 times less memory than other RTOSes. In fact, Salvo's memory requirements are so minimal that it will run where no other RTOS can. Salvo is ROMable, easily scaleable and extremely portable. It runs

on just about any processor, from a PIC to a Pentium.

Salvo is a purely event-driven cooperative multitasking RTOS, with full support for event and timer services. Multitasking is priority- based, with sixteen separate priority levels supported. Tasks that share the same priority will execute in a round-robin fashion. Salvo provides services for employing semaphores (binary and counting), messages, message queues and event flags for intertask communications and resource management. A full complement of RTOS functions (e.g. context-switch, stop a task, wait on a semaphore, etc.) is supported. Timer functions, including delays, timeouts

and cyclic timers, are also supported. The Salvo RTOS by Pumpkin is a lightweightsystem designed for use on small embedded systems.Because the PIC24H, the considered microarchitecture, has hardware limitations that make it difficult to use an RTOS, Salvo was a perfect fit for the CDH hardware. It allows for a cooperative multitasking environment, which greatly simplifies the software organization. Each major subsystem (thermal, power, commands, sensors, and communications) has one or more tasks to handle specific functionality for the system, which allows the system to be modularized and easily debugged.

**5. Hardware Overview**

The primary component of the OBC is microcontrollers. The secondary components are

EPROM, FLASH and SRAM. These components were used for storing the software and

were built-in to the microcontroller. The OBC HW consists of a minimum flash memory,

I2C, UART and SPI bus interface and external logic to control the memory modules and the

payload sensors.

**WATCHDOG TIMER**

The one thing probably that limits the design of the architecture the most, is the spaceoperating

demand. As no one will be able to reset the processor manually in case of a

software malfunction, the system must be able to do this by itself. The way is done is to use a

watchdog timer. The timer is increased on a regular basis (typically it is clocked by the

master clock) and when it reaches a certain value, it resets the processor. Of course this is not

what we want when the computer is working correctly. Therefore a part of the software must

reset the watchdog before it reaches the reset value. This is done when the software is

working correctly, but if it gets in a deadlock the watchdog is not reset, and therefore it resets

the system. The watchdog can either be build-in in the processor or it can be a separate unit.

**MEMORY**

The three different types of memory are chosen because of the space-operating nature of this

architecture: ROM, flash, and RAM.

In space, the radiation may lead to bit-flips in memory after which the software may do

unpredictable things. The watchdog will obviously reset the processor if this happens, and the

processor starts loading the boot-software. It is essential that this software always work

correctly, and therefore it must be stored in a memory, where bit-flips do not occur. Some

types of ROM (read only memory) have this feature.

The flash will be used to store the operating system and other software. Some of this software

could be store in ROM, but as it might be necessary to change parts of it, when the TUSSAT

is in space, it will be stored in flash.

The RAM (random access memory) will be used as a temporary memory when the programs

run, and as a place where measured values can be stored. As both flash and RAM suffer from

bit-flips it might be desired to have some error detection and correction circuitry (EDAC)

between these memories and the processor.

**BUSES**

In order to communicate with other elements, the computer must provide different interface,

a bus interface to exchange data with other subsystem, the bus selection will be covered in a

later section, both analogue and digital interfaces. Since some processors may not be able to

supply very much current it might be necessary to insert drivers between the digital interface

and the processor.

**CLOCK**

Depending on which processor is chosen we may have to add a real time clock (RTC) to the

board, but if the processor has enough timers, this can also be implemented in software. The

purpose of the real time clock is to make it possible to schedule tasks.

**CONTROL LATCH**

As we described above, the CMOS component are sensible to latch-up. To avoid burn-out, a

latch-up protection circuitry must be implemented between power bus and sensible

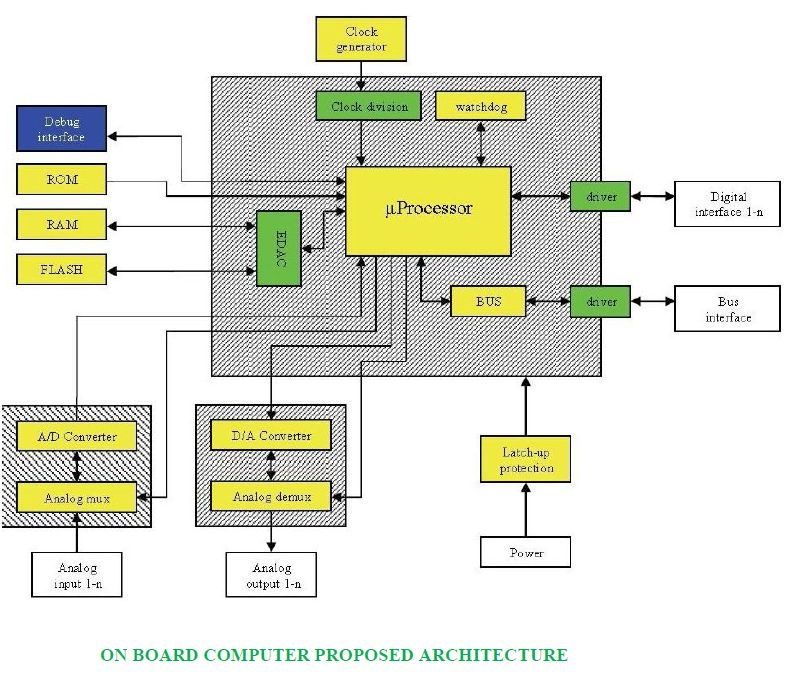
component. When a latch-up occurs the power must be turn off immediately.

**The debug interface**

The purpose of this is to make it possible to find and correct errors in the hard- and software

design. It also makes it possible to upload new software to the flash. It consists of measuring

points for important signals and some sort of interface to the processor and the flash.



**5.1. Choice of components**

This section describes the considerations we have made when choosing components

for the on board computer. These components are the MCU, external memory and

control logic. To improve the odds for success of the satellite’s development and

operational tasks, it is custom to choose components of great reliability for satellites.

To be absolutely sure of reliability, it is not uncommon to buy components that have

actually been tested in space.

A cheaper alternative is to choose a Commercial off-the-shelf (COTS) component that

has been used in space before. Another component just like it will probably work in

space too. The only problem with this alternative is, that development of satellites is a

long process, and when you find a component that has been successfully used in

space, it is most likely more than two or three years old. In these days, where new and

better technologies revolutionize the chip-marked every other day, it is very tempting

to choose new and more powerful components for your design. OBC team has

collectively agreed to yield to this temptation. Furthermore, tests will be performed to

verify that the components do not collapse with the first signs of space radiation.

**5.1.1. Analysis criteria**

In order to choose the perfect (or at least the best) processor, we have evaluated the

different processors in different categories and summed up the information in the table

below. The different categories have different weights, as they are not equally

important. In some categories, the rating has been made from a reference value, which

we have set from average specification of the component. The categories are:

1) **Radiation: -** When not protected by Earth’s magnetic field the satellite will be

exposed to high level of radiation. When electronics is exposed to radiation

three things will happen:

1. Slowly the chip will degrade until finally not working.

2. Bit errors occurring when a bit flips.

3. Latch up resulting in a short circuit.

By using code correction (e.g. Hamming) and outer protection (e.g. metal protection)

we minimize the amount of problems due to radiation without the cost of special space

approved components.

2) **Size: -** Since we are limited in weight and volume we will try to use compact

SMD components.

.

3) **Standby power consumption:** During the scheduling of the mission, some

subsystems will not be used. In order to save power those subsystems will be in

standby mode. The reference value has been set at 3[mW].

4) **Temperature Range:** The temperature will differ significantly depending on the

location of the satellite. In the sunny side of earth, the surface facing the sun will be

very warm and in the shadow of earth, everything will get very cold. The temperature

range has been set to 0°C to 40°C for normal operation. But in special case the

temperature might exceed this range in both directions, which is why we would like to

have components with great temperature tolerances. Luckily, it is normal for

military/industrial components to be operational from -40°C to 80°C, which we

believe is sufficient.

5) **Already used in space:** As the name says, by ‘already used in space’, we mean

whether or not the processor has been tried in space before, and if this processor has

work properly.

6) **I/O compatibility:** As the sensors are not yet defined we made the assumption that

5[V] tolerant I/O which is a standard voltage will be appreciable.

7) **MIPS:** We made the assumption that a microcontroller which provides between 10 to

20 MIPS will be sufficient to make the processing of the ADCS. The reference value

for 8 bit microcontroller is set to 1 to 10 MIPS.

.

8) **Multiplier:** As some processing will require multiplication, it should be interesting to

have an integrated multiplier, but not required.

9) **Digital and Analogical I/O number:** All sensors and actuators need access to the

processor via the I/O pins of the processor, it is preferable to have sufficient I/O pin in

order to connect all the peripherals. We made the assumption that 12 digital I/O and 8

A/D converters will be sufficient.

10) **UART:** The UART component will be used to program the chip on the earth during

the debug and test phase, and probably for the survival and science bus during spatial

mission.

11) **Development environment:** In order to have a fully functional microcontroller

software must be developed and implemented. This criterion compares the facilities of

development environment provided by the manufacturer. It is really important to have

powerful tools to manage the hardware in order to debug software.

5.2. **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

**5.2.1. Specifications**

To assist in selecting the right microcontroller (processor) and to meet the requirements

of the OBC, a set of minimum specifications was developed and is listed below:

**Data and Non volatile Program Memory:**

• EEPROM (***E****lectrically* ***E****rasable* ***P****rogrammable* ***R****ead* ***O****nly* ***M****emory*) – min. 8 kb

• Flash - minimum 188 kb

• SRAM (***S****tatic* ***R****ead* ***O****nly* ***M****emory*) - minimum 4 kb

**Desirable features**

• High processing speed – more than 10 MHz

• In-built Analog/Digital Converters

• Programmable UART (***U****niversal* ***A****synchronous* ***R****eceiver-****T****ransmitter*)

• Master/Slave SPI Serial Interface

• I2C bus interface

• Controllable I/O pins

• Programmable timers, especially the watch-dog timer

• Minimum 8 bit architecture

• Avoid Ball-grid-array (BGA) microcontrollers (difficult to solder)

**Low Power consumption** – less than 10 mA and voltage less than 5 volts

**Size –** should fit on a 5 cm x 5 cm Printed Circuit Board

**C compiler** - must be available for the processor (microcontroller)

**Operable in temperature range 0 – 40° C**; however, wider range is preferable.

Next, several microcontrollers were investigated that best met the specifications set above.

Table below lists the microcontrollers (that meet the minimum specifications) that were

examined along with the reason that they were rejected or selected.

|  |  |
| --- | --- |
| **COMPANY** | **Reason for Rejection or Selection** |
| **Motorola** | Hard to solder because of BMG configuration |
| **Hitachi** | Programming skills very limited |
| **Intel** | High power consumption |
| **Microchip** | Nothing Wrong |
| **PIC** | Nothing Wrong |
| **ATMEL** | Nothing Wrong |

**Table: Microcontrollers list and the reason for their rejection**.

**5.3. EEPROM**

EEPROM (also written E2PROM and pronounced "e-e-prom," "double-e prom" or simply "esquared")

stands for Electrically Erasable Programmable Read-Only Memory and is a type

of non-volatile memory used in computers and other electronic devices to store small

amounts of data that must be saved when power is removed, e.g., calibration tables or device

configuration.

EEPROM-store small amounts of data that must be saved when power is removed

When larger amounts of static data are to be stored (such as in USB flash drives) a specific

type of EEPROM such as flash memory is more economical than traditional EEPROM

devices.

There are different types of electrical interfaces to EEPROM devices. Main categories of

these interface types are:

 Serial bus

 Parallel bus

 **Serial bus devices**

Most common serial interface types are SPI, I²C, Microwire, UNI/O, and 1-Wire.

These interfaces require between 1 and 4 control signals for operation, resulting in a

memory device in an 8 pin (or less) package.

The serial EEPROM typically operates in three phases: OP-Code Phase, Address

Phase and Data Phase. The OP-Code is usually the first 8-bits input to the serial input

pin of the EEPROM device (or with most I²C devices, is implicit); followed by 8 to

24 bits of addressing depending on the depth of the device, then data to be read or

written.

 **Parallel bus devices**

Parallel EEPROM devices typically have an 8-bit data bus and an address bus wide enough to

cover the complete memory. Most devices have chip select and write protect pins. Some

microcontrollers also have integrated parallel EEPROM.

Operation of a parallel EEPROM is simple and fast when compared to serial EEPROM, but

these devices are larger due to the higher pin count (28 pins or more) and have been

decreasing in popularity in favor of serial EEPROM or Flash.

Flash memory is a later form of EEPROM. In the industry, there is a convention to reserve

the term EEPROM to byte-wise erasable memories compared to block-wise erasable flash

memories. EEPROM takes more die area than flash memory for the same capacity because

each cell usually needs both a read, write and erase transistor, while in flash memory the

erase circuits are shared by large blocks of cells (often 512×8).

EEPROM can be programmed and erased electrically using field electron emission(more

commonly known in the industry as "Fowler–Nordheim tunneling").

NOR Flash memory is a hybrid style—programming is through hot carrier injection and

erase is through Fowler–Nordheim tunneling.

EEPROM stands for Electrically Erasable Programmable Read-Only Memory. An EEPROM

is like an EPROM chip since it can be written in or programmed more than once. Unlike the

EPROM chip, however, an EEPROM chip need not be taken out of the computer or

electronic device of which it is part when a new program or data needs to be written on it.

Selective programming can be done to an EEPROM chip. The user can alter the value of

certain cells without needing to erase the programming on other cells. Thus, sections of data

can be erased and replaced without needing to alter the rest of the chip's programming.

Data stored in an EEPROM chip is permanent, at least until the user decides to erase and

replace the information it contains. Furthermore, the data stored in an EEPROM chip is not

lost even when power is turned off.

LIMITATIONS

While the EEPROM can be reprogrammed, the number of times it can be altered is limited.

This is the main reason why EEPROM chips are popular for storing only configuration data

such as the computer's BIOS code which doesn't require frequent reprogramming. The oxide

insulating layer can be damaged by frequent rewrite. Modern-day EEPROMs can be

rewritten up to a million times.

The EEPROM required by us needs to have a size of 1Mb.

5.4. **FLASH MEMORY**

Flash memory is a non-volatile computer memory that can be electrically erased and

reprogrammed. It is a technology that is primarily used in memory cards andUSB flash

drives for general storage and transfer of data between computers and other digital products.

It is a specific type of EEPROM (Electrically Erasable Programmable Read-Only Memory)

that is erased and programmed in large blocks; in early flash the entire chip had to be erased

at once. Flash memory costs far less than byte-programmable EEPROM and therefore has

become the dominant technology wherever a significant amount of non-volatile, solid

state storage is needed. Example applications include PDAs (personal digital assistants),

laptop computers, digital audio players, digital cameras and mobile prhones. It has also

gained popularity in the game console market, where it is often used instead of EEPROMs or

battery-powered SRAM for game save data.

Since flash memory is non-volatile, no power is needed to maintain the information stored in

the chip. In addition, flash memory offers fast read access times(although not as fast as

volatile DRAM memory used for main memory in PCs) and better kinetic shock resistance

than hard disks. These characteristics explain the popularity of flash memory in portable

devices. Another feature of flash memory is that when packaged in a "memory card," it is

enormously durable, being able to withstand intense pressure, extremes of temperature, and

even immersion in water.

Although technically a type of EEPROM, the term "EEPROM" is generally used to refer

specifically to non-flash EEPROM which is erasable in small blocks, typically bytes. Because

erase cycles are slow, the large block sizes used in flash memory erasing give it a significant

speed advantage over old-style EEPROM when writing large amounts of data.

**5.4.1. Serial flash**

Serial flash is a small, low-power flash memory that uses a serial interface, typically SPI, for

sequential data access. When incorporated into an embedded system, serial flash requires

fewer wires on the PCB than parallel flash memories, since it transmits and receives data one

bit at a time. This may permit a reduction in board space, power consumption, and total

system cost.

There are several reasons why a serial device, with fewer external pins than a parallel device,

can significantly reduce overall cost:

 Many ASICs are pad-limited, meaning that the size of the die is constrained by the

number of wire bond pads, rather than the complexity and number of gates used for

the device logic. Eliminating bond pads thus permits a more compact integrated

circuit, on a smaller die; this increases the number of dies that may be fabricated on

a wafer, and thus reduces the cost per die.

 Reducing the number of external pins also reduces assembly and packaging costs. A

serial device may be packaged in a smaller and simpler package than a parallel

device.

 Smaller and lower pin-count packages occupy reduced PCB area.

 Lower pin-count devices simplify PCB routing.

Flash memory required by us will be of the size of 128 KB. The Amtel 128 microcontroller

provides an inbuilt 128 K flash memory which is sufficient for data storage on the onboard

computer system.

If however we were to employ a controller that does not have a built in flash memory, we

shall need to install it externally.

It is a well known fact that flash memory is highly susceptible to corruption due to exposure

to radiation. Data stored in the flash memory will be completely lost and cannot be recovered

if the memory is corrupted. Hence we will need to ensure adequate protection from radiation

for the flash memory, or ensure that the exposure period is not too long so that the memory

can last the minimal mission duration of 9 months.

**5.5. SRAM**

Random-access memory (usually known by its acronym, RAM) is a form of computer data

storage. Today, it takes the form of integrated circuits that allow stored data to be accessed in

any order (i.e., at *random*). The word *random* thus refers to the fact that any piece of data can

be returned in a constant time, regardless of its physical location and whether or not it is

related to the previous piece of data.

By contrast, storage devices such as tapes, magnetic discs and optical discs rely on the

physical movement of the recording medium or a reading head. In these devices, the

movement takes longer than data transfer, and the retrieval time varies based on the physical

location of the next item.

The word RAM is often associated with volatile types of memory (such as DRAM memory

modules), where the information is lost after the power is switched off. Many other types of

memory are RAM, too, including most types of ROM and flash memory called *NOR-Flash*.

The computer needs RAM as working-memory in the execution of programs. There are

generally two types of RAM: Dynamic and static. Dynamic RAM (DRAM) is characterized

by its need for a clock in order to dynamically update the contents of the RAM in each clock

cycle. You can clock dynamic RAMs with very high frequencies – for example 100 MHz.

The expense of this capability is that dynamic RAM consumes more power, and since we

plan to run at low frequencies (which lowers the power consumption) we turned to look at

static RAM. Static RAM (SRAM) is not refreshed in each clock cycle (hence the name) and

communications are performed by a relatively simple protocol (for further information see

section on ”Timing Analysis”). Power consumption of static RAMs is quite low, which is

very important as the RAM will be active at almost all times.

To compare the possible choices of static RAM we must look at the following categories:

· *Size*

As the requirement specification states a minimum of 512 KB is needed – preferably 1 MB.

1 MB would be very good, as it would allow the computer to store more than a single

picture from the camera.

· *Supply voltage*

A supply voltage in same range as that of the processor would be preferred.

· *Active current / Stand-by current / Power dissipation*

Active current is the current drawn during reads and writes, stand-by current is the current

drawn when the component is passive. Power dissipation is calculated by multiplying active

current and the supply voltage.

All microcontrollers already have built in RAMs in them. Depending in the ultimate mission

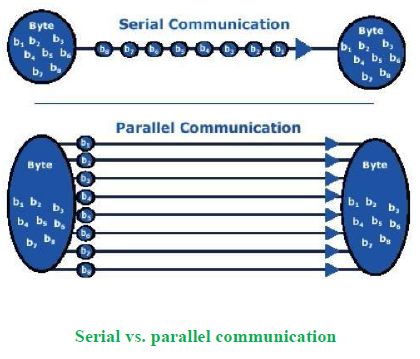
of the satellite and its payload, we will need to check whether the onboard RAM is sufficient

for the required work or external banks of RAM need to be added to the computer system.

**5.6. Type of bus**

In computer architecture, a bus is a subsystem that transfers data or power between computer

components inside a computer or between computers.

****

In a digital communications system, there are two methods for data transfer: parallel and

serial. Parallel connections have multiple wires running parallel to each other (hence the

name), and can transmit data on all the wires simultaneously. Serial, on the other hand, uses a

single wire to transmit the data bits one at a time.

It is a natural question to ask which one of the two transmission methods is better. At first

glance, it would seem that parallel ports should be able to send data much faster than serial

ports. Let's say we have a parallel connection with 8 data wires, and a serial connection with

a single data wire. Simple arithmetic seems to show that the parallel system can transmit 8

times as fast as the serial system.

However, parallel ports suffer extremely from inter-symbol interference (ISI) and noise, and

therefore the data can be corrupted over long distances. Also, because the wires in a parallel

system have small amounts of capacitance and mutual inductance, the bandwidth of parallel

wires is much lower than the bandwidth of serial wires. An increased bandwidth leads to a

better bit rate. We also know that less noise in the channel means we can successfully

transmit data reliably with a lower signal to noise ratio (SNR). In addition, because of the

increased potential for noise and interference, parallel wires need to be far shorter than serial

wires. Moreover, as the bus will connect the different subsystems, it will be preferable to use

less wire than possible.

In conclusion, for TUSSAT’s use, it is preferable to use a serial link between the subsystems.

**5.6.1. Bus candidates**

As shown before, the serial link will provides the best choice for our needs. All the following

bus use serial link. The descriptions following are only summaries which present the main

features of the bus.

**SPI BUS**

We are using it in following:

1. SPI BUS:

- Bidirectional (uses SPI protocol)

- 1 permanent master (Microcontroller ) and Slave(CC1020)

Used to program/initialize CC1020 after launch

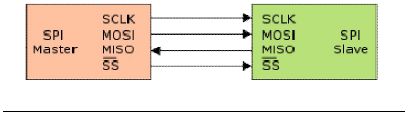
EXPLANATION

The **Serial Peripheral Interface Bus** or **SPI** bus is a synchronous serial data link that

operates in full duplex mode. Devices communicate in master/slave mode where the master

device initiates the data frame. Multiple slave devices are allowed with individual slave select

(chip select) lines.



INTERFACE

The SPI bus specifies four logic signals.

 SCLK — Serial Clock (output from master)

 MOSI/SIMO — Master Output, Slave Input (output from master)

 MISO/SOMI — Master Input, Slave Output (output from slave)

 SS — Slave Select (active low; output from master)

OPERATION

The SPI bus can operate with a single master device and with one or more slave devices.

To begin a communication, the master first configures the clock, using a frequency less than

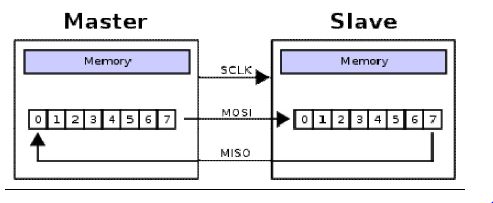
or equal to the maximum frequency the slave device supports. The master then pulls the slave

select low for the desired chip.

During each SPI clock cycle, a full duplex data transmission occurs:

 the master sends a bit on the MOSI line; the slave reads it from that same line

 the slave sends a bit on the MISO line; the master reads it from that same line



EXPLANATION

Transmissions normally involve two shift registers of some given word size, such as eight

bits, one in the master and one in the slave; they are connected in a ring. Data is usually

shifted out with the most significant bit first, while shifting a new least significant bit into the

same register.

After that register has been shifted out, the master and slave have exchanged register values.

Then each device takes that value and does something with it, such as writing it to memory.

If there is more data to exchange, the shift registers are loaded with new data and the process

repeats.

Transmissions may involve any number of clock cycles. When there is no more data to be

transmitted, the master stops toggling its clock. Normally, it then deselects the slave.

**UART**

USED IN:

UART BUSES

- 2 independent bidirectional UART lines

- GPS line

o Used to configure GPS

o Used to poll for and receive GPS data at regular intervals

- Magnetometer line

o Used to program magnetometer after launch

o Used to poll for and receive magnetic field data at regular intervals

OPERATION

A **universal asynchronous receiver/transmitter** is a type of "asynchronous

receiver/transmitter", a piece of computer hardware that translates data between parallel and

serial forms. UARTs are commonly used in conjunction with other communication standards

such as EIA RS-232.

A UART is usually an individual (or part of an) integrated circuit used for serial

communications over a computer or peripheral device serial port, UART are now commonly

included in microcontrollers.

The UART takes bytes of data and transmits the individual bits in a sequential fashion. At the

destination, a second UART re-assembles the bits into complete bytes. Each UART contains

a shift register which is the fundamental method of conversion between serial and parallel

forms.

External signals may be of many different forms. Examples of standards for voltage signaling

are RS-232, RS-422 and RS-485 from the EIA.

UARTs are commonly used with RS-232 for embedded systems communications. It is useful

to communicate between microcontrollers and also with PCs.

UARTs send a "start" bit, five to eight data bits, least-significant-bit first, an optional "parity"

bit, and then one, one and a half, or two "stop" bits. The start bit is the opposite polarity of the

data-line's idle state. The stop bit is the data-line's idle state, and provides a delay before the

next character can start. (This is called asynchronous start-stop transmission). In mechanical

teletypes, the "stop" bit was often stretched to two bit times to give the mechanism more time

to finish printing a character.

Asynchronous transmission allows data to be transmitted without the sender having to send a

clock signal to the receiver. special bits are added to each word which are used to

synchronize the sending and receiving units.

The Start Bit is used to alert the receiver that a word of data is about to be sent, and to force

the clock in the receiver into synchronization with the clock in the transmitter.

After the Start Bit, the individual bits of the word of data are sent, with the Least Significant

Bit (LSB) being sent first. Each bit in the transmission is transmitted for exactly the same

amount of time as all of the other bits, and the receiver “looks” at the wire at approximately

halfway through the period assigned to each bit to determine if the bit is a 1 or a 0. For

example, if it takes two seconds to send each bit, the receiver will examine the signal to

determine if it is a 1 or a 0 after one second has passed, then it will wait two seconds and then

examine the value of the next bit, and so on.

When the entire data word has been sent, the transmitter may add a Parity Bit that the

transmitter generates. The Parity Bit may be used by the receiver to perform simple error

checking. Then at least one Stop Bit is sent by the transmitter.

If the Stop Bit does not appear when it is supposed to, the UART considers the entire word to

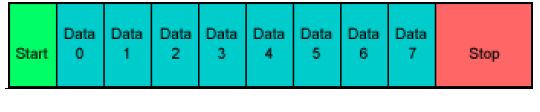
be garbled and will report a Framing Error to the host processor when the data word is read.

UART automatically discards the Start, Parity and Stop bits.

If another word is ready for transmission, the Start Bit for the new word can be sent as soon

as the Stop Bit for the previous word has been sent. Because asynchronous data is “self

synchronizes”, if there is no data to transmit, the transmission line can be idle.



The start bit is always a **0** (logic low), which is also called a **space**. The start bit signals the

receiving DTE that a character code is coming. The next five to eight bits, depending on the

code set employed, represent the character. In the ASCII code set the eighth data bit may be a

parity bit. The next one or two bits are always in the **mark** (logic high, i.e., '1') condition and

called the stop bit(s). They provide a "rest" interval for the receiving DTE so that it may

prepare for the next character which may be after the stop bit(s).

TRANSMISSION FROM UART

Transmission operation is simpler since it is under the control of the transmitting system. As

soon as data is deposited in the shift register, the UART hardware generates a start bit, shifts

the required number of data bits out to the line,generates and appends the parity bit (if used),

and appends the stop bits. Since transmission of a single character may take a long time

relative to CPU speeds, the UART will maintain a flag showing busy status so that the host

system does not deposit a new character for transmission until the previous one has been

completed; this may also be done with an interrupt. Since full-duplex operation requires

characters to be sent and received at the same time, practical UARTs use two different shift

registers for transmitted characters and received characters.

Transmitting and receiving UARTs must be set for the same bit speed, character length,

parity, and stop bits for proper operation. The receiving UART may detect some mismatched

settings and set a "framing error" flag bit for the host system; in exceptional cases the

receiving UART will produce an erratic stream of mutilated characters and transfer them to

the host system.

The four following bus are based on the UART component, but used with different drivers.

 **Driver RS232**: The RS232 specification defines Mechanical, Electrical, and

Functional characteristics. RS232 is an Unbalanced (Single Ended), unidirectional

(point-to-point) interface. Signal is referenced to ground. RS232 drivers feature a

controlled slew rate. Normal output levels are +5 [V]. RS232 use asynchronous

framing with a known data width of 8bits, and NRZ (non-return to zero) encoding.

 **Driver RS422:** RS422 define a balanced (differential) interface; specifying a single,

unidirectional driver with multiple receivers (up to 32). RS422 will support Point-to-

Point, Multi-Drop circuits, but not Multi-point.

 **Driver RS485:** RS485 define a balanced (differential) interface; defines the Physical

layer (OSI Layer 1), signalling protocol is not defined. RS485 specifies bidirectional,

half-duplex data transmission. Up to 32 transmitters and 32 receivers may be

interconnected in any combination, including one driver and multiple receivers

(multi-drop), or one receiver and multiple drivers. RS485 is the Multi-Point version of

RS422.

 **Driver LVDS EIA-644:** LVDS as Low Voltage Differential Signalling defines the

electrical layer only. The EIA- 644 provides a point to point topology with differential

interface.

**I2 C BUS**

We are using in:

1. POWER STATUS COMMUNICATION BUS

- Bidirectional I2C bus

- Microcontroller 1 is always the master with the power microcontroller always the slave

o Current status (on/off) of various loads on power bus ,sent from power microcontroller

2. EEPROM R/W BUS

- Bidirectional I2C bus

- Used to store/retrieve house-keeping data from EEPROM

OPERATION

**I²C** (**Inter-Integrated Circuit**) is a multi-master serial computer bus that is used to attach

low-speed peripherals to a motherboard, embedded system, or cellphone.

A sample schematic with one master (a microcontroller), three slave nodes (an ADC, a DAC,

and a microcontroller), and pull-up resistors Rp

I²C uses only two bidirectional open-drain lines, Serial Data (SDA) and Serial Clock (SCL),

pulled up with resistors. Typical voltages used are +5 V or +3.3 V although systems with

other, higher or lower, voltages are permitted.

The I²C reference design has a 7-bit address space with 16 reserved addresses, so a maximum

of 112 nodes can communicate on the same bus. The most common I²C bus modes are the

100 kbit/s *standard mode* and the 10 kbit/s *low-speed mode*, but clock frequencies down to

DC are also allowed. Recent revisions of I²C can host more nodes and run faster (400 kbit/s

*Fast mode*, 1 Mbit/s *Fast mode plus* or Fm+, and 3.4 Mbit/s *High Speed mode*), and also

support other extended features, such as 10-bit addressing.

he reference design, as mentioned above, is a bus with a clock (SCL) and data (SDA) lines

with 7-bit addressing. The bus has two roles for nodes: master and slave:

 Master node — node that issues the clock and addresses slaves

 Slave node — node that receives the clock line and address.

The bus is a multi-master bus which means any number of master nodes can be present.

Additionally, master and slave roles may be changed between messages (after a STOP is

sent).

There are four potential modes of operation for a given bus device, although most devices

only use a single role and its two modes:

 master transmit — master node is sending data to a slave

 master receive — master node is receiving data from a slave

 slave transmit — slave node is sending data to a master

 slave receive — slave node is receiving data from the master

The master is initially in master transmit mode by sending a start bit followed by the 7-bit

address of the slave it wishes to communicate with, which is finally followed by a single bit

representing whether it wishes to write(0) to or read(1) from the slave.

If the slave exists on the bus then it will respond with an ACK bit (active low for

acknowledged) for that address. The master then continues in either transmit or receive mode

(according to the read/write bit it sent), and the slave continues in its complementary mode

(receive or transmit, respectively).

The address and the data bytes are sent most significant bit first. The start bit is indicated by a

high-to-low transition of SDA with SCL high; the stop bit is indicated by a low-to-high

transition of SDA with SCL high.

If the master wishes to write to the slave then it repeatedly sends a byte with the slave

sending an ACK bit. (In this situation, the master is in master transmit mode and the slave is

in slave receive mode.)

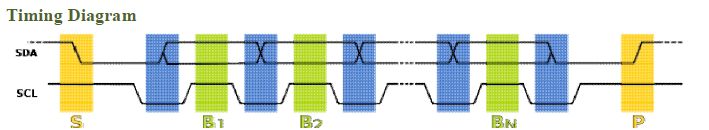
If the master wishes to read from the slave then it repeatedly receives a byte from the slave,

the master sending an ACK bit after every byte but the last one. (In this situation, the master

is in master receive mode and the slave is in slave transmit mode.)

The master then ends transmission with a stop bit, or it may send another START bit if it

wishes to retain control of the bus for another transfer

****Data transfer is initiated with the START bit (**S**) when SDA is pulled low while SCL stays

high. Then, SDA sets the transferred bit while SCL is low (blue) and the data is sampled

(received) when SCL rises (green). When the transfer is complete, a STOP bit (**P**) is sent by

releasing the data line to allow it to be pulled up while SCL is constantly high.

A particular strength of I²C is that a microcontroller can control a network of device chips

with just two general-purpose I/O pins and software. I²C is appropriate for peripherals where

simplicity and low manufacturing cost are more important than speed.

**Messaging Example: 24c32 EEPROM**

One specific example is the 24c32 type EEPROM, which uses two request bytes that are

called Address High and Address Low. (Accordingly, these EEPROMs aren't usable by pure

SMBus hosts, which only support single byte commands or addresses.) These bytes are used

to address bytes within the 32 kibit (4 kiB) supported by that EEPROM; the same two byte

addressing is also used by larger EEPROMs, such as 24c512 ones storing 512 kibits (64 kiB).

Writing and reading data to these EEPROMs uses a simple protocol: the address is written,

and then data is transferred until the end of the message. (That data transfer part of the

protocol also makes trouble for SMBus, since the data bytes are not preceded by a count and

more than 32 bytes can be transferred at once. I2C EEPROMs smaller than 32 kibits, such as

2 kibit 24c02 ones, are often used on SMBus with inefficient single byte data transfers.)

To write to the EEPROM, a single message is used. After the START, the master sends the

chip's bus address with the direction bit clear (*write*), then sends the two byte address of data

within the EEPROM and then sends data bytes to be written starting at that address, followed

by a STOP. When writing multiple bytes, all the bytes must be in the same 32 byte page.

While it's busy saving those bytes to memory, the EEPROM won't respond to further I2C

requests. (That's another incompatibility with SMBus: SMBus devices must always respond

to their bus addresses.)

To read starting at a particular address in the EEPROM, a combined message is used. After a

START, the master first writes that chip's bus address with the direction bit clear (*write*) and

then the two bytes of EEPROM data address. It then sends a (repeated) START and the

EEPROM's bus address with the direction bit set (*read*). The EEPROM will then respond

with the data bytes beginning at the specified EEPROM data address—a combined message,

first a write then a read. The master issues a STOP after the first data byte it NACKs rather

than ACKs (when it's read all it wants). The EEPROM increments the address after each data

byte transferred; multi-byte reads can retrieve the entire contents of the EEPROM using one

combined message.

**5.6.2. Analysis criteria**

The criteria needed to select the different types of buses are defined below. On the basis of

study of following criteria points and seeing their weight points OBC will select the buses to

be used.

1) **Reliability:** The reliability provides a comparison between different buses; this

comparison is made upon the immunity to electromagnetic interferences, the

possibilities offered to check if errors occur and prioritization it several nodes will

transmit at the same time.

As said further main bus and science/survival bus have not the same constraints, in that way

the reliability factor will be adapted.

2) **Power consumption:** In accordance to the embedded system specification, we only

have a few watts available, in that way the power consumption is a really important

criterion.

3) **Availability on commercial market:** As the name says, this criterion compares the

provided component supporting the bus, in other words it indicates if the bus is

commonly used or not.

4) **Data rate:** The assumption was made that the main bus will provide a high data rate,

for this case the reference is set as 1[Mbps]. For the other bus, the reference is

100[kbps].

5) **Topology**: This criterion indicates if the bus supports multi-point or multi master

topology. The main bus requires a multi-point or multi-master topology.

As only two devices will be connected on the science or survival bus, the requirements are

different and a point to point topology will be sufficient.

6) **Implementation facilities**: As implementation facilities, we say if the bus needs

specific component like transceiver, buffer, or a specific protocol.

We will also compare the bus on a few others subjects like voltage, development tools and if

this bus has already be used in space.

**OBC: Interface with ADCS**

**Attitude Determination and Control**

The ADCS 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 algorithms 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 Algorithms Execution:**

The On-Board Computer is to execute the control algorithms as designed by the ADCS subsystem.

The control algorithms are provided by the ADCS subsystem. According to which

the OBC team will take care of its implementation. This includes performing the requisite

numerical calculations with the desired accuracy.

**Actuation:**

The On-Board Computer will interface with and actuate the magnetorquers according to the

control algorithm results. This implies provision of suitable pulse-width-modulated signals

for suitable time intervals.

**Hardware Control**

In order to execute various tasks, it is necessary to use a number of the peripherals. One part

of software design is to write the 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 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.

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 command to the magnetometer via the

UART peripheral. The magnetometer will send a 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.

 The GPS requires no command to be sent. As soon as it is switched on it will start

transmitting GPS data. The GPS unit will send one packet of GPS data every second.

Each data packet consists of messages conveying position, velocity, time-date and

other information.

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

control the select lines of the multiplexer and the multiplexer output will be connected to one

of the pins associated with the ADCS peripheral.

CONTROL LAW ALGORITHM IMPLEMENTATION:-

The electronic PCB that implements the Attitude Determination and Control system is based

on 8-bit microcontroller, selected in order to have the maximum amount of peripherals that

are necessary to perform the satellite attitude control, using magnetic coils and a momentum

wheel, and the satellite attitude determination, using the sun sensors and the magnetometers.

**The implemented peripherals and control units are:**

**1.** 8-bit microcontroller: one used to perform the control algorithms, the other one used

to perform the estimation algorithms, connected together by 2 UART serial ports.

**2.** 10-bit PWM signals used to drive the H-bridge that controls the magnetic coils.

**3.** 10-bit DAC, used to generate the analog reference voltage that drives the

momentum wheel brushless motor.

**4.** 8-bit ADC channels, used to acquire information on the ADCS behavior, such as

current consumptions and voltage levels of the various units

**5.** 1 Real Time Clock with independent power source.

**6.** EEPROM units, used to store various important data during the satellite mission.

The connections between ADCS and OBC are discussed below:-

**SPI**: - The serial peripheral interface bus or SPI bus is a synchronous serial data link. Devices

communicate in master/slave mode where the master frame initiates the data frame. The main

purpose of the SPI bus is to allow access from the OBC to ADCs and DACs on other boards.

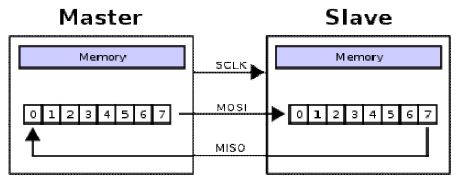
Each board connected to the SPI bus has 4 input lines as follows

• Master Out Slave In (MOSI) – data input

• Master In Slave Out (MISO) – data output

• Serial Clock SCLK – clock input

• Slave Select A SS A – enable slave A, input



During each SPI clock cycle, a full duplex data transmission occurs:

 the master sends a bit on the MOSI line; the slave reads it from that same line

 the slave sends a bit on the MISO line; the master reads it from that same line

**PWM**: - The PWM signals are used to drive H-BRIDGE that controls the magnetic coils. The

pulse width modulation feedback system can be used for attitude control for space vehicles.

In it is usually required that power be modulated in an on-off fashion. The PWM control is

also connected to ON-BOARD COMPUTER.

**MAGNETOMETER RESET**: - Sun-sensor/Magnetometer has become the usual

instruments combination for micro-satellite attitude determination with the low-weight/highreliability

character. Although these instruments have already been calibration strictly in

ground testing, error items may appear during the launch and on-orbit running period. The

magnetometer is affected not only by mechanical deformation but also by the disturbance of

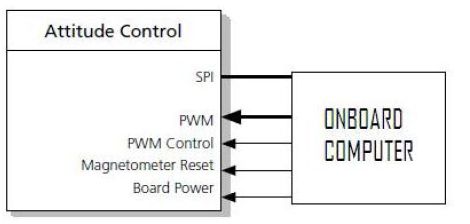
residual magnetic field of the satellite body. The sun-sensor is only affected by the

mechanical deformation. So there is a need for magnetometer reset control from ONBOARD

COMPUTER. Hence a connection is required from OBC towards ADCS for magnetometer

reset.

**THE NECESSARY CONNECTIONS ARE SHOWN BELOW**

****

**OBC: Interface with power subsystem**

The power subsystem is concerned with acquiring, regulating and distributing power

to the various components of the satellite. 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 and On-Board Computer subsystem

must interact to exchange data about satellite health. 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.

**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 misbehaviour on the part of any component.

The On-Board Computer will respond with suitable redundancy measures.

Health Monitoring (HM)

Data Acquisition

1. Primary microcontroller requests and receives (stores in temporary buffer) load data

packet from Power microcontroller (via I2C). (Once every 2 minutes).

2. Primary microcontroller creates a data packet containing the HM data. (Once every

2 minutes)

This data is then send to the ground station so that the satellite administrator would

know the health status of the data. This data is useful because it will help the TUSSAT

team to find the various design flaws. These results can be used as a research report

for the other Universities developing student satellites.

**OBC: INTERFACE WITH COMMUNICATION SUBSYSTEM**

Programming the CC1020 chip

All initialization for the CC1020 will be done by the primary microcontroller by using the

SPI interface. The SPI interface consists of an 4 lines: MOSI ( Master Out Salve In ), MISO (

Master In Slave Out ), SS ( Slave Select ) and the clock line. The primary microcontroller can

have multiple slaves and the slave needed to be programmed will be selected by the SS line (

set to low for selection of particular chip ).

Once the CC1020 chip is selected, an address of 1 byte will be first sent along the SPI to

indicate the location of the register on the chip to be programmed. Thereafter the appropriate

data to initialise or reconfigure the registers is sent on the next clock pulse. Like this, all

registers are programmed on the chip through the primary microcontroller.

Sending data to CC1020 for downlink transmission

The data to be transmitted to the ground station is sent to the CC1020 chip by the secondary

microcontroller through the GPIO line connecting them. The bit to be sent is shown by the

status of the line ( high/low ) and is modulated and transmitted by the CC1020 chip. The

transfer rate of the GPIO line will be 1200 baud.

To ease the receiving of the down linked data, the AX.25 format will be used as it is a

common format used by amateur radio operators like us. The data sent will deal with the

health and status of the various components on board the satellite and will be packaged into

AX.25 frames for transmission by the secondary microcontroller.

AX.25 Details

AX.25 is a data link layer protocol derived from the X.25 protocol suite and is used

extensively on amateur packet radio networks. The following illustrate the basic structure of

the AX.25 frame:

Flag Address Control PID Info FCS Flag

0x7E 112/224 bits 0x3F 0xF0 N\*8 bits 16 bits 0x7E

Flags

The flags are used to denote the separation between two sent data packets. When the

transmission is set to a delay on the Terminal Node Controller, the during that period of delay

these flags are sent over and over again. They are represented by the hex value 7E (

01111110 in binary ). There must necessarily be one flag between two adjacent data packets

to allow the transmitter to differentiate between the two.

Address

To study the address format, we use an example as follows:

“W2FS-4> CQ,RELAY:Example”

In this frame, the different components are:

 Source- W2FS-4

 Destination- CQ

 Digipeaters- RELAY

 Information to be transmitted- Example

In the frame, the call signs of all the components must be of a fixed size of 7 bytes according

to the internationally set standards. The name is fit into the first 6 bytes ( using blanks if

necessary ) and the last byte is used for the SSID ( Service Set IDentifier ). The receiver

would need to know when the address field would have ended. To solve this, we shift all the

bits left to bring in a 0 in the least significant bit of the addresses. The least significant bit of

the 7th byte is set 1 to signify the end of call signs, or left 0 otherwise. For example:

Source call sign: W2FS SSID: 4

Character Hex Value from ASCII table HEX value after left shift

W 57 ( 01010111 ) AE ( 10101110 )

2 32 ( 00110010 ) 64 ( 01100100 )

F 46 ( 01000110 ) 8C ( 10001100 )

S 53 ( 01010011 ) A6 ( 10100110 )

Space 20 ( 00100000 ) 40 ( 01000000 )

Space 20 ( 00100000 ) 40 ( 01000000 )

This is how the first six bytes will be coded. As for the seventh byte, we shall follow the

pattern of 011SSSSx, where SSSS represents the SSID in binary. The value of x will be 1 if

this is the last call sign and 0 otherwise. Here the SSID is 4 or 0100 in binary. Since this is

not the last call sign, the value of x will be 0. Hence the 7th byte becomes: 01101000 which is

68 in the hexadecimal form.

The next call sign is that of the destination which is CQ. Applying the same rules to it we

have:

Character : C Q space space space space SSID=0

Shifted Hex Value : 86 A2 40 40 40 40 60 ( 01100000 )

The final call sign is that of the digipeater RELAY. We can have a maximum of 8

digipeaters. Using the same set of rules, we have:

Character : R E L A Y space SSID =0

Shifted Hex Value : A4 8A 98 82 B2 40 61 ( 011100001 )

The last bit if the SSID byte ere has been set to 1 as this is the last call sign in the address.

Control and PID

We are using the hex value 3F for the control byteand the value F0 for the PID byte.

Information to be transmitted

Here we are transmitting the information “Example” which will be represented by 7 hex

bytes of: 0x45 0x78 0x61 0x6D 0x70 0x6C 0x65

There is no need to shift these bytes to the left unlike the address bytes.

Frame Check Sequence

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

correction section.

The Complete Picture

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 30 hex bytes as follows:

C Q sp sp sp Sp SSID=0 W 2 F S

sp 86 A2 40 40 40 40 60 AE 64 8C A6

40

sp SSID=4 R E L A Y SSID=0

Control PID

40 68 A4 8A 98 82 B2 61 3F F0

E x a m p l e

45 78 61 6D 70 6C 65

Or, as an initialized C array: Send Data[30] = {0x86, 0xA2, 0x40, 0x40, 0x40, 0x40, 0x60,

0xAE, 0x64, 0x8C, 0xA6, 0x40, 0x40, 0x68, 0xA4, 0x8A, 0x98, 0x82, 0xB2, 0x40, 0x61,

0x3F, 0xF0, 0x45, 0x78, 0x61, 0x6D, 0x70 0x6C, 0x65 }

**FUTURE DEVELOPMENT**

At this point of the project there is still much work to do, the first thing is to design a final

architecture for the OBC, and provide a test board to the software group in order to validate

or modify this design. Many different hardware tests like radiation test, risk studies will be

done on this board in the way to certify that the final board will work properly during space

mission. In a more remote future, the other subsystem board will be analyzed according to the

same method.