An Optical Remote Inspection System for the Surrey Nanosatellite Applications Program

Richard Lancaster

Submitted for the Degree of
Master of Science in Satellite Engineering
from the University of Surrey

Surrey Space Centre, University of Surrey
Guildford, Surrey GU2 5XH
United Kingdom

August 2001

Abstract

The Surrey Nanosatellite Applications Program was a long running Surrey Space Centre project adopted by Surrey Satellite Technology in late 1999, to develop technology to enable the construction of sub 10kg "nanosatellite" spacecraft. This report details the development of an imaging system that would enable satellites built as part of the program to inspect other spacecraft in their vicinity.

The report details the development of the generic imaging system for the SNAP program and the flight of the first production model on the SNAP-1 spacecraft in June 2000. On orbit results returned from the SNAP-1 spacecraft are included in the report.

Acknowledgements

Craig Underwood (Supervisor, SNAP program instigator and director)

Ndedi Monekosso (SNAP OBC designer)

Hans Tiggeler (EDAC FPGA design and coding, general technical support)

Andrew Dachs (SNAP OBC applications software, general technical support)

Mark Fouquet (Analogue electronics guidance)

Trevor Edwards, Chris Evenden (PCB layout)

Teresa Lowe and Kathy Verrier (Clean room staff)

Ed Stevens (Long suffering SNAP-1 assembly, integration and test director)

Neville Bean (On orbit operations)

Lee Cowie (Camera mechanics)

Mark Tucknott, Guy Richardson, Paul Charman (Mechanics and integration)

Phil Kinsey (Parts procurement)

All the other SSTL and SSC engineers, academics and staff who made SNAP-1 possible.

Contents

Chapter 1

Introduction

A nanosatellite is a class of satellite weighing less than 10kg. This puts them at the very bottom end of the satellite weight spectrum, with large communications satellites at the other weighing several tonnes. However it is only recently, due to the continued miniaturisation of microelectronic devices, that it has become viable to build nanosatellites capable of performing useful mission functions.

In late 1999, seeing the emerging viability of nanosatellite platforms. Surrey Satellite Technology Limited (SSTL), adopted a five year old Surrey Space Centre (SSC) program to develop the technology necessary to build a nanosat, with the view of launching their first nanosatellite spacecraft in mid 2000. This program was called the Surrey Nanosatellite Applications Program (SNAP), with the first spacecraft being called SNAP-1.

One of the possible application areas that was envisaged for the new nanosatellite platform was that of remote inspection. What is meant by this is that a nanosat with on board camera systems could be permanently docked with a larger satellite or space station. Then be deployed when there is a need to check for damage or inspect some other problem.

The basic aim of this project was therefore to develop a camera system for the SNAP program. That would enable the nanosatellites developed as part of the program to perform worthwhile remote inspection missions. It was also the intention to fly the first such camera system on the SNAP-1 nanosatellite when it was launched in June 2000.

1.1 An aside

At first glance the problem may seem trivial. Just place a CCD and a frame grabber onto the nanosat and you have a remote inspector. Unfortunately however this is not quite the case. This is because there are two main ways in which a nanosat is likely to be operated, and both require the vision system to be more than a simple

CCD and frame grabber.

The first mission profile is that an inspection mission would be orchestrated, or flown directly by a human operator. To enable this to happen the nanosat must therefore be capable of relaying real-time video to the operator. However, nanosats are small. This means that they can only carry a limited area of solar arrays and a limited mass of batteries. Hence, in general, they will not be able to provide the electrical power required by a transmitter that can relay the bandwidth of uncompressed real time video. Which leads directly to the requirement for the nanosat to carry the processing power required to perform compression on the video stream to reduce its bandwidth to a transmittable size.

The second mission profile is that the nanosat would be autonomous and fly a predetermined mission, whilst relaying key images back to the operator. It is also likely that the nanosat would be deployed when the parent craft becomes disabled in some way or another. Therefore it may be the case that all of the parent's radio and other electrical systems have failed. Hence the only way that the nanosat will be able to hold its position relative to the parent and finally re-dock, is by optically tracking the parent using its cameras. Other techniques such as radar sounding will require more power than the nanosat's power systems can provide. This therefore means that the nanosat must also be capable of performing high level computer vision/target tracking functions.

Hence the need for processing power on-board the nanosat is clear. However it is unlikely that the nanosat's on-board computer (OBC) will be able to play this role. This is because it will be tied up performing the attitude control and house keeping tasks which it classically performs. Therefore it makes sense for all the camera systems and processing power required for the vision tasks to be integrated into a single self contained module. This can then simply be plugged into a nanosat when it is required to perform a remote inspection task.

1.2 Objectives

The precise objectives of this project were therefore to:

- Develop a self contained "Machine Vision System" (MVS) module. Which consists of both the camera systems and processing power necessary for carrying out remote inspection tasks, and is compatible with the SNAP program spacecraft interfaces.
- Manufacture an MVS module to fly on the SSTL SNAP-1 nanosat in June 2000.

Note that the launch date of the SNAP-1 spacecraft was only 9 months after the date on which this project was initiated. Therefore time was of the essence.

Chapter 2

Specifications and Constraints

There were three independent sets of specifications and constraints governing the design of the Machine Vision System (MVS):

- The constraints imposed by the space environment.
- The SNAP spacecraft interface specification.
- The generic mission objectives of the MVS.

There were also an additional set of mission objectives for the version of the MVS that was to be flown on the SNAP-1 spacecraft.

2.1 Space environment constraints

The fact that the MVS was to be launched into space, meant that constraints were imposed on it due to the space environment its self. Some of these constraints are now detailed.

Launcher characteristics

To get into space a satellite needs to be launched on a rocket. This presents a number of mechanical problems, as rockets accelerate rapidly and produce harsh vibrations over a wide spectrum of frequencies. Therefore any payload put into space needs to be mechanically designed in such a way as to be able to withstand such one off conditions. Even if this means that once it is in orbit it is effectively massively over engineered.

Radiation effects

Space is filled with fast moving, hence high energy, charged particles, such as protons, electrons, alpha particles and some heavier ions. For example at the L1 point between the Sun and the Earth, there are on average around 2 high energy protons per cm^3. On the Earth's surface we are protected from these particles, both by the deflective properties of the Earth's magnetic field and the absorbing properties of the atmosphere. However for Earth orbiting satellites, which are mostly located between 300km and 40000km from the Earth's surface, the situation is not so favorable. Firstly there is no atmosphere between them and the incoming particles. Secondly the magnetic field which performs such an efficient job at deflecting particles from the Earth's surface has a tendency to trap particles in orbits around the Earth. These trapped particles form belts abound the Earth, ranging in altitude from 500km to 35000km. These belts are known as the Van Allen belts. The upshot of this is that almost all Earth orbiting spacecraft encounter significant fluxes of charged particles.

Unfortunately, when a charged particle hits a VLSI transistor on a modern microchip, it causes a large quantity of charge to be generated within the transistor. In the worst case this could cause the transistor to burn out and either short circuit the chip or enter a fixed state. This is known as a Single Event Latch-up or SEL, and is generally fatal to the functionality of the circuit. However in most cases it simply causes the transistor to flip state. So if it was representing a digital 0, it might change to representing a digital 1. This is known as a Single Event Upset or SEU.

This kind of event would cause havoc in any digital circuit that had been designed to work in Earth's benign radiation environment. Therefore when designing circuits for use in space, special care has to be taken to cope with such errors. So for example, on a computer system, Error Detection And Correction (EDAC) hardware, is normally placed between the CPU and the RAM. This hardware transparently encodes data as it is written into the RAM with something like a 16:8 forward error correction code [1]. Then when the data is read back, most of the errors which have since occurred in the data due to radiation effects, can be transparently corrected by the EDAC hardware before the data is delivered to the CPU.

For more information on charged particles and upset rates refer to [2] and [3].

Vacuum considerations

Space is a vacuum. This generates a set of constraints which are not immediately obvious.

- Certain plastics "out-gas", and effectively disintegrate. Hence in general plastic components cannot be used on space missions.
- Most electrolytic capacitors out-gas or explode. Hence in general only non-

electrolytic capacitors can be used. This often causes problems because it is not normally possible to purchase high value non-electrolytic capacitors.

- The standard D-type connectors found on most electronic components contain Cadmium. In a vacuum this grows crystals, which given time can grow between two contacts and cause a short circuit. Therefore space rated non Cadmium D-type connectors have to be used [4].

2.2 SNAP spacecraft interface specification

As part of the SNAP program, a generic system architecture was defined for all SNAP spacecraft. The MVS, like all SNAP payloads, therefore needed to comply with this architectural specification.

Payload box

The module box for SNAP payloads was defined as being a box of dimensions 160x120x20mm. This in itself presented a significant difficulty, because by comparison the module boxes that had housed systems such as the 386 On-Board Computer (OBC) on previous SSTL satellites were of dimensions 320x290x30mm. So to develop the nanosatellite, SSTL had to reduce the volume of all of their systems by a factor of 4. Even trying to cram a new system such as the MVS into such a small form factor looked like it was going to present significant challenges.

Electrical interface

Under the SNAP system architecture, the MVS was given a 9-way D-type connector interface to the rest of the spacecraft. This contained the signals detailed in Figure 2.1. This allowed the MVS to communicate with the rest of the spacecraft via a CAN bus [5] and directly with the ground via an RS-232 like async link. The connector also supplied the MVS with its power supply.

In terms of power budget the MVS was allowed an on orbit average power consumption of 1W, with a peak demand limit of 2W. To put this in perspective the 386 OBC on previous SSTL satellites draws an on orbit average of around 3W.

2.3 Generic MVS mission objectives

The generic mission objectives of the MVS were defined to be as follows:

- The MVS must be a “Black Box” plug in module, containing all the cameras and processing power necessary, to enable a SNAP spacecraft, to obtain and

Pin	Description
1	CAN HI
2	CAN HI RED
3	Case GND
4	CAN LO
5	CAN LO RED / Async RX
6	Switched 5V
7	Switched Raw Battery
8	Async TX
9	GND

Figure 2.1: *Specification of the SNAP MVS 9-way D-type spacecraft interface connector.*

relay to a remote human observer, TV quality images of other spacecraft in the local vicinity.

2.4 SNAP-1 MVS additional mission objectives

It was decided to give the MVS to be flown on the SNAP-1 spacecraft the following additional mission objectives:

- As the SNAP-1 spacecraft is deployed from the launch vehicle. The MVS board should power up and start capturing a time-lapse movie sequence of the deployment. This movie should then be stored until it can be downloaded to the ground once the spacecraft has been commissioned.
- Once in orbit the MVS should be capable of taking medium resolution images of the Earth's surface.

Chapter 3

Hardware Design

Between September 1999 and January 2000 the SNAP MVS hardware was designed. This chapter details that design process.

3.1 Modular decomposition

It was rapidly decided that it would be logical to break the MVS down into three discrete conceptual modules. These were as follows.

- **The camera systems:**
  The physical cameras themselves.
- **The image digitiser module:**
  The module that controls the cameras and converts the signals they output into digital data in a useful format.
- **The processing module:**
  The module that provides the processing power necessary to perform the compression/vision functions, and also handles communication with the rest of the spacecraft.

3.2 Camera systems

Camera selection

Due to the particular mass and volume constraints presented by trying to develop a nanosatellite system. It was decided that the choice of cameras to be used on the MVS must meet four overriding requirements to make them viable for nanosat use:

- Low cost

- Low mass
- Small size
- Low power consumption

A notable absentee from this list is image quality. This is because to perform a remote inspection task you don't need Landsat quality multi spectral imaging [6]. Instead you probably only require low quality monochrome video. Indeed this is reflected in the original spec, which calls for the ability to relay TV quality images to a remote human observer.

Having considered these requirements it became clear that classic CCD imaging elements would probably not be the best solution. This is because they require significant analogue domain drive electronics, hence making them neither particularly low mass, low volume or low power relative to the mass, volume and power budgets available on a nanosat.

It was therefore decided to employ a new class of imaging element. The CMOS imaging element. Each pixel in these elements is a photo diode. Which means that the intensity of the light incident on the pixel can be read off directly by the element's electronics and output at the edge of the chip in a convenient format. This is as opposed to CCD technology in which the charge that builds up in each pixel during each exposure has to be clocked out and sensed using complex timing and sensing electronics. Hence in general CMOS elements and their support electronics are smaller, lighter and consume less power.

A further advantage of CMOS elements is that they are fabricated using fairly standard CMOS chip fabrication techniques, as opposed to the NMOS or PMOS technology used for CCDs. This means that other digital and analogue processing functions can be trivially fabricated onto the same die as the imaging array. Hence the imaging array and its support electronics can all be fabricated on the same chip, massively reducing their mass, volume and power consumption.

A number of CMOS elements were considered for the role. Including the IMEC's FUGA element [7] and the VLSI Vision [8] range of elements. Each of these elements has interesting features facilitated by the integration of extra processing functions onto the imager die.

For example the IMEC element has address and data bus pins on the imaging element. Hence using the address bus the drive electronics can randomly request the value of any pixel in the array without having to read all the pixels in the array as would be necessary with a CCD sensor. This functionality is useful in target tracking applications, as once a lock has been acquired, rapid scans can be performed of the object of interest without having to scan the whole array.

The VLSI vision array on the other hand is a more conventional array in which the entire contents are continually clocked out. However it does contain an on board ADC and a digital control port to set gains and exposure parameters.

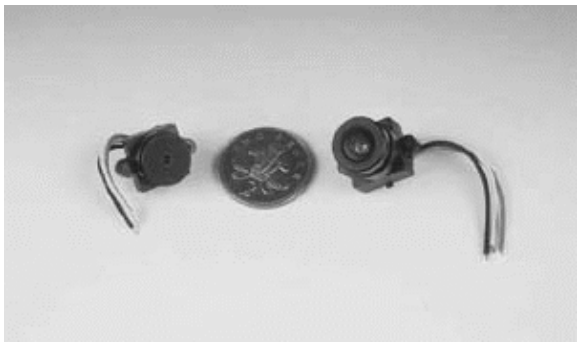


Figure 3.1: *The CMOS cameras selected for the SNAP MVS.*

However in the end a simple element available off the shelf from Farnell electronics was selected. This was selected because:

- Low cost - 40 UK pounds.
- Small size - The imaging element is only 10x10x5mm. See Figure 3.1
- Low power - The element only consumes 140mW
- Simple 3 wire interface - Only power, GND, and a composite video signal. This is useful because it means that there isn't a bundle of wires snaking across the spacecraft from the camera to the digitiser input. The FUGA element would have required around 20 wires.
- Short procurement lead time - Hence it was actually going to be possible to obtain a set of cameras before the first MVS board was to be launched on the SNAP-1 spacecraft.

The elements had a 352 column by 288 line resolution, which we deemed as acceptable for remote inspection missions.

Camera arrangement

It was decided to make the MVS support up to 4 cameras. This would enable different views of the object being inspected to be obtained without the physical maneuvering of the nanosat. It would also allow different lenses to be fitted to each camera so that both wide angle and detailed views could be obtained simultaneously.

In the specific case of the SNAP-1 spacecraft. It was decided to mount the cameras directly to the walls of the MVS module box. This was possible because on the SNAP-1 spacecraft the side walls of the box were exposed to space. It also had the benefit that all of the mounting and wire routing problems were local to the MVS payload, so mounting points didn't have to be placed elsewhere on the spacecraft etc. Further this allows the MVS to simply be plugged into future SNAP spacecraft without having to worry about where to mount the cameras.

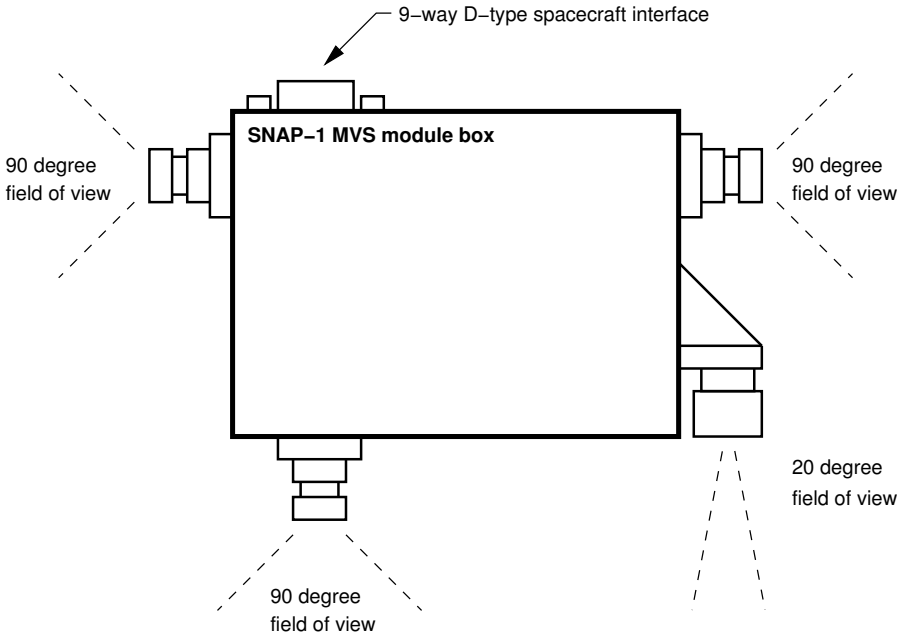


Figure 3.2: *The layout of the cameras on the SNAP-1 version of the MVS module.*

The layout of the cameras and the lenses used on the SNAP-1 version of the MVS module is shown in Figure 3.2. As can be seen from this there were three, 3.6mm, Wide Angle Cameras (WACs) that gave a view of a 270 degree arc of space. There was also a 50mm Narrow Angle Camera (NAC) that gave a detailed view of whatever the middle wide angle camera was looking at. The narrow angle camera also contained a near infra-red (NIR) pass filter to enhance the contrast between land and sea features on the Earth's surface [6].

NOTE: The mechanical mountings for the cameras and the mechanics of the module box were designed by SSTL mechanical engineers.

3.3 Processing module

Performing any kind of vision, compression or target tracking function, particularly if a relatively high frame rate is required, by its nature demands a significant amount of processing power. Hence early on in the project it was realised that the processing power that the MVS board would require to make it a useful platform would be significant. Therefore nothing short of a modern 32bit processor was likely to be adequate.

It has already been indicated in this document, that the processor technology (186, 386) that SSTL had been flying on spacecraft up until the SNAP program was not going to be suitable for this purpose. Both because its form factor was too large to fit into the module box, and the power consumption too great for the power supply available. Fortunately however, at the same time a project was starting up within

SSTL to develop a new On Board Computer (OBC) for the SNAP program, based around the 32bit Intel StrongARM 1100 processor. This is a high performance, low power consumption processor, making it ideal for nanosat use. It was hence decided that it would be sensible to work on the development of this project and then integrate a cut down version of the OBC onto the MVS board to provide the processing power required.

The idealist might suggest that it would be better to use a DSP for performing such vision and compression functions. This is probably true. However the time available before the launch of the SNAP-1 spacecraft did not facilitate the expenditure of the resources required to develop two completely new computing platforms from scratch. Therefore we had to live with what we could reasonably design in the time available, and as such the StrongARM didn't represent all that painful a compromise.

The architecture of the cut down OBC used on the MVS board is shown in Figure 3.3. This has a classical embedded system architecture, with the following specific features:

- 220MHz StrongARM RISC processor (Run at 88MHz to give a favorable power/performance trade off).
- 2Mb of FLASH RAM for storage of firmware (120ns).
- 4Mb of EDAC protected RAM for code and data storage (70ns + 30ns for the EDAC).
- A CAN controller to link the processor to the spacecraft CAN bus.
- An async link from the processor directly to the spacecraft's RF transmitter and receiver.
- A watchdog device to reset the processor if it crashes.

An important side effect of the MVS's processing module being the same as the main OBC on a spacecraft, is that if the main OBC fails, then the MVS can be run as a replacement OBC, hence giving the mission an extra degree of redundancy.

Processor flight heritage

One of the things that spacecraft system engineers are eternally worried about is flight heritage. If you haven't flown a component before in a representative environment, then you've no idea how it will react when it gets hit by its first high energy particle. It could for example be extremely susceptible to single event upsets (SEUs) or worse it could be susceptible to latch ups which destroy the device. (Refer back to Section 2.1 for more details on SEUs and latch up). Further, fully representative testing is extremely expensive and difficult to do, not least because simulation of the MeV particles requires a visit to a facility such as the CERN

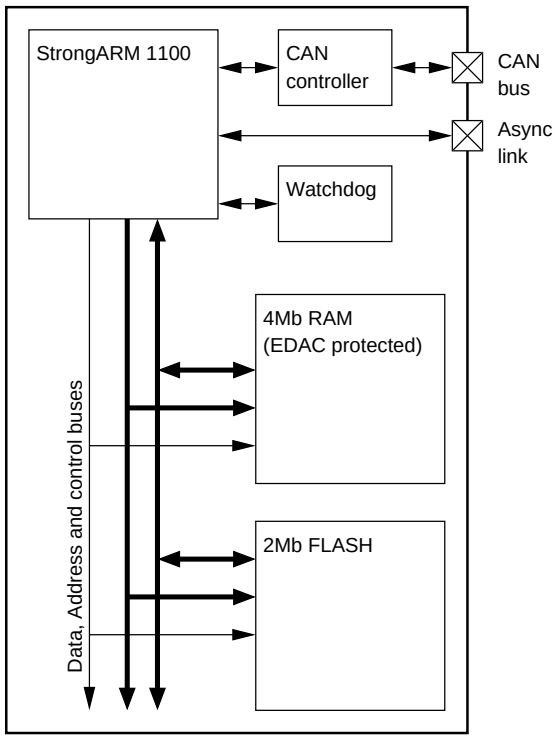


Figure 3.3: *The architecture of the MVS' processing module.*

particle accelerator. So the bottom line is that if you haven't flown a component in a similar orbit then you have no idea how it will fare.

To our knowledge, no-one had ever flown a StrongARM processor in space before. Hence we had a number of months of discussion and worrying about whether to effectively risk the entire mission by flying the StrongARM as the OBC. In the end we came to two conclusions. Firstly due to the constraints imposed by the nanosat volume, mass and power limitations, none of SSTL's tried and tested technology, such as the 186 or the 386ex, would even fit into the module boxes, let alone function off the meager quantity of electrical power supplied by the nanosat's small area of solar arrays. We therefore either had to fly the StrongARM, some other untried technology, or reduce mission functionality. Secondly as SNAP-1 was a 100% research and development mission, we decided that in lew of any evidence for or against, we might as well bite the bullet and fly the StrongARM.

FLASH RAM

The purpose of the FLASH RAM was to store the system's firmware. Refer to the software architecture section for details of this firmware.

The FLASH RAM presented yet another heritage problem, in that previously all SSTL spacecraft had used 5 Volt, 8 bit EPROMs for firmware storage. Unfortunately the StrongARM is a 3.3 Volt system that requires a 16 bit wide boot device.

So although we were fairly confident that most EPROM technology would work in orbit, we couldn't actually get hold of any 3.3 Volt EPROMs. This meant that we were forced to look toward FLASH technology to provide our non volatile storage.

FLASH technology has been used on orbit before. We knew for example that the Stellenbosch University, SUNSAT spacecraft had employed it successfully. We also knew that if it gets hit by a high energy particle while it is being programmed the die can be rendered inoperative. This is because during program an internal charge pump generates a high potential, perhaps 12V or 24V, across the cells being programmed. If a high energy particle then hits such a cell and generates an ionisation trail through it, then the resulting current flow can be fatal to the device. However we were planning to use the FLASH for firmware storage and no on orbit reprogramming was envisaged. So we were reasonably confident that FLASH technology would be acceptable. We did however have no precise knowledge on exactly which FLASH chips had been flown before on orbit. We therefore once again simply bit the bullet and selected the 2Mb, 16 bit wide, 3.3 Volt Am29LV160BB120EI FLASH chip from AMD.

Note that both the OBC and the MVS had the capability to reprogram their FLASH RAMs in orbit. However due to the risks already detailed, there were no plans to actually do this.

Volatile RAM

The StrongARM 1100 processor supports two types of volatile memory, SRAM and DRAM. Previously most SSTL spacecraft had flown SRAM as their volatile storage. This is partially because if employed correctly it consumes less power than DRAM. However the main reason is that it exhibits a better response to radiation hits than DRAM. Basically when a particle hits an SRAM, in general, a single bit flips. This type of corruption can easily be corrected by simple EDAC techniques (See the next section on EDAC techniques for more details). However when the sensing amplifier on a DRAM gets hit by a particle, whole rows of bits can be corrupted. This kind of corruption is very difficult for EDAC to correct, and hence leads to DRAM's lack of favor within SSTL for volatile RAM applications. For these reasons it was therefore decided to use SRAM on the OBC and the MVS.

However we then encountered some problems. We wanted to fly 4Mb of RAM, so that we could fit a decent quantity of software and log files onto the OBC and store a reasonable number of images on the MVS. However as we'll see in the next section this meant that 8Mb of RAM had to be flow to enable the EDAC to function. Thus given the SRAM memory densities available at the time, if we laid 8Mb of SRAM out on the PCB, then there wouldn't be enough room left for the other components required in the system.

We therefore turned to a company called HMP, who build hybrid stacked memory modules. Their PUMA device is a 4 layer stack of 8 SRAM chips, which has a footprint just larger than the size of a pair of the chips used in the stack. Each of

these stacks is a 32 bit wide block of 4Mb. Hence two blocks satisfied our SRAM requirements and made the footprint of the SRAM small enough to fit onto the PCB.

EDAC technique

Up until the SNAP program, some SSTL computer systems had used Triple Modular Redundancy (TMR) to protect their volatile RAM from SEU corruption. Under this technique, when a processor writes data into the RAM, EDAC hardware between the processor and the RAM writes three identical copies of the data into three independent RAM banks. Then when the data is read back the EDAC hardware reads all three copies and performs majority voting between them. Therefore if the data from one of the banks is corrupted, the other two banks will out vote it and the correct data will be returned to the processor.

However while this technique provides good protection from errors, by enabling the detection and correction of a corruption of any one of the three copies of a bit stored in the RAM. It does have the down side of having a 200 percent overhead. So for every 1Mb of storage you want to have available to software, you have to fly 3Mb of physical RAM to store all the copies.

On a nanosat where you are trying to minimise the volume, mass and power consumption of everything including RAM, having to fly three times more RAM than you can utilise is punitive. We therefore decided to use a new technology on the OBC and MVS, which had recently been developed at SSTL by Dr. Stephen Hodgart and SSTL engineer Hans Tiggeler. This technology is known as 16:8 EDAC [1].

Under this technology, for every 8 bits that are written to RAM a further 8 bits of encoded parity data are also written into the RAM by the EDAC hardware (An explanation of the coding technique used to generate the parity bits is far beyond the scope of this document. Therefore please refer to the original paper [1] for more information). Using this technique it is possible to correct two errors in each 16 bit block of data and parity bits stored in the RAM. So its error correction ability is lower than TMR, but it has the advantage of having only a 100 percent overhead. By selecting the 16:8 technique we therefore made a trade off between error correction ability and physical memory overhead. However we felt that our past experience [2] with SRAM memory technology in orbit, suggested that the reduction in error correction ability would still leave us with a workable system.

A block diagram of the EDAC and RAM structure used on the SNAP OBC and MVS is shown in Figure 3.4.

The EDAC hardware was implemented by Hans Tiggeler in a pair of Actel A54SX08-VQ100 FPGAs. Actel FPGAs are the FPGAs of choice for most space missions. This is because they do not contain a bank of volatile SRAM to hold the configuration of their internal cells, as is common on devices from Xilinx and most other manufacturers. Instead they are write once “PROM like” devices, in which physical

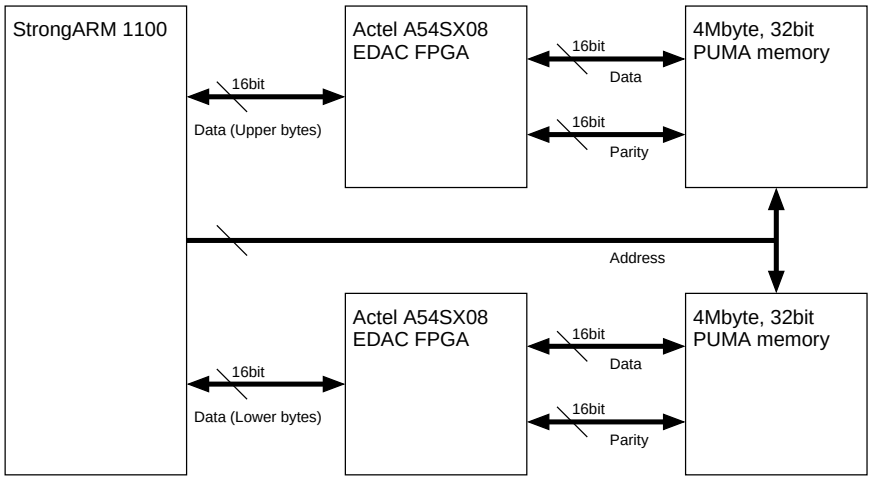


Figure 3.4: *The architecture of the EDAC subsystem used on the SNAP OBC and MVS.*

fuses are blown between cells during programming. This therefore means that SEU events cannot reconfigure their internal architecture, an occurrence which is entirely possible with SRAM based devices.

There's one further note that needs to be made here about the use of EDAC. When data is read from the RAM it is transparently corrected by the EDAC hardware and the processor never sees the erroneous data. However the data in the RAM remains corrupted. Therefore the cumulative build-up of corruptions over time, would eventually result in the EDAC failing to be able to correct a badly corrupted word. Hence a process termed "Washing" needs to be carried out. Basically every 10 minutes or so the processor reads in and then writes out each individual word held in the RAM. The process of reading causes the processor to obtain an error corrected version of the word, and the process of writing causes the corrected word to be written back into the RAM. Therefore errors are washed out of the RAM, hence preventing a cumulative buildup from occurring.

CAN controller

On previous SSTL satellites the on board computers had used the memory mapped Philips SJA1000 CAN controller peripheral to send and receive CAN packets via the spacecraft's CAN bus. However once again there were a number of issues with using this on the nanosat boards. Firstly it was 5V technology, and secondly it would have required significant glue logic to convert from its multiplexed data and address bus to the StrongARM's non-multiplexed buses, and we didn't have the board space available to do this.

Therefore once again a device with no flight heritage had to be used. At the time the only 3.3V CAN peripheral available was the Microchip MCP2510. However just by chance it happened to be a very convenient device for squeezing onto the nanosat

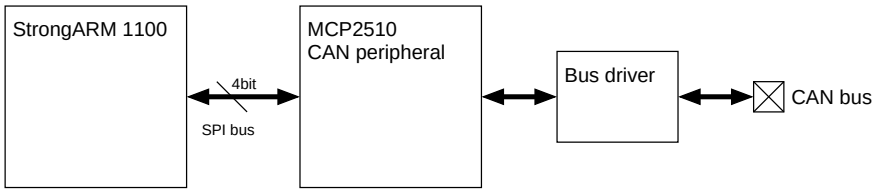


Figure 3.5: *The architecture of the CAN subsystem used on the SNAP OBC and MVS.*

board. This is because instead of being a memory mapped device and hence having a large number of pins for the address and data bus interface, it simply had a 4 wire SPI serial interface to communicate with the CPU. Hence these 4 pins were connected to the SPI bus output of the SA1100 without any glue logic. The other pins on the chip were then connected through a driver to the CAN bus, and that's all there was to it.

This architecture is shown in Figure 3.5.

Watchdog

All SSTL on board computers fly with a hardware watchdog. The purpose of this is to reset the processor if it doesn't pulse a pin on the watchdog chip every second. This means that if the software on the processor crashes, either because it is badly written, or an SEU corrupts some data causing it to crash, then the processor will no longer be able to pulse the watchdog and hence the watchdog will reset it, allowing the processor to restart and get on with performing its mission function.

For the watchdog on the OBC and MVS we selected the Maxim MAX706T. This had the additional advantage that it watched the power supply and also reset the processor if the voltage dropped too low.

3.4 Image digitiser module

Given the choice of cameras and the design of the processing module. The functionality required from the image digitiser module therefore boiled down to being able to, on demand from the CPU, grab a frame from a specified camera and write the frame as a bitmap into CPU accessible RAM.

Composite video

As was mentioned in Section 3.2, the cameras selected for the MVS output composite video signals. Composite video is a method of encoding a video sequence (E.g. a time varying set of images) for transmission on a single analog channel. Figure 3.6 contains an illustration of the composite video format.

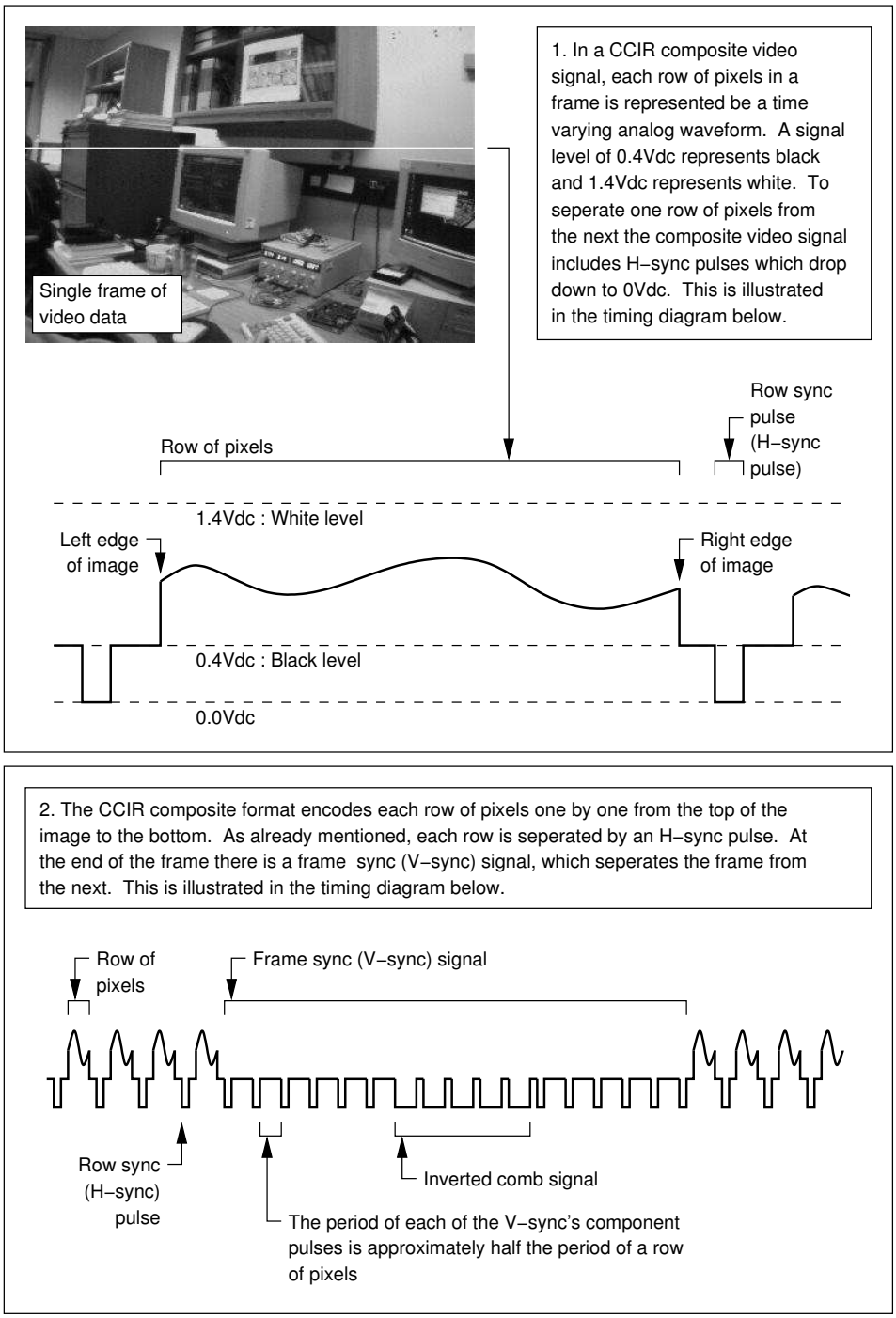


Figure 3.6: *Illustration of the CCIR composite video format.*

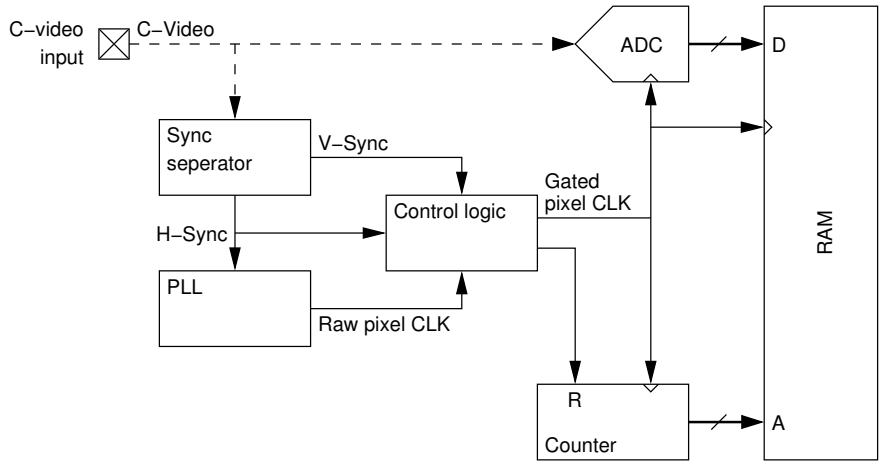


Figure 3.7: *The classical approach to digitising composite video.*

As an aside, note that the only sync references carried in the signal are the horizontal sync (H-sync) pulses and the vertical sync (V-sync) pulses. There isn't the clock edge per pixel which would be required to clock a digitised version of the pixels into a block of RAM. Therefore the receiving device, if it wants to clock each individual pixel into RAM, needs to generate a pixel clock (P-clock) from the H and V sync pulses. This is a throwback to the original purpose of the composite video signal, which was to transmit pictures to an analogue television set. Hence because a television simply alters the strength of its electron beam in accordance with the amplitude of the composite signal as it scans across the screen, it therefore doesn't require a pixel clock.

The classic circuit for converting a composite stream into a bitmap in a block of RAM is illustrated in Figure 3.7. The basic point of this circuit is to generate a pixel clock from the sync information in the composite stream, then gate the pixels sequentially into the RAM using this clock.

The first stage of this process is to extract the H and V sync pulses. Elantec is one of the major manufacturers that provide single chip solutions that accept the composite stream on one side and output digital logic H and V pulses on the other.

The horizontal sync pulse is then fed into a phase locked loop, which multiplies up the frequency of the H pulses by the number of pixels in each row of the image. This gives a clock which is aligned and locked to the H pulses and by definition each of its edges is aligned with the values representing the pixels in the composite stream. Hence this is the pixel clock. (Note that there are some subtleties involving the front and back porches that I'm glossing over for the sake of clarity).

The H, V and P clocks are then fed into some simple control logic which is responsible for synchronising the capture of a frame. So for example when given a command to grab a frame, it would wait for a V sync pulse. When a V sync arrives it knows that it is now at the start of a frame. It would therefore reset the counter to, for the sake of argument, zero. Hence the intention is that the first pixel be stored

in address zero and the consecutive pixels in consecutive addresses. It would then gate the P clock into the ADC clock input, the write strobe of the RAM and the clock input of the counter. Hence each pixel would be digitised and written into the correct address of the RAM. It would continue doing this, with some subtleties during H sync pulses and porches, until the next V sync pulse is encountered. At which point it knows that it has grabbed a whole frame. Hence it stops gating the P clock into the ADC, counter and RAM and flags up completion.

However the problem with this circuit is that it is messy analogue technology. For example the sync detector is basically a pair of RC circuits and comparitors, and the phase locked loop is not a particularly nice beast. Therefore when the use of this classic circuit was considered, two concerns were raised.

Firstly due to the deadlines for the SNAP-1 spacecraft, it was likely that the first prototype PCBs would arrive back from manufacture a matter of days before the final flight board had to be integrated with the spacecraft. Therefore it would be useful if the prototype and flight boards powered up and worked 100% correctly the first time power was applied. Hence using analogue circuits which were likely to need a bit of tuning would probably be mutually exclusive with this goal.

Secondly, it is likely that after the cameras had received a sizable dose of radiation, that the composite signals would no longer have nice defined 0V, 0.4V and 1.4V levels. Similarly the signals might have become noisy once the MVS was integrated with the spacecraft. Hence the simplistic RC circuits would no longer be able to cope, and there would be no retroactive fix that could be applied to the hardware once the system was in orbit.

Given these concerns it was therefore decided that something different had to be done.

ThinkPads and Mwave

Around 1995/1996, IBM was a clear leader in laptop technology. Apart from the styling, one of the reasons for this was their investment in R&D, and hence the way in which they pushed the edge of the technological envelope in their production models. One of their innovations at the time was the Mwave DSP chip. Basically instead of having a dedicated modem and a dedicated sound card in their laptops. They instead had an Mwave, connected via an ADC and DAC, directly to the sound and modem ports. This is illustrated in Figure 3.8. Modulated data coming in on the phone line was therefore digitised by the ADC without demodulation and passed directly to the Mwave. Similarly data output by the Mwave was converted to an analogue waveform, and without further modulation passed directly to the phone line. Therefore the Mwave was directly responsible for performing, in software, all the modulation and demodulation processes normally performed by modem hardware. This strategy paid off in 1995 when the new 28K8 modem speeds arrived on the scene. As IBM was able to upgrade existing customers ThinkPads to 28K8, simply by providing them with a new device driver for the Mwave. Hence giving

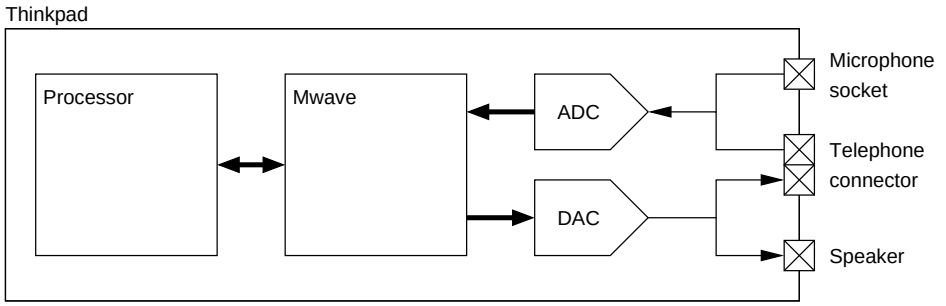


Figure 3.8: *The IBM ThinkPad Mwave architecture.*

strong credibility to the worth of performing all the processing in software.

Direct lessons could therefore be leant from this for the design of the MVS board. Basically, like ThinkPads out in the field, spacecraft hardware cannot be upgraded once in orbit. Therefore if like on the ThinkPads the MVS fed the composite video directly into an ADC, and then provided the raw output of the ADC directly to the processor. Then the processor could at any time during the lifetime of the spacecraft, be reprogrammed to extract the data from the composite stream, regardless of how the stream had become deformed by radiation or distorted by noise.

This would also be beneficial in the quest to get the board to work the first time it was powered on. This is because the hardware is minimised from the collection of sync separators and PLLs, into an ADC and some digital glue logic to interface the ADC to the processor. All the complexity is then contained in the software, which has a very rapid revision cycle of less than 5 minutes, as compared to a board revision which might take 3 weeks.

Software digitiser Mark I

The first design concept we produced for such a system is shown in Figure 3.9. The basics of this are that the composite stream is fed raw into the ADC. The ADC converts the stream into digital words and the words are then temporarily buffered in the FIFO. When the FIFO contains a significant number of words an interrupt is sent to the processor. The processor then starts reading the words from the other side of the FIFO and writing them into RAM.

However this concept was discarded for three reasons. Firstly we were unsure we could procure a FIFO before the SNAP-1 launch. Secondly because at the design stage we didn't have any functional StrongARM based hardware. We were therefore a little hazy on exactly how fast we could get the processor to read data from the FIFO and write it out into RAM. Hence we didn't want to risk building the system and then finding that the processor's memory bus wasn't actually fast enough to perform its job. Finally under this system the processor is likely to be spending all of its runtime transferring data from the FIFO to the RAM. Any interruptions

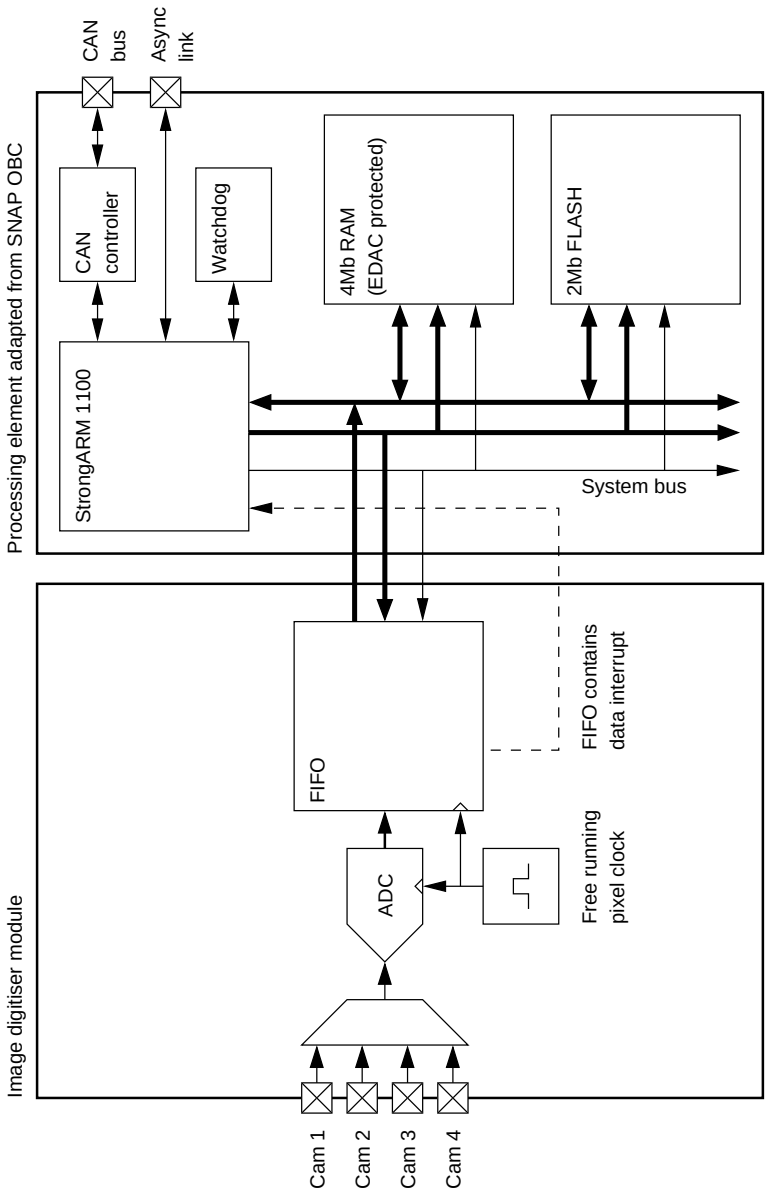


Figure 3.9: *The Mark I MVS architecture.*

in this process would result in loss of data when the FIFO overflowed. Therefore it was likely that the processor would not have enough processing capacity free to continue interacting with the rest of the spacecraft. Which isn't an ideal situation.

Software digitiser Mark II

To mitigate these problems the design in Figure 3.10 was developed. This design is similar to the mark I design, in that the ADC still converts the raw video stream directly into digital words. However in this design the processor isn't then burdened with the task of having to handle the digitised video in realtime. Instead there are two completely independent system modules, each capable of functioning entirely independently of the other. The first of these is the "Processing element", which is the cut down OBC discussed earlier. The second, termed the "Image digitiser module", contains the ADC to convert the raw composite video into digital words, a DMA controller and a block of RAM.

In normal operation the two buses are connected together by the bus divider. Therefore both buses are controlled by the processor, which can read and write to its own RAM, the video storage RAM and the command registers in the video DMA controller.

When the processor wants to grab a frame from the incoming composite stream it sends a command to the video DMA controller. The first thing this then does is instruct the bus divider to break the connection between the two buses. The processor now has control of system bus 1, but cannot access any of the devices on system bus 2. While the video DMA controller has control of system bus 2, but cannot access any of the devices on system bus 1. The DMA controller then grabs raw words from the ADC and writes them sequentially into the video storage RAM until an entire frame has been captured, at which point it signals back to the processor that it has finished and reconnects the buses. While the capture is underway the processor is free to get on with communicating with the rest of the spacecraft and performing processing, possibly of previously grabbed frames, in its own bank of RAM bank. Then, once the DMA controller has signaled it, it can read the raw composite video from the video storage RAM and perform the processing required to extract the pixel data.

This design therefore solved all the problems of procurement, timing and processor availability that were present in the mark I design.

Some of the nitty gritty low level details of this design will now be discussed in the following sections.

Frame boundaries

As discussed earlier the composite video entering the digitiser is a continuous sequence of frames separated by V-sync pulses. Un-synchronised to these frames the

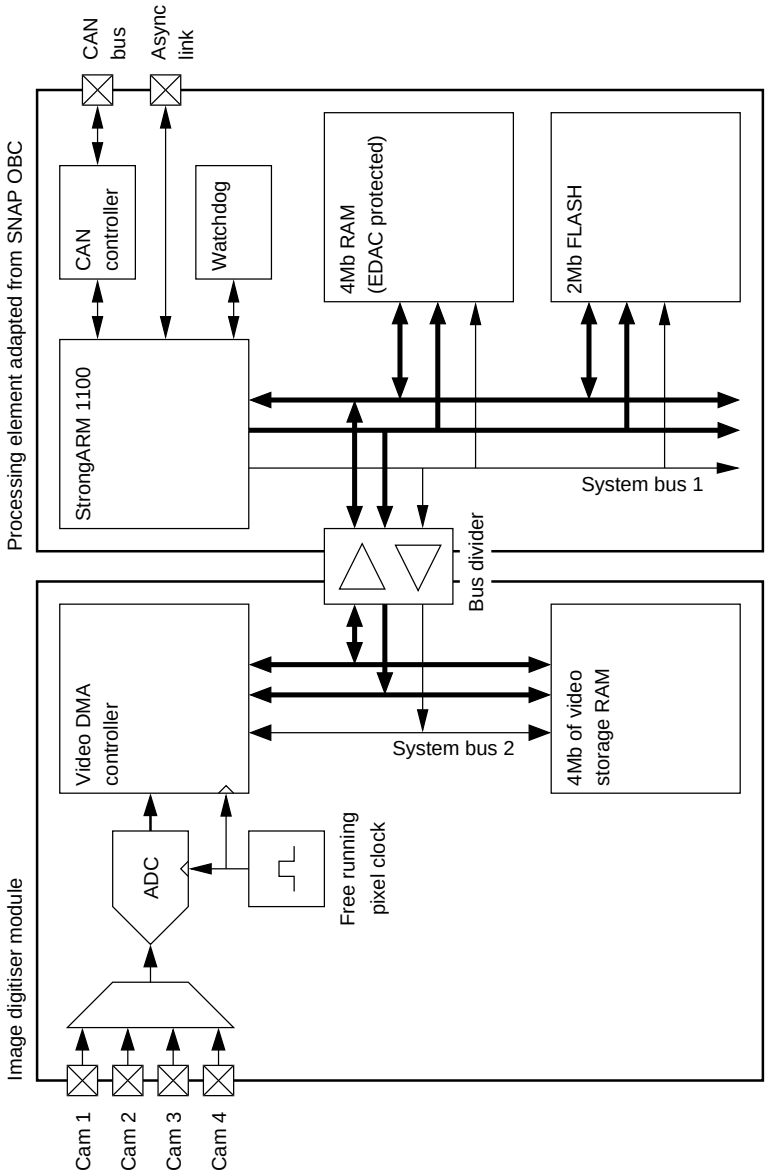


Figure 3.10: *The Mark II MVS architecture.*

processor request a frame to be captured. The video DMA controller then has to capture a single frame to the video storage RAM. But how does the video DMA controller know when the frame starts and ends? The bottom line is that it doesn't. This is because to know when a frame starts and ends, the video DMA controller would have to perform V-sync detection. Hence it would require V-sync detection hardware and we would have defeated the whole point of performing all of the composite video processing in software.

The solution we developed for this problem was to ignore where the frames started and ended. Instead we simply got the video DMA controller to start capturing data as soon as the processor commanded it to grab a frame. We then got it to continue capturing data for the period of time it takes the camera to output two frames. Therefore by definition you can always guarantee that there will be one complete frame in the data grabbed. Generally this will be surrounded by two partial frames, unless by coincidence the grab started exactly at a frame boundary, in which case the data will contain two complete frames. The software can then scan through the data until it finds the first V-sync. It can then extract the data from the frame that follows.

Pixel clock alignment

Earlier when we discussed the generation of a P-clock with the classical digitisation approach, we mentioned that the P-clock would be generated using a PLL which multiplied up the frequency of the H-sync pulses by the number of pixels in each line of the image. Therefore because of the nature of the PLL the first P-clock edge in each line would be almost exactly aligned with the edge of the H-sync pulse. Therefore the first pixel in one line would be sampled from the same place in the composite representation of the line as the first pixel in the next line. The second pixel would then be sampled in the same place as the second pixel in the next line etc.

However, in the Mark II system we were not detecting H-sync pulses in hardware. Instead we simply had to use a free running clock of the frequency that the pixels were being output in the composite stream (7.14MHz in the case of the cameras selected for the MVS). This meant that the same number of samples were clocked into the RAM as their were pixels in the line. However, unless it occurs by chance, the edge of the pixel clock that clocks in the first pixel of the line is unlikely to be aligned with the H-sync pulse. The upshot of this is that a vertical line in the image might appear to wobble to the left and right by up to half a pixel, as if it were viewed through a heat haze.

At the time that the system was designed we hoped that this effect would not be too noticeable. However only trying it out on the prototype would tell.

Camera support electronics

So far we've only discussed the fact that the composite video was fed directly into the ADC. However there were four cameras, outputting four different composite streams which somehow needed to be supported. The initial concept was to have each camera feeding into its own ADC, but this was dismissed as requiring too much board space and digital glue logic. Instead an Elantec EL4441C video multiplexer was employed. This could be switched by the processor to select which camera fed its data into the ADC. The side effect of this was of course that data could only be captured from one camera at a time. However the system could switch between cameras 25 times a second, giving a round robin frame rate from each camera of 6 frames per second. Which we deemed acceptable for our application.

To supplement this FET power switches were placed on each camera. Hence all of the cameras are nominally turned off to conserve power. Then when a camera it required it can be individually powered up.

Video DMA controller

Like the EDAC FPGAs the video DMA controller was implemented in an Actel A54SX08-VQ100 FPGA. Its internal block diagram is included in Appendix D.

The configuration registers of the video DMA controller are memory mapped onto system bus 2. So to grab a chunk of raw composite video the processor first writes to these registers across the system buses. It sets up in these registers the amount of composite video to capture (basically two frames worth as discussed earlier), the base address in the video storage RAM to start writing the data to, and the configuration of the camera power supplies and multiplexer. It then asserts the REQ line to command the DMA controller to start grabbing a frame.

The video DMA controller then asserts the appropriate control lines to divide the two buses. It is now the sole bus master of system bus 2, and hence can grab a block of raw video data into the RAM. Hence on every clock edge from the free running oscillator that is acting as the P-clock, it reads in the byte of data representing the next pixel from the ADC. It then internally buffers up 4 pixels before writing them out as a single 32bit word into the video storage RAM. Once it has captured the configured amount of data it reconnects the buses, hence handing system bus 2 back to the processor and signals to the processor that it has completed using the ACK line.

Chapter 4

SNAP-1 MVS Hardware Implementation

Between January and May 2000, the version of the MVS that was to fly on the SNAP-1 spacecraft was implemented. This chapter details the implementation process.

4.1 PCB design, manufacture and testing

During January 2000 the designs for the MVS' PCB were converted from sketches in notebooks into schematics using the CadStar design package. The resulting schematics are included in Appendix B.

February and early March 2000 saw SSTL engineer Trevor Edwards routing and fabricating the PCB. The masks of the 8 layer PCB which resulted from this process are included in Appendix C.

Two unpopulated PCBs were returned from fabrication late in March. SSTL clean room staff then populated these PCBs.

The first PCB became the prototype board, on which all the initial testing was performed. This is shown in colour slide 1. Fortunately the hardware complexity minimisation which had been key to the design process had paid off, as it was found that the board worked flawlessly first time. A couple of the images obtained by the prototype board are shown in Figure 4.1.

Unfortunately the pixel alignment problem we feared might rear its head, due to the use of a free running P-clock, rather than a phase locked P-clock, did rear its head. Fortunately we'd been planning for this eventuality. Our solution was to replace the 7.14MHz oscillator, which provided one edge per pixel, with a 14.28MHz oscillator which provided two. Hence the composite video would be over sampled by a factor of two. This would allow the software to more accurately locate the edge

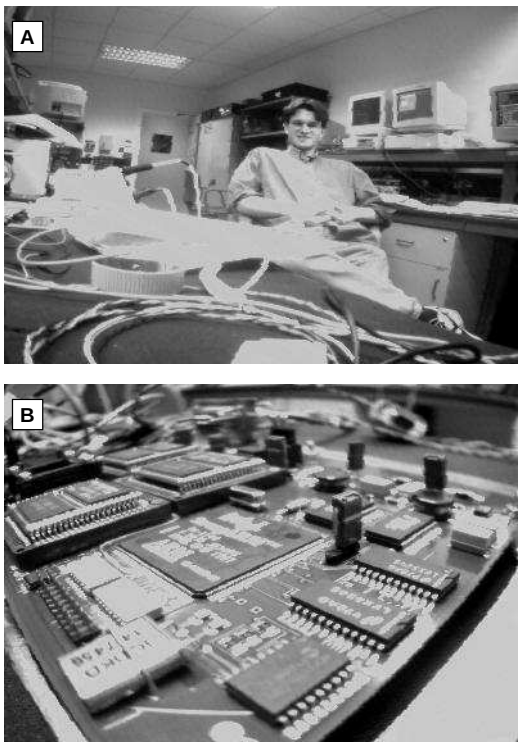


Figure 4.1: *(A) The first image ever taken by the prototype MVS board. (B) An image of the prototype MVS board taken by its self.*

of the H-sync pulse. Hence effectively allowing the phase locking to be performed in software rather than hardware. Unfortunately the only suitable crystal oscillator module that we had avaliable in our desk draws was a 20MHz unit, and we didn't have the time to procure a 14.28MHz one. Therefore we had to go with the 20MHz unit, which meant an over sample of approximately a factor of three. Which wasn't a particularly useful number for the software to have to down sample, but it had to suffice. This did of course mean that three times more raw data had to be stored in the video storage RAM at three times the speed. However we'd designed in enough headroom to allow for this eventuality. The result of all this was that you really needed to know what you were looking for to be able to notice the line jitter.

The second PCB then became the flight MVS board and was hence mounted into a payload box and had its cameras attached. This is shown in colour slide 2. After tests confirmed that the flight board was functioning correctly, the board was integrated into the spacecraft. This happened only two weeks after the first PCB was returned from fabrication.

4.2 Camera focusing

Focusing cameras for use in space is generally a difficult task. The reason for this is that if you focus them in air and then try to use them in a vacuum, then because

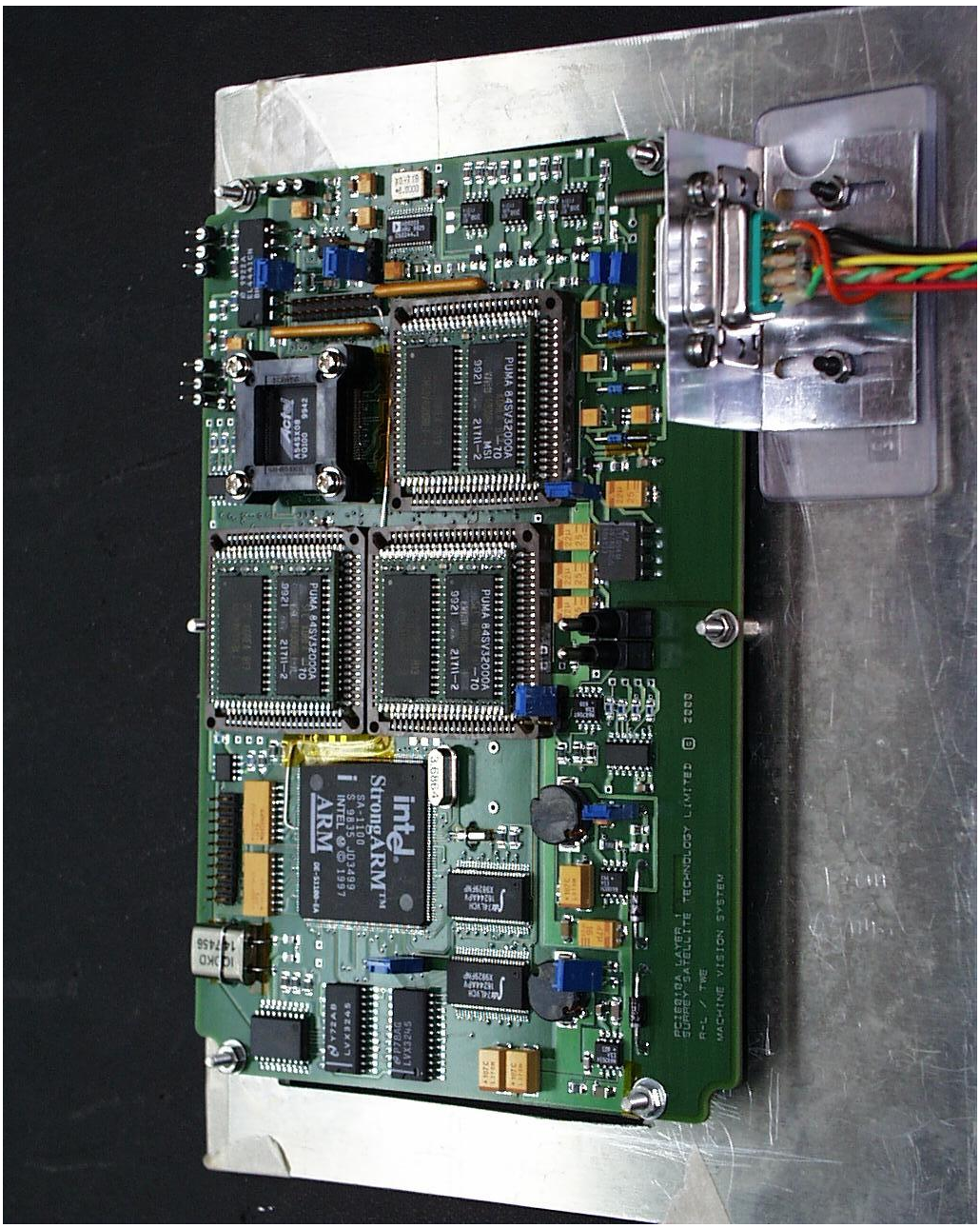
of the differences between the air/glass and vacuum/glass refractive indices, the cameras will be out of focus.

Fortunately however the small focal length of the lenses and low resolution of the imaging elements used on SNAP-1, meant that the focusing difference between air and vacuum was negligible. Hence the cameras were simply focused in air by pointing them out of the window to focus them on infinity.

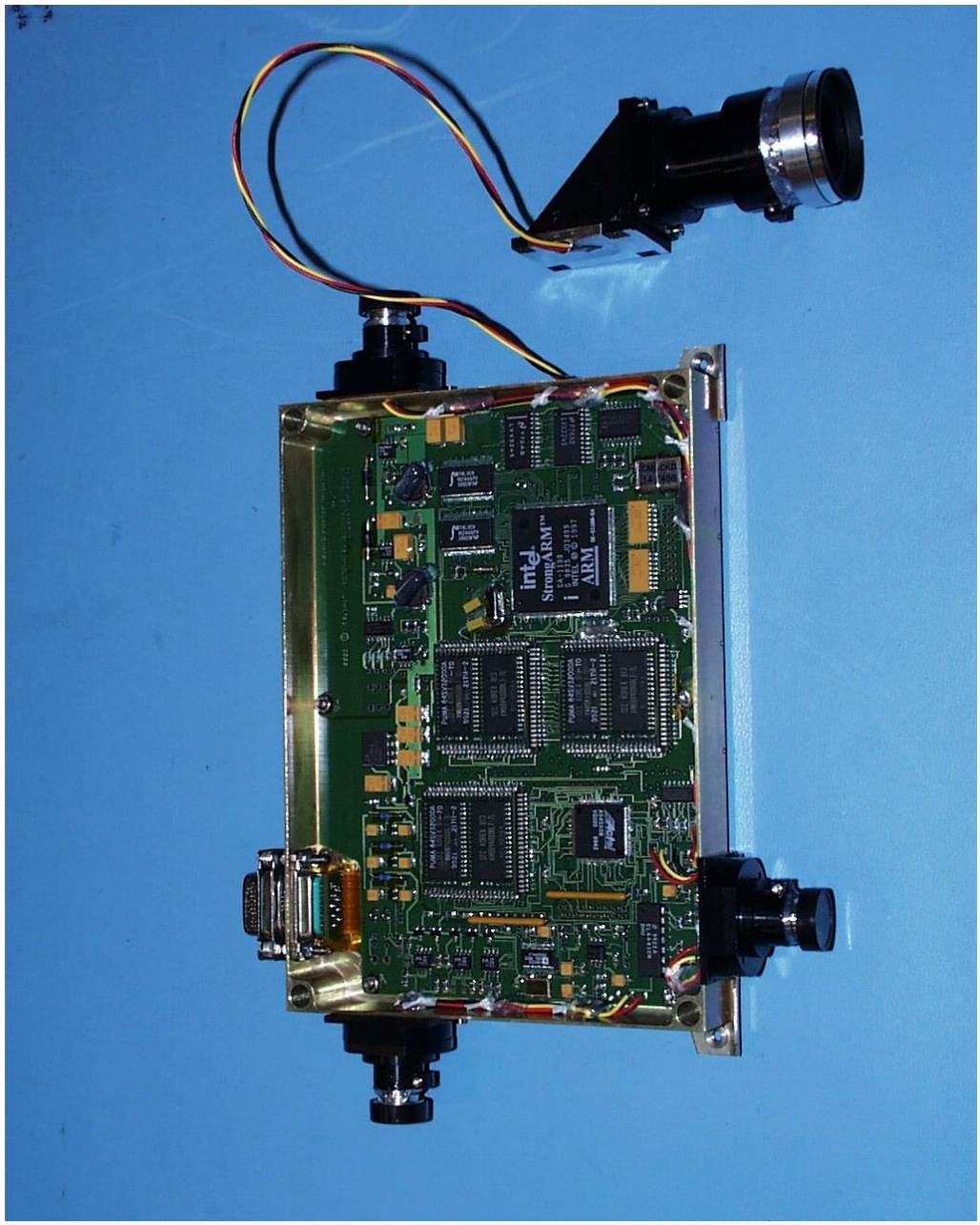
4.3 Spacecraft testing and shipping

Once the spacecraft had been assembled it underwent a set of tests as a complete unit. These included thermal cycling, operation in a vacuum, vibration tests and EM interference tests. During these tests a few small software problems were ironed out and the narrow angle camera had to be moved to stop it snagging on the separation system.

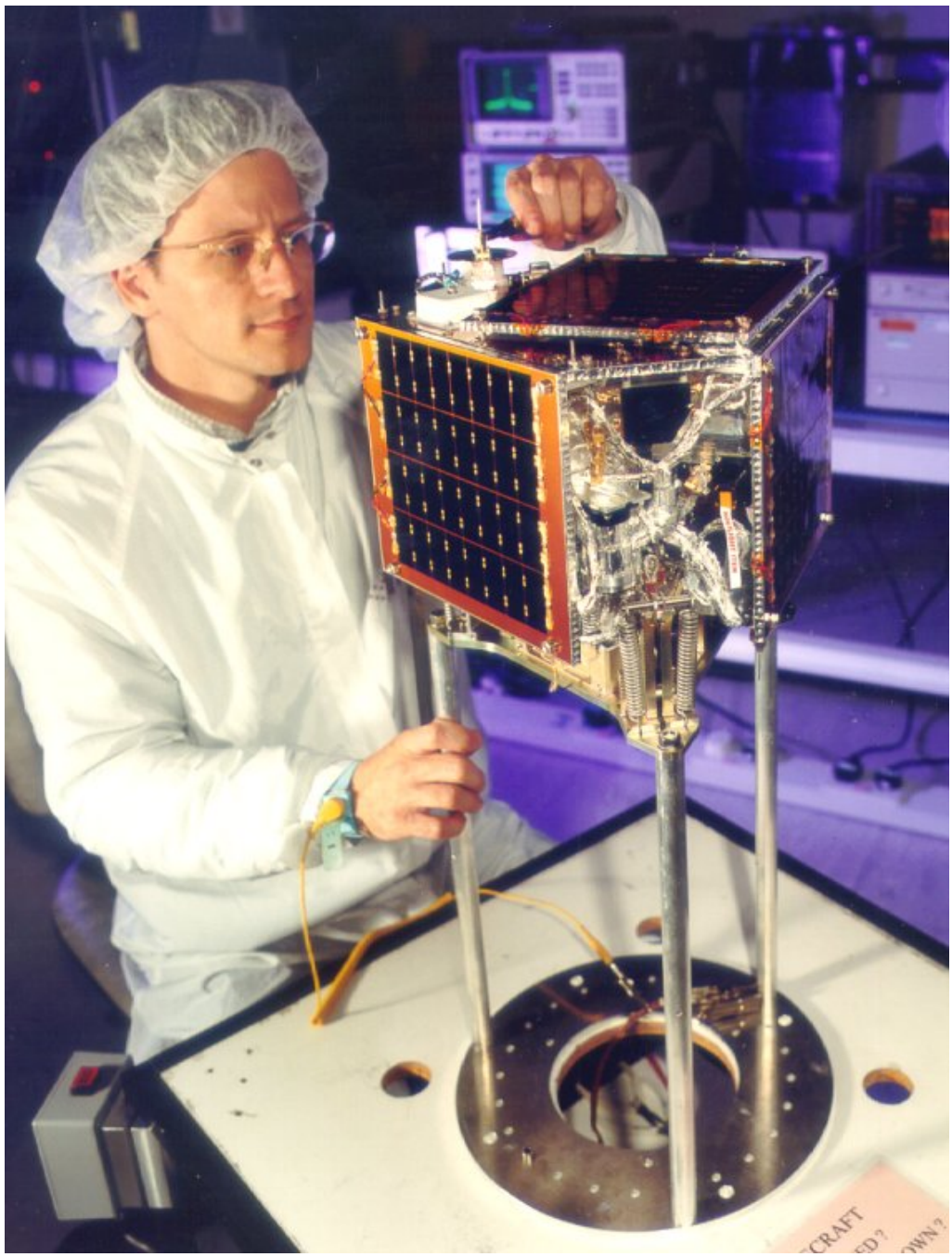
However the tests were successfully passed and the spacecraft was shipped to Russia for launch. Colour slide 3 contains a picture of the completed spacecraft just before shipping.



Slide 1: The prototype Machine Vision System



Slide 2: The flight SNAP-1 Machine Vision System



Slide 3: The SNAP-1 flight model undergoing final integration

Slide 4: The Cosmos 3M launch from Plesetsk northern Russia which carried the SNAP-1 spacecraft

Chapter 5

Software Architecture

Two pieces of software were written for the generic MVS and OBC boards with the intention that they be used on all missions containing these boards. They were the async bootloader and the BZL operating system. Two extra pieces of software were then written for the SNAP-1 MVS board in order to achieve its specific mission objectives. These were the SNAP-1 MVS mission commander and the SNAP-1 MVS AutoGrabber.

5.1 Async bootloader

The intention of the bootloader is that in general it will be the only piece of software in the FLASH RAM of a SNAP MVS or OBC board when it is launched. Its sole role is to run when the board is powered up or reset and support the SSTL async bootloader protocol [9]. In doing so it allows a standard SSTL ground-station to:

- Upload data into the board's RAM across a noisy link.
- Download data from the board's RAM across a noisy link.
- Issue a jump instruction to execute code at a specified address.

Hence the bootloader can be used to upload flight code to an MVS or OBC board when it is in orbit and execute it. It can also be used as a simple way of downloading results or logs stored in the board's RAM.

The bootloader is mission critical. If it fails to work then code cannot be loaded onto the board and hence the board becomes useless. For this reason the bootloader was designed to be as minimalistic as possible and was hand coded in assembler. Also the bootloader code configures the system clocks to their slowest possible setting, and adds several wait states to the memory timing. This means that even if the board becomes damaged in some way, there might be a chance that the slackening

of the timing allows the bootloader code to execute and hence it may be possible to perform some of the board's mission.

The bootloader code is included in Appendix E.

5.2 BZL

Once in orbit normal operations on both the MVS and OBC board need more than hard coded assembler. The software needs to handle interrupts from the CAN and async peripherals while at the same time performing image processing or attitude control computations. Therefore to handle this kind of concurrent activities you start to need a proper operating system.

When we were building the MVS and the OBC we had two options. We could either write our own custom operating system or port a currently existing operating system such as SCOS (The x86 operating system which ran on SSTL's previous 186 and 386 OBCs), Linux, RTems or VxWorks. Once again timescales made the decision. We only had 4 weeks between the point at which we started working on the OS and the launch of SNAP-1. We knew we could write our own custom, tight and compact OS within this time frame, but we didn't think we had the time to fully understand an off the shelf OS, port it and get over the obstacles its inherent design would inevitably present to our application. Therefore we took the path of known risk and went with the fully custom operating system. This decision was backed up by later experience when we attempted to port the Linux OS to the SNAP OBC board. It took us three months to get to the stage of booting it with a "Hello World" application, and then we discovered that there were some very difficult obstacles to overcome, such as trying to morph the CAN bus into Linux's concept of a network device.

The result of this custom OS development was the BZL operating system. It is a lightweight cooperative multitasking operating system, totaling around 10,000 lines of C and assembler code, which compiles to a 30Kbyte executable. A schematic overview of the modules within the BZL operating system and a typical applications load which might be flown on an MVS board is included in Figure 5.1.

For full details of the BZL operating system refer to its user manual in Appendix F.

At the time of writing the only application code written for BZL on the MVS board was a simple telecommand processor which provides a telecommand interface to the camera system. Hence a telecommand can be sent to the MVS to request an image be taken with a specified camera and stored at a specified RAM location.

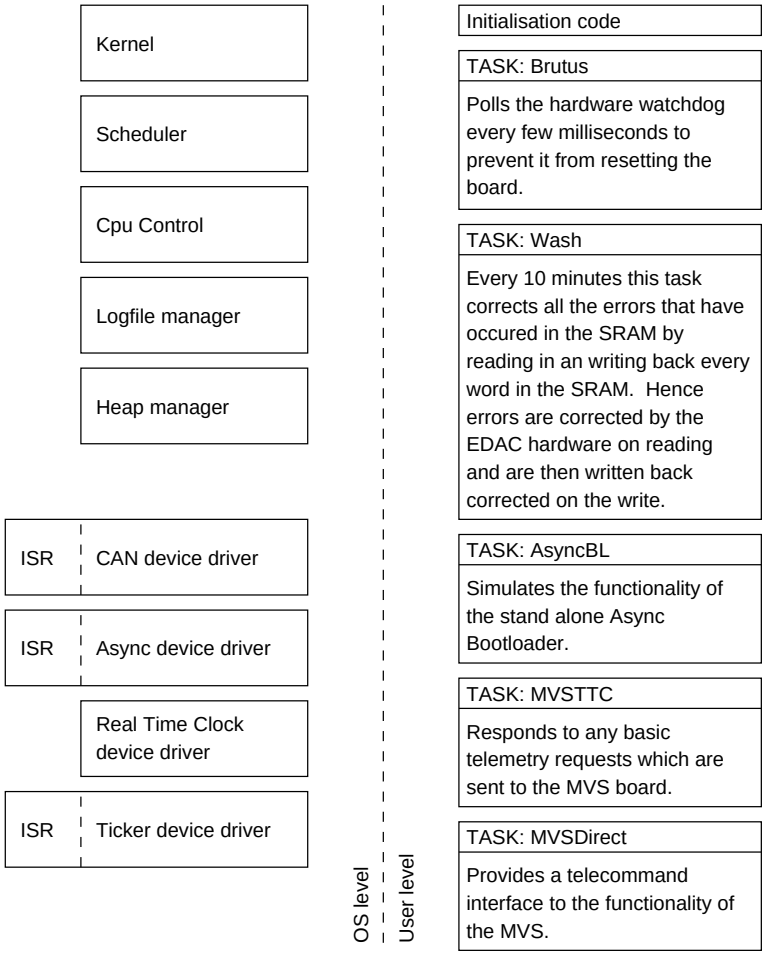


Figure 5.1: *The architecture of the BZL operating system with a typical applications load for the MVS board.*

5.3 SNAP-1 MVS specific code

One of the main mission objectives of the SNAP-1 MVS was to capture a time-lapse movie sequence of the deployment. This therefore necessitated that it carried extra code in its FLASH RAM to capture the sequence. These extra bits of software were the mission commander and the AutoGrabber.

The AutoGrabber's function was to perform the actual capture of the time-lapse movie sequence. The mission commander's function was to run the AutoGrabber when the MVS was powered up by the power system on deployment, then run the async bootloader when the AutoGrabber terminated. Hence once the deployment sequence had been captured and the AutoGrabber had exited, the bootloader could be used to download the results and upload new flight software.

5.4 C-Video codec

The most interesting part of the software load on the MVS boards is the C-Video codec. This is the piece of software which extracts a bitmap image from a digitised composite video signal. Hence it deserves a more detailed mention. Figure 5.2 contains a pseudo code overview of the algorithm developed for the task. You may wish to refer back to Figure 3.6 for details of the composite video format. Note that this overview glosses over some of the complexities involved in handling porches and calibration lines.

```
Set pointer to start of raw C–video block

Create empty output bitmap

Advance pointer to the end of the first V–sync pulse
    The codec locates the V–sync by looking for a contiguous series of bytes,
    longer than quater of the period of a row of pixels, all of which have a value
    significantly lower than the black level.

For n = 1 to number of lines in frame
    {
    Advance pointer to next H–sync pulse
        The codec locates the H–sync pulse by looking for the next series
        of contiguous bytes which have a value significantly lower than the
        black level.

    Align pointer to rising edge of H–sync
        Phase locks to the H–sync in software.

    For p = 1 to number of pixels in line of output bitmap
        {
        Copy pixel at pointer to output bitmap

        Advance pointer by 3 pixels
            Hence performing down sampling.
        }
    }
```

Figure 5.2: *Pseudo code representation of the C-Video codec.*

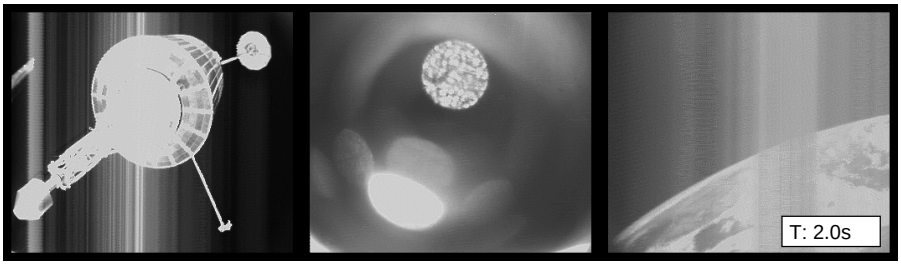
Chapter 6

Results

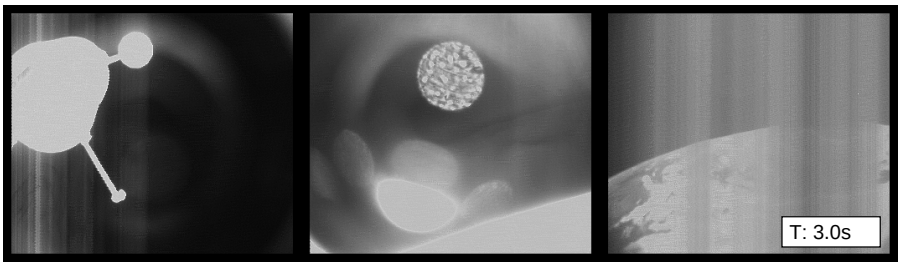
The SNAP-1 spacecraft was successfully launched on a Russia Cosmos 3M rocket from Plesetsk in northern Russia on the 28th June 2000. Slide 4 contains an photo of the launch.

6.1 Deployment images

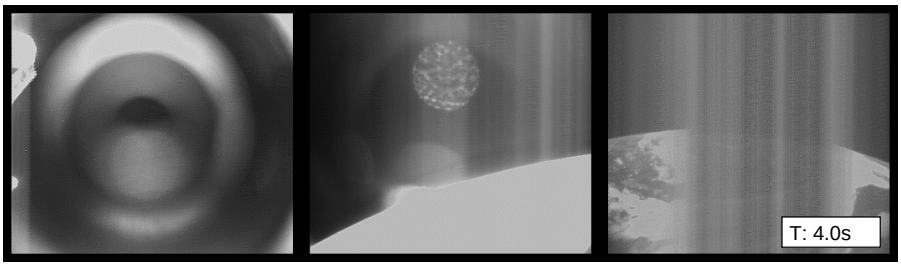
What follows is the set of time lapse images obtained by the AutoGrabber software during the deployment of the SNAP-1 spacecraft. At each point in time the images from the three wide angle cameras have been bound together into a panorama so that the full 270° arc of space can be observed.



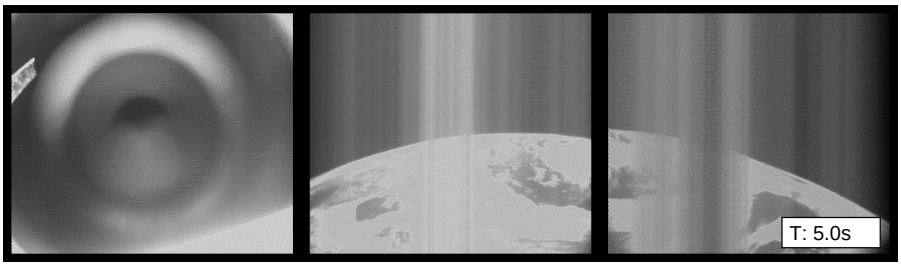
Taken two seconds after the SNAP-1 spacecraft deployed. This image shows three interesting features. On the left is the Russian Nadezhda spacecraft from which SNAP-1 deployed and is now drifting away from at 1ms^{-1}. In the middle is glare from the sun which is just out of shot. On the right is a limb of the Earth.

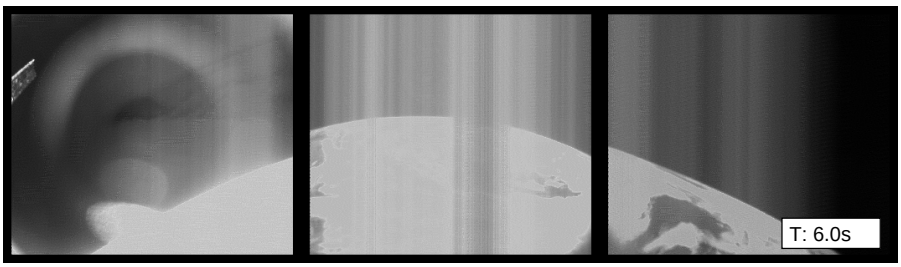


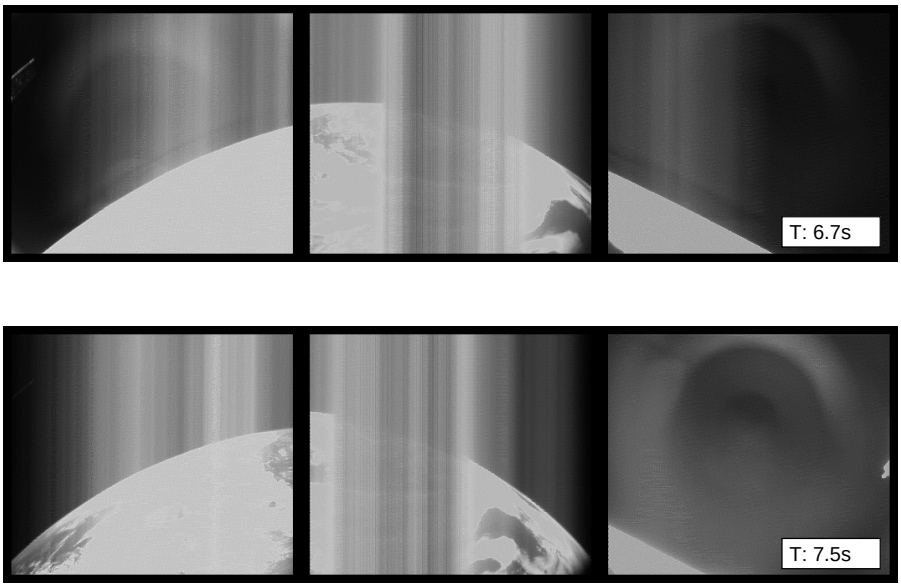
One second later and the Russian spacecraft is spinning out of view while the Earth moves in.



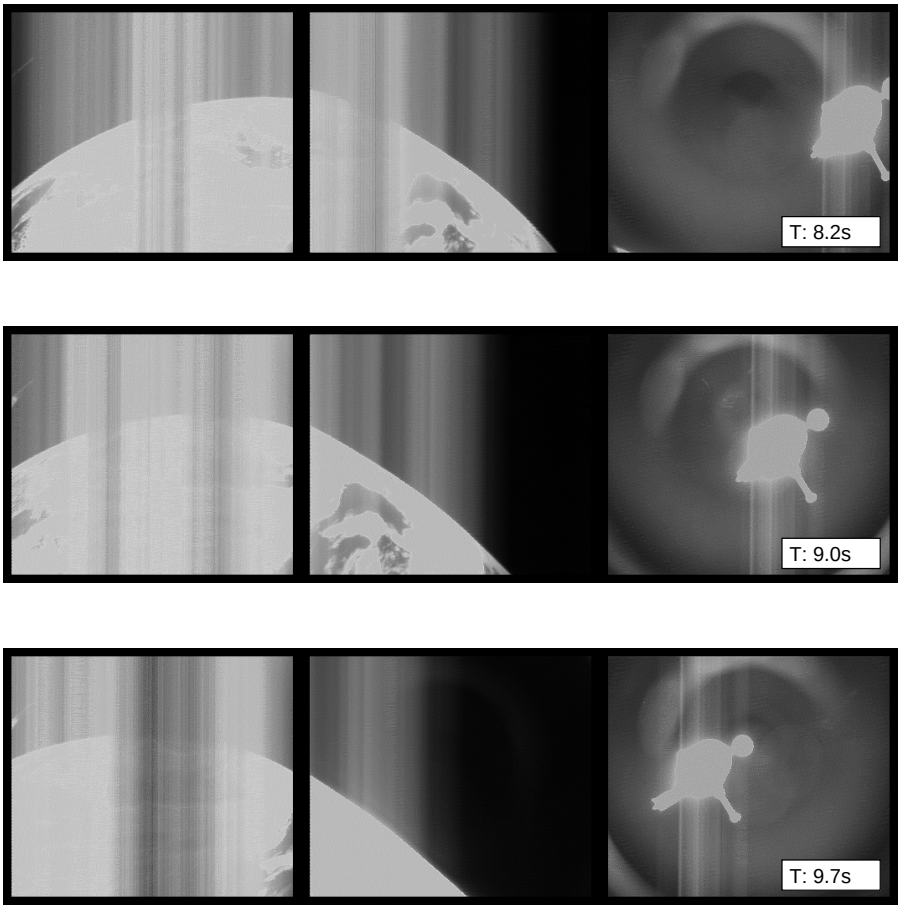
The Russian spacecraft has now spun almost completely out of view. Meanwhile the vertical banding on the Earth is due to camera saturation which is covered in Section 6.3.



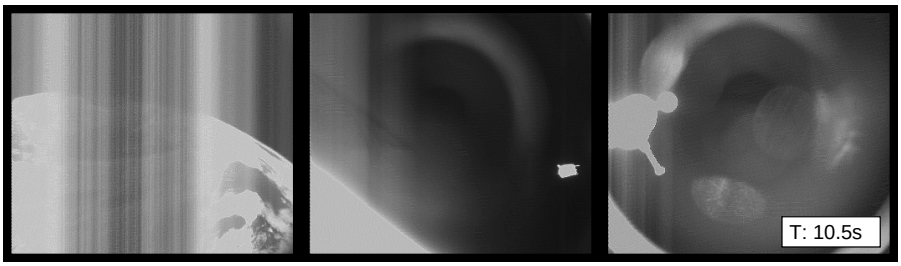




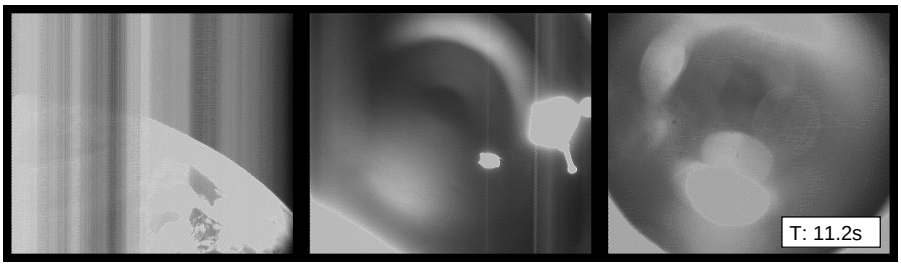
The Russian spacecraft just pokes back into view on the right hand side of the image.

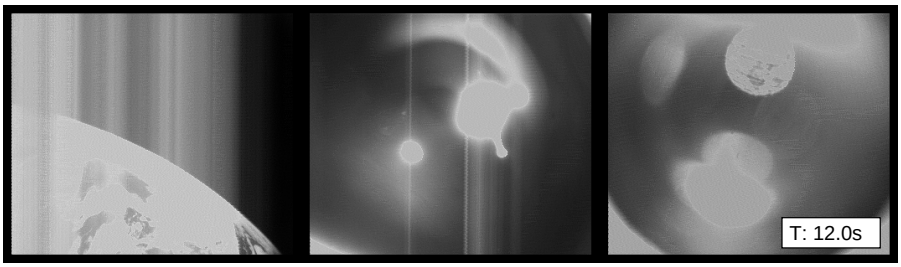


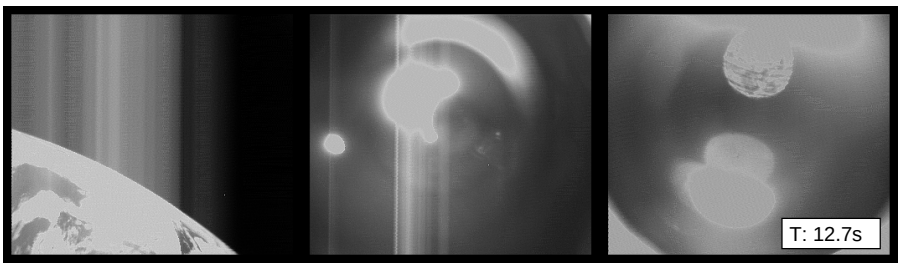
The other SSTL satellite on the launch, Tsinghua-1, starts to deploy from the Russian spacecraft.



Tsinghua-1 has separated from the Russian spacecraft and is drifting away from it at 1ms^{-1}.







6.2 Earth images

During the year following the launch of the SNAP-1 spacecraft the MVS payload was used to capture a number of images of the Earth. A couple of these are included in Figure 6.1.

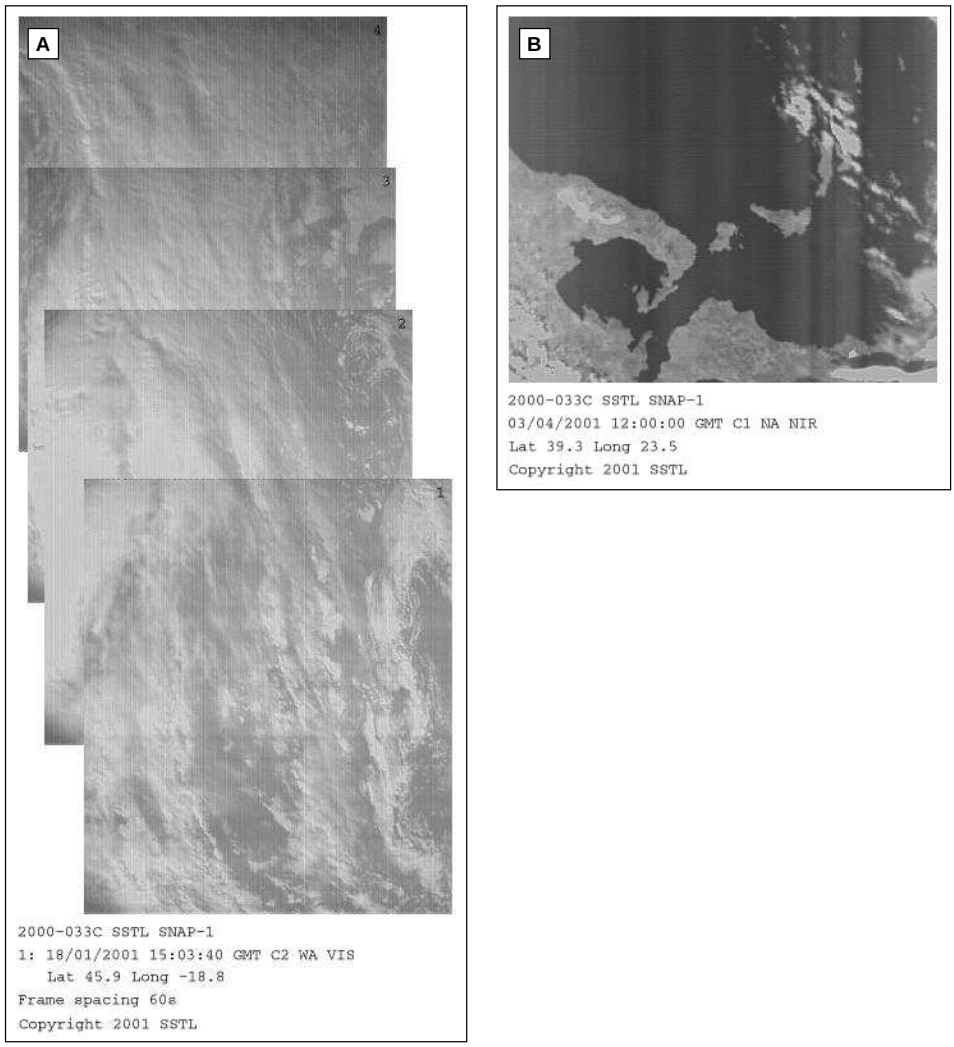


Figure 6.1: *(A) A strip of images taken over the northern Atlantic showing the cloud belts surrounding a large anti-cyclone which is just visible in the top left of the image. (B) An image of the Greek port of Volos. Taken in the near IR band with the narrow angle camera at a ground sample distance of approximatly 500m per pixel.*

6.3 Evaluation

From the imagery in this chapter it is clear that both the hardware and software worked almost flawlessly. However two problems are evident and these will now be discussed.

Gain control problems

In all of the images of the Russian spacecraft and Tsinghua-1 after $T = 7.0s$, although the spacecraft have well defined outlines, their bodies are saturated to white and contain no detail. It was determined that the culprit was the automatic gain control that is contained in the cameras. The purpose of this is to adjust the gain of the output amplifiers of the cameras so that the grey levels in images output are consistent across varying light levels.

The problem is that the gain control is a simple analog device. It works by employing a negative feedback loop, which attempts to keep the average of all the pixels in each image output equal to a mid range grey. This works fine for most of the images for which the camera would conventionally be used on Earth. However when the frame is filled with the blackness of space, the gain control cranks the gain up to its maximum to try to make space a mid level grey. Hence when you then have a small spacecraft in the field of view against such a black background, then because of the resulting high gain setting, the spacecraft will completely saturate, as seen in the images after $T = 7.0s$.

Two possible solutions present themselves for this problem. The first is of course to allow the CPU to set the gain factor of the camera's amplifiers. This is indeed possible with the VLSI vision sensors considered earlier. The problem then is exactly what value does the CPU set the gain to? For example when imaging a spacecraft which is being specularly illuminated by the Sun, the gain needs to be low. However when in partial eclipse the gain needs to be high. But how does the processor work out exactly what the gain setting needs to be for any particular image. In the end you might need to go as far as having to get the CPU to optically track the spacecraft you are trying to image and actively alter the gain such that it maximises the spread of the spacecraft's histogram. This is certainly an area for further research.

The second approach to this problem is to use logarithmic response sensors rather than the linear response sensors used on SNAP-1. The FUGA sensors discussed earlier are an example of such a sensor. Because of their logarithmic response they are capable of imaging massively contrasting illuminations in the same image. Hence reducing the importance of having the gain control so finely tuned. However this comes at a price, as their logarithmic response makes them very poor at imaging subtly shaded objects.

Camera saturation problems

In the deployment sequence the images of the Earth are all accompanied by a large amount of vertical streaking. This is because the SNAP-1 spacecraft and the Sun were aligned such that the Sun was specularly reflecting in the ocean, and hence saturating the imaging elements them selves. This is a fairly simple problem to solve, we just over looked it in the rush to get SNAP-1 launched. Basically all that is required is a strong enough filter or a small enough aperture stop to cut down the light intensity to a level that the cameras can tolerate.

Chapter 7

Conclusions

In terms of the SNAP-1 specific part of the project, the MVS board was about 99 percent successful. It achieved both its objective of capturing a time lapse movie sequence of the deployment and of capturing images of the Earth's surface, which given that only nine months elapsed from concept to launch, was a fairly staggering result. However it wasn't a perfect result. There were of course the problems of light intensity and gain control, but over all the SNAP-1 MVS mission was a success.

However in terms of the objective of producing a generic plug in MVS module for future SNAP missions, there is some work to be done. The core system performed well, but the gain control problems need to be researched and resolved.

Bibliography

[1] H A B Tiggeler M S Hodgart. A (16,8) error correcting code (t=2) for critical memory applications. In *DASIA*, 2000.

[2] C Underwood. 18 years of flight experience with the UoSAT microsatellites. In *ESCCON*, 2000.

[3] C Underwood. *Single event effects in commercial memory devices in the space radiation environment*. PhD thesis, Surrey, 1996.

[4] Positronic industries. http://www.positronic.com/.

[5] Bosch. *CAN Specification*, 2 edition, 1991.

[6] R Kiefer T Lillesand. *Remote sensing and image interpretation*. Wiley, third edition, 1994.

[7] IMEC FUGA imaging elements. http://www.fillfactory.com/.

[8] VLSI Vision imaging elements. http://www.vvl.co.uk/.

[9] H Tiggler and N Bean. The sstl async bootloader protocol. Internal SSTL document.

[10] J Sellers. *Understanding Space*. Mc Graw Hill, 2000.

[11] D Jagger. *ARM architectural reference manual*. Prentice Hall, 1996.

[12] Intel. *Intel StrongARM SA-1100 microprocessor developers manual*. Intel, 1999.

[13] B G Schunck R Jain, R Kasturi. *Machine Vision*. McGraw-Hill, 1995.

[14] R Boyle M Sonka, V Hlavac. *Image Processing, Analysis and Machine Vision*. Chapman and Hall, 1995.

[15] J Watkinson. *MPEG-2*. Focal press, 1999.

[16] M Brady. *Robotics Science*. MIT press, 1989.

[17] M Boas. *Mathematical methods in the physical sciences*. Wiley, 1983.

[18] D Lewine. *POSIX programmer's guide*. O'Reilly, 1994.

[19] B Kernighan and D Ritchie. *The C programming language*. Prentice Hall, 1988.