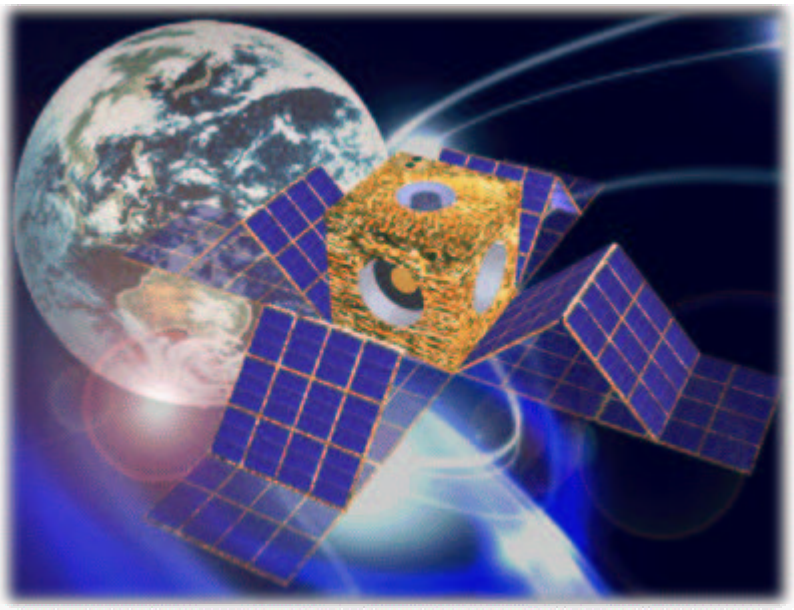
Onboard Computer for Pico Satellite



Ørsted • DTU
Technical University of Denmark
January 2002

Jonas Sølvhøj, c973442
Malte Breiting, c973568
Morten Briand Thomsen, c973709

Abstract

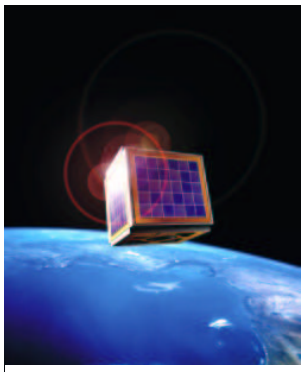
This paper is the result of a mid-curriculum project carried out at The Technical University of Denmark. It is about the design and construction of an onboard computer for a student picosatellite. The paper describes how the requirements of the computer affect the choice of components and the overall design. Furthermore, the choice of a debug interface and the effects of space radiation are discussed.
As the construction of the onboard computer is not yet complete, the future plans of the project are described.

Table of Contents

The DTUsat

This paper is about the construction of an onboard computer for a student satellite, which is being built at The Technical University of Denmark.

The satellite is categorised as a picosatellite, which refers to the small size of the satellite. Commercial satellites are, although varying in size, usually quite big and weighing in the range of 1000 kg's. Our satellite is restricted to 1 kg and a size of 10×10×10 cm – and is therefore referred to as a cubesat. Because of its small size and low weight, many cubesats can be placed in the same launch as secondary payloads. In our case, there will probably be one or two commercial satellites as primary payloads and 16 cubesats as secondary payloads. Because of our secondary status, we have no influence on the orbit our satellite will have, we have to do with whichever orbit the companies of the primary payloads select. It seems that a polar orbit (north-south) in a height of about 600 km is likely. As the exact orientation and height of the orbit is not of the greatest importance to us, this will do fine.



Picture of a cubeSat

Because of the relatively humble conditions mentioned above, the price of getting the satellite in space is very low compared to the prices of commercial satellite launches. This is a key factor allowing us to dream of our own satellite in space.

In addition to the restrictions on size and weight there is a range of other specifications described in "Cubesat – Design Specifications Document", which can be found at Stanford University's internet pages[1]. These specifications primarily concern the exact measurements and shape of the satellite, so that it can be ejected successfully from the spacecraft without interfering with the other cubesats. It also specifies requirements of robustness, in order not to have bits and pieces tumbling about during the launch and ejection of the cubesat.

Time of launch is yet to be determined, but our goal is to have a computer ready for implementation in the satellite in the summer of 2002. This will give opportunity to have a launch in the fall of 2002 or the spring of 2003.

As building a satellite is a very big project, the various tasks of designing the different parts have been assigned to different groups. The satellite has been divided into the following parts that will be constructed by one group each:

- Mechanical Design and Construction
- Power
- Onboard Computer (our group)
- Onboard Software
- Radio Communication Hardware
- Communication Software
- Satellite Antenna

[1] http://ssdl.stanford.edu/cubesat/specs-1_files/CubeSat Developer Specifications.pdf

- Ground Station Software
- Attitude Control and Determination
- Camera Payload
- Tether Payload
- System Engineering

The satellite has two payloads: a camera and a tether. A tether is a device that can change the orbit of the satellite. It is done by rolling out a long live wire in the magnetic field of earth, which will assert a force on the satellite.

As it is imperative that the different parts function together, communication between the groups is of outmost importance. Budgets have to be made, interfaces have to be defined and the knowledge contained in the groups has to be shared in the best possible way in order to hope for a successful satellite. To help communications a system-engineering group has been formed, consisting of minimum one person from each group. This group meets once a week to discuss various issues, progresses and setbacks.
Furthermore, a homepage has been created for sharing of information and files. It can be found at the address: *www.dtusat.dtu.dk*

The different groups have different levels of involvement. Some groups design their part in a small project rated as a 5-point course, some as a 10-point course and some as a 15-point mid-curriculum project (Polyteknisk midtvejsprojekt). The three of us have chosen to let this project be our mid-curriculum project. This means that we have spend three days a week working on the onboard computer during this semester. All groups have a supervisor affiliated, who administers and evaluates the course, but it has been a student project from the start and it is meant to be the ideas of the students that are realized in the project.

Why is a Computer Needed in a Satellite?

The satellite could be designed, so that each part was intelligent, enabling them to accomplish their tasks and communicate without a distinct central computer. This would imply that all parts would need to have some sort of circuitry to perform intelligent decisions, e.g. microcontrollers and memories. This solution would be very costly in terms of developing time, powerdissipation, and space compared to a solution with only one centralized and more powerful computer. Furthermore, internal communication could turn out to be problematic with numerous microcontrollers. So there are many advantages in having an onboard computer (OBC). One disadvantage with an OBC is that the different parts rely heavily on one and another. A fatal error in the computer will mean that nothing will function. Furthermore, one can fear that if one of the parts fail, it might jam the entire satellite.
To prevent total failure in case of this disaster, the radio group are designing the radio to be partly autonomous. This means that basic communication with the satellite will be possible without an operating onboard computer.

The OBC will be the part controlling the functions of the satellite. It will have an operating system installed that will manage the programs, which handle various tasks. For example, a program will perform attitude control (attitude is changed by a mechanical mechanism). This program will read the status of the attitude sensors and then regulate the attitude via actuators. Running a program like

this on a computer instead of making an attitude control system of traditional components for regulation like operational amplifiers, greatly improves the flexibility of the system.

Criterias of Success

In the satellite project there has been made a list of objectives from which we can determine our level of success. These are the prioritised objectives:

1. That all groups learn something
2. To finish and document all the different modules, so others, e.g. future designers of DTU satellites, can use them
3. To put a satellite into space
4. To receive a beacon signal, telling that the satellite is up and that something works in space
5. To receive a smart-beacon-signal from the on-board computer that sends more information to earth
6. To establish two way-communication with the satellite
7. To obtain three dimensional attitude control
8. To deploy two specific payloads

In our group, we have a similar list of objectives, being:

1. We learn something
2. To obtain experience in the process of an engineering project
3. A functioning test computer
4. A flight model computer that does not exceed any budgets (power, dimensions and durability)
5. A computer that works in space

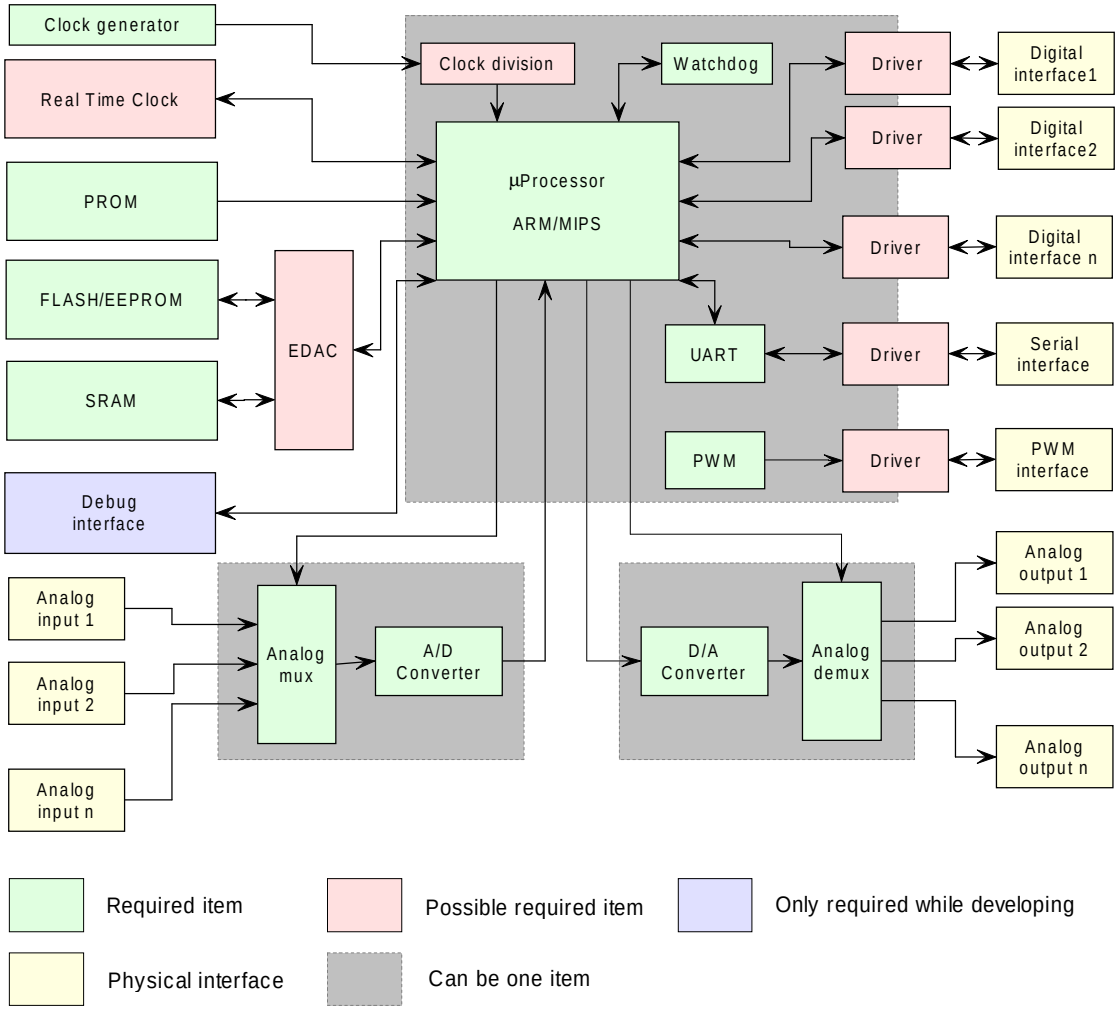
Without giving away the plot we can say, that the first three criteria have been met, leaving us at least partially successful.

System Overview

As mentioned earlier the computer has to perform several different tasks. The most important are:

- Regulation of attitude
- Communication with earth
- Measurements of analogue values e.g. temperature, battery voltage, and tether current
- Payload specific tasks: Tether deployment and control of camera

Each of these has different demands concerning the OBC. The fact that the computer has to operate in space also implies that the system has to be designed in a special way. In this section we will describe how these demands affect the design of the OBC. We will describe which blocks the computer must be build from, but not which chips we choose to use, as this will be covered in a later section. The system described in this section is depicted in the following figure:



The one thing that probably limits the design of the OBC the most, is the space-operating demand. As no one will be able to reset the processor manually in the case of a software malfunction, the system must be able to do this by itself. The way this 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 trapped 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 as shown in the figure.

When studying the figure you will find three different types of memory: ROM, flash, and RAM. These three types are also chosen because of the space-operating nature of this computer. 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 works 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 stored in ROM, but as it might be necessary to change parts of it, when the OBC is in space, it will be stored in flash.
The RAM 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.

In order to communicate with earth the OBC must be connected to a radio. The radio sends and receives data serially. The easiest way to implement this is to use an UART. The UART converts between serial and parallel data and also handles the serial timing. The UART is build-in in many processors, but can also be a separate unit.

The OBC has a number of different interfaces to other parts of the satellite – both analogue and digital. Since some processors may not be able to supply very much current it might be necessary to insert drivers between the digital interfaces and the processor. It has yet not been decided if the A/D- and D/A-converters will be placed on the OBC-board or if they will be placed on other boards. In a later section (”Future Development”) we will describe a likely placement of these.

The A/D converters described above will among other things do measurements of sun intensity and magnetic fields. Depending of these values control signals must be sent to the attitude control actuators. The actuators must be supplied with pulse width modulated signals (PWM), and therefore a PWM-controller must be included in the design either on the OBC-board or on the attitude board. Another solution is to generate the PWM-signals using software.

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.

One last thing of special interest is shown on the figure: 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.

Requirement Specification

In order to have a complete and fully functional satellite, it is imperative that all groups comply with a set of specifications, so that the resources in terms of space and power are used in the best manner without exceeding the budgets. The different groups, who individually have made preliminary budgets for their own parts, have formed these specifications and budgets. These budgets have been summed up in system-engineering perspective, mainly by the power- and the mechanics group, in order to have an idea of whether or not they are realistic. The final measures, weight constraints etc. are still to be determined, but below the present guidelines are listed.

- Resources – Three types of memory must be present:
  - ROM – minimum 8 KB
  - Flash – minimum 128 KB
  - RAM – minimum 512 KB

- Architecture
  - A number of A/D – D/A converters must be present
  - The CPU must have individually controllable I/O pins
  - The CPU must have a watch-dog to handle mal-function

- Power
  - There will be about 1W available for all systems in the satellite, so this is an absolute maximum

- Size
  - There will be room for a board of 6×6 cm

- Radiation - The computer must be prepared for two types of impact from radiation
  - Latch-up
  - Long term effects (must withstand minimum 2 KRad)

- C or ADA compiler must be available for the processor

- Desired features:
  - The software groups would greatly appreciate a processor with the following two characteristics:
    - 32 bit architecture
    - MMU (memory managing unit)
  - Use of ball-grid-array (BGA) IC's means difficult soldering process. Because of this we would very much like to avoid BGA components
  - Preferably the C or ADA compiler must be available at a low cost or free of charge

- The components must be able to function within the temperature range 0-40°C. However, a wider temperature range will be preferred

Choice of Components

Satellites are normally extremely expensive and account for years of development. To improve the odds for success, 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. This form of certainty is very expensive – for example, we found that the price of a space-tested 8 KB memory block was Dkr 13.000.
A cheaper alternative is to choose a kind of 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. Since our satellite is build primarily for educational purposes, all groups in the satellite project have agreed to yield to this temptation. The way we see it, a university is the perfect place for such experiments. Furthermore, we will perform tests to verify that the components do not collapse with the first signs of space radiation.

As the section "System Overview" showed, the computer consists of various components – processor, RAM, oscillator etc. The abilities of the computer and fulfilment of the requirement specification is, of course, highly dependent on the choice of components. Because of this, these choices have been made after many considerations.

The Processor

We need a processor for the computer that can execute the onboard programs in a reasonable way. The compulsory tasks of the satellite (power control, communication, attitude control etc.) could be managed by a quite humble processor or microcontroller. Processors and microcontrollers from a few years back would do the job with no problems and we could choose a very reliable one.
Nevertheless, we examined both new and old processors because of the above argumentation on why to take a chance on newer products.
We found that newer processors have made drastic improvements on power consumption even though the speed of the processors has greatly increased. In other words, we can have a 32-bit processor, which will please the software groups very much, that uses less power than an older and less powerful 16-bit processor. So, from the beginning the search for a processor has been narrowed down to a search for a low-power 32-bit processor.

Which characteristics did we look for?

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 the average specification of the component. The categories are:

- *Power consumption*
  As we only have a maximum of 1W available to all systems in the satellite, it is of outmost importance that the processor has low power consumption. The different companies have specified their numbers in different ways, for instance Atmel states the power consumption relatively to the frequency applied to the processor, as this is variable. Others state it at a fixed frequency but here the workload (which also has influence on power consumption) varies. Furthermore, we have been told by MIPS (see ”External Contacts”), that these figures very often are exaggerated in the direction that serves them!
  This means that comparison between the different competitors is not straightforward and has to be made carefully.

  Power consumption has been given top weight of 5.

- *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. By taking the different materials, weight, size and so on into account, the temperature range can be calculated. The problem is that the satellite project is in lack of a group performing these calculations. This means that we have had to make an educated guess on the temperature range based on similar satellite projects. The range has been set to 0°C - 40°C for normal operation. But in special cases 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.

  Temperature range has the weight of 5.

- *Placement of pins*
  In ball grid array (BGA) components the pins are distributed underneath the chip. This means that soldering cannot be done with a traditional soldering iron. Typically, BGA chips have soldering paste applied to the pins from the factory. To solder the chip you have to warm up the paste in an oven.
  Checking for short circuits and dead connections is quite tricky and furthermore we do not know anyone with experience with BGA components. Because of this, we would like to have the components in packages where the pins are located along the edges (for example TSOP) so ordinary soldering can be used to attach the components to the PCB.

  Placement of pins has the weight of 5.

- *Number of pins*
  Even if the pins are situated along the edges of the component, soldering can be very tricky if there are hundreds of them on a small chip. In order to reduce the number of soldering errors a relatively small amount of pins is preferred.

  Number of pins has the weight of 3.

- *Number of I/O pins*
  All parts of the satellite need access to the processor via the I/O pins (in/out pins) of the processor. Though we would like to have a reduced number of total pins, it is preferable to have as many general-purpose I/O pins as possible.

  Number of I/O pins has the weight of 3.

- *Scalable clock frequency*
  Most processors run at a fixed speed, but some processors can be fed by a scalable clock signal. This means that it can run whichever frequency you would prefer up until a maximum frequency. This feature would be very nice to have in the satellite, as it would enable us to slow the computer down and save power in situations where great speed in program execution is not needed.

  Scalable clock frequency has the weight of 4.

- *Availability*
  Far from all processors are in stock at Danish electronic resellers (like Arrow and Farnell). If a component is not in stock, it has to be shipped from the manufacturer's headquarters (this is often in the Far East), which normally takes 8 to 9 weeks. In our case this is far too long, as the mid-curriculum project is carried out in 13 weeks. This makes availability very important.

  Availability has the weight of 5.

We have also compared the processors on a few other subjects like space rating, supply voltage, onchip memory and some extra features. By space rating, we mean whether or not the processor has been tried in space before. The supply voltage is included, as it is preferable to run a standard supply voltage (like 3,3 Volt), which also can be used for other parts of the computer instead of having numerous supply voltages. Onchip memory is a nice feature as it is often very fast compared to external memory. Also if a program runs solely in onchip memory, the external memory can go to stand-by status, which consumes less power.
Comparisons are made on basis of the datasheets of the different components.

	Power consumption	Temperature range	Placement of pins	# of pins	Archi-tecture	# of I/O pins	Space rating	Supply Voltage	Onchip memory	Scalable clkfreq.	Other features	Avail-ability	TOTAL
Reference	<1W	-40-80 C	edges	100	32 bit			3.3V					
Weight	5	5	5	3	2	3	1	2	1	4	2	5	
Motorola:													
MCF5206e	--- (400mW)	+	+	0 (160)	+	0	-	+	-	-	+		-5
MMC2001	+ (80mW)	+	+	+ (144)	+	++ (30)	-	+	+++	-	+++	----	12
Atmel (ARM):													
AT91M40800	+++ (4mW/MHz)	+	+	+ (100)	+	++ (32)	-	+	-	+++	+	+++	73
AT91M40807	+++ (4mW/MHz)	+	+	+ (100)	+	++ (32)	-	+	+	+++	+	---	45

	Power consumption	Temperature range	Placement of pins	# of pins	Archi-Tecture	# of I/O pins	Space rating	Supply Voltage	Onchip memory	Scalable clkfreq.	Other features	Avail-ability	TOTAL
Hitachi:													
SH7709A	+ (80mW)	- (-20-75C)	+	-- (208)	+	++	-	+	-	-	+		5
H8/3334V	-- (180mW)	+	+	+ (80)	- (16 bit)	- (16)	-	+	+	-	++		0
MIPS:													
V850, NEC	++ (50mW)	+	+	+ (100)	+	+	-	+	-	-	+	----	11
TinyRISC, LSI													
Intel:													
StrongARM:	-- (300mW)	--- (0-70C)	+	+ (144)	+	++ (32)	-	+	-	+	+		-3
Siemens:													
TC1775		+	----	--- (350)	+	+++ (100)	-	+	+	+	+++		4
c167CS	-- (100-500mW)	+	+	+ (144)	+	+++ (111)	-	+	+	+	++		24
Microchip:													
PIC16F877	+++ (20 mW)	+	+	++ (40)	--- (8 bit)	--- (10)	+	+	+	+	++		31
PIC16C77	+++ (20mW)	+	+	++ (28)	--- (8 bit)	--- (10)	+	-	+	+	++		27

Choice of Processor

As the table shows the winner is: *Atmel AT91M40800*. The score of the Atmel shows that it has many nice qualities and features.
Where it not for lack of availability, we would have preferred the R40807 version of the Atmel processor because of its presence of onchip memory. However, if it cannot be supplied it is not worth betting on.
The table does not do justice to the MIPS processors we were introduced to at our visit at MIPS. The reason they are not fairly represented in the table is, that MIPS processor mostly are used as processor cores in larger system-on-chip systems, which have too many unusable features and thereby too high power consumption for our needs. However, MIPS (the company) are in possession of “clean” processor chips without all the unnecessary features. These processors seem to meet the high standards of the Atmel processors. Furthermore, they include something called “memory management unit” (MMU), which is something the software groups are very interested in. A MMU ensures that different programs (processes) cannot accidentally perform memory writes in forbidden areas of memory, which leads to program errors and system crashes. Unfortunately, MIPS only use ball grid array (BGA) packages, which is why we did not investigate the possibilities of these processors futher.

In spite of the above, we feel that we have chosen a very capable and powerful processor that fulfils our needs. The lack of a large onchip memory can be counterbalanced with external memory and the lack of an MMU simply means that the software groups cannot write their program thoughtlessly (which we are sure they would not have done anyway…).

The RAM

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 looked 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.
  The size is stated so the organization of the RAM is shown. As 16 data lines are available from the processor the optimal width of the RAM is 16 bits. (512K times 16 bit equals 1 MB – 512K times 8 bit equals 512 KB and so on)
- *Supply voltage*
  A supply voltage of 3.3 Volt is preferred as this is the supply voltage of the other processor.
- *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.
- *Number of pins / Availability / Price*
  These categories were investigated in order to be sure they did not make the use of the component impossible.

Manufactor	No	Size	Voltage	Active Current	Standby Current	Power Diss.	Pins	Availability
Samsung	K6F8016V3A	512x16	3.0-3.6	4mA	0.5uA		44	yes
G-Link	GLT6400M16	64x16	2.2-2.7	50mA	15mA/5mA		44	
Alliance	AS6VA25616	256x16	2.7-3.3			132mW	44	
IXYS	PDM31096LL	512x8	3.0-3.6			65mW	32	Preliminary
IXYS	PDM21096LL	512x8	2.4-3.0			65mW	32	No
Alliance	7C254096LL	512x8	2.3-3.0			90mW	44	
Brilliance	BS62XV4000		1.2-2.4	15mA	0.25uA			
G-Link	GLT6400M08	256x8	2.2-2.7	45mA	15mA/5mA		32	
ISSI	IS62VV25616L/LL	256x16	1.65-1.95	36mA	9uA		44	yes

As the table shows, the Samsung RAM has very nice qualities in both size, supply voltage and power consumption. Furthermore, the price of the components is $13, which makes it affordable. So the Samsung RAM is our choice.
Unfortunately it is only availably from international resellers, which means that delivery will take 8-9 weeks. As mentioned before 8-9 weeks is too long in a project like ours. Because of this, we are using a bit older 512KB version of the Samsung RAM in our first version of the computer, as this is available from Danish resellers. The big advantage in using another Samsung RAM is that it has the

exact same pin configuration as the new K6F8016U3A mentioned in the above table. During the spring we will be able to order the new RAM, wait 8-9 weeks and implement it in a new version of the computer without having to change the design of the computer board. Both the new and older RAMs have propagation delays of 70 ns (corresponding to 14 MHz).
The older Samsung RAM we are using now has the following characteristics:

Manufactor	No	Size	Voltage	Active Current	Standby Current	Power Diss.	Pins	Availability	Price
Samsung	K6T4016V3B	256x16	3.0-3.6	60mA	15uA		44	yes	~13$

Static RAMs are developing very fast in these days. New technologies allow the power consumption to drop and the size to increase. The reason why the new Samsung K6F8016V3A is not available in Denmark is that it has only just been released internationally, but during the spring it will probably become available domestically, which means that we might not have to wait the 8-9 weeks mentioned above.

The Flash

Flash memory is needed to store the operating system and various programs in the satellite – its role is similar to a hard disk of an ordinary desktop computer.
The selection of available flash memories is not as big as it is the case with processors and RAM. Comparison of the ones available is primarily made on power consumption, size and packaging (we cannot use BGA components).
The flash memory Am29LV017D from AMD meets our demands easily:

- It has a size of 2 MB (the minimum requirement is 128 KB!)
- Supply voltage is 3.3 V and it uses 9 mA read current, 15 mA write current and 200 nA in stand-by mode
- The propagation delay of reads and writes is 120 ns (8.3 MHz)
- It was available from Arrow, Denmark

The flash will only be used during boot of the computer, updates of onboard software and possibly in special cases of data storage. This means that most of the time it will only consume 200 nA $\times$ 3.3 V = 660 nW, which is extremely low. Furthermore, its vast size can turn out to be useful.

Instead of a flash we could have chosen to use an E^2PROM, which has similar qualities. With an E^2PROM it is possible to rewrite a single byte at any address. This is a big advantage compared to a flash where you have to erase an entire sector at a time (in our case 64 KB). This would make modifications in onboard software easier and faster.
But it seems that development is done primarily in flash'es as flash'es consume significantly less power than E^2PROM's and at the same time can store greater amounts of data. Because of this we chose a flash.

External Contacts

As building an onboard computer for a satellite is not a standard assignment on DTU, we have also had input from outside DTU – both in terms of design ideas and funding. Here is a list of the people/companies who are contributing to the project:

The final printed circuit board for the computer has to be made professionally by a company with the equipment and expertise to produce it in 4-layers. Four layers will be necessary in order not to exceed the specifications on the size of the board, but a PCB like this is rather expensive. We have had contact to Martin R. Jørgensen, who is in the sales department of Elcon[2]. Elcon makes PCB's and is specialized in small quantitative productions, for example proto-type PCB's. This fits our needs perfectly and furthermore Elcon has offered to produce our PCB free of charge, as it is a student project. We have visited Elcon in Horsens, Denmark, where we saw the production facilities and made arrangements for the actual production of our board.

Jimmy Malmkvist is running PowerCAD[3] (a one-man company). He is a professional in the area of routing PCB's – he has several years of experience. Jimmy has offered us to either help us make or do the entire routing of the final board free of charge. This is a great opportunity for us, because it might prevent many problems in the process of producing the board (at Elcon), since he knows exactly how they want the details of the PCB specified.

We have had Jimmy evaluate if it is possible to compress our present design to the size of 6×6 cm in a 4-layer PCB. Even after adding components for latch-up protection and possibly error-detection of memory accesses, Jimmy believes it is possible. The reason for this is that we will have two extra layers and we will be able to narrow the signal lines and vias.

As mentioned the computer will be designed with a processor from Atmel[4] because of its great features, low power consumption etc. After the processor-choice had been made, we contacted Atmel to let them know of our project. Atmel found it interesting and has supplied us with an

[2] www.elcon.dk
[3] www.powercad.dk
[4] www.atmel.com

evaluation board for the processor and 20 processors free of charge. We have used the evaluation board for comparisons with our own board, and the 20 processors are necessary in order to have specimens for the test boards, the radiation tests and the final boards.

Peter Davidsen works as an engineer at Terma[5]. We were introduced to Peter during an introductory course this summer, where he gave a lecture on digital design. He was a part of the team designing the computer for the successful ”Ørsted” satellite. In the beginning of the project Peter contributed with ideas to the design of the computer – mainly regarding mechanisms to solve problems with radiation. These ideas have not been realized yet, but when we move forward in the project during the spring, this area will be very important.

MIPS[6] processors have many interesting features, so in the process of choosing the processor for the computer we contacted MIPS, who are situated in Ballerup, Denamrk. MIPS found the project of great interest and offered to supply us with processors, testboards and support on these processors in case of questions in the process of creating the design. The possibility of technical support on a processor is quite unique, as support is not available from ordinary resellers (like Arrow or Farnell). Unfortunately, we had to turn down MIPS in an early stage, because all MIPS processors are in ball grid array (BGA) packages. BGA components would make it impossible (or very hard) to produce test boards ourselves, which would mean, that we would have had to change our entire approach to the problem (all soldering would have had to be made professionally, which would be very expensive and time consuming).

We would like to take the opportunity to thank all of these companies for their help. Without them, a student project like this would be hard to complete – especially financially. Furthermore, we have been introduced to the companies, given insight to their business and learned something about what it would be like to work as an engineer in such a company. This has been a very exciting and positive experience.

[5] www.terma.dk
[6] www.mips.com

Component Description

CPU Description

As mentioned we have chosen an AT91M40800 from ATMEL as CPU. The AT91M40800 is a 32-bit RISC processor built on an ARM Thumb core (ARM7TDMI)

"Designers can use both 16 bit Thumb and 32 bit ARM instructions sets and therefore have the flexibility to emphasise performance or code size on a sub-routine level as their applications require."[7]

The ARM core is widely used in several processors. Since all the processors, which use the same core, use the same instruction set compilers are easier to find. The fact that the processor can execute both 16 bit instructions and 32 bit instructions, gives an opportunity to use the fast and less power demanding 16-bit data width whenever possible.

All the CPU functions are set-up by writing to the different internal control registers of the CPU.

Memory

The AT91M40800 only has 8KB of internal RAM. The internal RAM is usually used for the stack memory because it usually is faster than the external memory and has a 32-bit data width.
The CPU has a maximum of 64MB memory address space, which is more than sufficient for this project.
The external memory is mapped after restart of the processor. Placing the remapping software on address 0x0 in non-volatile memory does this. The remapping software contains the information for the CPU about the different kinds of memory modules connected to the CPU, such as how fast the memory is and what chip-select it is connected to.
The external memory can have a 16-bit data width or 8-bit data width. To determine whether the boot memory is 16-bit or 8-bit the BMS pin is held low or high 10 clock cycles before reset is released.

I/O lines

The processor has several programmable I/O pins – 32 in total. 6 of these pins are dedicated as general-purpose I/O-pins, the rest are multiplexed with other functions such as USART, chip-selects, external interrupts and timer/counter signals.

Timers/Counters

AT91M40800 has 3 timers/counters, which all have a resolution of 16 bits. All 3 timers have a waveform mode that allows each timer to generate two PWM[8] signals, both with the same frequencies.

USART

USART means Universal Synchronous/Asynchronous Receiver Transmitter, which is a serial communications device.

---

[7] Quote from *http://www.arm.com/armtech/Thumb*
[8] PWM (Pulse Width Modulation) This option is needed by the attitude group to control the solenoids in the satellite.

There are two build-in USARTs in the AT91M40800. They can be configured to work in either synchronous or asynchronous modes and the baud rate is selected as a division of the master-clock.

Tri State Option

The CPU has a feature that puts all output pins in tri-state mode; holding NTRI low during the last 10 cycles before reset is released does this. This feature can be very good for error detection. This is because it completely disables the CPU from the rest of the components on the board.

Watchdog

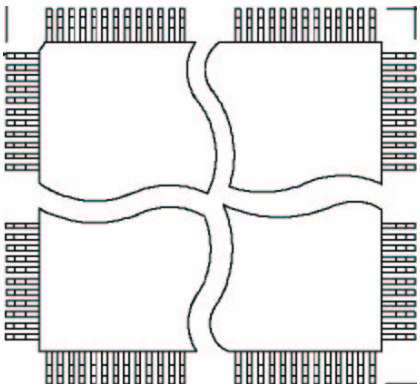
The CPU has a build-in watchdog. The watchdog is capable of making both an internal reset and sending an external signal, holding the watchdog dedicated pin NWDOVF low for 8 master clock cycles. This external signal could be used to notify external components that the processor has been reset.

Power Features

The power dissipation of the CPU is directly proportional to the speed of the master clock. When using external memory and a 3.3V supply the power consumption is 3.77 mW/MHz[9]. However, as mentioned before[10], this number should be taken lightly because the manufactures embellish the facts a little with these competitive numbers.
This means that it is possible to select at clock frequency that complies with the amount of power available on the satellite. It is possible to use a voltage supply between 1.8V-3.6V, though it is only possible to use a 16MHz or less clock frequency for 1.8V.
There are also other power features. It is possible to shut down the parts of the processor that are unused and thereby save power.

Package

The processor is available in a 100-lead, Thin Quad Flat Pack (TQFP). This means that it is a 100pin thin surface mounted component with outer dimensions of 16mm per side.



JTAG/ICE

The JTAG/ICE feature that the AT91M408000 possesses is quite significant and will be discussed in a later section (“The JTAG/ICE”).

[9] See “AT91M40800 Electrical Characteristics” (*http://www.atmel.com/atmel/acrobat/doc1393.pdf*)
[10] See section on ”External Contacts”

Flash

The flash component we have selected is an Am29LV017D from AMD[11]. It is able to use a voltage supply from 2.7V-3.6V. The Am29LV017D has a capacity of 2M×8 bit (=2 MB).
The Flash only has a databus width of 8-bit. The control signals for the flash are:

- CS - Chip Select
- WE - Write Enable
- OE - Output Enable
- RESET - Hardware reset
- RY/BY - Ready/Busy output

The RY/BY is used as a hardware detection to see whether or not an embedded algorithm is in progress or complete. RY/BY is an open-drain output and it is therefore possible to connect several together with a pull-up resistor in parallel. This is useful when several flash components are connected to the same CPU and the CPU only has one pin dedicated to detect whether or not the memory module is finished or not. The algorithms for the read and write process for the flash will be discussed in the software section of this report.

RAM

The RAM module we have selected for the testboard is not the same kind as the type that we intend to use on the flight-board but it is similar in the design. It is the type K6T4016U3B and is manufactured by Samsung[12]. It has a capacity of 256Kx16 bit. The K6T4016U3B is controlled by 5 control signals:

- CS - Chip Select
- WE - Write Enable
- OE - Output Enable
- UB - Upper Byte
- LB - Lower Byte

The UB and LB signal are used for selecting upper or lower part of the databus. This allows the circuit to save power when it only is required to access 1 byte.

Boot PROM

Since the first board that we are to make is a test board that will be used as a development tool for the satellite, it is not convenient with a ROM that only can be programmed once. Therefore we have decided to use a flash-component to simulate the boot ROM. To avoid making more drivers than necessary we will use a flash of the same kind as the flash for the flight-board even thought it is about 2000 times larger than the boot ROM will be on the flight-board.

[11] See *http://www.amd.com/us-en/FlashMemory/ProductInformation/0,,37_1447_1623_1468,00.html*
[12] See *http://www.samsungelectronics.com/semiconductors/SRAM/SRAM.htm*

Power Monitor

To ensure that the CPU does not begin to interact with external devices such as flash etc. before the supply voltage has reached a fixed threshold, we have included a power monitor in the design. It is connected to the reset pin of the processor, which it holds low until an acceptable supply voltage is present. The component we have selected for power management is a MAX811T. The MAX811T has a threshold at 3.08V before it releases the reset signal. It also supports and debounces a manual reset button. This component will only be used on the flight-board if the power group does not deliver a "power good" signal.

RS232 Transceiver

Since it is a test board that we are making we want to be able to test the two build-in USARTs. For this purpose we need a transceiver device to convert the 3.3V logic from our CPU's USART dedicated pins to the ±12V that is used on a PC's serial port[13]. The component we have found for this task is a MAX3223. The MAX3223 is designed to convert from 3.3V logic to standard RS232 logic and has support for 2 transmit channels and 2 receive channels, corresponding to 2 USARTS.

[13] RS232

Programming the Flash

The boot memory on the test computer is a flash component. It is necessary to program this module with the software to remap the memory connected to the CPU and software to upload new software to other memory components.

For this task we have explored several solutions.

- One way to program a flash component is to buy a programming device like the ones for programming PLDs[14]. But it would also require an interface capable of connecting our SMD component to a burning device, which is quite expensive and difficult to find.
  Another problem would be how to reprogram the flash after it was mounted. This could however be solved by applying a plug consisting of all the connections from the flash, on our board, and then use a probe between the programmer and computer-board.
  Then the next problem would be: How not to interfere with the other components on board. But since it is only the CPU that could cause problems and not the RAM or the other flash-component, it could be solved with the tri-state[15] option that is available on the CPU.
- Another way of programming the flash could be by making a programmer with the use of a microcontroller and then make the interface to the onboard flash as described above. The only problem is that this would be a very time consuming task because it also would require us to write the software for the microcontroller.
- The last option that we thought of was to use the embedded JTAG[16] interface. This could enable us to control the CPU and via the CPU program the flash. When considering the PCB design the JTAG solution would be much easier than the other solutions because it would not require the connective plug with connections to all the 36 pins on the boot flash. The JTAG would consist of a plug with only 4 connections to the 4 dedicated pins on the CPU (and a few capacitors and resistors) that constitute a standard JTAG output.

The JTAG is clearly the best way to program the boot-flash and will also provide other debug possibilities. Therefore have we decided to rely on this solution.

The JTAG/ICE

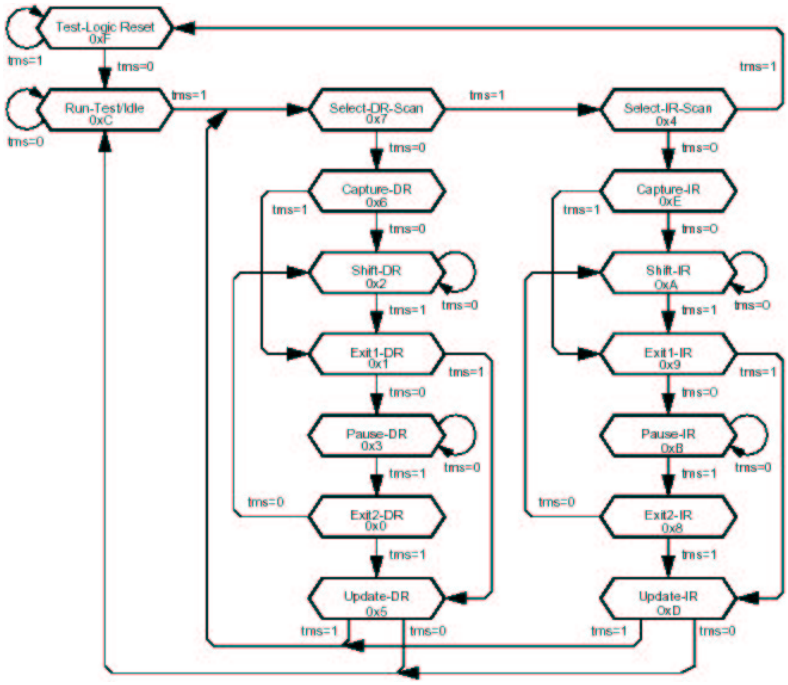
According to the last section we chose to use the JTAG/ICE feature of the processor for programming the flash. In this section we will provide a short description of what the JTAG/ICE is, how it works, and what it is capable of.

The standard

JTAG is short for “Joint Test Action Group”. As indicated by the name, JTAG is a standard proposed by a number of vendors of integrated circuits. Actually the correct name of the standard is IEEE 1149.1, as it has become an official IEEE standard.

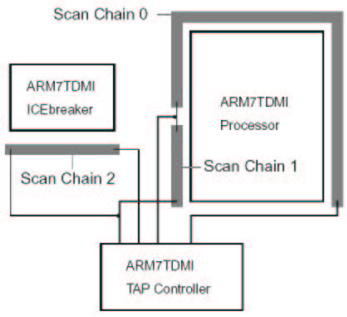
[14] Programmable Logic Devices
[15] See the section “Tri State option”
[16] See the section ”JTAG”

The purpose of the standard is to make a simple common interface for testing integrated circuits. It describes how a serial interface can be used to get data in and out of an integrated circuit, and describes the protocol the IC must implement in order to be compliant with the standard. The physical interface consists of 4 digital lines: TCK, TDI, TDO and TMS. As the interface is synchronous a clock TCK is required. The test-hardware must supply this clock. Data is shifted in serially using the TDI, and the result is returned using TDO. The TMS signal, which must be supplied by the test-hardware, controls an internal state machine of the device being tested. The purpose of this state machine is to control, what the applied serial stream of data is, and where it must go. The state machine is depicted in the figure to the right. Without going into much detail, it shows that the TMS controls the next state. It also shows, that two different vertical lines of boxes are almost equal. The purpose of the left column is to insert data into a register called the data-register (DR), and the right one to insert data into an instruction register (IR). With the correct data inserted into these registers the user can take control over the IC being tested.



By using the instruction-register it is among other things possible to halt the execution of a program (if the IC is a processor), ask the IC to do a self-test, and ask the IC for an identification code. It is also possible to select different so-called scan chains.

A scan chain is actually a serial shift-register. The scan chain is used as the data-register mentioned earlier. The easiest way to explain what is can be used for is to look at an example, namely the processor we have chosen to use.

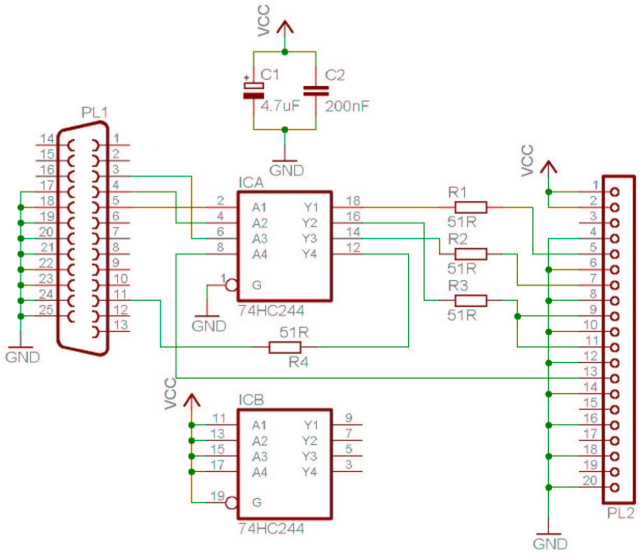


In the figure to the left it can be seen that the processor has 3 scan chains. Look at the “Scan chain 0”. It surrounds the processor-core. Each shift register cell of this chain is multiplexed with a signal controlling the core signal (i.e. the address bus and the databus). If we set the mux’es, so that the shift registers and not the normal control signals are connected to the core, we are actually able to load instructions into the core. If we let this instruction be a request to store the value of a register in the RAM, the value is written to the databus-interface of the core, but as this interface is connected to the shift register by the mux, the data is not written in RAM, but in the shift register. By shifting these data out we are actually able to see the contents of the registers. In this way it is possible to get data in and out of the processor. By changing if it is the shift register or the normal control signals that are connected to the core, one can also read from and write to the RAM. This is what we make use of, when we load a program into the RAM.

As mentioned the instruction register (IR) of the JTAG interface can be used to select different scan chains. In the previous figure it can be seen that our processor has a scan chain (no. 2) which connects to an ICEBreaker. ICE is short for "In Circuit Emulation". The ICEBreaker makes it possible to set breakpoints, that automatically halts the processor if for example an attempt was made to read from a specific memory address. In this way it is possible to use some debugging software at a host computer that downloads a program into the memory of the target processor and executes it. If a breakpoint is reached the debugging software can show this to the user and show the contents of all registers etc. This is a very powerful debugging facility.

Our JTAG interface

As can be seen from the above, only 4 signals are required in the JTAG. Therefore it is quite simple to connect a device to a host computer using this interface. The simplest way is probably to use the parallel port of a desktop computer. Commercially available devices can be bought to make this connection. An example of such a device is the Wiggler manufactured by Macraigor Systems[17]. We decided however to build our own interface, as we found a schematic of a device on the Internet that is pin compatible with the Wiggler. The nice thing about this is, that the software available for the Wiggler can be used with the device we build.



The schematic found at *http://sourceforge.net/projects/jtag-arm9* is shown above. PL1 is the parallel port connector and PL2 is the JTAG connector. In between those two a 74HC244 IC has been added. The purpose of this is to do a voltage level conversion from the 5V of the parallel port to the 3.3V of our processor. The supply voltage for the IC is taken from the JTAG interface. Therefore our test-board must supply this voltage. The 16 GND connection in the JTAG interface (PL2) serve as shielding lines.

[17] http://www.macraigor.com/

Design Description

We have decided to make a testboard with extra debug possibilities and with some simple test interfaces. The board's primary function is to show whether or not we have understood the datasheets properly and to locate logical errors in the design. The following components is used in the design:

- AT91M40800 – CPU
- 2 × Am29LV017D – Flash
- K6T4016U3B – RAM
- MAX811T – Power monitor
- 2 × LP2981 – Voltage regulator
- IQXO-71 – Clock generator at 12.288 MHz
- MAX3223 – Serial transceiver
- JTAG/ICE plug
- I/O connection plug

Since all of the components are described in the "Component Description" section we will concentrate on how to connect the components correctly in this section. The CPU is by far the most complicated of all the components and it is the central component binding all components together.

The CPU Connections

The pins on the CPU basically have 5 types of functions and some of the pins have more than one function. The 5 types are:

- Voltage supply
- CPU control
- External memory control
- I/O – Input/Output
- JTAG/ICE

Voltage supply

On the CPU there are 10 GND pins, 6 VDDIO pins, and 3 VDDCORE pins. The VDDIO gives voltage supply to the I/O-lines and VDDCORE supplies the CPU-core.

In the design of the evaluation-board[18], made for this processor by ATMEL, the VDDCORE and VDDIO are connected as one net. The evaluation-board has 4 layers and uses 2 layers for power planes and 2 layers for signals.
But since our board is a 2-layer board we have to mix the power lines and the signal lines in the same layers, which means that we cannot be sure that our supply will be as steady, as in the case of the evaluation board. To improve the supply voltages, we have decided to separate the two nets as close to the voltage supply as possible. Placing a jumper between the VCC and VDDIO and one

[18] See *http://www.atmel.com/atmel/acrobat/doc1706.pdf* page 6-8

between VCC and VDDCORE does this. Since none of the datasheets describe how to place decoupling capacitors and what size they should be, we have once again looked in the manual for the evaluation-board. On the evaluation-board they have used four 100nF capacitors, one for each side of the board. Because of this we have decided to also use 100nF-decoupling capacitors. However in our design we will use 9 capacitors, 3 close to VDDCORE and 6 close to VDDIO, just to be safe.

Since the computer is to operate in space, some considerations about how to handle a latch-up are necessary. A latch-up can occur when a proton hits a CMOS component and causes short-circuit between the layers inside the component. A latch-up can destroy a component completely and it is therefore important to protect the CMOS components. It is fairly easy to detect a latch-up because it causes the component to draw a lot of current. When this happens it is vital to turn off the power for a short period of time. But when power is turned off in one subsystem it is difficult to predict how all the others subsystems react. The easiest solution to this problem is to turn off the whole satellite and then reboot all once again. Since a latch-up is to turn the entire satellite off the latch-up protection task has been delegated to the "Power Group". Because of this there are not implemented any latch-up protection in the design of the test-computer. It is however still not known what and how the "Power Group" will implement to ensure latch-up protection.

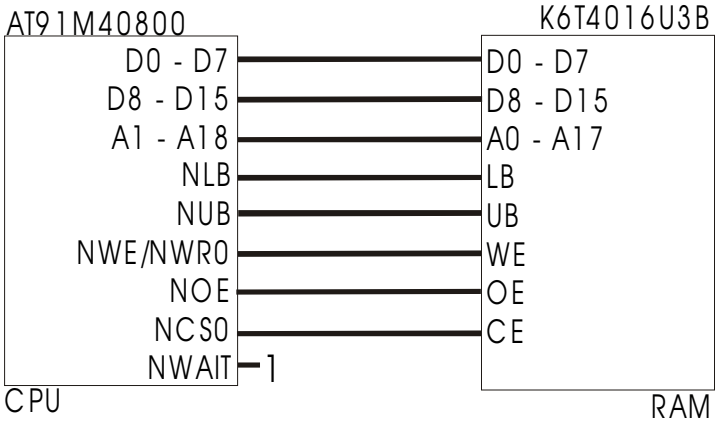
CPU Control

The CPU samples some information about the boot mode, 10 master clock cycles before NRST (reset) is released. The 2 pins that are sampled are BMS and NTRI. These two pins also have other functions after the boot sequence is over. It is therefore evident that the applied signal must be sufficiently weak. Applying the desired logic signal through a 400kΩ pull-up/pull-down resistor does this according to the datasheet[19] for the processor. In our design we have an 8-bit data width for the boot flash and therefore BMS is set to logic 1[20] during boot. The NTRI provides an option for all the outputs of the processor to be disabled and enter tri-state. To be able to use this option during error detection of the board we have included a jumper that selects between logic 1 and logic 0.
The NRST pin is connected to a power monitor (MAX811T) that pulls NRST low until the power supply is stable above 3.08V. The power monitor is also connected to a manual reset button so that it delivers a denounced reset when the reset button is pressed. The most important reason to apply this MAX811T to the design is to prevent the CPU from trying to read from the external devices, before they have the power supply they need. Only the CPU can operate at 1.8V. The reset signal is also connected to the two flash modules to make sure they are in a known state after reset. Another vital input signal to the CPU is the master clock, MCKI. A crystal oscillator of the type IQXO-71 at 12.288 MHz creates the clock signal. The IQXO-71 is a simple component to implement, as it only requires a voltage supply and 2 capacitors. The IQXO-71 also has an enable/disable option but it is unused in this application. The frequency of the oscillator is selected so that the baud rate on the USART can be exact set up to the standard speeds used for serial communication, and because it gives a reasonable proportion between calculating speed and power dissipation.

[19] See *http://www.atmel.com/atmel/acrobat/doc1354.pdf* top page 9.
[20] See *http://www.atmel.com/atmel/acrobat/doc1354.pdf* table 3.

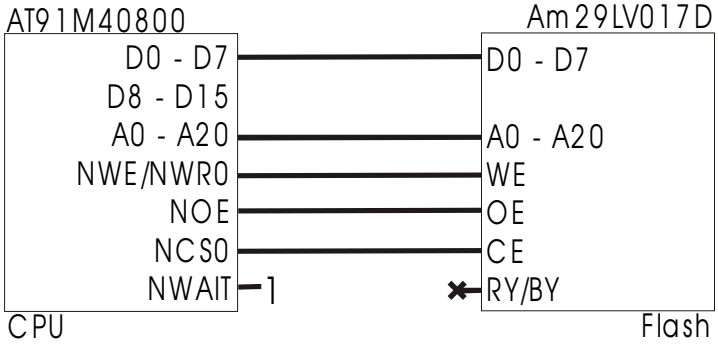
External Memory Control

The CPU has 3 external memory components to control, one RAM module and two flash modules. (One of the flash modules will be replaced by at ROM on the flight-board)
The CPU is connected to the external memories via the databus, the addressbus and some control signals. The addressbus is only used from A0 to A20. The used width of the addressbus is determined by the size of the largest memory module, which is the flash that has a capacity of 2 MB = 2^{21} bytes. The RAM and the two flash modules share these address lines as well as the data lines. But since they do not all have the same amount of memory they do not need an equally amount of address lines.
The RAM is connected so that half-word (one byte) access is also possible. Connections are shown in the figure below. The reason why it appears that A0 on the CPU is unused is that NLB is multiplexed with A0.



The connections between the RAM and the CPU

The two flash components are a bit different from the RAM since they only have an 8-bit data bus. The flash has an output pin called RY/BY. This pin is used as hardware detection to see if the component is active erasing or programming. This pin corresponds to the NWAIT on the CPU which adds wait states to the read or write cycle when pulled low. However it might be possible that the flash is so slow that it is better to use an interrupt pin instead of NWAIT so that the processor can work while the flash works in the background. We have however decided not to connect the pins because the detection can be done by software instead and to keep the design as simple as possible. The connection may be used in a later version of the computer.



The connection between the CPU and the boot-flash

The connections for the other flash module are the same as show in the figure of the boot-flash, the only difference is that NCS0 is changed to NCS1.

I/O – Input/Output

As mentioned, the processor contains 32 programmable I/O lines, but only 6 of these are dedicated general-purpose I/O lines. The rest of the I/O lines are multiplexed with other functions.
One of the hardware supported I/O functions is the USART. The CPU has two built-in USARTs. The USARTs can run in both a synchronous mode and an asynchronous mode. Since the USART on the satellite will be used for the connection to the radio, and the radio will be designed to use asynchronous mode, this is also the mode that we will use. The serial communication in the satellite will be conducted at the levels 0V and 3.3V. These are not convenient voltage levels for testing the computer through a standard PC's serial port. To convert the levels to standard RS-232 (to which the serial port of a PC comply) we use a MAX3223. The MAX3223 is a dual RS-232 transceiver similar to the popular MAX232 apart from a few extra features such as autoshutdown and enable/disable. But more importantly the MAX3223 is able to use a power supply down to 3V and not 5V as the MAX232.
The two RS-232 adjusted outputs from the computer are connected to two DB9 female plugs for easy connection to a PC.
To be able to use the board for evaluation of hardware-near software, where software controls I/O pins, we have added a plug with connections to all the unused I/O pins, GND and VCC. This makes it possible to construct a suitable interface and connect it to the test computer. VCC on the plug is delivered from a voltage regulator that is not used on the board for any other applications. The reason to use a dedicated voltage regulator is to ensure that an external device can't interfere with the computer's voltage supply. For test purposes we have made a small extension-board[21] with 8 LEDs and 4 buttons that fits this plug.

JTAG/ICE

As discussed in the "Component Description" section we use a JTAG interface to program the boot-flash and as a debug option. To ensure that we can use the board with a commercial JTAG/ICE[22] we will follow the standard interface description, which can be found on www.at91-forum.com[23]. To make sure that the JTAG interface will not affect the processor when it is unplugged all the input pins are pulled high. The JTAG connection design is basically the same as on ATMEL's evaluation board so that the two are compatible with the same interface.

Timing Analysis

The speed of a component will always be limited. The maximum speed of our processor is 40 MHz (at 3.0 V power supply), which means that every clock cycle must at least have a length of

$$\frac{1}{40 \cdot 10^6 \, s^{-1}} = 25ns \, .$$

[21] See Appendix B
[22] ICE In Circuit Emulation
[23] More precisely here: *http://www.at91-forum.com/viewfaq.php3*

The reason for this minimum time is that all transitions inside the chip have to be able to complete before the next clock cycle. In our computer, we run the processor at 12.288 MHz, which gives a cycle time of

$$\frac{1}{12.288 \cdot 10^6 s^{-1}} = 81.4 ns .$$

The same limitations exist in memory components. Time is needed in order to either read or write a value. Our RAM has a propagation delay of 70 ns for both reads and writes, which means that it takes 70 ns from the RAM receives a request until the task is completed. The flash is a bit slower and has a propagation delay of 120 ns.
Communication between processor and memory is generally done in the following way:

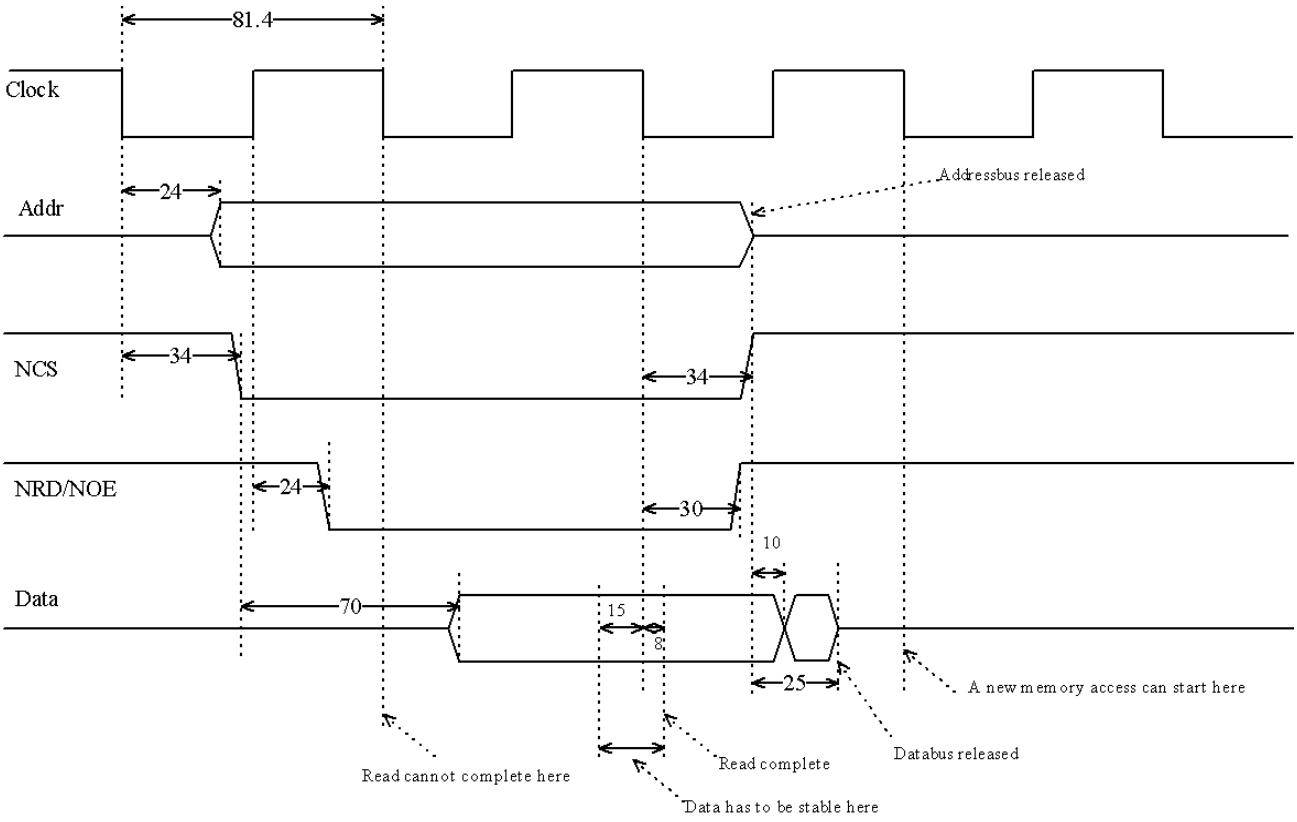
- The processor asserts these signals:
  - The chip select (CS / NCS) of the relevant memory component
  - The address of the data of interest
  - Output enable (OE / NOE) is deasserted in case of a read
  - Write enable (WE) is deasserted in case of a write
- When the memory component has CS, the address and either OE or WE, it performs the desired memory access.
- When the data has been either read or written, the databus is released along with the address bus and the other signals

In order to know when data is ready from a read operation or when data has been written in a write operation, the processor has to be set up to comply with the speed of the memory. This is done by inserting *wait states* in the communication mentioned above. Inserting a wait state means that the control signals are held for one extra clock cycle, so that the memory component has time to complete.
To determine if it is necessary to insert one or more wait states we have to investigate the timing in greater detail, which will follow.

The RAM

The diagram below shows the timing of the different signals in a read operation. In this diagram a single wait state has been inserted. If it had not been there, the processor would have tried to read data from the databus at the time marked with “Read cannot complete here”. This would have resulted in an unsuccessful read operation, as there are no valid data on the databus at that time.
But with the wait state inserted the signals are held one extra clock cycle, which allows the RAM to complete the access and place the correct data on the databus for the processor to read.
All timing data is stated in the electrical characteristics datasheet for the processor[24] and the datasheet for the RAM.[25]
The timing values for the processor depend on the speed and supply voltage of the processor. In the Atmel datasheet values are specified for the processor running at 40 MHz with 3.0 V, 33 MHz with 2.7 V and 16 MHz with 1.8 V. As 16 MHz is the speed closest to the speed we are running, we have chosen these values for our analysis.
All values in the diagram are in ns.

[24] http://www.atmel.com/atmel/acrobat/doc1393.pdf
[25] http://samsungelectronics.com/semiconductors/SRAM/Low_Power/Low_Power_&_Low_Voltage/4M_bit/K6T4016U3B/K6T4016V(U)3B.PDF



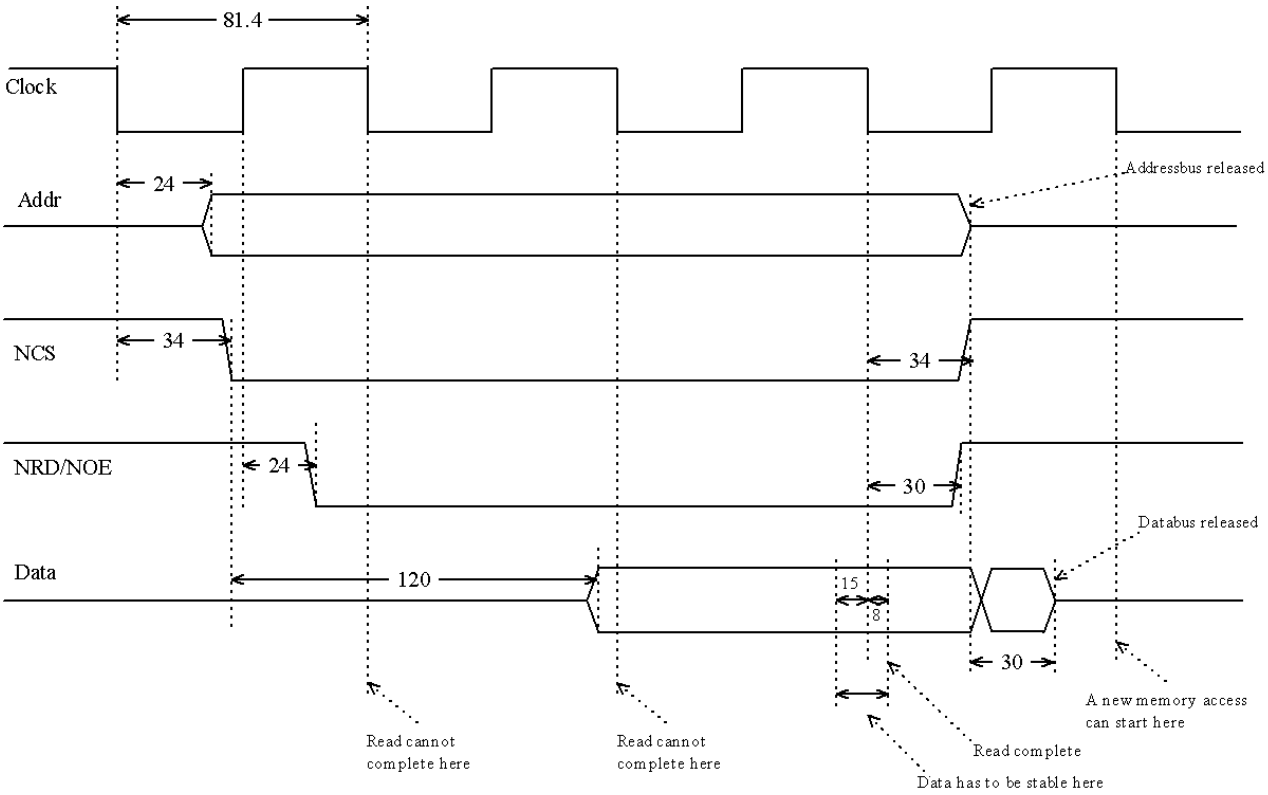
Timing of read access in RAM

In our first tests of the computer we inserted as many wait states as possible to make sure that the memory had enough time. Extra wait states can never cause errors in the communication, as it is the processor's release of chip select (NCS) and output enable (NOE) that triggers the RAM to release the databus. Until then RAM will hold the correct data on the databus.
Nevertheless, the insertion of extra wait states slows down the communication between the processor and the RAM.

The timing of writes to RAM is very similar to the one of reads. The only difference is that the processor deasserts write enable (NRW) instead of output enable (NOE). As mentioned above the propagation delay of the RAM in case of writes is 70 ns, just as it was with reads. This means that insertion of a single wait state also is sufficient in write operations.

The Flash

Determination of the required number of wait states in flash-memory accesses is done just like with RAM accesses. The timing of a read operation can be seen in the diagram below.



Timing of read access in flash

In the diagram two wait states have been inserted and, as data is ready at the time the processor reads from the databus, the timing demands are met.
As the diagram shows, it is very close that a single wait state would have been enough – data is ready at the second negative edge of the clock. But the processor demands a setup time of 15ns before the clock edge, where data also has to be stable, which it is not. So to be on the safe side two wait states are inserted.

PCB Layout

To ensure that the logical design of the computer is correct we have decided to make a large test board with several measuring points and test options.

Self-Production capabilities

This project is a student project with no commercial value and we have therefore a limited budget. Because of this we decided to make the first physical version of the test board ourselves. The process of making a PCB[26] goes like this:

1. Make a print of the design on a transparent paper
2. Illuminate a light sensitive PCB with the transparent paper in front
3. After the illumination the board is developed and the tracks appear
4. The board is then placed in an acid that removes the unwanted cobber

[26] PCB stands for “Printed Circuit Board”

5. Clean the PCB and put it in a pewter dissolution
6. Drill the holes

Since our production method is a little primitive compared to the professional manufactures, there are some limitations to the design:

- Only two layers are available
- The vias[27] must be larger than 50mil[28]
- Vias must be assembled manually
- Vias cannot be placed under SMD components
- Tracks must be at least 8mil

Though there are disadvantages there are also some advantages. Since it is a test board it is good to have as many test points as possible to help with the error detection, and all the vias serve as test points. It is also an advantage that there only are two layers when an error is found in the design and needs to be corrected. This will not always be possible if the error is located in a mid layer.

Using Protel 98se and Protel 99se

We have used Protel 98se and Protel 99se to make the PCB design. During this process we have found that both programs are quite unstable but it seems that the '99 edition is a little better at some points.
One of the biggest advantages in '99 is that it is possible to define different width constraints for groundplanes (polygonplane) and the rest of the board. This is useful, because otherwise the space between the groundplane and the tracks will be as narrow as the distance between the legs on the most compact component, witch will increase error possibilities significantly.
Another great improvement in 99' is that it is possible to set a preferred and a minimum track width for the autorouter. This assures that the router uses a wider track wherever possible.
The last improvement we like to mention is that the net name is labelled on the tracks which makes orientation easier when a part of the board is viewed close up.
We will recommend the 99' edition for similar projects in the future.

Board Size

Since the computer is supposed to fit into a pico-satellite, the board proportion has to follow some guidelines. The group handling the mechanical design of the satellite sets these guidelines. However the mechanical design is not complete yet so the final guidelines are still not available. We do know that the size of the computer board is expected to be about 60mm $\times$ 60mm. But since we need to build a test board first we have decided to build it the size that makes the routing easiest, which is 164mm $\times$ 154mm.

Component Footprint Choice

In the design we have two kinds of components: The components, which only have a test purpose, and the components that also will be needed for the flight-board. Since it is not necessary to make the testboard very compact, we have decided that the components used only for test purposes does not have to be SMD mounted components. This reduces the error possibilities significantly. The

[27] A via is a connection from one layer to another
[28] 1mil = 0.025mm

components that will be used for the flight-board, is however mostly SMD components. This is because they are smaller and they do not take up space on both sides of the board.

Track

The minimum size for the track width is dictated by the component with the smallest footprint. The pins on the processor are only 8.6mil, and therefore 8.6mil is the smallest track width that is used. For the power supply we have decided to use 25mil to ensure that the different components gets a steady supply. For all other wires we have used 13mil width. This corresponds with the size of the pins on the RAM.

Via

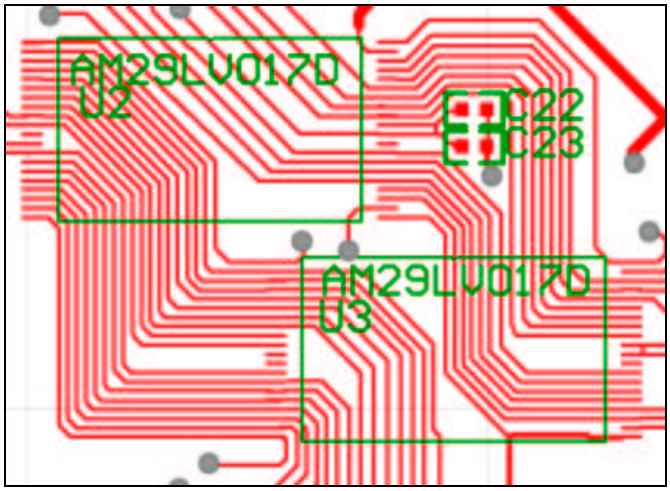
We have to manually assemble the vias, which is done in the following way:

1. Drill a hole in the wanted via pad
2. Insert a small piece of wire in the hole
3. Solder the wire on both sides of the PCB

With this procedure one can see why the via-pad must have a minimum size of 50mil in diameter. The large size of the vias makes the routing much harder and forces a larger distance between the components because it takes 5 times as much space to make a connection to the other side of the board. Fortunately, the vias will be much smaller on the professionally made PCB for the flight-board. During the error detection removing a via has been an easy way of isolating short circuits on the board.

Component Placement

In this design the databus and addressbus constitute a large part of the connections. To prevent that too many tracks has to cross each other the components are placed so that it is possible to connect the two busses without any lines crossing. This is illustrated on the figure below where the two flash-components, which share all lines except the chip-select signal.



Footprint and connections between the two flash-components.

Routing

When we started with this design we encountered some problems with the cauterization of the PCB board. If the tracks were too close the acid couldn't get trough. Because of this we decided to make as large distances as possible between the tracks. Since Protel's auto router had problems with our design and found it impossible to route, we had to route the whole board by hand. When a board is routed manually, it makes it easier to error detect because the logic in the paths of the tracks is more obvious and there are fewer connections sent on long devious tours.

The Software

Background

The primary goal in this project has been to develop a working computer for the satellite. Even though we have been very careful in the design process it is most unlikely that the hardware (the PCB) does not contain errors, and even then the physical manufacturing may lead to errors, which we need to know about. This includes bad solderings, short-circuits, and tracks that are broken as a consequence of the way we make the PCB.
In order to verify our hardware design we therefore need some software, which can help us in the debugging and test of the hardware.

The software this section covers is only written to be used during the design and verification of the hardware. The software that will run on the computer, when it operates in space, is designed and written by other groups of the satellite project. As mentioned in the beginning, groups covering bootstrap, protocols, and overall software architecture exist. The last group has proposed that the computer will run an embedded operating system called eCos[29]. eCos is an open source operating system and thereby it is not only free of charge, the source code is also available.
Even though the following software is written primarily for tests, parts of it may be used in the flight software, as eCos does not contain low level drivers for all the parts the computer is build of. These drivers can be based on the following, as this is very low level.

We have written 4 different small programs:

- *blink* - The purpose of this program is to verify that we are able to compile and link a program that can be downloaded and run on the computer. It is very small and can be run from the internal memory of the processor, so that the external bus interface is not needed to be working probably - It can however also be run from the external RAM, in order to verify that this is functional.
- *flash* - This program is used to program the onboard flash. As the flash is soldered to the board, it has to be programmed using the processor. Parts of this program may be used on the computer, when it operates in space.
- *Boot* - This is a simple program written in assembler. All it does is a remap of the memory, and then it calls another program.
- *usart* - This program sets up the processor, so that serial communication can be performed.

All of the programs with exception of the boot-program are written in the C language. The reason for choosing C is, that it is quite easy to write programs, which interact with hardware, because it is possible to work with addresses. Another reason is that we have access to at compiler, which can generate machine code for the ARM core. The compiler we use is the “gnu”-compiler[30]. This compiler has the advantages that it is free and it is known to produce quite well optimised code.

[29] http://sources.redhat.com/ecos/
[30] http://www.gnu.org/

Compilation and Linking of Programs

Because the compiler has to generate programs that will run on our embedded system, it must be set up in a special way. This section provides a small description of what is needed to make it generate binary files, which can be downloaded and executed on the computer. We have included this section because some of the things needed are hardware specific.

We have set op the compiler (actually compiled it) to generate code for the arm-elf-target. This means that the compiler will generate 32-bit machine code. We do not use the thumb-mode[31] of the processor.

Compiling C programs is straightforward if you are familiar with the C-compiler gcc. The only thing needed is to issue the following command:

```
arm-elf-gcc –c file.c
```

A lot of options can be given to the compiler, but it will be out of the scope of this rapport to list them here. The important thing is that the command will generate a binary file named file.o. The reason for mentioning the compiler command at all is, that the code generated (.o) will not run directly on the processor. The problem is, that the .o-file doesn't contain any information about where in memory the program must be run.

In order to provide this information we must use the linker tool. This tool assembles different .o-files and makes them run in a specific place in memory. A "linker script" controls the tool. The one we use is listed in appendix E. The script makes the program run from address 0x02000000. This is where we have mapped the external RAM. This value can of course be changed to something else. With this linker script the linking can be done using the following command:

```
arm-elf-ld -T ldscript -o out.elf file.o
```

The output of this command is a file named out.elf. This file can be downloaded and run on the target.

There is one thing about the above linker script that the user needs to know. As it is very simple, it does not contain all the symbols needed to generate a working program. This means that the user will have to insert the following function in the C-source, otherwise the linker will exit with an error:

```
void __gccmain() {
}
```

When a program written in C is compiled, every call to a function is done by pushing the arguments of the function to the stack before branching to the address of the function. Therefore it is needed to set up the stack-pointer of the processor before the main-routine of the C-program is executed. We do this with a program written in assembler. The program is shown in appendix F. The program sets up the pointer to a place in memory after the program code. The linker script controls the exact place. The assembler code is assembled with the command:

[31] See section on CPU in "Component description"

```
arm-elf-as -c -o crt0.o crt0.S
```

To sum up, the following commands are needed to generate a program, which can be run on our computer (note that the crt0.o is added to the linker command):

```
arm-elf-as -c -o crt0.o crt0.S
arm-elf-gcc –c file.c
arm-elf-ld -T ldscript -o out.elf crt0.o file.o
```

Both the linker script and the program setting up the stack was included with the binary gcc-package available at *www.ocdemon.com*. We have only made small changes in these sources to affect the size and placement of our RAM.

After this short introduction to the compiler tools we are ready to look at the 4 different test-programs.

The blink Program

As mentioned earlier the purpose of the blink program is to test that the processor is working properly. In appendix G we have included the source code for the program. The program sets up the processor, so that the lower 8 I/O-pins of the processor (P0-P7) are outputs. This is done using the following two lines:

```
PIO_PER = 0xff;
PIO_OER = 0xff;
```

The `PIO_PER` register controls whether the individual pins are connected to an internal device or to the PIO-bus. The `PIO_OER` controls whether a pin is an input or an output. Therefore, by writing the value `0xFF` to the registers, the 8 lower I/O-pins get connected to the PIO and are set as outputs.
The registers are defined as follows:

```
#define PIO_PER  *(volatile unsigned int*)(0xFFFF0000)
```

What is important in this line is the `volatile`-keyword. This makes sure, that the compiler *does* make machine code that executes this line. If it was omitted, the compiler may chose not to generate the required code, if it finds, that the line does not do anything. The `0xFFFF0000` is the address of the PIO_PER register.

By setting the individual bits in `PIO_SODR` or `PIO_CODR`-registers the corresponding output can be turned on or off. The fact that two different registers are used to set and reset a bit, is actually a neat feature of the processor. Most other processors only have one register for this. If you want to set a single bit in a register without changing the others in such a processor, three commands are needed (read, or, write). On the Atmel processor only one command is needed to do the same, which not only makes the code smaller but also run faster.

The remaining of the blink program sets and resets different outputs making the LEDs connected to the corresponding pins of the processor flash. This produces a nice light-show on our extension-board.

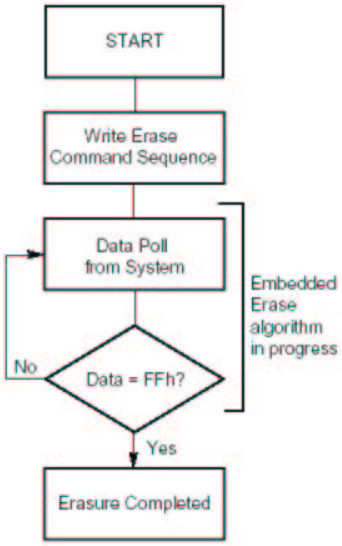
As this is all the program does, it is very small. This make is perfect as the first tests-program, as it can be run from internal RAM, external RAM and flash.

The flash Program

As mentioned earlier we use the JTAG feature of the processor to download code into the RAM of the processor. Even though the flash is mapped to the external bus interface in the same way as the RAM, the JTAG is not able to write to the flash directly. This is because the flash requires a special algorithm for programming. Therefore, we have developed a program that can be downloaded to the RAM where it is executed. This program takes care of programming the flash. This seems to be the way commercially available JTAG flash-programmers works. An example of such a program is the "Flash programmer" available at *www.macraigor.com* for the price of £500.

In order to have the data, which is to be programmed to the flash, included in our program, we use at small program called gen_progfile available at *http://www.ahare.btinternet.co.uk/*. This converts the binary data of a file into a c-array and writes the result in prog_array.c. This file is then included in our flash-program, and thereby we have access to the data.

The flash has an embedded programming algorithm, but because of the way the flash is made, it is only able to change one's into zeros. Therefore the address being programmed must be erased (i.e. all bit set to one's) before the wanted value is written into the address. To do this the flash features an embedded erase algorithm. This algorithm erases an entire sector, i.e. 64KB of data are erased. In order to make use of the embedded algorithm the processor must do, what is depicted in the figure to the right[32]. In appendix H the source code for our implementation can be seen.



The first step is to issue the "Sector Erase Command Sequence" This command sequence consists of 6 values that must be written to the sector. This is what happens in the following lines:

```
*flash = FLASH_Unlock_first;
*flash = FLASH_Unlock_second;
*flash = FLASH_Unlock_sector_erase;
*flash = FLASH_Unlock_first;
*flash = FLASH_Unlock_second;
*sect_adr = FLASH_Sector_erase;
```

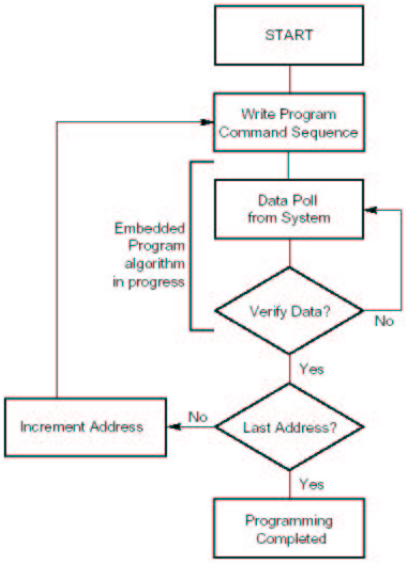
Next the processor must read from the sector to determine if the data has changed to the value `0xFF`. If this is the case, then the erasing has completed – otherwise the embedded algorithm is still in progress.
As the algorithm may never end, if the value read does not change to `0xFF`, we have included a timeout; just to make sure the loop stops. This makes the algorithm look like:

```
while (--timeout > 0) {
   if (*sect_adr == 0xFF ) break;
```

[32] AM29LV017D datasheet page 19

```
}
```

To make sure the entire sector has been erased, we test if all values in the sector have been set to 0xFF. This should always be the case when the embedded algorithm is finished, but the radiation from space may destroy some memory cells making it impossible to erase them. Therefore this check has been added.



The writing of a value to an address in the flash can now be done. The way is quite similar to the way the erasing was done. A command sequence must be issued, the processor must wait until the writing has finished, and the data must be verified. The algorithm the processor must follow is depicted to the right[33]. Our implementation of this algorithm can be seen in the appendix.

In order to see what happens when our flash-programmer runs on the board, we use the LEDs on the extension board. The way they are used is quite similar to the way they were used in the blink-program.

The boot Program

The boot program is shown in Appendix I. We have not written this program ourselves, but only made some small changes to some code included with the gen_progfile mentioned above. The original code does a remap of the external bus interface and then goes into an infinite loop. We changed the remap values to the ones used by our RAM and flash, and instead of just making a infinite loop, we jump to an address in the flash (the value of `PtProg, 0x01010000`), so that the program in this address is run after the remap.

The USART Program

The last program is the USART program. It was written to have a way of testing if the serial communication is working correctly. The serial port 0 on the board must be connected to a desktop computer. The program sets up the USART of the processor, and wait until it receives something (a character from the computer). Then it sends back the received value+1, and the received value+2. Typing A in a serial terminal on the desktop computer returns BC, typing 5 returns 67 etc.

The USART is set up with:

```
/* Initialize the channel */
PIO_PDR = ( PIOTXD0 | PIORXD0 ) ;
US0_MR = ( US_ASYNC_MODE | US_CHMODE_NORMAL ) ;
US0_BRGR = 80; /* baud rate */

/* Start channel */
US0_CR = ( US_RXEN | US_TXEN );
```

[33] AM29LV017D datasheet page 17

The first line makes sure the `TX0` and `RX0` of pins are connected to the internal USART. Then the mode of the USART0 is set to asynchronous, and `US_CHMODE_NORMAL` makes the USART operate without any local/remote loopback.

Writing 80 to `US0_BRGR` (the baud rate generator) sets the baud rate to 9600 bps. Since the USART is to operate in asynchronous mode, the baud rate is calculated as[34]:

$$\text{Baud rate} = \frac{\text{Selected Clock}}{16 \times \text{CD}} = \frac{12.288\,\text{Mhz}}{16 \times \text{CD}} = 9600 \Rightarrow \text{CD} = 80$$

Finally, we start the channel by enabling the RX and TX signals.

Then we start waiting to receive something. This is done by:

```
value = US0_RHR;
```

The program 'hangs' at this line until something is received. The USART could be set to generate an interrupt when data is received, but in this simple program we do not make use of this feature.

Finally we want to show how data is sent:

```
/* Send byte */
US0_THR = value + 2;

/* Wait Tx ready */
for (; (US0_CSR & 2) == 0;);
```

When something is written to the `US0_THR` register it is instantly sent out. In order to know when all data is sent, we look at bit 1 of the `US0_CSR`-register, which is 1, when all data is sent.
The complete source code can be seen in appendix J.

[34] AT91 ARM Thumb microcontrollers manual page 98.

Test and Verification

Even though much care was taken in the process of making the schematic for the system, producing the PCB, and soldering components, the board did contain some small errors. Using the programs just described we were able to find and correct these errors at our board. In this process the JTAG interface was a great help, as we could write and read values to the RAM and use the information gained to find the soldering errors we had in our databus.
It was nice that the evaluationboard supplied by Atmel gave us the possibility to test the software before we downloaded it to our own board. This way it was possible to exclude software errors.

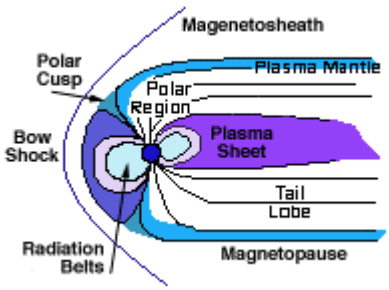
After finding and correcting these errors we were able to make some measurements concerning how much power the board requires. As the design of the OBC has yet not been finished (e.g. A/D and D/A converters are missing) these values are not the final ones, but as the most power consuming components are included on the current board, the values make it possible to verify if the OBC can be made using an acceptable amount of power.
The measurements were made without the JTAG cable and the extension board attached to the test board. This way only the current drawn by the processor, flash, RAM, oscillator, and voltage supervisor are included.
We have made two measurements: One with a program running in RAM, and one with a program running in FLASH. The program running was the blink program described above.

	Program running in RAM	Program running in flash
Current	47 mA	39 mA
Power (at 3.3 V)	155.1 mW	128.7 mW

As can be seen the values are far below the required value (1W). We therefore believe that even though more components are added, the power consumption of the OBC will still not go beyond a critical value, but it will most likely be close when all subsystems are connected. One problem exists with the measured values: They may (and most likely will) change when the components are irradiated. Unfortunately, information on component degradation is not something that can be found in a datasheet, and therefore we have to do some tests concerning radiation ourselves. The next section (”Radiation in space”) will cover this topic.
The power consumption of the computer running programs in RAM will be lowered, when we replace the RAM with the newer version.

Radiation in Space

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. Some places our satellite might cross a so-called "solar wind", which is a burst of particles (mostly protons and electrons) emitted from the sun, and here radiation will be substantial. At 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 our circuitry than low-energy particles (often referred to as back-ground radiation), as they penetrate deeper into the components and cause greater damage in case of collision. Much research is carried out in making maps of intensities and different kinds of radiation in space. Such a map is shown in the figure. The figure shows that the radiation is not only depending on the sun but also very much on the magnetic field of earth (magnetosphere). [35] As further details of this area are quite complex and out of the scope of this project, we will not discuss it further.



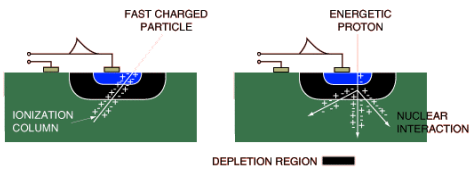
Radiation and magnetosphere near earth

Radiation effects on a satellite like ours can be divided into three categories:

1. *Single event effects*
   A single-event effect results from, as the term suggests, a single, energetic particle. If such a particle hits one of our components, we might encounter three different errors:

   - Bitflips (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 RAM/flash (as mentioned in "System overview"). 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 bitflips.

   - Latch-up (hard error)
     We mentioned this kind of error in "Design description" and that permanent damage can be avoided by turning off the power.
     The picture shows part of a CMOS component being hit by a high-energy particle. This particle can either short-circuit the source or drain to ground, which is located in the bottom of the picture. The short circuit arises because the particle creates a conductive "tunnel". A short-circuit will draw



Particle colliding with a CMOS component

[35] http://www.eas.asu.edu/~holbert/eee460/spacerad.html

a lot of current and the component is very likely to take permanent damage if the power is not turned off immediately.

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

2. *Spacecraft charging*
   Spacecraft charging is a phenomenon caused by charged particles. For example: In a given area, there might be a surplus of free electrons, which gradually will charge the entire satellite to a negative voltage. This voltage can be very high – up until tens of kilovolts. However, if a common ground is used in the entire satellite, this is no problem. On the other hand, it will be very dangerous if voltages like these are suddenly discharged to an isolated component of the satellite. [36]
   Actually, the payload group designing the tether has been thinking about trying to use these high voltages, as the tether requires a voltage of at least 100 V in operation. This way we could save both room and weight, because a transformer would not be needed. But for now it is just an idea.

3. *Total ionizing doze*
   The total ionizing dose (TID), 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, secondary Bremsstrahlung photons, and solar flare protons.[37]
   As TID increases, material degradation increases. 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 our components will react to these long-term effects, all groups in the satellite project have sent samples of their components to Risø for tests. At Risø the components will be exposed to different amounts of radiation: 0.9, 1.8, 4.5 and 9.0 kRad at 25.05 Rad/s. 2 kRad is the average amount of radiation we can expect in one year.
   When the components come back from Risø, we can start to replace the existing components on the board with the ones exposed to radiation. By doing this we will see whether or not they still function and see the changes in their characteristics.

   Hopefully, we will see that at least the ones exposed to only a little radiation still function without characteristics like power consumption having changed too much.

   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. During the spring, we will decide whether or not to implement shielding, depending on the results of the Risø-tests.

[36] http://www.hq.nasa.gov/office/codeq/relpract/1258jsc.pdf
[37] http://www.eas.asu.edu/~holbert/eee460/spacerad.html

Future Development

A number of things on our current board will have to change, before it can be sent into space. The most obvious thing is the size. As mentioned earlier, the size of the current test-board is 164mm×154mm. The mechanics group has proposed a maximum size of 60mm×60mm excluding space for connections to other subsystems. This means that our board must be reduced in size. In the section about the PCB-layout we saw, that the current large size was due to at number of things: Via-size, space between tracks, minimum track-size, and problems with the routing. Since the final board is to be produced by professionals, these things change. The most important change is that we are able to use a 4-layer board. Our plan is to use the two mid-layers for VCC and GND. This means that these two nets are removed from the top and bottom layers. Because the two supply nets are connected to all components, routing will become much simpler, as they do not cross the busses anymore. Furthermore, the fact that the vias and tracks can be narrowed in makes it simpler to route the remaining signals on the top and bottom layer. We therefore believe that the board can be made much smaller, so that it indeed meets the requirements from the mechanics group. Jimmy Malmkvist from PowerCAD also stated, that he did not se any problems reducing the size at our meeting with him[38].

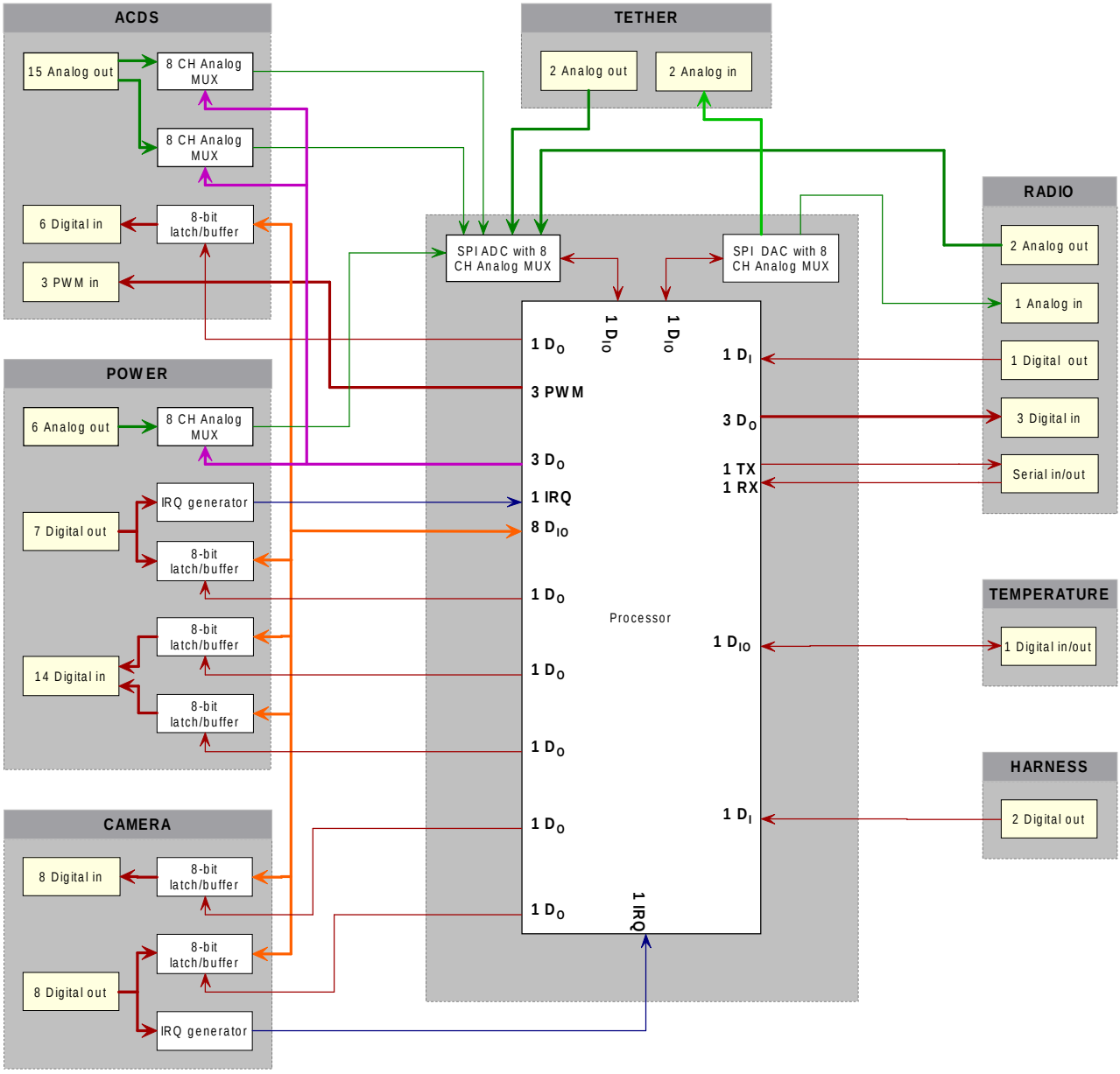
As it can be seen by the previous discussion the current test-board only contains the necessary components for a computer, which is operational. In the "System Overview"-section we saw that in order to have the computer do something meaningful, it must be connected to other subsystems of the satellite. One of the things that has been difficult in this project was (and still is) to find out exactly which connections are needed by these subsystems. At present time these specifications are still not completed, but at the last system-engineering meeting, the following requirements were set. The table shows how many outputs each system has, and how many inputs it requires.

	Analogue		Digital					
	Output	*Input*	*Input*	*Output*	*In/output*	*PWM*	*Serial*	*IRQ*
ACDS[39]	15			6		3		
Radio	2	1	1	3			2	
Power	6		7	14				1
Camera[40]			5	5				1
Tether	2	2						
Harness			2					
Temperature					1			
Sum	25	3	15	28	1	3	2	2

Summing up there are 28 analogue signals and 51 digital signals to the OBC board. The processor only provides 32 digital signals, which even if none of them were used to control the ADCs and DACs, is far too few. As the processor was chosen before we knew these numbers, we obviously have to do something to reduce the signal count requirement. The good thing about limiting the signal-count is that fewer connections in-between subsystems will be necessary, which reduces the total space and weight used by interconnections.

[38] See the External Contacts section.
[39] As the attitude group were not present at the meeting, this may contain errors
[40] These values are guesses, as the Camera group has not yet started their work

At the system-engineering meeting we discussed how we could best do this limiting without interfering with work already done by the groups. It was also considered if the changes would make the satellite more intolerant to failures and if the changes would require more power etc.
By the following figure the result of this work is shown:



When looking at the requirement table it can be seen that some systems (the ACDS and the power) have a lot of analogue outputs. In order to limit this number of outputs, analogue multiplexers have been added to each of these boards. If an 8 channel mux is used (as shown on the drawing) the total connections to the computer can be as few as 6. 3 are used to connect each mux to an A/D-converter on the OBC board (thin green lines). The muxes share the 3 other connections in parallel and use this signal to control which input channel is selected (fat purple line). The A/D-converter on our board needs to have a mux inserted, so that only one of the analogue signals from the muxes is measured. On the drawing this is shown as a single block entitled "SPI ADC with 8 CH MUX". This is because several single chip solutions exist that both have mux and converter. Some of these

are controlled via SPI (a single wire bus), and therefore only one connection to the processor is needed. Since only 3 of the analogue inputs of this ADC are used, 5 are left. This is enough to connect the analogue signals from the remaining subsystems. Summing up 3 outputs and 1 in-/output is needed to control the measuring of analogue signals in the satellite.

In order to limit the number of required digital lines from the processor, an 8-bit wide bus has been introduced in the drawing (the fat orange line). It connects the subsystems requiring a lot of digital signals to the computer. On these systems an 8-bit latch/buffer is added. The purpose of those is to make sure only one system uses the bus at a time. This buffer requires an enable signal, when it is to control the bus. Therefore 6 dedicated digital outputs from the processor are connected to each of the buffers. When something is to be read from or written to a specific subsystem, the enable line is first driven low by the processor (assuming active low enable signal) and then the required value is written to/read from the bus. When the enable signal goes high again the value on the bus is latched in the subsystem, and the bus can be used for communication with other subsystems without interfering with the current subsystem.
The introduction of a bus causes another problem. In some subsystems it is urgent to tell the processor if anything changes on its output line. The power-board has such a demand. If for example the power manager detects a latch-up in a subsystem, it is vital to tell the processor that the manager has cut off the power to this subsystem. Therefore an "IRQ generator" has been added to the power-board. The purpose of it is to generate a signal that can be used as an interrupt for the processor. This interrupt must be generated, if any of the lines change state and therefore the IRQ generator is probably something simple as an 8-input nor-gate
It can be seen from the requirement table, that only 3 DAC-lines are required. Instead of placing 3 DACs on our board, the same solution as described under the ADC are used. A "SPI DAC with 8 CH MUX" can be bought in a single chip. This means that the generation of the 3 analogue voltages can be done with only 1 in-/output line from the processor.

Counting the required number of digital in- and outputs after these changes yield, that 32 lines are required. This is exactly what is available on the processor. With the currently available requirements for the different subsystems, it is therefore possible to connect all parts of the satellite to the OBC. This implies that the chosen processor (and the computer in general) can fulfil the requirements of the satellite.
Since the specification of the requirements for the subsystems is not yet final, changes in the proposed design of the satellite interconnections may occur. This is especially true for the camera, as the number of signals needed to control this is yet not known. The consequence of this is that more I/O-lines may be needed from the processor, but at present time no more are available. It might be necessary to make further optimisations in the way we connect the subsystems. One of these could be to insert a latch controlling the 6 lines required to control the bus-buffers. This optimisation would make 5 I/O-lines available on the processor.

We will conclude this section by a short look at the physical interface with the other groups. As mentioned 32 lines from the processor to other subsystems are needed. Also a number of wires from the power-board to the other boards are needed. In addition some of the subsystems may have connections in-between.
In the beginning of this section we stated, that the size of the PCB must be 60×60 mm excluding space for connections to other subsystems. Therefore there will be added some space for these connections. As the number of connections is quite high, different kinds of connectors are considered: A slot like the PCI-slot known from the motherboard of a desktop computer (would

require a connection board in the satellite), a flat cable with connectors (like the one connecting a harddrive of a desktop computer to the motherboard), or maybe a socket that would make it possible to stack the different PCBs on top of each other. The choice has not yet been made, as the total number of connections will influence on which solution is best suited for our application. Also things like robustness concerning vibrations must be considered.

Conclusion

We have designed and produced a functioning computer for our satellite. So far, the computer complies with the requirement specification:

- It has sufficient memory resources
- The architecture of our design satisfy the needs in the satellite (we have a sufficient number of I/O pins, a watch-dog, possibility of serial communication and so on)
- We have a power consumption of about 150 mW, which will become lower when we implement the newer, low power RAM
- A flightboard with a dimension of 60mm×60mm will be possible because of our opportunity to have the PCB professionally made free of charge
- We have C-compilers available for the processor free of charge

Furthermore, we have avoided the use of BGA components, which we believe has saved us a lot of debugging time. When it comes to the wishes of the software groups, we have been able to fulfil the request for a 32-bit processor but not the request for a MMU (memory managing unit).

There is one big question left from the requirement specification: Can our components cope with the effects of the radiation in space? This will be answered during the spring, when our components come back from Risø and we have the time to test them.
These tests are not all that has to be done in the spring:

- The RAM has to be exchanged
- The ROM must be found
- A/D converters must be added
- Shielding may have to be implemented
- Error detection and correction circuitry (EDAC) will be implemented
- Production of the PCB
- The interface between the subsystems has to be made

So there is still a lot of work to be done, but for now we feel very content with what we have achieved and learned. Our choices of components and overall design seem to be reasonable.
With our JTAG and our flash driver we found a way to program the flash. The use of the JTAG interface of the processor has generally been very positive. It is a very reliable interface and as JTAG is a widely used standard, we will be able to use our knowledge and our homemade “Wiggler” in other projects.

One of the greatest challenges has been to work with a big project (our computer) that is only part of an even bigger project (the satellite). This has taught us the importance of communication between the different groups in a big project, which will most certainly be useful when we get to work as engineers in the future.
We believe that we have explored many different areas of the engineering world in this project; we have had contact with many different companies, designed a system conceptually, found the optimal parts for this system, build the system ourselves and tested the system via our own software.

This process is typical for development of any kind of device and with this project, we feel that we have been through all the steps.

Jonas Sølvhøj, c973442

_____

Malte Breiting, c973568

_____

Morten Briand Thomsen, c973709

_____

List of References

Am29LV Flash Memory Family
http://www.amd.com/us-en/FlashMemory/ProductInformation/0,,37_1447_1623_1468,00.html

AT91EB40 Evaluation Board User Guide
http://www.atmel.com/atmel/acrobat/doc1706.pdf

AT91M40800 Datasheet
http://www.atmel.com/atmel/acrobat/doc1354.pdf

AT91M40800 Electrical Characteristics
http://www.atmel.com/atmel/acrobat/doc1393.pdf

Cubesat Developer Specification
http://ssdl.stanford.edu/cubesat/specs-1_files/CubeSat Developer Specifications.pdf

JTAG Interface for ARM9 Processors
http://sourceforge.net/projects/jtag-arm9

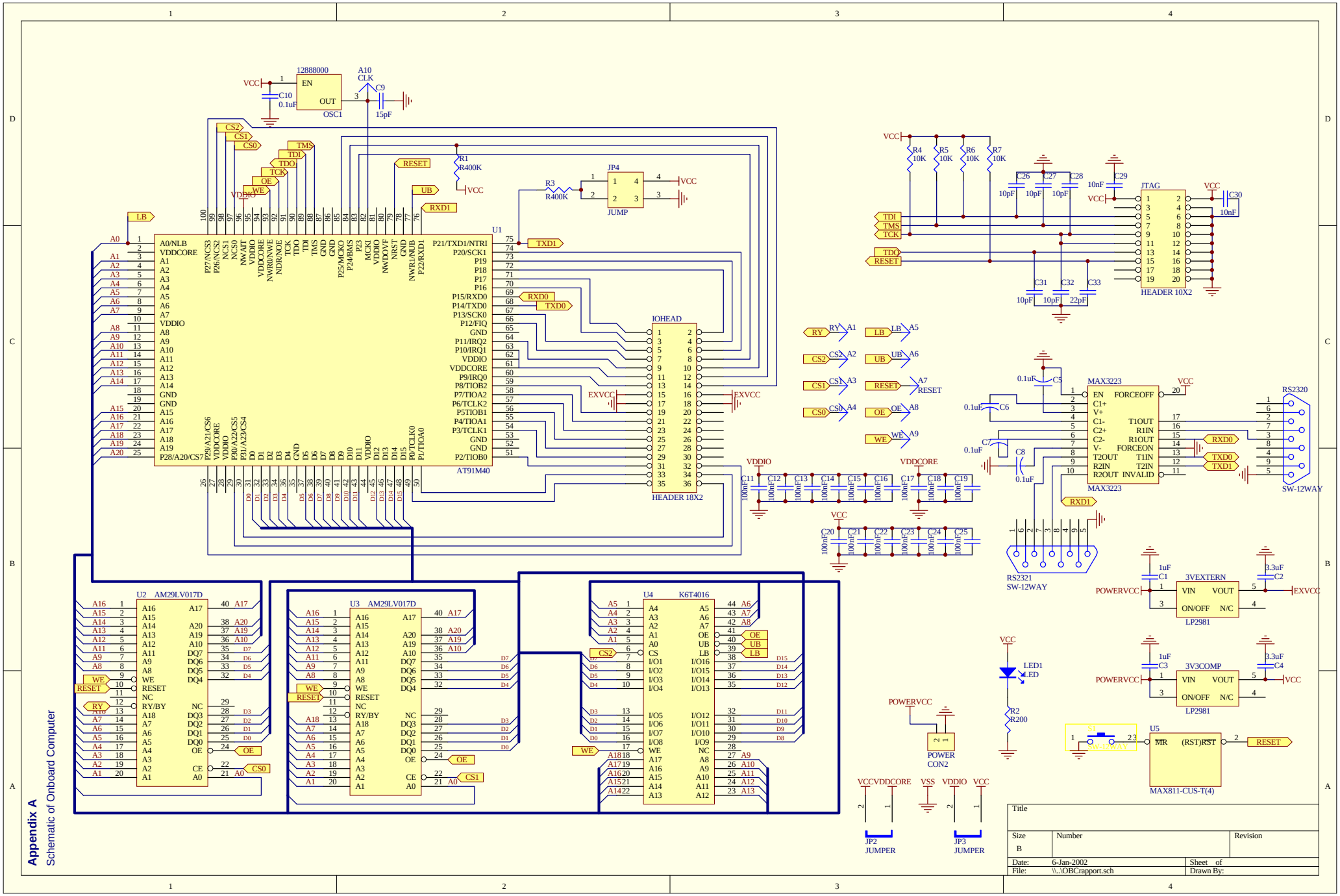
Samsung Semiconductor SRAM Family
http://www.samsungelectronics.com/semiconductors/SRAM/SRAM.htm

Space Radiation
http://www.eas.asu.edu/~holbert/eee460/spacerad.html

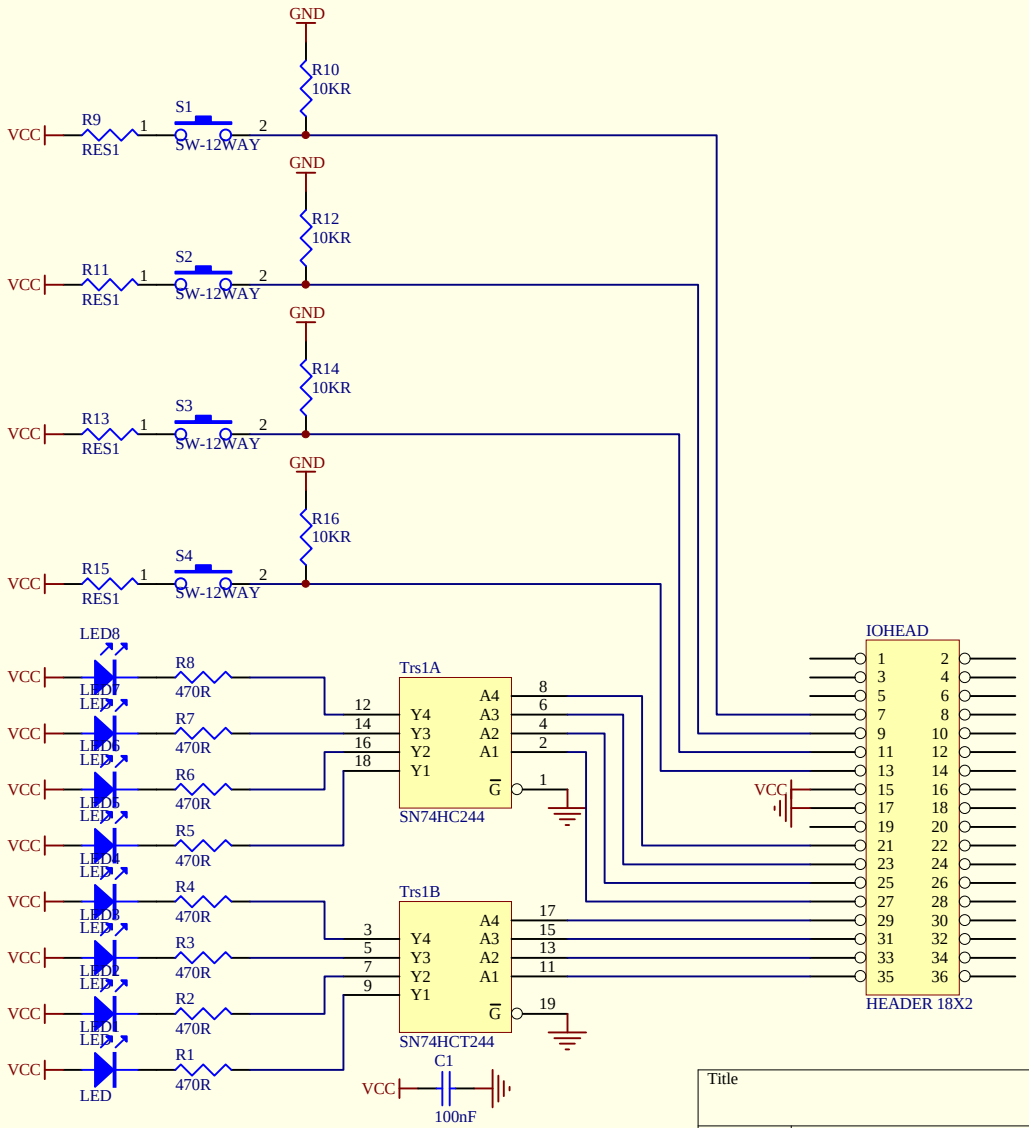
Space Radiation effects on Electronic Components in low-earth orbit
http://www.hq.nasa.gov/office/codeq/relpract/1258jsc.pdf

The AT91 Forum
http://www.at91-forum.com/

The Thumb Architecture Extension
http://www.arm.com/armtech/Thumb

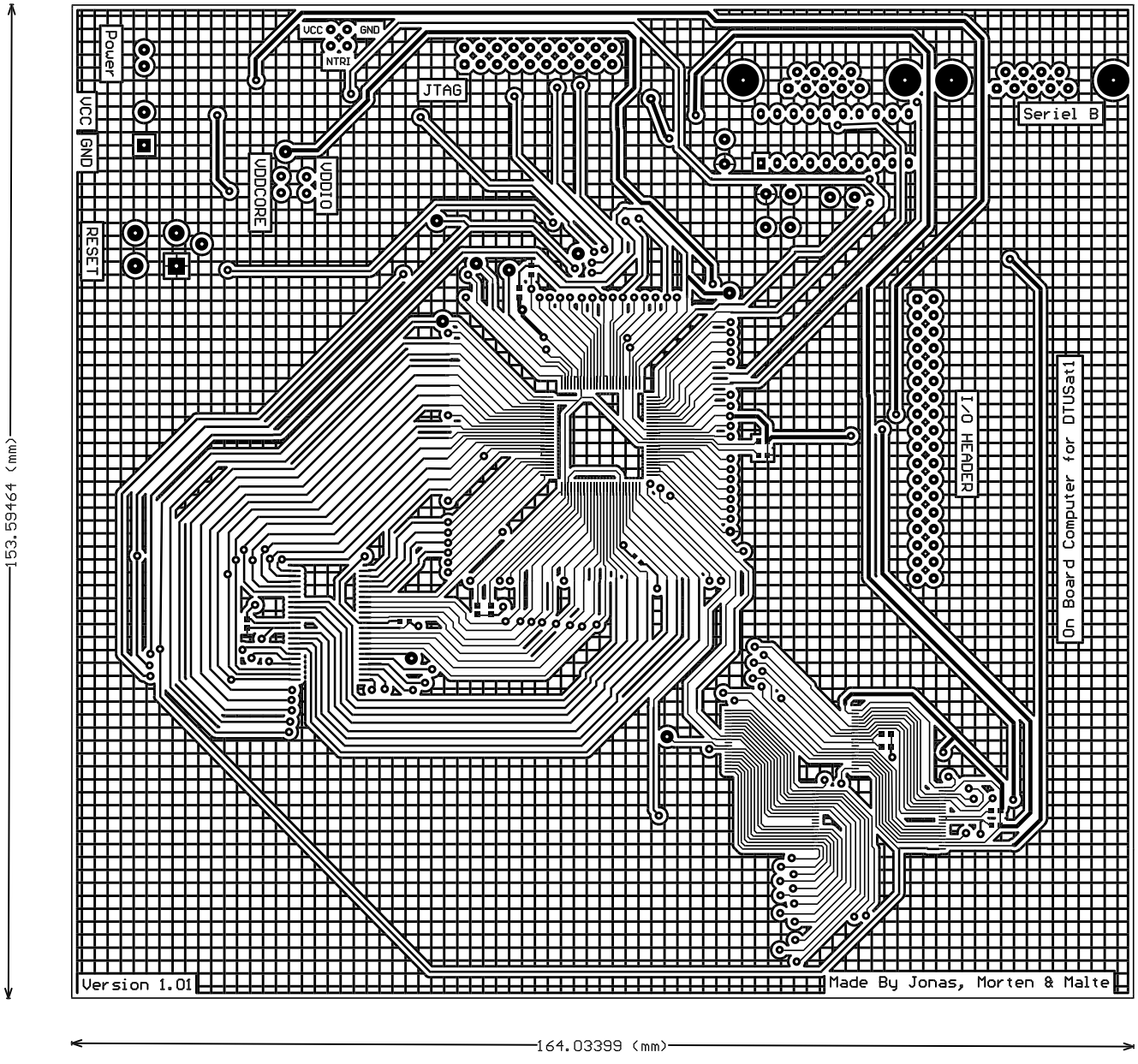

Appendix A
Schematic of Onboard Computer
U1
AT91M40
IOHEAD
HEADER 18X2
JTAG
HEADER 10X2
JUMP
JP4
OSC1
12888000
U2 AM29LV017D
U3 AM29LV017D
U4 K6T4016
MAX3223
RS2320
SW-12WAY
RS2321
SW-12WAY
3VEXTERN
3V3COMP
LP2981
MAX811-CUS-T(4)
POWER
CON2
JP2
JUMPER
JP3
JUMPER
LED1
LED
Title
Size
B
Number
Revision
Date: 6-Jan-2002
File: \\..\OBCrapport.sch
Sheet of
Drawn By:

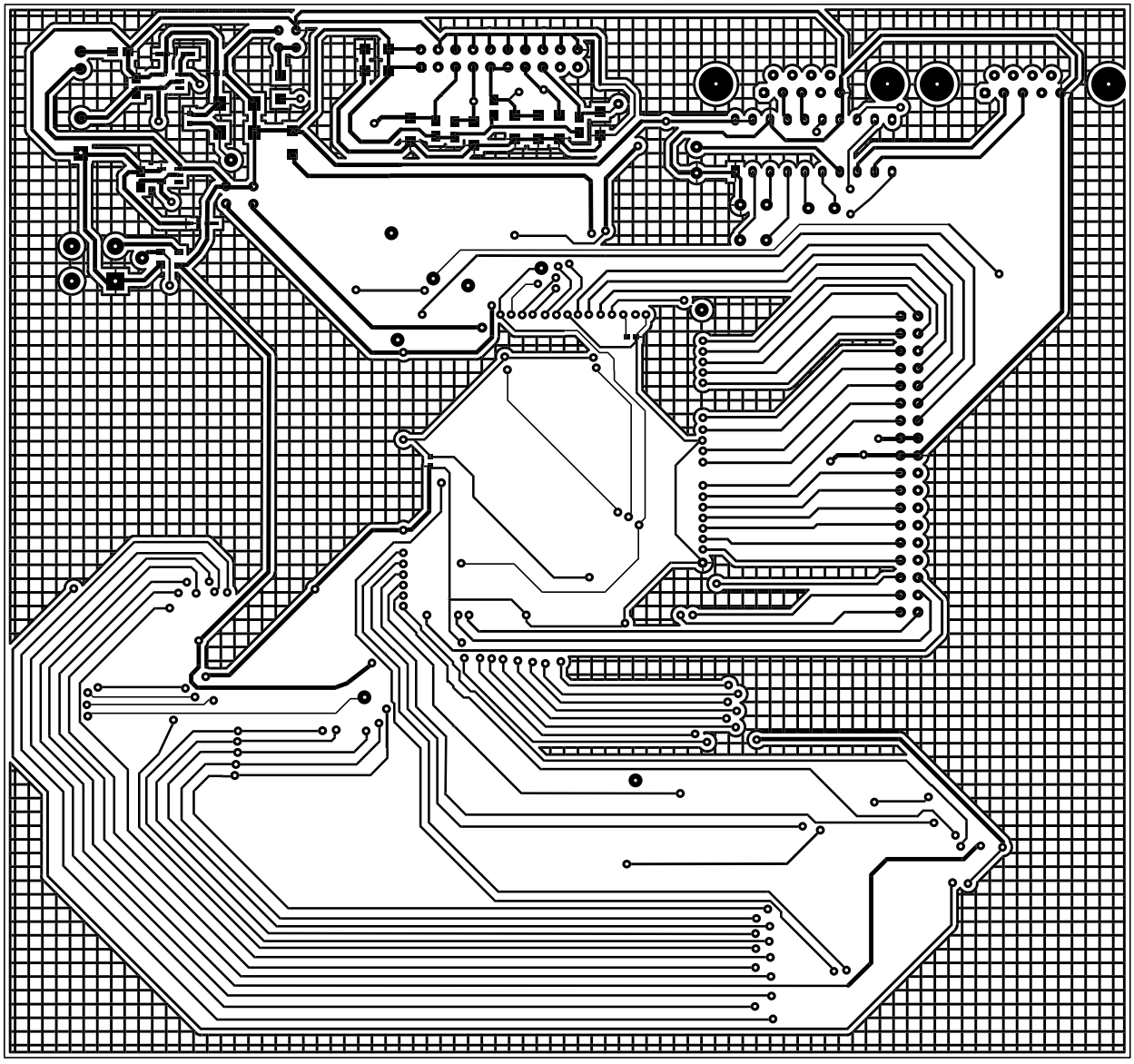
Appendix B
Schematic of Extension-board



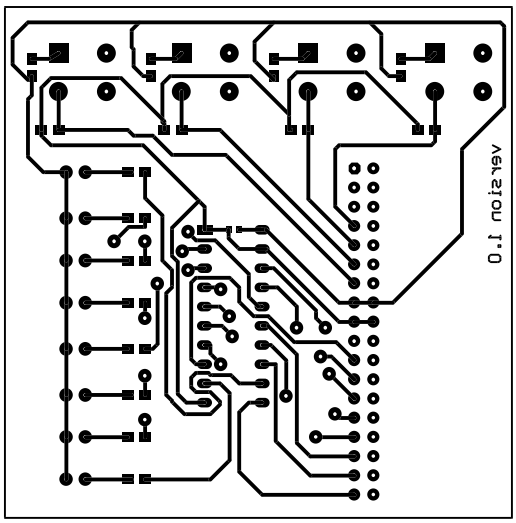
Title		
Size: B	Number	Revision
Date: 6-Jan-2002		Sheet of
File: \\..\OCCTESTCONrapport.sch		Drawn By:

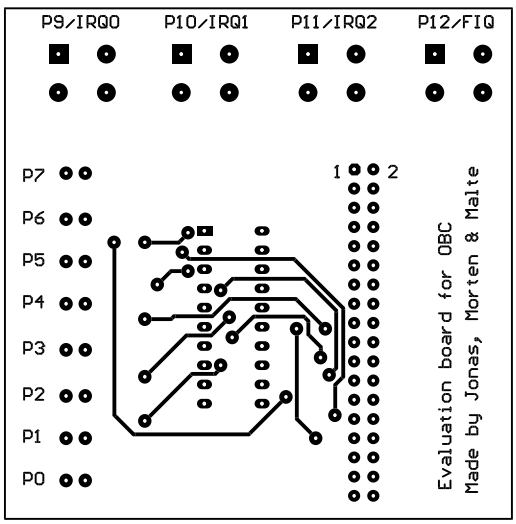
Appendix C – PCB Layout of Onboard Computer





Appendix D – PCB Layout of Extension-board





Appendix E – Linker Script

```
SECTIONS
{
   . = 0x02000000;
   .text : { *(.text) }
   . += 0x9000;
   .data : { *(.data) }
   . += 0x1000;
   .bss : { *(.bss) }
   __bss_start__ = .;
   . += 0x1000;
/*__sbss_start = .; */
/*__sbss_end = .; */
   __bss_end__ = .;
   . += 0x1000;
PROVIDE (__stack = .);
   _end = .;
   .debug_info     0 : { *(.debug_info) }
     .debug_abbrev   0 : { *(.debug_abbrev) }
     .debug_line     0 : { *(.debug_line) }
     .debug_frame    0 : { *(.debug_frame) }
/*     .debug_str      0 : { *(.debug_str) } */
/*     .debug_loc      0 : { *(.debug_loc) } */
/*     .debug_macinfo  0 : { *(.debug_macinfo) } */
}
```

Appendix F – Source of Stackpointer Setup

```
   .externmain
   .externexit

/* .text is used instead of .section .text so it works with arm-aout too.  */
   .text
   .code 32
   .align 0

   .global_mainCRTStartup
   .global_start
   .globalstart
start:
_start:
_mainCRTStartup:

   /* Start by setting up a stack */
   /*  Set up the stack pointer to end of bss */
   ldr  r3, .LC2
   mov  sp, r3

   sub  sl, sp, #512  /* Still assumes 512 bytes below sl */

   mov  a2, #0    /* Second arg: fill value */
   mov  fp, a2    /* Null frame pointer */
   mov  r7, a2    /* Null frame pointer for Thumb */

   ldr  a1, .LC1  /* First arg: start of memory block */
   ldr  a3, .LC2  /* Second arg: end of memory block */
   sub  a3, a3, a1  /* Third arg: length of block */

   mov  r0, #0    /*  no arguments  */
   mov  r1, #0    /*  no argv either */

   bl main
   bl exit   /* Should not return */

   /* For Thumb, constants must be after the code since only
   positive offsets are supported for PC relative addresses. */

   .align 0
.LC1:
   .word  __bss_start__
.LC2:
   .word  __bss_end__
```

Appendix G – Source of blink

```
/* This is blink.c
 * A simple led-flasher for the OBC test-board
 * LEDs must be connected to P0-P7 */

#define PIO_PER  *(volatile unsigned int*)(0xFFFF0000)
#define PIO_OER  *(volatile unsigned int*)(0xFFFF0010)
#define PIO_SODR *(volatile unsigned int*)(0xFFFF0030)
#define PIO_CODR *(volatile unsigned int*)(0xFFFF0034)


int main() {
   int i, dir = -1,lys;
   int last;

   PIO_PER = 0xff;  /* PIO controls lower 8 bits */
   PIO_OER = 0xff;  /* Lower 8 bits are outputs */
   PIO_SODR = 0xff; /* Turn on lower 8 bit, i.e. turn off the LEDs */

   last = 1;
   lys = 2;

   while (1) {
      PIO_CODR = lys;  /* Turn on LED 'lys' */

      for(i=0; i< 10000; i++) {
         /* Flash LED 'last' while waiting */
         if (i%10==0) PIO_CODR = last;
         if ((i+9)%10==0) PIO_SODR = last;
      }

      PIO_SODR = lys;  /* Turn off LED 'lys' */
      last = lys;

      if (dir==-1) {
         lys = lys << 1;
      } else {
         lys = lys >> 1;
      }

      /* Change directon if limit reached */
      if (lys == 256 || lys==1) dir =- dir;

   }
}


/* main() returns to this function */
int exit() {
   while(1);
}


/* Cheat gcc, so that it will compile the program */
void __gccmain(void) {
}
```

Appendix H – Source of Flash Driver

```
/* This is prog.c
 * A flash-programmet for the amd29lv017D flash
 * The first programming adress is hard-coded into the main function */

#define FLASH_Unlock_first         0xAA
#define FLASH_Unlock_second        0x55
#define FLASH_Unlock_sector_erase  0x80
#define FLASH_Program              0xA0
#define FLASH_Sector_erase         0x30

#define FLASH_Sector_size        0x10000

#define BIT_3_MASK                 0x08

#define FLASH_ERR_OK               0x00
#define FLASH_ERR_TIMEOUT          0x01
#define FLASH_ERR_VERIFY           0x02

#define TIMEOUT                   50000

#define PIO_BASE             0xFFFF0000
#define PIO_PER          PIO_BASE + 0x00
#define PIO_OER          PIO_BASE + 0x10
#define PIO_SODR         PIO_BASE + 0x30
#define PIO_CODR         PIO_BASE + 0x34

#define LED_ALL         0xFF

extern unsigned char prog_array[];
extern long prog_array_size();


void led_on(int what) {
   *(volatile int*)(PIO_CODR) = what;
}


void led_off(int what) {
   *(volatile int*)(PIO_SODR) = what;
}


void led_init() {
   *(volatile int*)(PIO_PER) = LED_ALL;
   *(volatile int*)(PIO_OER) = LED_ALL;
}


int program_byte(void *adr, unsigned char byte) {
   volatile unsigned char *flash = (volatile unsigned char *)FLASH_BASE;
   volatile unsigned char *prg_adr = (volatile unsigned char *)adr;
   int timeout = TIMEOUT;
   int res = FLASH_ERR_OK;

   /* Turn on the LEDs to indicate something happens */
```

```
  led_off(LED_ALL);
  led_on(byte);

  /* Write Program Command Sequence */
  *flash = FLASH_Unlock_first;
  *flash = FLASH_Unlock_second;
  *flash = FLASH_Program;
  *prg_adr = byte;

  /* Data Poll and Verify */
  while (--timeout > 0) {
    if (*prg_adr == byte) break;
  }

  if (timeout < 0) res = FLASH_ERR_TIMEOUT;

  return res;
}


int erase_sector(void *sect) {
  volatile unsigned char *flash = (volatile unsigned char *)FLASH_BASE;
  volatile unsigned char *sect_adr = (volatile unsigned char *)sect;
  volatile unsigned char *adr;
  int timeout = TIMEOUT;
  int res = FLASH_ERR_OK;

  /* Write Erase Command Sequence */
  *flash = FLASH_Unlock_first;
  *flash = FLASH_Unlock_second;
  *flash = FLASH_Unlock_sector_erase;
  *flash = FLASH_Unlock_first;
  *flash = FLASH_Unlock_second;
  *sect_adr = FLASH_Sector_erase;

  /* Wait for erase timer to timeout */
  while ( (*sect_adr & BIT_3_MASK) == 1 );

  /* Data Poll from system */
  while (--timeout > 0) {
    if (*sect_adr == 0xFF ) break;
  }

  if (timeout < 0) res = FLASH_ERR_TIMEOUT;

  /* Verify data */
  for (adr = sect_adr; adr < sect_adr + FLASH_Sector_size; adr++) {
    if (*adr != 0xFF && res == FLASH_ERR_OK) {
      res = FLASH_ERR_VERIFY;
      break;
    }
  }

  return res;
}

int main() {
  int testarray[20];
  volatile unsigned int i;
```

```
  int len;

  led_init();

  erase_sector((void*)0x01000000);

  /* Flash the LEDs to indicate erasing finsihed */
  led_on(LED_ALL);
  for (i=0; i<500000; i++);
  led_off(LED_ALL);


  len = prog_array_size();

  /* Do the programming */
  for (i=0; i<len; i++) {
    program_byte( (unsigned char*)(0x01000000+i), prog_array[i]);
  }

  /* Turn on every second LED to indicate programming has finished */
  led_off(LED_ALL);
  led_on(0xAA);

  return 0;
}


void __gccmain() {
}


exit() {
  while(1);
}
```

Appendix I – Source of Bootstrap

```
.equ EBI_BASE,0xFFE00000

   .global__main
__main:
   .long  InitReset   /* reset */
undefvec:
   .long  undefvec    /* Undef */
swivec:
   .long  swivec      /* SW */
pabtvec:
   .long  pabtvec     /* P abt */
dabtvec:
   .long  dabtvec     /* D abt */
rsvdvec:
   .long  rsvdvec     /* reserved */
irqvec:
/* ldr  pc, [pc,#-0xF20] /* IRQ : read the AIC */
fiqvec:
/* ldr  pc, [pc,#-0xF20] /* FIQ : read the AIC */

InitReset:

/*
 * Initialise the Memory Controller
 * --------------------------------
 * Copy the Image of the Memory Controller
 */

   mov  sp,#0x1000

   ldr r10, PtInitTableEBI/*get the address of the chip select register image */

   movs r0, pc, LSR #20    /* pc > 0x100000 */

   moveq  r10, r10, LSL #12  /* Mask the 12 highest bits of the address */
   moveq  r10, r10, LSR #12

   /* Load the address where to jump */
   ldr  r12, PtInitRemap /* get the real jump address ( after remap ) */

   /* Copy Chip Select Register Image to Memory Controller and command remap */
   ldmia  r10!, {r0-r9,r11}  /* load the complete image and the EBI base */
   stmia  r11!, {r0-r9}    /* store the complete image with the remap command */

   mov  pc, r12     /* jump and break the pipeline */


InitRemap:
   ldr  r10, PtProg
   mov  pc, r10

PtProg:
   .long  0x01010000
PtInitTableEBI:
```

```
   .long  InitTableEBI            /* Table for EBI initialization */
PtInitRemap:
   .long  InitRemap               /* address where to jump after REMAP */

InitTableEBI:
   .long  0x01002EFE
   .long  0x10000000
   .long  0x02003121
   .long  0x40000000
   .long  0x50000000
   .long  0x60000000
   .long  0x70000000
   .long  0x00000001    /* REMAP command */
   .long  0x00000006    /* 7 memory regions, standard read */
   .long  EBI_BASE      /* EBI Base address */
```

Appendix J – Source of USART test

```
/* This is usart.c
 * A USART test program for the OBC test board
 *
 * The byte format is: start + 8 data (without parity) + 1 stop
 * The baud rate is counter is 80: 9600 bauds with MCKI = 12.288 MHz
 */

#include "parts/m40800/reg_m40800.h"

int main(void) {
   unsigned int loop_count;
  unsigned int value;

   /* Initialize the channel */
   PIO_PDR = ( PIOTXD0 | PIORXD0 ) ;
  US0_MR = ( US_ASYNC_MODE | US_CHMODE_NORMAL ) ;
  US0_BRGR = 80; /* baud rate */

   /* Start channel */
  US0_CR = ( US_RXEN | US_TXEN );
  for (loop_count = 1000; loop_count>0; loop_count--);

   value = 'A';

   while(1) {
     /* Wait Tx ready */
     for (; (US0_CSR & 2) == 0;);

     /* Send byte */
     US0_THR = value + 1;

     /* Wait Tx ready */
     for (; (US0_CSR & 2) == 0;);

     /* Send byte */
     US0_THR = value + 2;

     /* Wait Tx ready */
     for (; (US0_CSR & 2) == 0;);

     /* Receive and check byte */
     value = US0_RHR;
   }

   return(0);
}


void __gccmain() {
}


exit() {
   while(1);
}
```