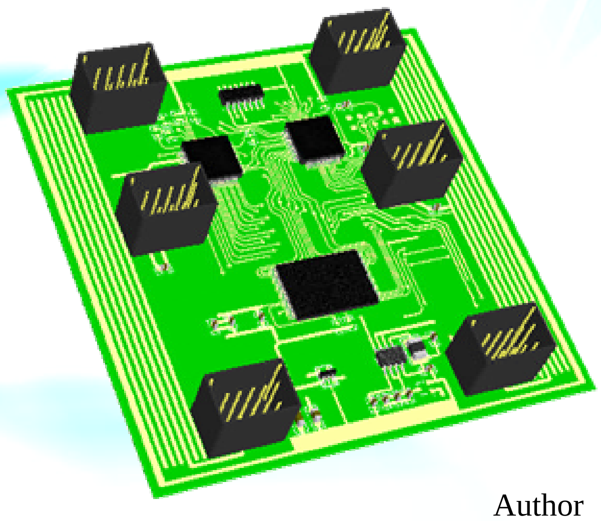
Diplomarbeit

compass one

DESIGN AND DEVELOPMENT OF A CDHS FOR A PICO SATELLITE



Author

Artur Scholz
Matrikel Nr. 185871
Fachhochschule Aachen

www.raumfahrt.fh-aachen.de

FH AC Hochschule Aachen University of Applied Sciences

DLR

Scope

This paper documents the design and development of the on-board computer system for the Compass-1 picosatellite, i.e. the so-called command and data handling system (CDHS). The work is based upon the preceding detailed definitions elaborated in previous studies [1] and goes in accordance with the CubeSat specifications [2] wherever applicable.

All work was carried out by me personally. Whenever there is a quotation, the corresponding reference is noted and can be found in the appendix.

Artur Scholz
Aachen, 24.09.2004

Acknowledgements

First I like to thank **Prof. Blome**, **Prof. Ley** and **Mr. Plescher** for their efforts and support for me personally and the Compass-1 project in general. In particular Mr. Plescher has become the point of contact for our questions and organizational issues concerning the realization of this project.

It is a big venture to parallel design a computer system for a picosatellite in terms of hardware and software. Since I could not revert to extensive electrical engineering background I had to start a lot of research to improve my capabilities in this area. But I would not have been able to complete this activity without the enormous help of **Prof. Dr. Timo Bretschneider** and the team of the Satellite Engineering Centre, Singapore.

The development activities of the engineering model turned out to be more convenient. The file with the PCB layout was handed over to **Mr. Nieren** at the electric workshop at the FH Aachen and the board was produced within a few days.

I also want to express my thanks to the German Academic Exchange Service, **DAAD**, for the scholarship that made it possible for me to conduct main parts of my diploma work in Singapore.

Contents

1. Overview

1.1 Introduction

The design and development activities for the command and data handling system of the Compass-1 picosatellite were largely carried out at the Satellite Engineering Centre[1] (SEC), which is part of the Nanyang Technological University[2] (NTU) at Singapore. The time frame was defined within three months. Beforehand, within a preparation time of about two months, confidence was gained in handling the PCB software by means of preliminary drafts created at the satellite lab of the FH Aachen[3].

The outcome of this work is wholly dedicated to the Compass-1 satellite[4], which is being developed at the FH Aachen. At the same time, the work corresponds to the obligatory final year thesis that has to be completed by each student to eventually graduate.

The expected results of the works are twofold: Primarily it shall result in a fully developed engineering model (EM) of the CDHS in order to expose it to physical testing and integration test for final assembly of the whole spacecraft. The second important aspect of this work is to supply a comprehensive and comprehensible documentation of how this system was elaborated. Expectantly it will serve as a guideline for future ventures with similar characteristics.

The engineering model shall path the way for the subsequent flight model, which in turn is supposed to have only slight modifications; in the best case it would have virtually none. This means, that the engineering model reflects the corresponding system in such a way, as that all necessary measurements have been applied and all critical aspects found consideration. Essentially the engineering model encompasses the main board with its electrical components and connectors and the dedicated software, implemented in the main controller unit.

The software which was used for the hardware layout is Protel 2004. Altium supplied a license that allows the full utilization of their software CAD product until the end of 2004. The software package has proven to be powerful and at the same time intuitive in its handling.

For the development of the software code the Integrated Development Environment (IDE) from Silicon Laborites was used. It is included in the Evaluation Board package sold by Silicon Laboratories and makes use of the Keil ‘C’ Compiler and Assembler. The ‘C’ Compiler is however limited to a code size of 4 Kbytes in this free version.

[1] http://www.ntu.edu.sg/centre/sec/
[2] http://www.ntu.edu.sg/
[3] http://www.fh-aachen.de
[4] http://www.raumfahrt.fh-aachen.de

1.2 Purpose of the CDHS

The satellites command and data traffic has to be routed and scheduled in an appropriative manner that takes into account the specific mission objectives and requirements. Furthermore the payload data may usually imply the need of computing functions to be carried out on board the spacecraft.

The CDHS of Compass-1 does this by providing hardware and software solutions for the controller unit, the system bus, the data memory unit and the payload interface unit.

The command and data handling subsystem is responsible for running the spacecraft in an ordered and ‘intelligent’ manner according to its flight program. It is the ‘brain’ of the operation. Whereas the Electrical Power System (EPS) is responsible to guarantee a minimum working system in order to keep the satellite alive, only the CDHS allows the satellite to fully carry out the mission goals and hence distinguish the spacecraft from useless space debris.

The basic functions of the Compass-1 CDHS are to execute commands coming from ground (and arriving at the CDHS via the COM subsystem) and periodically gather vital information from other subsystems and store them in memory.

The commands from ground encompass the change of ADCS parameters, image capturing and transmission and request for housekeeping data. The CDHS ensures that this data is available on by providing a non-volatile mass memory unit. It also provides the system bus interface to the other subsystem boards.

1.3 The Design and Development Process

The design process shall derive a physical architecture and design (including software), from functional analysis and requirement allocation. The output from the design will describe the system physically and in terms of software on the lowest assembly levels, budgets, interfaces and relationships between external and internal items. The subsequent development activities then cover the production of the system. The block diagram in figure 1.1 illustrates the entire subsystem life cycle.

Hardware and Software design and development generally goes hand in hand. Yet, the software is by nature the more flexible part that can be modified more easily at later stages to justify ad-hoc changes. The hardware however, once it is finally developed, is less likely to adopt changes since in most cases this would result in the production of a new piece of its kind. Therefore it is very advisable to finish with the hardware design first but wait with the development until the software has advanced far enough, so that modifications that occurred due to software design can still be integrated in the hardware design.

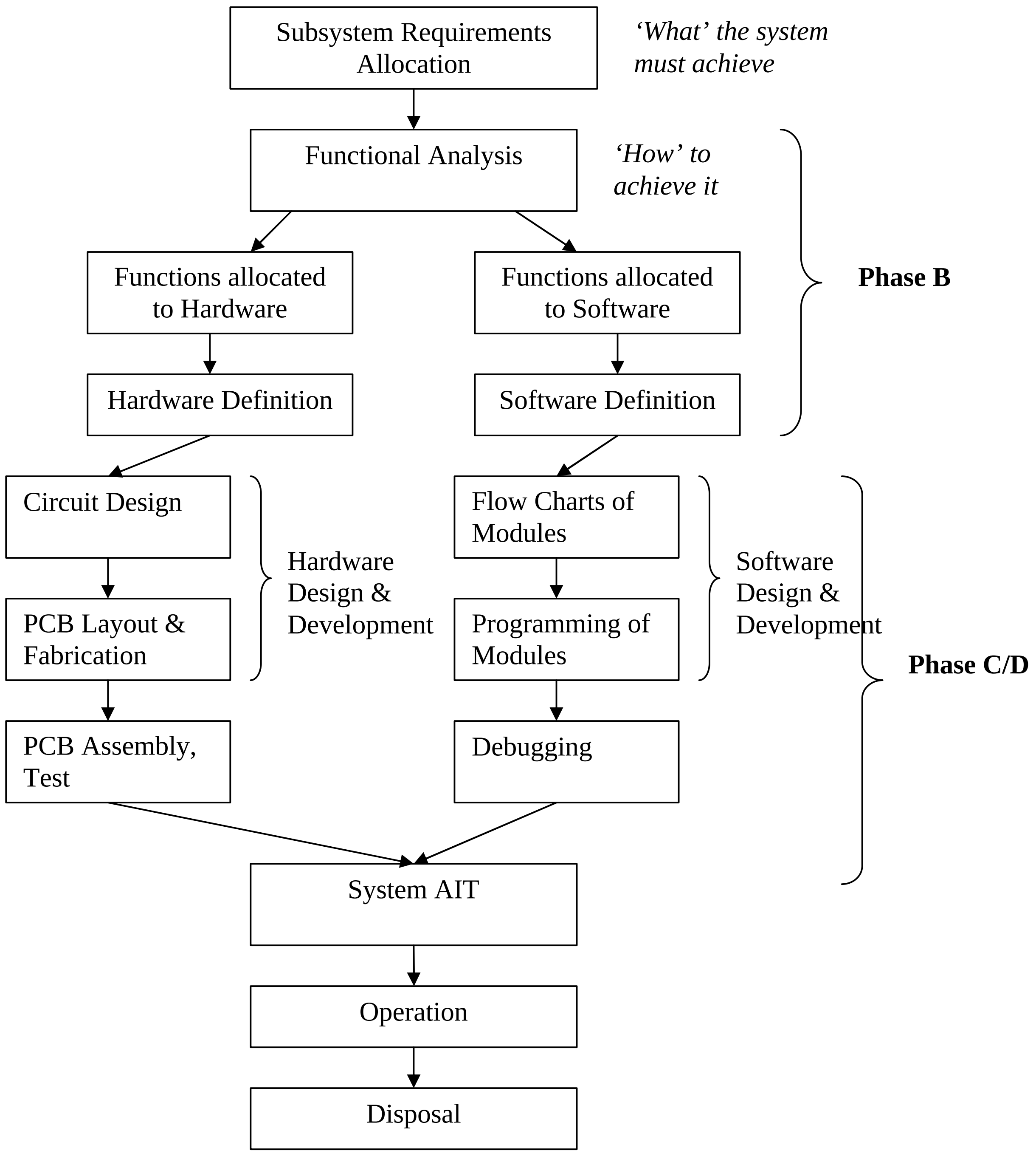


Figure 1.1: The product life cycle CDHS subsystem

Subsystem Requirements Allocation

This first step concentrates on the requirements that are exposed to the subsystem. Beforehand, the overall system had been divided into smaller functional blocks, and passed on certain requirements to them. Each subsystem consequently has to fully comply with the requirements to make the system work as a whole. And even so a satellite might be composed of the same set of subsystems that another satellite has, the specific requirements of the particular subsystems might differ significantly according to the spacecraft mission objectives. Generally, the requirements can be obtained by asking 'what' the subsystem shall achieve. As an example, one can figure out that the CDHS has to *store* data from the payload.

Functional Analysis

An analysis is then carried out to translate the requirements into functions. Functions describe the proposed solution to the requirements by asking ‘how’ they can be accomplished. This step does not come up with a suggestion of specific components that can be used. Instead, it describes the functions that have to be fulfilled by this subsystem. In our example, the CDHS requirement to *store* the data would be formulated as to include a *memory* as functional element.

Allocation of Functions

The elaborated functions are then grouped and partitioned in order to address them to the next lower assembly level, i.e. the hardware and software parts. This process is strongly influenced by constraints and requirements, such as available technology, cost, schedule, risk, manufacturing and test capabilities, the space environment factors, and other considerations. The implementation of each functional element is based on these decisions.

As for electronics, either hardware or software, and sometimes a combination of both, can be used to implement those functions. In fact, very few electronic functions are now performed solely with software or hardware. Most require a combination of both to meet their requirements. By its nature, software tends to be the more flexible solution as it allows modifications even in advanced stages of the product life cycle. Hardware though, once developed, will be more cost and time consuming if changes have to be implemented because in most cases this would cause the manufacturing of a new piece of it. It is nevertheless advisable to address as many functions to the hardware as possible because in space the software is more delicate to malfunctions.

Hardware Definition

Following the allocation of functions, components are selected using trade-off studies that again take into account availability, cost, heritage, risk, schedule and so on to find the most suitable product for that specific task. It shall however be noted, that there will always be products that might suit the purpose better than the chosen one, especially in the electronics industry where performance increases are very fast paced. Therefore, the final selection has to be seen as the most appropriate solution at that time, which was chosen with due diligence.

Software Definition

Also for the software there exist fully developed solutions, e.g. operations systems (OS). The OS enables the programmer to write code on a highly abstractive level, which does not take into account the final hardware selection but rather concentrates on the functions to be carried out. The main advantage of this is that the code can be easily modified and transferred to other hardware configurations of even other projects.

The other possible way to define the software would be to write a code that is dedicated to the chosen hardware configuration. In that way, the hardware resources can be fully utilized and the code size will remain relatively small compared to an OS (however, the programmer will have to write more lines of code since all low level routines have to be written as well).

By either method, a top-down approach is highly desirable, which enables the programmer to write from the highest level (the flight software) downwards to the lowest level (i.e. the OS or hardware interfaces, respectively). Thus the top level flow chart shall be created at this stage.

1.3.1 Hardware Design and Development

Next to the software design and development this is the one major part covered within this documentation. All activities attached to the design and development process are carried out within phase C/D and its application and results can be found in the next chapters. Figure 1.2 shows those activities again in more detail.

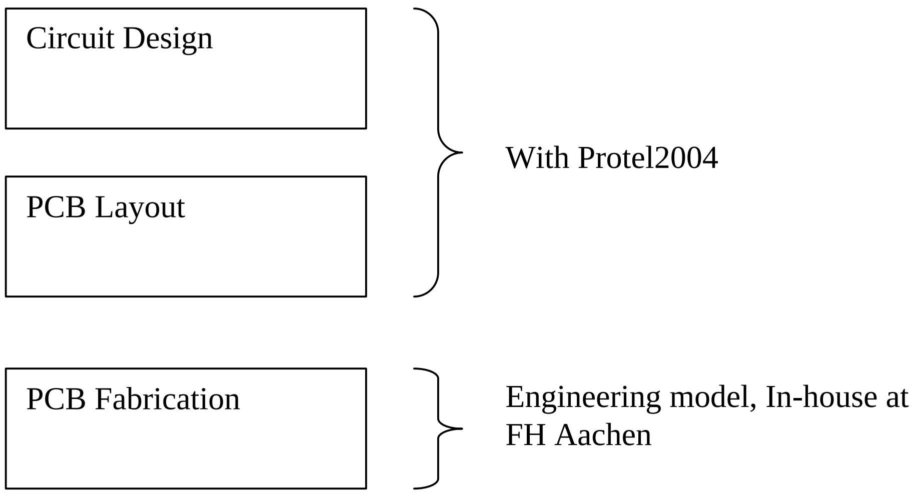


Figure 1.2: The hardware design and development process

Ideally, each step in the process should be accomplished in a logical sequence, avoiding "loop-backs" or repeated activities in order to save time, money and human resources. However, such a perfect approach is hardly ever achieved. Therefore it is very advisable to make use of gateways or reviews after each block in order to 'freeze' a successful design. For a student project with a very limited time schedule, this will not be done in an official manner but rather as an individual attitude towards the own work.

Circuit Design

After the functions are addressed to the subsystem and the technical solutions to implement those are defined, the design of the circuitry can begin. Initially the interoperability of the devices is identified, i.e. how they will work with each other. Then the signal flow is determined, covering signals within the subsystems components (i.e. the internal interfaces) and with external interfaces. These interfaces have to suit certain requirements, such as data structure and content, compatibility, physical connection and signal timing if appropriate.

Included in the circuit design is the selection of parts that are necessary to implement in order to adjust the main devices in a proper manner, e.g. resistors and capacitors. Finally, all circuitry required for each function on the subsystem can be designed in detail.

It should be mentioned that the circuit design activities usually are the most time-consuming part of the hardware design. Once the circuit has been proven to be correct, the downstream operations ought to be straightforward.

At this stage of the design process it might be appropriate to set up breadboards or construct a simulation model in order to demonstrate that the design is conceptually sound before it is handed over to the layout process.

The result of the circuit design will be one or a set of schematically drawings, i.e. schematics, which build the basis for the next step: the layout process.

PCB Layout

The circuitry created in the previous step is based upon logical interconnections. Now these logical connections consequently are translated into physical tracks, which will eventually lead to the layout of the final PCB. This step also takes into account other important issues, e.g. the physical dimensions, such as length, width, height and mounting methods.

The PCB layout activity consists of the definition of the board layout, establishment of layout design rules, the transition from the schematic to layout, part placement and finally, the routing.

PCB Layout activities:

- **Definition of board layout**: Primarily, the board is defined in its size and shape by the spacecraft structure and the location where the PCB will be installed. These are the external requirements. The internal configuration of the board (e.g. the number of layers and its material) are determined by other factors, such as fabrication possibilities and costs. The layout also covers the definition and location of mounting hardware (e.g. holes for screws) and keep-out areas (where no tracks shall be placed).

- **Establishment of layout design rules**: The design of the PCB layout is influenced by a set of constraints, which are formulated as design rules. Those rules take into account mostly the later manufacturability of the board when the final layout is handed over to the workshop. Critical aspects are the minimum track width, the minimum clearance between tracks and so on. Taking those constraints into account before the actual physical layout design has begun, saves the designer a lot of work and time because it will be assured that the PCB layout will be producible.

- **Transition from schematic to layout**: The most important ingredient to the board layout process is the schematic that has been created in the circuit design process. It specifies what components are used and how they have to be interconnected. In a completely automated environment (such as offered by Protel) the logical parts from the schematic are replaced by their physical footprints, which are either found in the comprehensive built-in libraries or manually created by the designer. The footprints (also called land patterns) are then displayed in the layout sheet and represent the position where the corresponding component will be placed.

- **Part placement**: This step is one of the most critical ones in the board layout process. It affects how well the interconnections can be realized, its manufacturability (concerning the soldering of the components on the board) and even environmental issues (thermal, radiation). As each new board design is a unique venture, there exist very few standard rules for the placement process. To some extend, this is a quite arbitrary work. But as can be seen during the routing activities, there are placement configurations that are much better than others, because the tracks lengths are much shorter and the whole interconnections are less complex.

A good approach to optimize placement is to control the so-called *rats nest* (the point-to-point interconnections among the components) to avoid too congested areas. In addition, ones the placement is meant to be complete, performing a test route with the autorouter helps evaluating the proposed placement. The location and quantity of unrouted lines indicate problems with the placement. Hence, the designer can rearrange the parts to solve those problems iteratively.

- **Routing**: After the part placement the point-to-point interconnections which are shown as lines from pin to pin are now replaced by physical tracks, taking into account the previously defined design rules. Depending on the quality of the autorouter it is advisable to first route most or at least the important interconnections by hand first and then let the computer finish the work. Completely autorouted layouts tend to look mazy; even so they follow all the defined design rules.

PCB Fabrication, Assembly and Test

When the routing has been completed and brought up a satisfying board layout, the files can be handed over to the manufacturer (commonly in form of ‘Gerber’ files). The manufacturer produces the PCB according to the layout drawings. Tracks are created either by chemical or mechanical removing of the conducting metal foil areas next to the lines. The fabrication also comprises the drilling of holes and Vias (interconnections between layers). Including a solder mask will facilitate the application and flow of solder employed during component assembly. Furthermore it will provide environmental protection or insulation between closely spaced conductors.

Before starting the soldering of components onto the board, the quality of the board shall be verified. This is done by visual inspection as well as electrical testing. The visual inspection can faster identify broken tracks or short circuits. Also critical contaminations might be found. Though, only the following electrical test will even recognize defect tracks, which the human eye would not be able to find. This verification might take some time but it is a critical activity that should not be left behind.

After the soldering of the components there shall be again inspections and tests. This is done to confirm that all parts work properly and that they are correctly interconnected. The tests however can only verify that the parts of the board as a standalone will work whereas interfaces to external parts can only be taken into account to some extent, such as by emulating inputs/outputs. The functional testing on the next higher level can only be carried out when the board is being installed together with the other subsystems into the wholly system.

Apart from inspections and electrical tests, the board shall also be exposed to harsh environmental conditions as they appear in orbit, such as thermal cycles, extreme cold and hot temperatures, radiation and high vacuum. A board that still is fully operational after those tests can be seen as reliable to work in space as well.

1.3.2 Software Design and Development

Software programming is an activity that can cause deep frustration or great joy, depending on the fact if the program is erroneous or if it works correctly. Simultaneously the programmer is exposed to high pressure on effectiveness, because for a program to work it requires a code that has no mistakes!

Some people (mainly hobby programmers) use a trial and error approach to get their programs to work as they wish. They usually implement little planning or detailed design into their software. Doing so also makes this a very time consuming activity, too.

Trial and error is the least effective way to write software. Instead, the software code shall be derived from a deliberate and careful design approach that facilitates later changes and modifications. It also brings more transparency into the code, which is highly appreciated by other people who want to get themselves familiar with how the program works.

There are two design approaches that are well established among software engineers. Those are called *structured programming* and *top/down design*. Both are mutual important and only the simultaneously use of both of them can form an efficient synergy.

Many high and middle languages for programming have benefited greatly from those techniques and in turn reflect those characteristics in their design tools. However, even highly sophisticated design environments do not enforce good programming. Good programming depends on the discipline of the programmer. Therefore the programmer has to understand and embrace the techniques of the mentioned design approaches to benefit from it.

In order to visualize a program and to make the algorithmic process of a code better understandable, flowcharts are commonly used. A flowchart is a diagram that displays several standard symbols connected by flow arrows. Each flowchart symbol represents actions to be carried out. A flowchart is used as a design tool and will eventually serve for documentation, when the design is finally established. It clearly illustrates what the program does and how it does it.

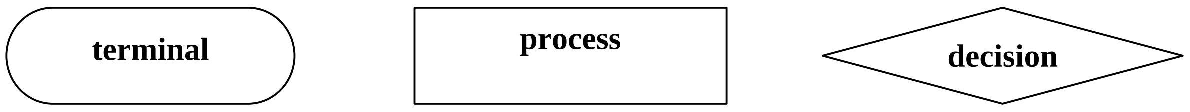


Figure 1.3: Flowchart symbols

Flowcharts consist of three basic elements: the terminal, the process block and the decision (figure 1.3).

Intensive use of comments in the code together with flowcharts is a valuable source of information for the programmer as well as others.

Structured Programming

Simplicity (much like it is postulated by the KISS idea: “Keep It Simple, Stupid”) is the key to good programming. Complex programs have a propensity to be less understandable (even by experts), harder to modify and worst: to write.

The application of flowcharts brings a structure into the code design. Apparently the flowcharts with only three types of symbols are a simply concept, however connecting those building blocks in a disorganized manner will result in *spaghetti-code* rather than a lucid diagram.

There are again three fundamental ways to interconnect the flowchart symbols. Those fundamental program structures are: SEQUENCE, IF-THEN-ELSE and DO-WHILE. They represent the building blocks of structured programs. In fact, only programs that orderly make use of those structures are referred to as being correctly structured.

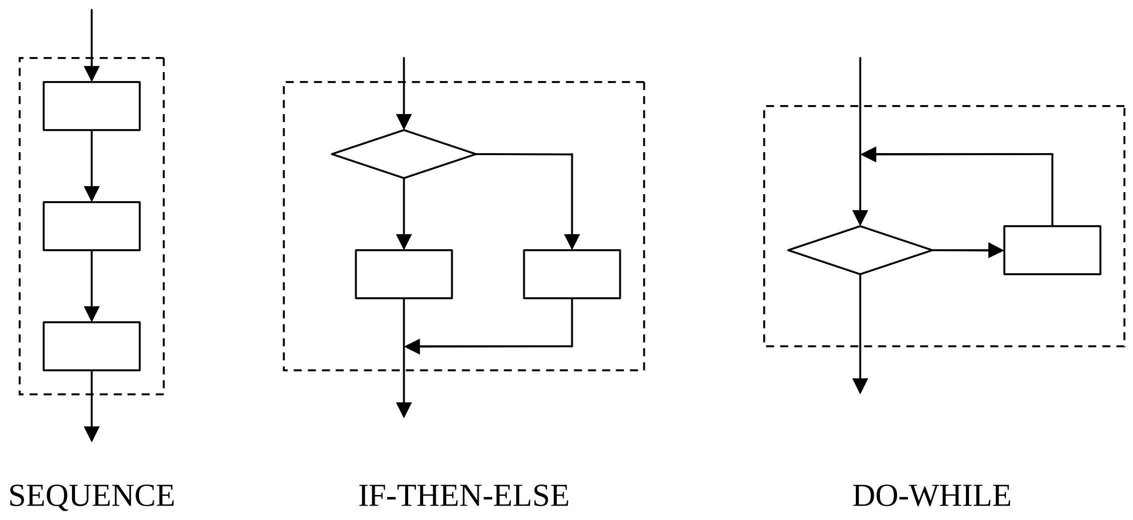


Figure 1.4: Fundamental program structures

The SEQUENCE structure is an alignment of process boxes, which are gone through one after another.

The IF-THEN-ELSE structure chooses between two alternative processes. In case of an IF-THEN, which means that something is only done under certain conditions, one box would be left empty. This way, a process is either carried out or bypassed.

The DO-WHILE is a loop that exits upon a decision. If the condition for exiting the loop is not fulfilled, the process is repeatedly executed.

Consequently applying and enforcing structured flowcharts may seem as a limitation to the program code for some people. Nevertheless, if the problem cannot be solved in the flowchart, neither can it be solved with the code.

Top/Down Design

The top/down approach is an old and simply idea but its application for software coding is relatively new. It means that one shall look at the bigger picture first and then elaborate the subsequent levels into finer and finer details during the design activity. It is much the same approach that is used for the whole spacecraft itself. At first the major functional parts are identified, which form the top level design. Following that, these functions are broken down into smaller parts that form the next lower design level. This is continued until enough detail exists to write and document a software program.

On all levels, the modules shall be practical and easy to understand. Most important: only related functions should be together in the same module.

The top/down design approach greatly benefits form the use of flowcharts. Flowcharts shall be used for each module at each level, whereby a lower level module, which comprises of various decisions, sequences etc., can be represented by a single process block in the next higher level. This can be correctly achieved because each process block has a single beginning point and a single ending point, which is also true for each module flowchart.

It is the programmer's decision to decide the number of levels that seem to be useful to implement in the top/down design. Generally, the flowcharts shall be kept hardware independent, i.e. no hardware details should be placed in the building blocks of a flowchart.

The programmer does not have to delay the programming activities (which represent the development process) until the entire software design is complete. Instead, the programming can and shall start even before the lower levels are designed!

From the high level flowchart the lower level modules can be perceived as *black boxes*, which fulfill their designated purposes as soon as they are correctly and completely designed and programmed. In order to test the high level designs and hence their program code, those black boxes are replaced by dummy modules, so-called *stubs*. Stubs do not carry out correct functions but substitute the real code in a very simply way. They might either return with a certain value or in many other cases they will just do nothing. The stubs will be later replaced with the actual code. By doing so, the seemingly impossible suggestion to start programming (and testing) on higher level before the entire design has finished can be realized.

1.3.3 System Assembly, Integration and Test

The final assembly of the subsystem, its integration into the system and tests on component, subsystem and system level are all essential procedures within phase C/D. They are however not covered in this document since it would have gone beyond the scope of this diploma work. They will be executed afterwards and then separately documented. But the following will give an idea on how those steps will be realized.

The PCB assembly is done by soldering the devices and components on the manufactured board. A person who is confident in soldering such small footprints is required for this and there are some available in the team and at the university also. The equipment is at the lab of the FH Aachen, too. The program code will then be programmed into the MCU internal FLASH RAM through the JTAG interface. This is very convenient and allows in-system debugging as well. That means that the EM board will serve as the test bed for the software. The complete board will undergo the necessary tests on subsystem level. Previous to that, some critical devices might be tested beforehand.

The integration into the system means to mount it inside the EM structure and to adjust the subsystem boards on it. Another connection has to be done from the CDHS board towards the camera module.

The tests on system level comprise the simulation of in-orbit operation in terms of data and command exchange. Appropriative ways to emulate the other subsystems and to verify a proper function of the CDHS have to be created. This phase will finally conclude with the qualification tests, which will be exposed to the satellite as a whole.

1.4 Design Approach to Failure Minimization

1.4.1 Component Classification

Contrary to most earth-bound gadgets, space technology has to withstand very harsh environmental conditions that exist in the orbit or trajectory the spacecraft is traveling. In particular the electronic devices are susceptible to failures. Hardware can be affected in such a way as that a part will fail temporary or even permanent. Software malfunctions will result from hardware errors and may lead to the complete breakdown of the system if no adequate countermeasures are implemented.

For that reason industry has developed dedicated components that fulfill the functional characteristics of their original devices but are qualified to be used in space. But the improved reliability of those components makes them by magnitudes more expensive. And their performance can in most cases not draw level with the up-to-date products from the commercial off-the-shelf (COTS) industry.

For Compass-1 the same is true as for all university satellites: cost budget is tight and therefore the use of space-graded products is simply excluded. Instead, COTS components will be implemented nearly exclusively. In fact most organizations apply some combination of approaches to justify the use of COTS components; usually considering the overall life cycle costs and trading off risks, performance, and costs. For the majority today however, the main driver for the use of commercial parts is performance and availability [4].

In the following sections the sources for possible malfunctions in the on-board computer system are analyzed, which might affect in particular commercial devices and practical countermeasures are discussed.

1.4.2 Failure Sources and Intended Countermeasures

For the electronics of the CDHS board three types of environmental influences are dangerous:

- mechanical deformations
- extreme temperatures
- radiation

Mechanical Deformations
They might be caused by different thermal expansion coefficients of coupled parts, e.g. the SMD devices and the PCB. In addition, the launch will expose much structural stress on all of the implemented components. The way to circumvent those failures is to design the whole spacecraft in such as to avoid centers of extreme stress points. And to conduct tests and checks that will prove survivability.

Extreme Temperature
All electronic devices will only operate within certain temperature boundaries. If those limits are exceeded the part will almost certainly fail permanently. The industrial temperature range specifies proper functionality from -40...+85 degrees and is available for most devices and parts. Those shall be used preferably for the Compass-1 satellite. Some devices however

do not comply with this characteristic and have to be accommodated in such as that they still stay within their temperature limits for all orbits.

Radiation

The CDHS board is placed inside the CubeSat frame and thus surrounded by shielding aluminum plates with a thickness of about 1mm. And even so this will block a big amount of radiation, there is still a fraction going through. Sources of radiation are radiation belts (Van Allen belt), solar winds and galactic cosmic rays. The effects of radiation can be distinguished in two groups [5], [6]:

Cumulative long-term degradation

- Total Ionizing Dose (TID)
  Causes degradation of devices

Single event effects (SEE)

- Single event upset (SEU)
  Causes bitflips (data and code)
- Single event latch-up (SEL)
  The affected device will draw extensive current and eventually burn out

For the Compass-1 picosatellite, with mission duration of six month, the TID effects are not of big concern. This can be said because the radiation dose in LEO orbits is in the range below 1krad/yr and compared to state-of-the-art devices (Flash: up to 10krad, CMOS: approx. 100krad) this is not significant.

On the other hand the SEE are a significant issue! To reduce the probability of their occurrence there are two discussable methods: shielding as well as radiation hardening. Both methods aim for fault avoidance. And both are not practical to the picosatellite approach, due to the strict mass limitations and the tight cost budget. Note also that the techniques of shielding as applied to address total dose issues, often prove ineffective for addressing single event issues, and in some cases can lead to worse situations as particle interact with the shielding to cause secondary effects.

An efficient approach for a picosatellite like Compass-1 can therefore only be failure tolerance together with graceful degradation. This encompasses the application of redundancy in hardware or software. It also includes testing of components to verify their functionality in space as good as possible. And it means to address priorities to subsystems and sub-parts and to construct them in such as to have guaranteed minimal working systems in cases of major malfunctions.

The SEU problems are going to be addressed by a combination of hardware and software. This implies the use of a watchdog timer and possibly error detection algorithms.

The SEL effect is being solved by the EPS board that cuts off the power supply temporarily in case of extensive current draw.

2. Hardware

2.1 Overview

The hardware of the command and data handling system essentially comprises a Printed Circuit Board (PCB) with the electronic devices soldered on both sides and some mounting aids to attach it properly to the spacecrafts structure. In addition, there are connectors on one side of the board that connect them to the other subsystem boards. This is illustrated by the figure 2.1 below.

The PCB, which supports and interconnects the electronic components, is composed of two basic elements. One is the base substrate, which is a combination of an insulating dielectric and a reinforcing material that is embedded within. The other constituent element is a metal foil from which conductors are formed to produce the circuit paths between the components.

The CDHS circuit design is pure digital because all the processed signals are in binary format, i.e. they are either logical ‘0’ or‘1’. No analog conversion or processing is done on this board.

Apart from the subsystem connectors and the header for the JTAG interface, all components are surface mounted devices (SMD). Nowadays almost every type of standard part can be obtained as SMD. This configuration satisfies the need for more complex and highly miniaturized PCBs. It will also facilitate the manual soldering process. However, there are critical aspects attached to the use of SMD parts. The different coefficient of thermal expansion (CTE) of the board and the device may lead to cracks and broken leads, which would cause a discontinuity of the connection.

Components are placed on both sides, whereas critical ICs are located on the top side in order to make us of the radiation shielding effects of the boards. It can be seen from the CAD figure of the inner configuration that those parts that are facing towards the geometric centre will be less exposed to the harsh radiation that enters from outside the spacecraft.

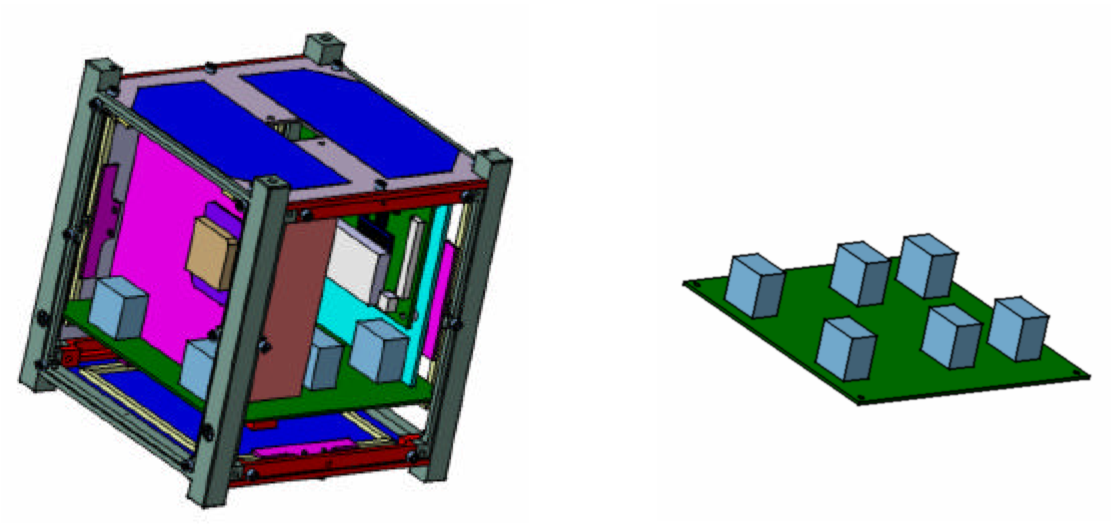


Figure 2.1: The main board location inside the spacecraft

The board is not going to be multilayer but will be limited to use top and bottom layer only. A double-sided board is much easier and cheaper to develop and due to the relatively small number of components used on the board (most functions are provided as on-chip solutions) it does not become necessary to have more then two layers for the tracks. Also, the in-house workshop of the FH Aachen is limited to the production of double-sided PCBs.

The main electronics that reflect the systems functionality are integrated circuits (ICs) for digital signal processing. Those devices are the active parts.

Discrete passive devices comprise resistors, capacitors, conductors and inductors and are necessary to shape the functionality of the ICs and to respond to possible fluctuations in the signal strengths.

Following the selection of each of the parts to be used for the system, the layout for their interconnectivity is designed and eventually developed, which is described in detail in the next sections. The development process itself comprises the manufacturing of the PCB and the soldering of the components. Finally, tests are carried out to verify the functionality of the system.

2.2 Hardware Design Tools

The hardware part of the CDHS design and development process concentrates on the creation of a square printed circuit board with given dimensions. Usually the term PCB refers to boards that are manufactured by chemical etching (e.g. with iron (III) chloride or ammonium persulfate), hence it’s referred to as ‘printed’. This approach however is not very practical for our case, as only a few boards (two engineering models and a flight model) are being built. PCBs can also be manufactured by mechanical etching, in which a trace is etched by milling away the copper along its perimeter. Such a CNC milling gadget is available at the FH Aachen. The final layout has to be delivered to the workshop in form of a ‘Protel 99’ or ‘Gerber’ file.

Prior to the layout activities, which comprises component placement and the routing of tracks, a schematic has to be created first as will be explained later on.

There are computer aided design tools available, some of them even freely available at the internet. The software ‘Eagle’ would be an example for a cost-free program that allows the creation of Gerber files and provides an easy to use graphical interface. By its nature it aims the private, non-commercial user market and is by far powerful enough for those purposes.

When it comes to more complex projects, with hierarchic levels of schematics, professional engineers turn to sophisticated products, such as OrCad and Protel, which provide a solution for the whole design process, from schematic to final layout. For the design of the COMPASS-1 CDHS, the company Altium has generously provided a full Protel2004 license free of charge. Protel2004 enhances the capabilities of its previous version and is backward compatible. Figure 2.2 shows a snapshot of the software.

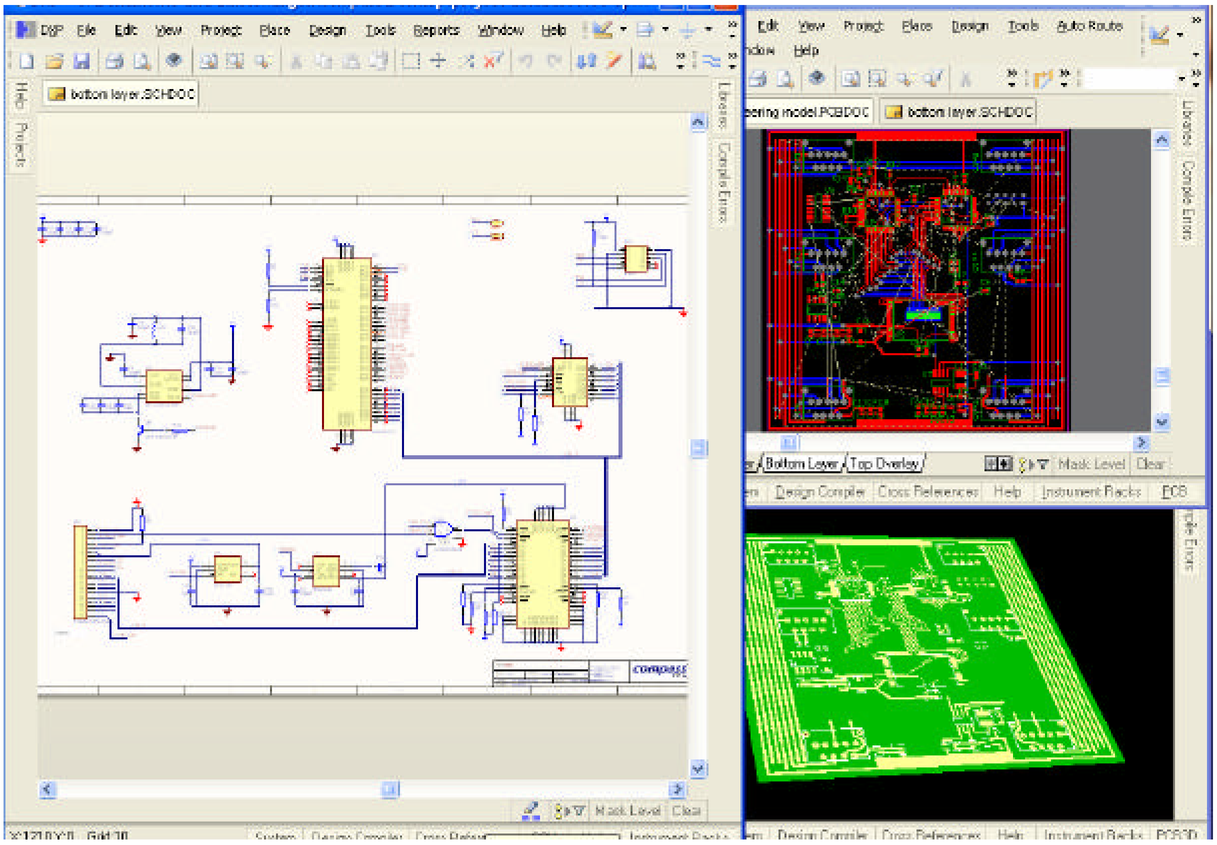

DXP File Edit View Project Place Design Tools Reports Window Help
bottom layer.SCHDOC
Bottom Layer Top Overlay
Mask Level Clear
Design Compiler Cross References Help Instrument Racks PCB

Figure 2.2: Protel 2004

2.3 Design and Development Activities

Outlined in figure 2.3 are the several steps that constitute the design and development process for the PCB hardware. It starts with the creation of a schematic, which is the reflection of the circuitry of the devices to be implemented. The critical input to this activity is the mechanical definition of the PCB, i.e. the constraints that evolve from the specific purpose of the board and its location and interfaces in the system it is being installed. Following that, the logical layout is transferred into a physical one with footprints reflecting the position of the components. Those are then placed in an appropriate manner and will eventually being routed. When this step has been completed, the board is fabricated and tested.

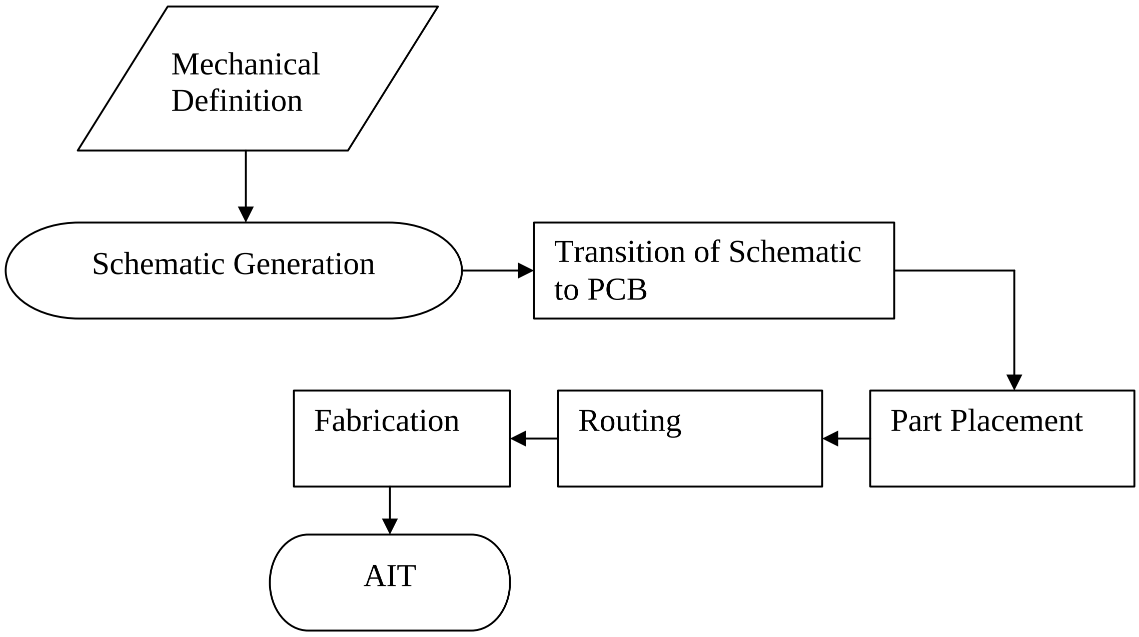


Figure 2.3: The hardware design and development activities

2.3.1 Mechanical Definition

A description of the board’s physical/mechanical properties and constraints is the input to start the circuit layout process.

The workshop of the FH Aachen is able to produce two-layer boards, which means that there will be a top and a bottom layer for conductor paths. Layer-to-layer connections are realized by vias. The available space for tracks is a limited resource and will require a proper placement of the components at a later stage.

Apart from the other components, which are going to be placed in an iterative manner to find the most suitable configuration, the connectors for the subsystem boards are defined by structural considerations. Together with the group that is responsible for the spacecraft structure it was decided to place every two connectors in line with a distance of 54mm from each other. There will be three lines of connectors; hence three subsystem boards will be accommodated. The horizontal dimensions for the placement of the connectors are fixed, whereas the vertical dimensions are slightly negotiable for the following flight model in order to react to physical requirements, such as the position of the centre of gravity.

Figure 2.1 illustrates the location of the CDHS board (referred to as main board) inside the spacecraft. Refer to figure 2.4 for the dimensions of the connector’s placements. Figure 2.5 helps to identify the orientation of the board inside the structure by means of a reference model of the satellite that was established in previous phases of the project.

Located on each corner of the board are the mounting holes, with a hole size of 1.3mm.

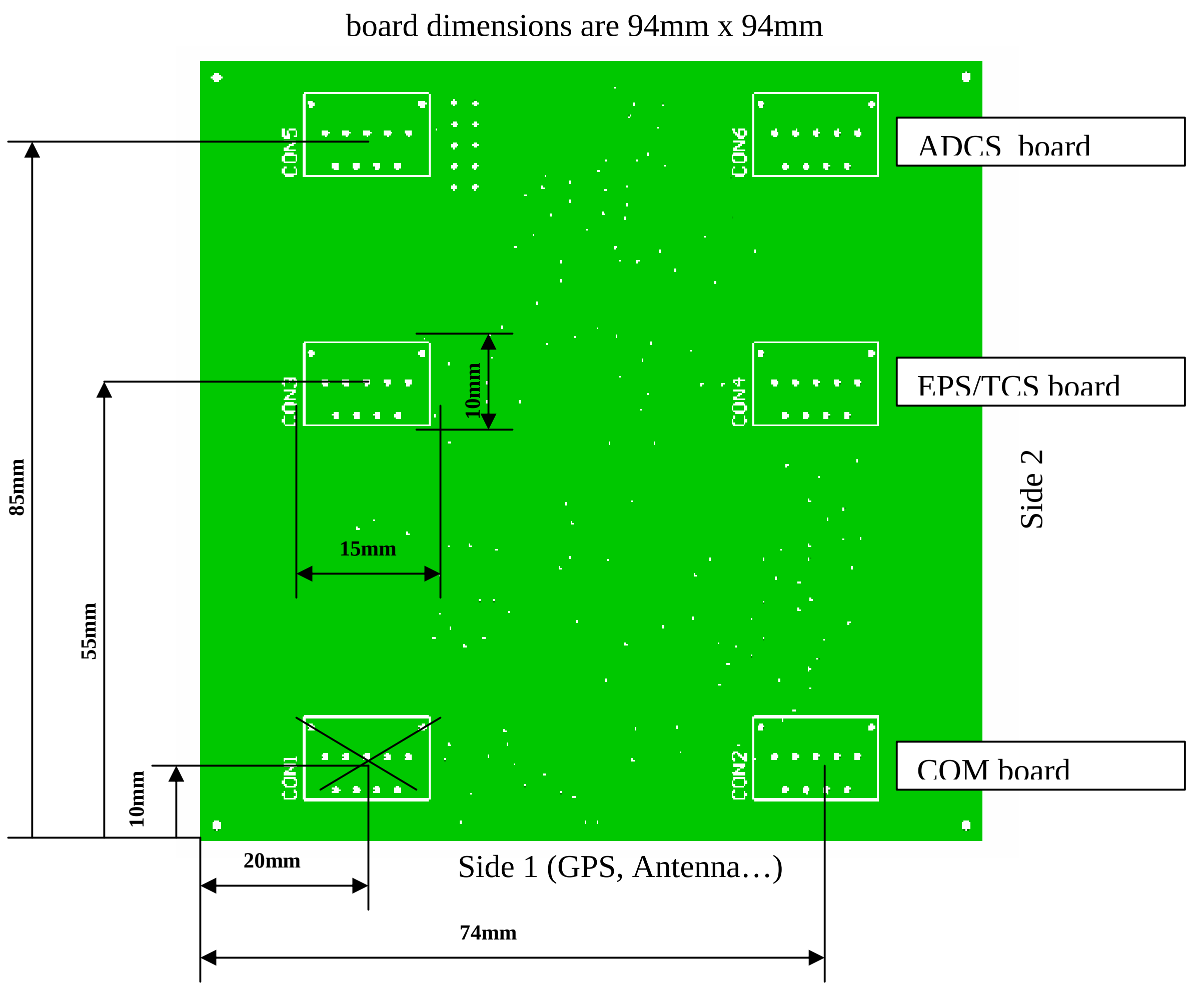


Figure 2.4: Placement of connectors

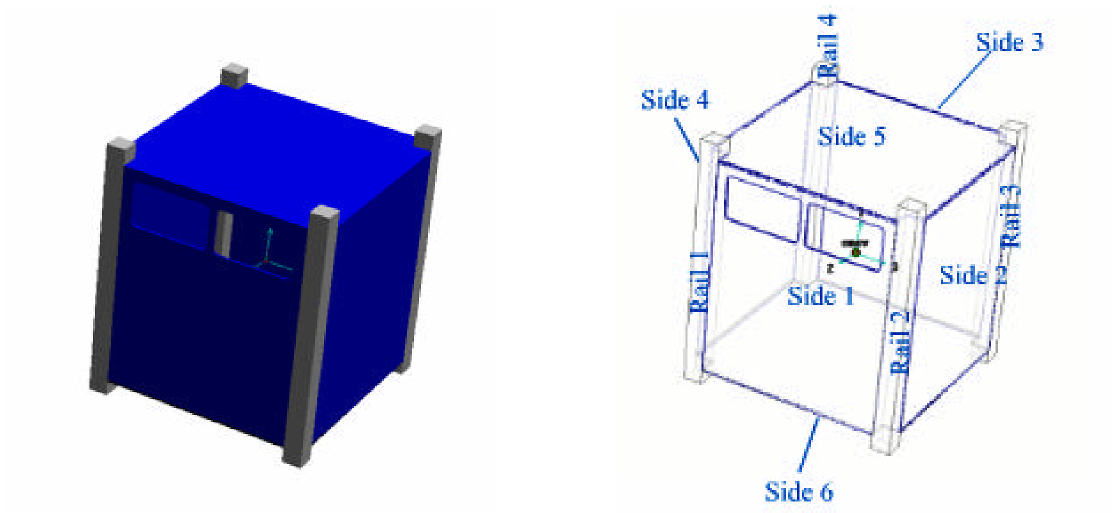


Figure 2.5: Reference model

Board Dimensions are 80mm x 65mm

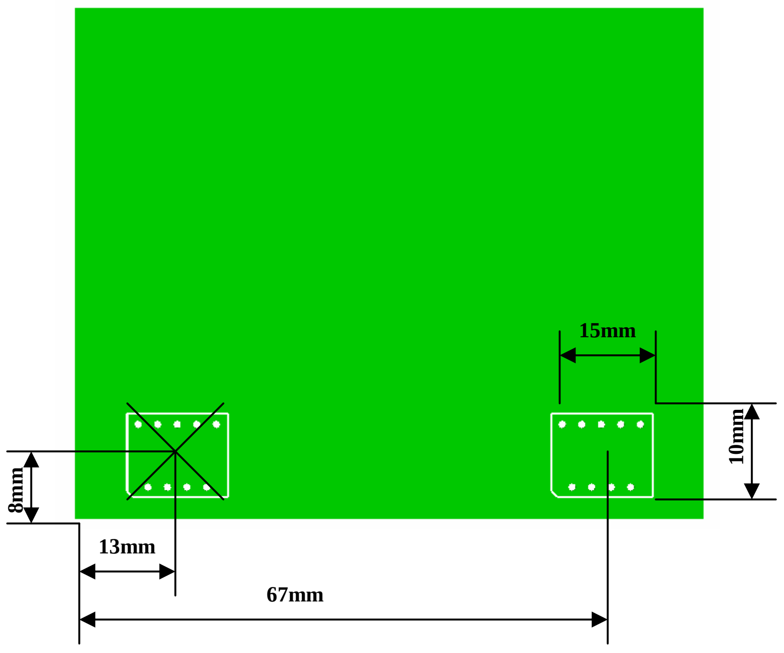


Figure 2.6: Corresponding subsystem board connector layout

2.3.2 Schematic Generation

The schematic layout process derives from the formerly defined functional layout. The functional layout gives the idea about the interconnections of the electrical devices. It is shown in figure 2.7 and 2.8.

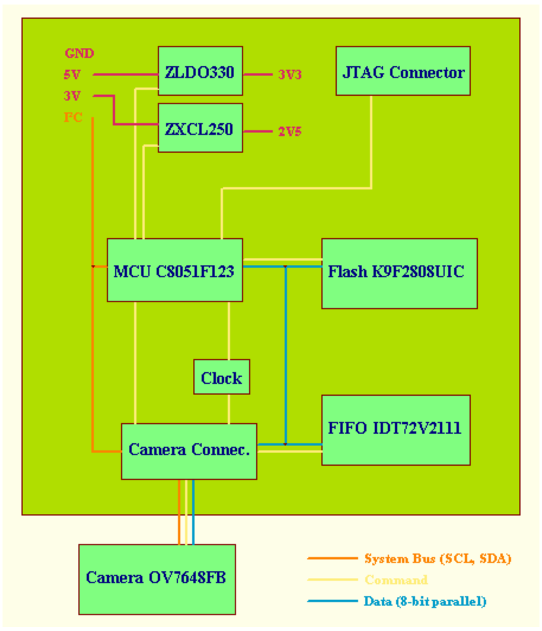


Figure 2.7: Main devices

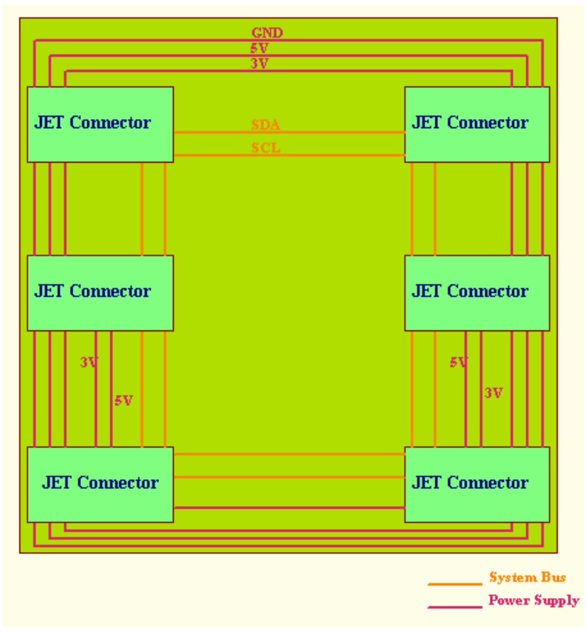


Figure 2.8: Connectors

It is a logical approach to divide the overall schematic into two sheets, one for the electrical devices that make up the CDHS and one sheet for the connectors. Please note that the figures reflect the logical interconnection rather than the physical position of the components.

Taking only into account the device-specific pin configurations, the representation of the physical components in a schematic is an abstract drawing. Still, to facilitate the recognition of the parts and for preliminary layout configurations, the size and shape of the models are kept similar to its real world devices. Each component model is furthermore attached with a specific footprint model that will be used during transition from schematic to layout.

For those components where there are no integrated models in the library (because they are too new or no standard components) models were manually created and are listed in the appendix for the sake of completeness.

Top Level Schematic

Since there are two sheets involved in this project, a schematic that is hierarchically placed one level above them is needed to identify the interfaces between them.

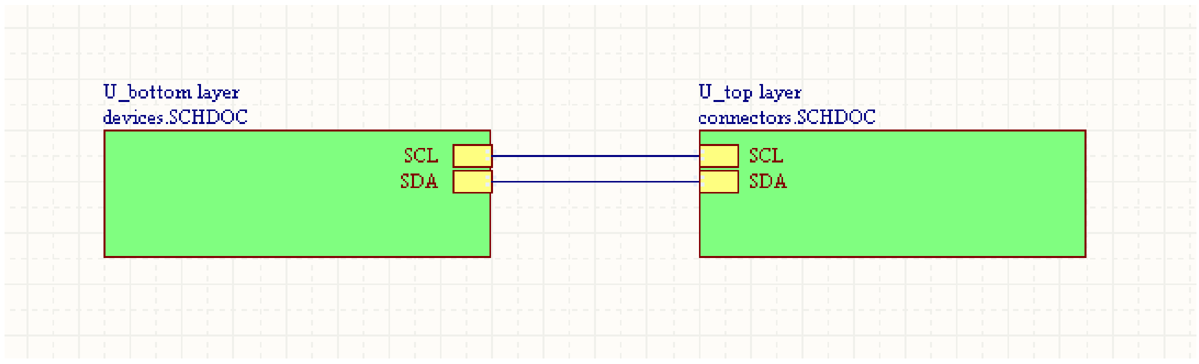


Figure 2.9: Top level schematic

The interfaces are as easy as can be seen in the figure above. Essentially only the two I²C wires connect the main board devices with the subsystem boards. Not shown are the power lines, such as the 3V, 5V, GND, etc. They are interconnected by default.

Connectors Schematic

Although not indicated by the drawing, the nets with same identifiers are also interconnected by default. That means for example, that all lines with a '5V' identifier belong to the '5V' power net, which is going to interface all of those pins. In the upper left corner the I²C wire is pulled high (to 5V) with resistors (2.2KOhm) according to its specification [7]. All connectors share the 5V, 3V, GND and I²C lines. The COM subsystem board and the EPS/TCS board share additional lines for power (3V_E and 5V_E: both are always connected to power, even in power-save mode) and two lines (A and B) for the emergency beacon signal.

Due to increased stiffness two-part connectors are chosen instead of a card edge connection, where the subsystem boards would be inserted in slots.

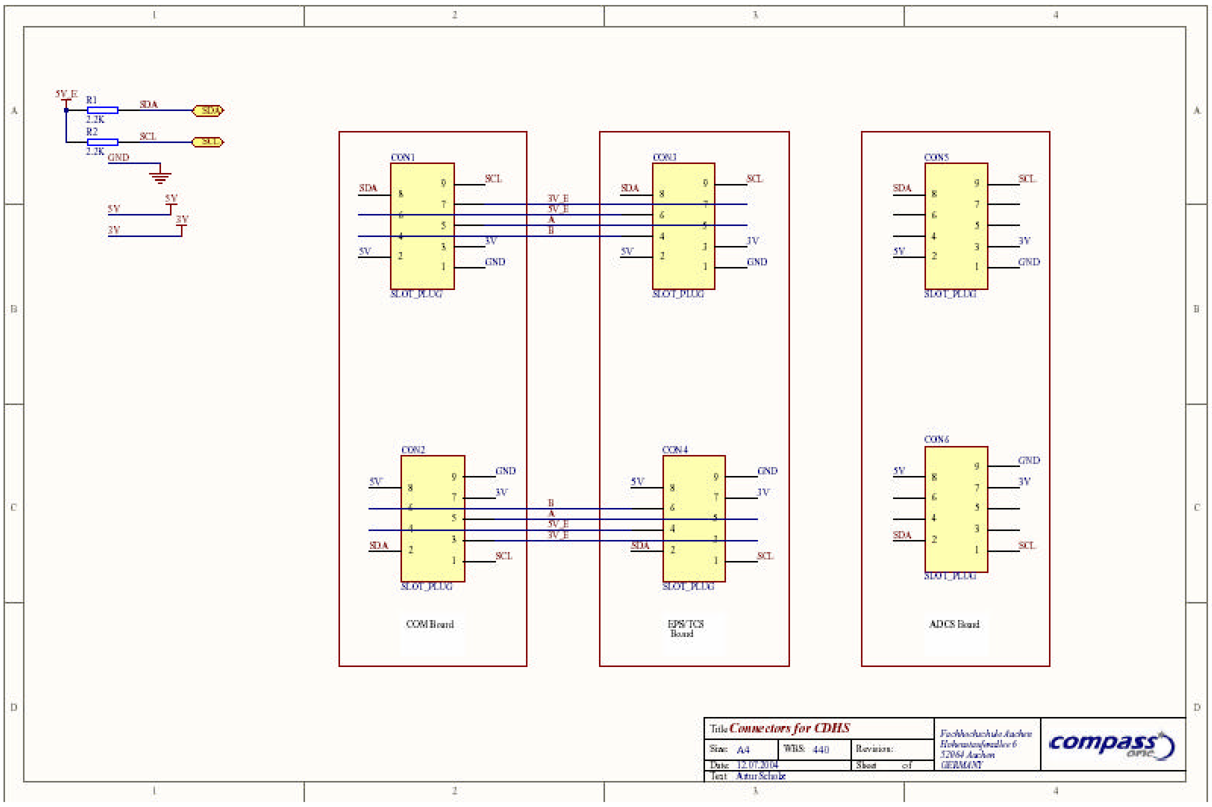


Figure 2.10: Connectors schematic

Devices Schematic
The more complex circuitry of the devices is given in figure 2.11.

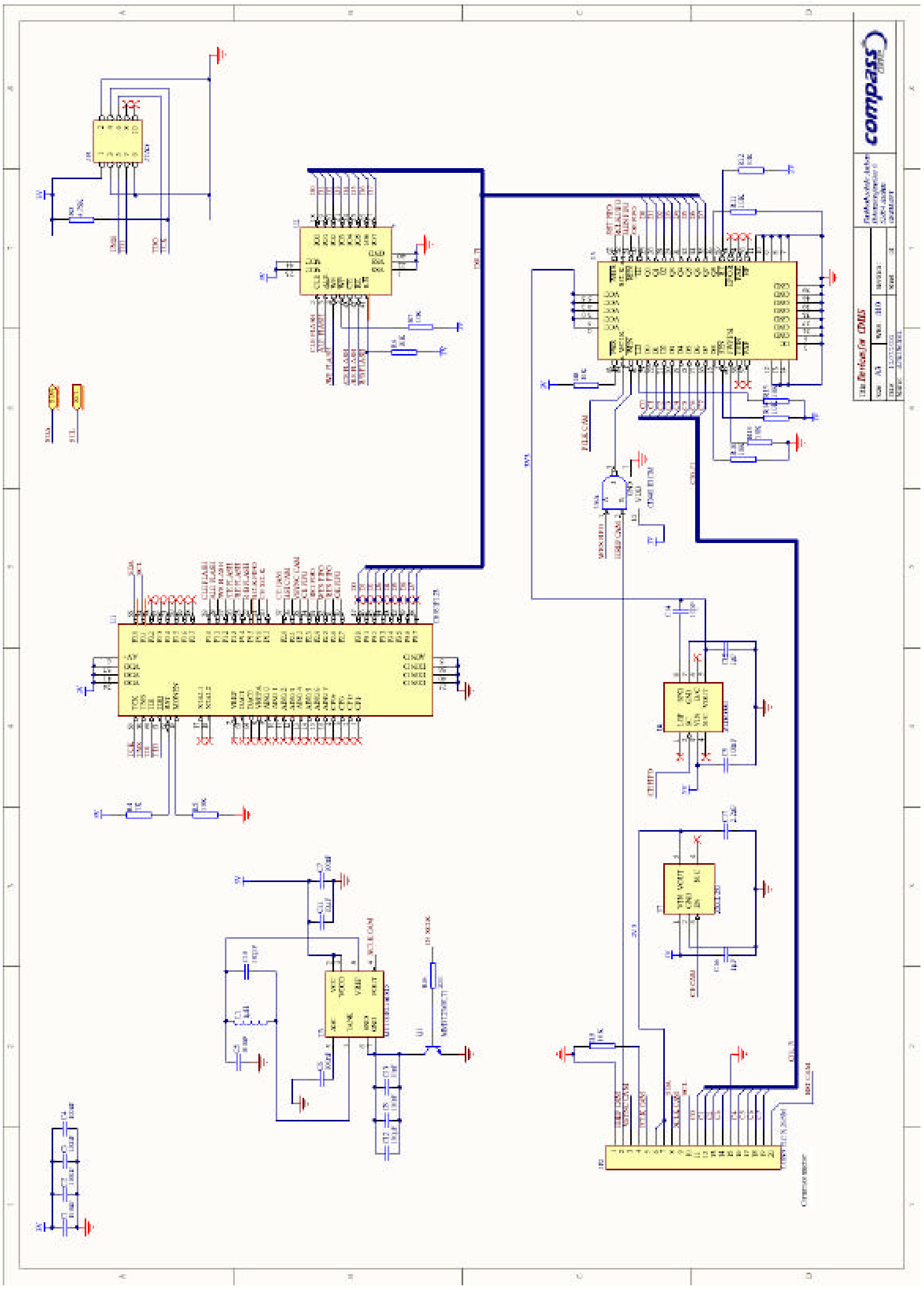


Figure 2.11: Devices schematic

The following section will describe the components in more detail and explain their interconnections. Some fast guidelines to understand the schematic are given below.

- The left upper corner accommodates the de-coupling capacitors that ensure a constant voltage level for the ICs because they compensate short fluctuations in the power supply. In the placement process, they have to be moved towards their corresponding ICs to work properly.
- In the right upper corner is the JTAG header that is used to program the MCU.
- The MCU is placed in the middle of the sheet and connects to virtually all other devices.
- As mass storage device, a Flash memory is chosen and can be seen in the middle right.
- The very left and the lower third of the sheet suit all the components that comprise to the payload interface unit, i.e. the clock generator, the camera connector, the voltage converters and the FIFO (from left to right).

The selection of the components has been performed at the previous phase B of the project. Included in its definition there was a brief description of each component. The next paragraphs will elaborate the components pin layout in more detail and lists the created nets and interconnections.

2.3.3 Components Description and Interconnections

MCU [8]

The internal 128Kbyte Flash memory for the program code is being programmed via the JTAG interface (TCK, TMS, TDI and TDO), which is accessible through a standard 10pin header JP1 (figure 2.13). The programming and debugging is done *in-system*, which means that the board will be programmed when all its components are assembled on it. This also allows in-circuit debugging to verify that the software works correctly with interfacing the MCU with the other devices.

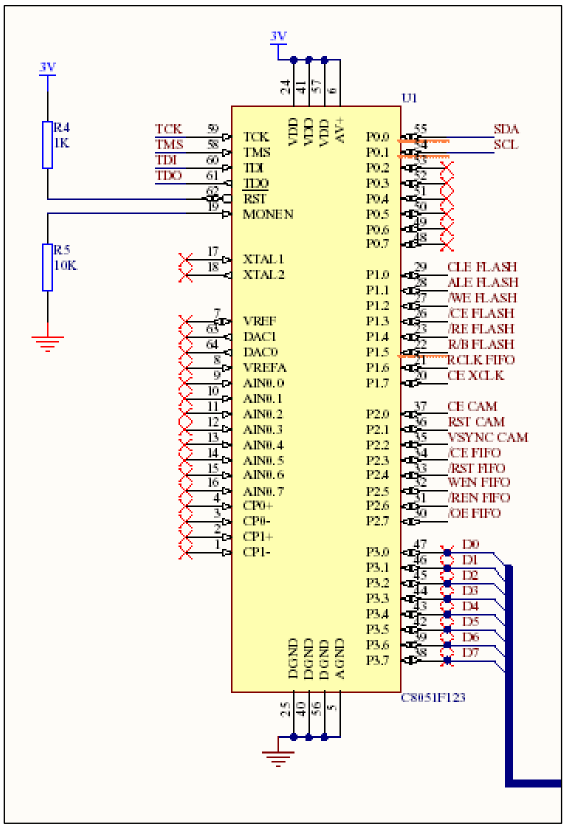


Figure 2.12: MCU schematic

The /RST pin is pulled high via resistor to disable external reset, which would be triggered if this pin is pulled low.

Pin MONEN is tied low via a resistor to disable the internal VDD monitor, which would force a reset if the supply voltage drops below a certain level. This may be a critical issue, but assurance of a constant voltage level will be the responsibility of the EPS subsystem instead of the CDHS.

The other pins on the left side are not used and left floating. They are for analog and digital conversion, a feature that is not used by the CDHS.

The upper and lower sides are connected to the power supply and ground.

On the right side the general purpose input/output pins (GPIO) are located which can be configured according to their purpose by software. For the sake of an easy configuration, the two first pins of port 0 are assigned to be the I²C interface (i.e. P0.0 and P0.1). The rest of port 0 is left open.

Port 1 and 2 control the functions of the other devices. A description of the acronyms is given in the table below. Please refer to the particular components pin description for more details on its function.

Pin	**Net Name**	**Interfaces with**	**Function**
P1.0	CLE FLASH	Flash	COMMAND LATCH ENABLE The CLE input controls the activating path for commands sent to the command register. When active high, commands are latched into the command register through the I/O ports on the rising edge of the /WE signal.
P1.1	ALE FLASH	Flash	ADDRESS LATCH ENABLE The ALE input controls the activating path for address to the internal address registers. Addresses are latched on the rising edge of /WE with ALE high.
P1.2	/WE FLASH	Flash	WRITE ENABLE The /WE input controls writes to the I/O port. Commands, address and data are latched on the rising edge of the /WE pulse.
P1.3	/CE FLASH	Flash	CHIP ENABLE The /CE input is the device selection control. Note that when the device is in the busy state, /CE high is ignored, and the device does not return to standby mode in program or erase operation.
P1.4	/RE FLASH	Flash	READ ENABLE The /RE input is the serial data-out control, and when active drives the data onto the I/O bus. Data is valid after the falling edge of /RE which also increments the internal column address counter by one.
P1.5	R/B FLASH	Flash	READY/BUSY OUTPUT The R/B output indicates the status of the device operation. When low, it indicates that a program, erase or random read operation is in process and returns to high state upon completion. It is an open drain output and does not float to high-z condition when the chip is deselected or when outputs are disabled.
P1.6	RCLK FIFO	FIFO	READ CLOCK When enabled by /REN, the rising edge of RCLK reads data from the FIFO memory and offsets from the programmable registers.

P1.7	CE XCLK	External Clock	CHIP ENABLE Triggers the transistor that is used as a switch to turn on the external clock oscillator.
P2.0	CE CAM	Camera	CHIP ENABLE CE high activates the voltage supply (2V5) for the camera module.
P2.1	RST CAM	Camera	RESET Resets the chip when active high.
P2.2	VSYNC CAM	Camera	VERTICAL SYNCRONIZATION VSYNC high indicates that a new frame is being transmitted.
P2.3	/CE FIFO	FIFO	CHIP ENABLE /CE low activates the voltage supply (3V3) for the FIFO.
P2.4	/RST FIFO	FIFO	RESET FIFO /RST low initializes the read and write pointers to zero and sets the output register to all zeroes.
P2.5	WEN FIFO	FIFO	WRITE ENABLE WEN enables WCLK for writing data into the FIFO memory and offset registers. Refer to payload interface unit for details.
P2.6	/REN FIFO	FIFO	READ ENABLE /REN enables RCLK for reading data from the FIFO memory and offset registers.
P2.7	/OE FIFO	FIFO	OUTPUT ENABLE /OE controls the output impedance of the pins connected to the data bus [D0-7]. When high, the output data bus goes into a high-impedance state.

The pins of port 3 are the data bus that connects the MCU, the Flash and the FIFO with each other. It is an eight bit bus system that is used to transfer the image data stored in the FIFO to the Flash and to send command and address information from MCU to Flash.

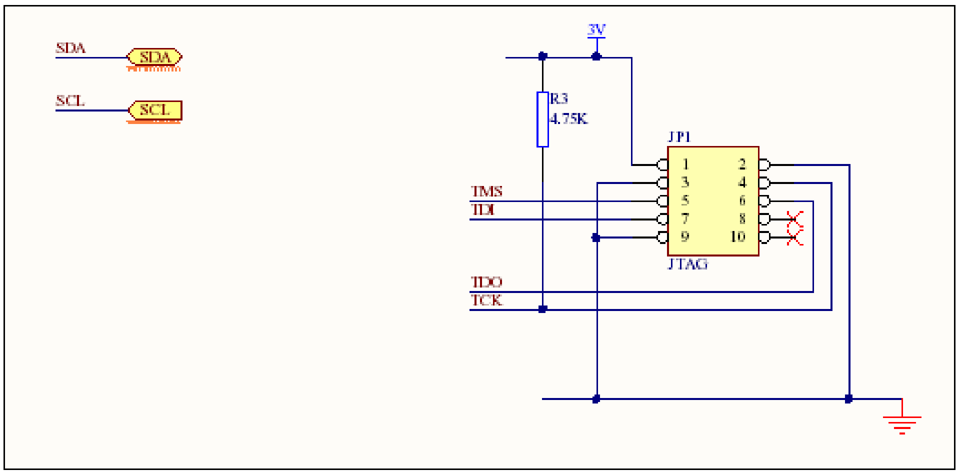


Figure 2.13: JTAG schematic

Mass Storage Flash Memory

The Flash device from Samsung [9] is the non-volatile memory solution that stores the images taken by the camera module together with the housekeeping data from the various subsystems and other system information. The chip is offered in 16Mx8bit which yields to a total capacity of 128Mbit. The I/O pins serve as the ports for address and data input/output as well as command input. The on-chip write control automates all program and erase functions. It has an extended reliability of 100K program/erase cycles, which makes it suitable for this write-intensive application.

The I/O pins are used to input command, address and data and to output data during read operations. The I/O pins float to high-z when the chip is deselected or when the outputs are disabled.

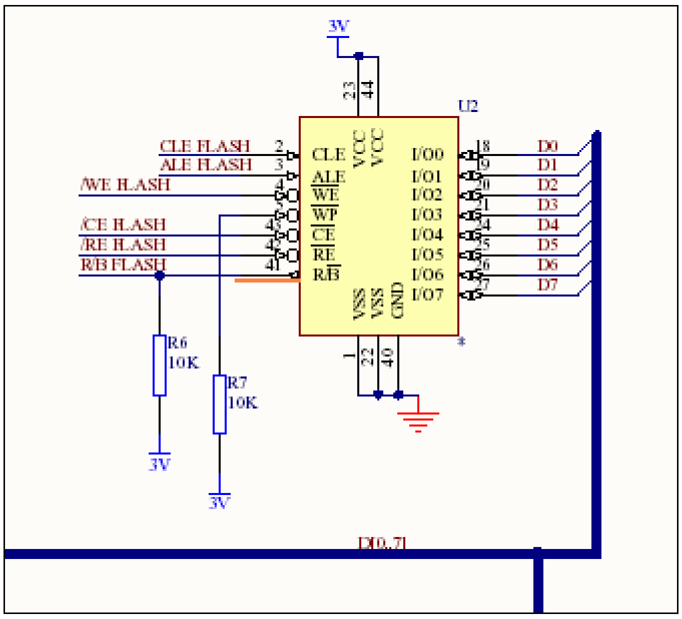


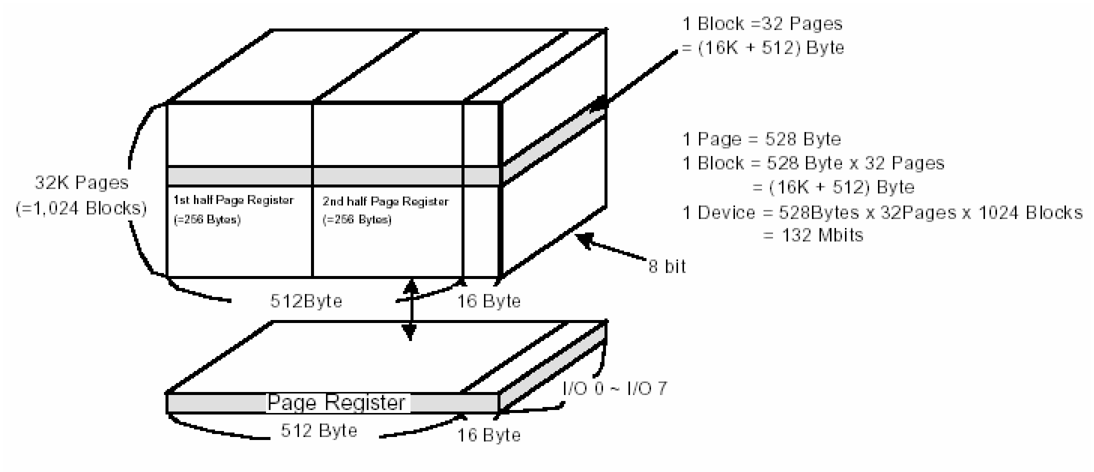
Figure 2.14: Flash schematic

It is connected through the 8 bit data bus D[0..7] with the MCU and the FIFO device. The ‘R/B FLASH’ wire is tied high via a resistor of 10kOhm because it has an open drain output that goes low when the chip is busy.

For explanation of how the chip works, figure 2.15 illustrates the internal memory organization of the Flash memory. After a read command has been input, there are three more cycles required to address a singly byte. The first cycle with one address byte (A0...A7) is for horizontal position. By this, bytes 0 to 255 are addressable. To address the following bytes 256 to 511 a different command code is used.

The second two cycles address the vertical position, i.e. the page number. There are 32K (that is 32768 or 2^{15}) pages, hence requiring exactly 15bit for addressing (A9…A23).

Summarizing, the Flash is controlled by the MCU in terms of commands and address information and gets most of its content from the FIFO, which is interconnected to the data bus and holds the images from the payload.



	I/O 0	I/O 1	I/O 2	I/O 3	I/O 4	I/O 5	I/O 6	I/O 7	
1st Cycle	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	Column Address
2nd Cycle	A_9	A_{10}	A_{11}	A_{12}	A_{13}	A_{14}	A_{15}	A_{16}	Row Address (Page Address)
3rd Cycle	A_{17}	A_{18}	A_{19}	A_{20}	A_{21}	A_{22}	A_{23}	L*	

Figure 2.15: Flash internal memory configuration

Payload Interface Unit

Involving the majority of ICs in the CDHS circuitry, the payload interface unit is more complex to handle. The reasons for the big number of involved devices are the special voltage requirements for the camera and the FIFO, the triggering of the FIFO [10] via a NAND [11] gate and the need for an external clock oscillator circuit to drive the cameras clock signal. This is due to the very fast output of the data from the camera module, which is too fast for the Flash to be handled, thus device that is able to fetch and buffer an image becomes compulsory. This is done by the FIFO.

The signal flow starts at the camera connector. As can be seen, the connector (which is going to be a 20pin connector) is connected to GND (pin 1 and 15) and to a 2V5 power supply from the voltage regulator. Pin 4 is the power down switch and is tied to GND to disable it. The camera will be disconnected from power supply when not operating.

VSYNC is usually low and indicates the start of a new frame when it shortly peaks to high. Refer to figure 2.17.

HREF CAM when high shows that data bytes are available at the output bus C[0…7]. It remains high for all data bytes of a horizontal row. Refer to figure 2.18.

The camera module is programmable via I²C interface.

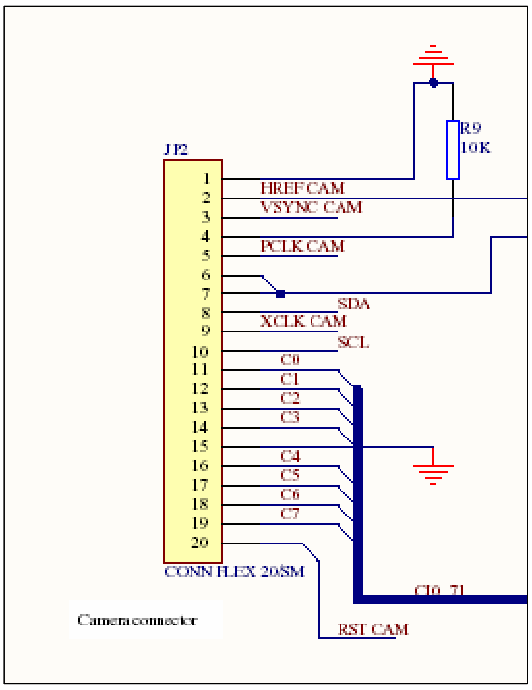


Figure 2.16: Camera connector schematic

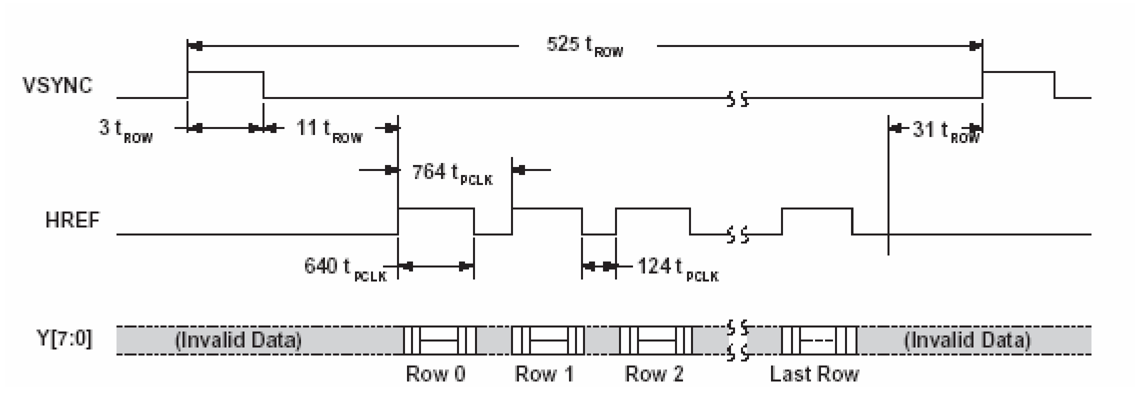


Figure 2.17: VGA timing diagram

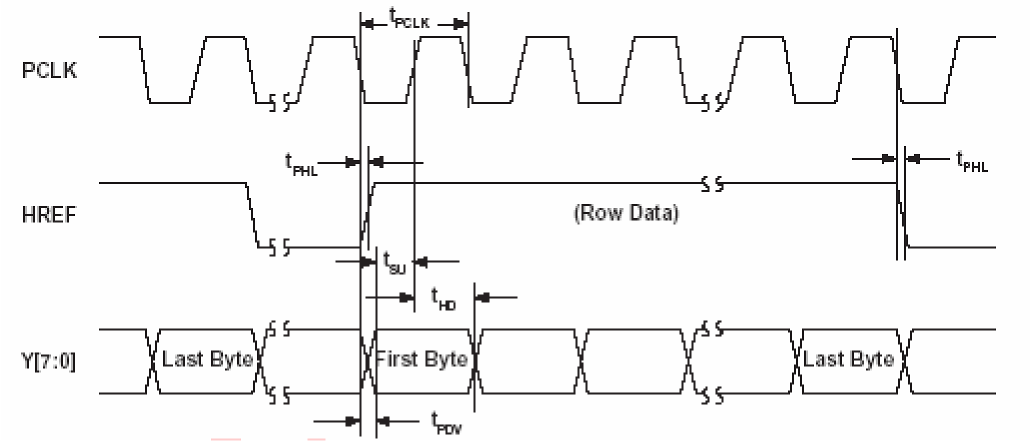


Figure 2.18: Row output timing diagram

The camera will take VGA images and operates therefore with 30fps. Thus, a single frame is obtained within 1/30 seconds. From the values in the above figures t_{row} and t_{PCLK} can be calculated:

$$1/30s = 525 * t_{row}$$
$$t_{row} \approx 63ms$$

$$t_{row} = 640 * t_{PCLK}$$
$$t_{PCLK} = 99.2ns \qquad (1)$$

The XCLK CAM pin is the input for the clock signal, which is generated externally as described in the following.

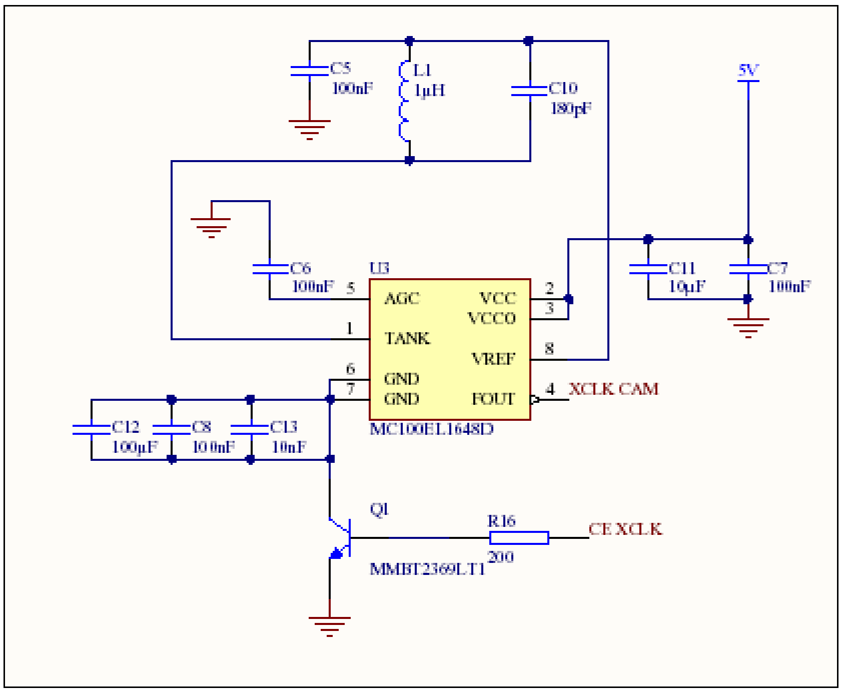


Figure 2.19: The external clock circuitry

The above shown circuitry outputs a clock signal at the XCLK CAM line when the CE XCLK is high. The frequency of the clock signal is set by the following equation:

$$f_0 = \frac{1}{2p\sqrt{L_T * C_T}}$$

With the chosen values of L_T = L1 (1µH) and C_T = C10 (180pF), this yields to a resulting frequency of

$$\underline{f_0 \,\tilde{}\; 11.863 \text{ MHz}} \qquad (2)$$

This is in the lower range of the allowed frequencies (minimum is 10MHz) to drive the camera. All other conductor and resistor dimensions are specified by the chip's documentation [12]. The duty cycle of this device is specified with 50%.Note that the device is connected to the 5V power supply.

Below are the two voltage regulators, the ZXCL250 and the ZLDO330 from Zetex [13], [14]. Both require several capacitors to work properly. They are connected to the common ground plane and both are selectable via an input pin. The regulator for 2.5V is activated with CE CAM high, whereas the 3.3V regulator needs a low on /CE FIFO to run.

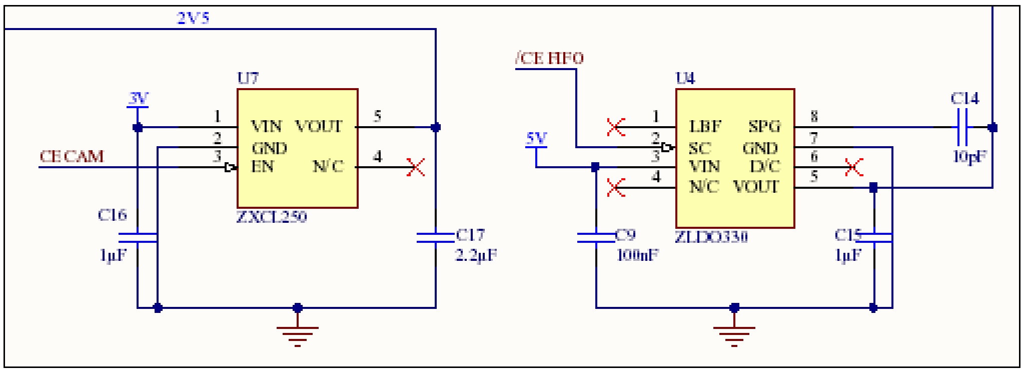


Figure 2.20: The voltage regulators for FIFO and camera

The 3.3V regulator supplies the FIFO with power and thereby controls if it is on or off. This is why the wire is labeled CHIP ENABLE FIFO (/CE FIFO).

The device with the most interconnections is the FIFO and is shown below.

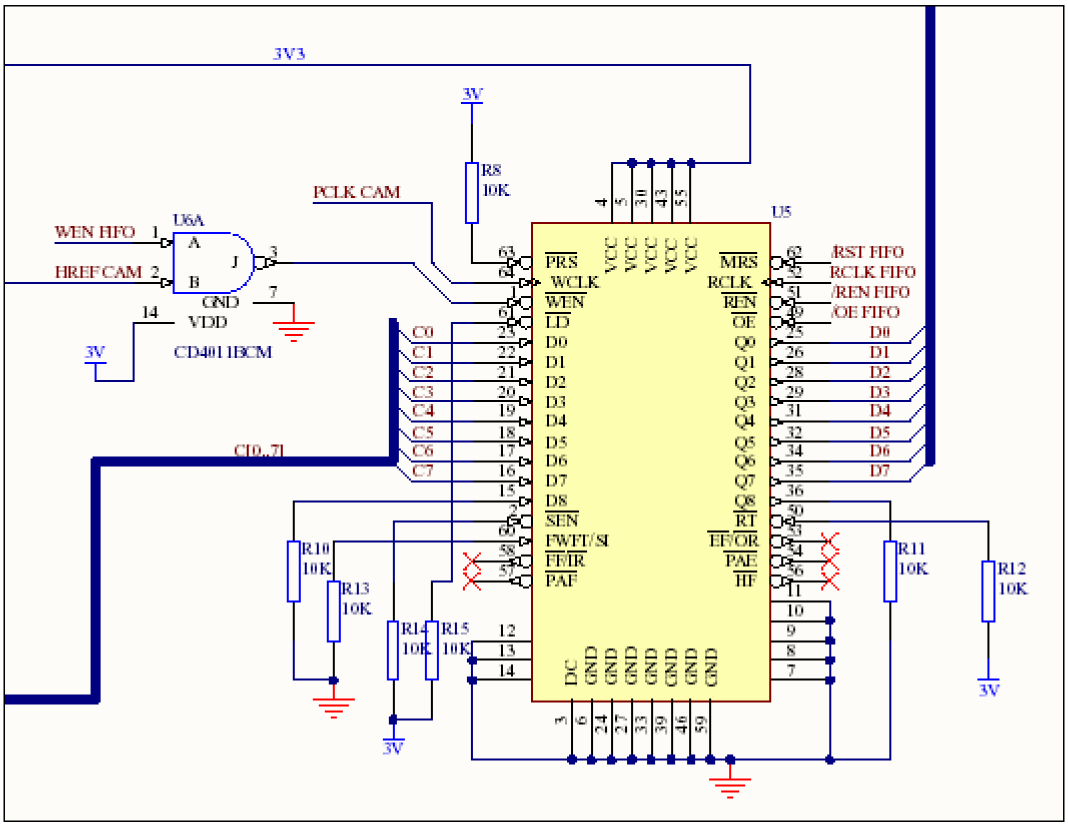


Figure 2.21: The FIFO interconnections

Again, top and bottom sides are connected to the 3V3 supply from the regulator and to the common ground, respectively. The /PRS (PARTIAL RESET) is deactivated by pulling it high via a resistor. The input for the WCLK (WRITE CLOCK) comes from the camera, which outputs a PCLK (pixel clock) of

$$1/t_{PCLK} \sim 10\text{MHz} \qquad (3)$$

Refer to equation (1) for t_{PCLK} value.

The FIFO can be written to when the /WEN pin is low, which is realized through a NAND gate that goes low only when there is a high on the WEN FIFO line from the MCU and a high on the HREF line from the camera. The NAND is supplied with 3Volt constantly.

The 8bit output from the camera is transferred to the FIFO input ports via the data bus C[0…7]. The ninth bit (D8 as well as Q8) is disabled and pulled to ground. The same is true for the other features which are not being used. They are disabled by either tying them high or low, according to the specification.

A reset of the FIFO can be triggered by driving a logic one on the /RST FIFO line. The RCLK FIFO (READ CLOCK) dictates the speed data is read out and put on the data bus D[0…7] when simultaneously the /REN FIFO is low.

Finally the /OE FIFO line is used to set the data bus in a high impedance state when pulled high. Doing so will allow the MCU to program the Flash via the data bus and without interfering the FIFO data.

2.3.4 Transition from Schematic to PCB

To transfer a logical drawing as the schematic is, into a physical one, the symbol and physical properties must be linked to each other. Part terminations defined in the symbol are assigned pin numbers and interconnections between them are assigned net names. The same pin numbers are used in the physical part definition and, together with the interconnection data from the schematic, establish a net list that will be used to define how the parts placed on the layout are to be interconnected. These interconnections will be displayed as a so-called *rats nest* and later on translated into conductor paths during the routing activity.

As said, the basic information for a transition of a schematic to a PCB derives from the libraries that combine the symbol and the physical properties (in form of a footprint) of each part together. For some parts however, there were no standard footprints in the library and they had to be created manually. They are included in the appendix for reference. A table with all parts to be mounted on the board can be found in the appendix as well. It is the so-called Bill of Material (BOM).

All parts are surface mount devices (SMD). Only the connector for the JTAG and the subsystem board connectors are through hole. The latter require more stiffness, which would not be provided by SMD.

2.3.5 Part Placement

The placement of the devices is one of the most critical layout activities as it has a significant influence on how (and if at all) a board is completely routable. The mentioned rats nest serves as a good identification of too jam-packed areas, which should be avoided.

The first major decision for each part was whether to place it on the top or on the bottom side of the PCB. This was not a question for the subsystem board connectors however as there location is defined by integration issues discussed in the mechanical definition part.

Ultimately, due to expected better shielding effects the majority of devices (all ICs and many passive parts as well) are located on the top layer.

Also in terms of thermal control it should be the best solution, as the satellite will generally get too cold instead of too hot (please refer to the results from the preliminary thermal analysis). Each component radiates a small amount of heat. When components face each other, the hotter one will reduce its temperature by radiating it to the colder one.

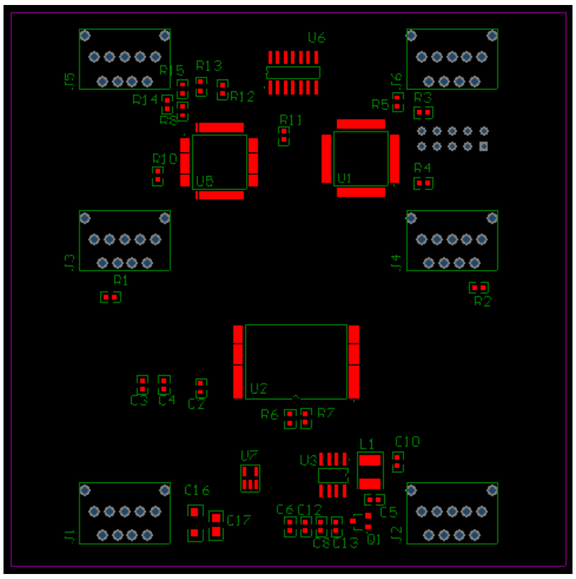


Figure 2.22: Placement on top side

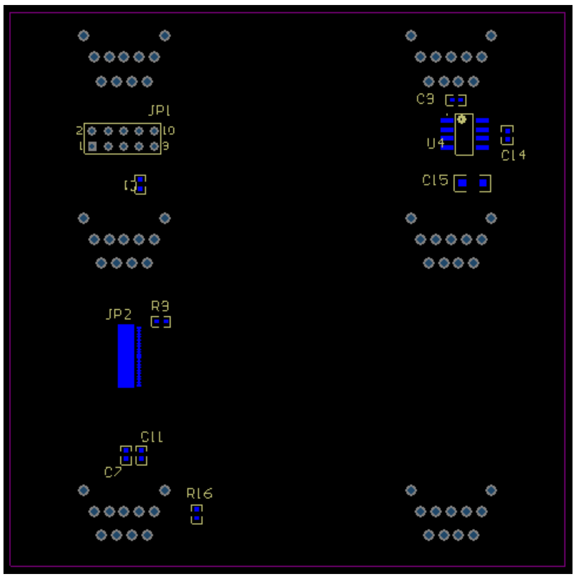


Figure 2.23: Placement on bottom side

The above figures display the chosen component placement for the engineering model of the CDHS board. There is still room for changes and most likely the subsystem board connectors will have to be moved a little bit for the next model due to structural reasons. This will however not be a big problem. The shown configuration is the result of an iterative process and by no means claims to be the one and only solution.

2.3.6 Routing

The placement and routing activities are deeply interwoven with each other and changes on one affect the other notably.

Initially the autorouter has to do the routing job. But before that some specific tracks were done by hand. Those were the power supply lines and the system bus lines, which run in parallel on both sides and interconnect with each connector. Although the autorouter finished the entire board, the result was not satisfying from the aspect of maintainability and clearness of the board. A test board was manufactured, too. After that, each and every track was routed manually. The results can be seen in figure 2.24 and 2.25.

Since all prototype boards are manufactured in-house at the workshop of the FH Aachen, the following design rules for PCB layout were taken into account:

- minimum track width: 8mil
- minimum clearance: 8mil
- via style: hole size of 20mil, shape size 40mil

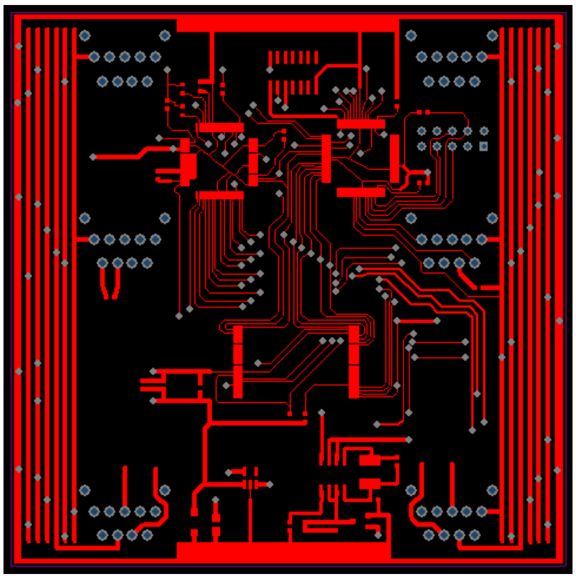


Figure 2.24: Routing of top layer

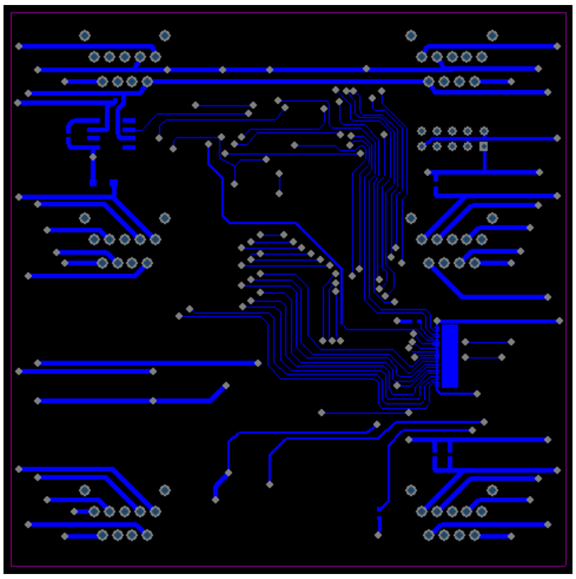


Figure 2.25: Routing of bottom layer

2.3.7 Design Verification

The routed board was checked with the design check option of the Protel2004 software. Short-circuits and other mistakes were not detected. This is also due to the online check, that is active all the time and avoids the introduction of errors already in the design activity.

Thus the hardware design part is theoretically error-free.

2.4 Development Activities

As soon as the PCB design has finished and the design is verified, the files can be sent to the fabricator for production. It would also be possible to produce a self-made board but this option was not considerable as the FH Aachen has a dedicated laboratory for that. The final model is most likely being produced externally to ensure best possible quality.

2.4.1 Board Fabrication

The PCB manufacturing itself is the first step in the development activity. As mentioned already, the FH Aachen has facilities for board production. This includes a milling machine that can realize a minimum track width of 8mil, which is at the same time the smallest footprint that appears on the CDHS board. The workshop also provides the copper plates and cuts it according to the given dimensions.

It is a very convenient process, as the only thing to be done was to send the design files (saved as Protel 99) via email to the workshop. The result is shown in figure 2.26.

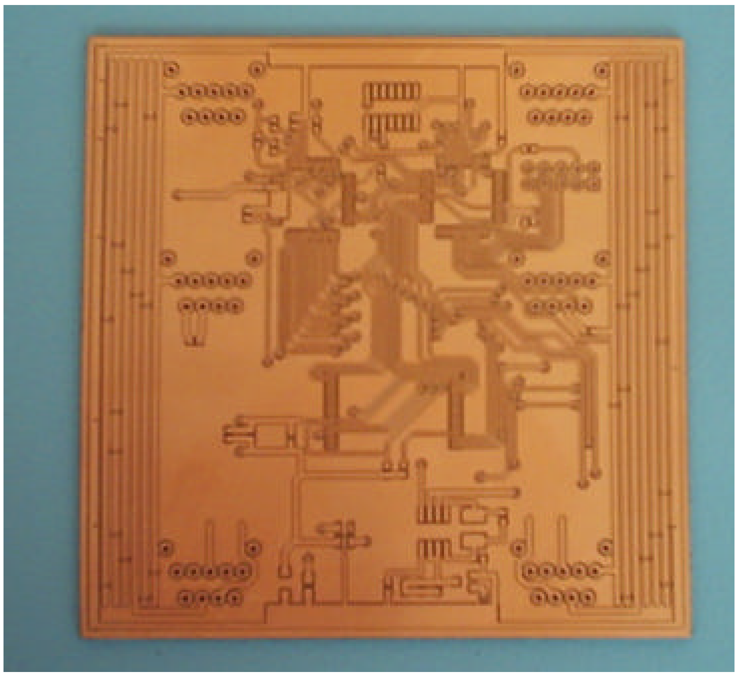


Figure 2.26: The engineering model of the CDHS board

2.4.2 Soldering

The soldering activity is a more delicate one, because it needs a person with good experience in that field. It is going to be done by one of my fellow students. The necessary tools and equipment are all available at the electronic workshop.

2.4.3 Inspection and Testing

Inspection of the soldered components has to be conducted to check for any type of failures, such as broken connections, short cuts, etc. Most of them will be identifiable with the naked eye. Others might be too small and may only be detectable with help of a voltmeter and other instruments.

Next to the necessary hardware test that are specified in the CubeSat documents, the board as a whole will have to endure additional tests, set by the Compass-1 group.

The results of the testing will be documented separately.

3. Software

3.1 Overview

The flight software is the core item of the CDHS. It brings the devices ‘to life’ and utilizes their features. Software programming for the CDHS refers to the development of the program code that is being stored in the MCU.

The program code has to accommodate the various mission modes of the spacecraft as illustrated in figure 3.1. During boot, the MCU initializes the other devices and goes into a pre-defined state. Then the nominal mode is active, in which the CDHS interacts with the COM subsystem, the Payload and the ADCS. When in power save mode, the CDHS is off. No code will be executed at this time. As soon as the EPS switches from power save into normal mode again, the boot procedure is passed through again.

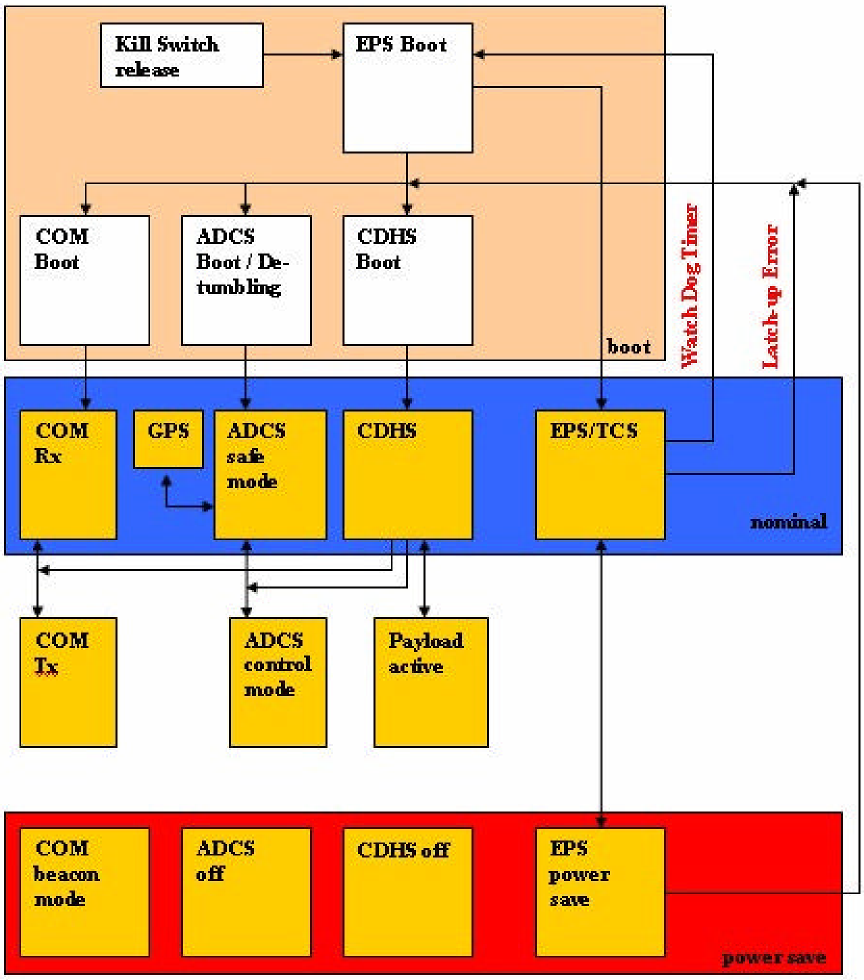


Figure 3.1: The mission modes

Rather than using a real-time operating system a designated kernel is going to be programmed that will keep the resulting code compact and small in size. Also it can be furnished much better to the hardware on the lowest level.

3.2 Integrated Development Environment (IDE)

For the chosen MCU (the C8051F123 from Silicon Laboratories) there exists a development kit that comprises of a target board and a complete solution for software development using a Windows PC. The target board is in-system programmable (alike the CDHS will be) and can be used to verify and debug code modules. Essentially, the integrated development environment (IDE) is a complete, standalone software program that provides all the tools needed for developing and testing. Quoting the product package [15]:

“A full-version Keil A51 macro assembler and BL51 banking linker are included with the development kit. Also an evaluation version of the Keil C51 ‘C’ compiler is included. The evaluation version of the C51 compiler is the same as the full professional version except code size is limited to 4K bytes and the floating point library is not included.

The Cx51 Optimizing ‘C’ Compiler from Keil Software is a complete implementation of the American National Standards Institute (ANSI) standard for the ‘C’ language. Cx51 is not a universal ‘C’ compiler adapted for the 8051 target. It is a ground-up implementation dedicated to generating extremely fast and compact code for the 8051 microprocessor. The ‘C’ language on its own is not capable of performing operations (such as input and output) that would normally require intervention from the operating system. Instead, these capabilities are provided as part of the standard library. Because these functions are separate from the language itself, ‘C’ is especially suited for producing code that is portable across a wide number of platforms. Since Cx51 is a cross compiler, some aspects of the ‘C’ programming language and standard libraries are altered or enhanced to address the peculiarities of an embedded target processor.”

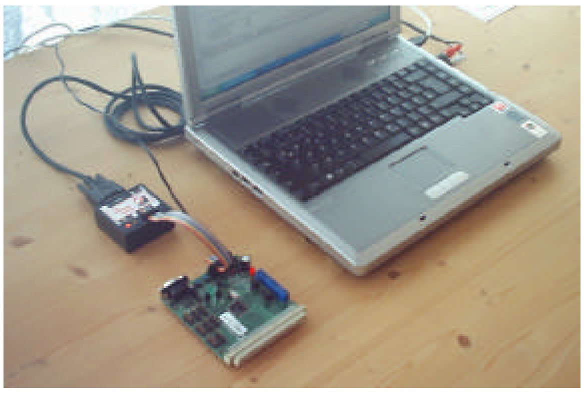


Figure 3.2: The target board assembly

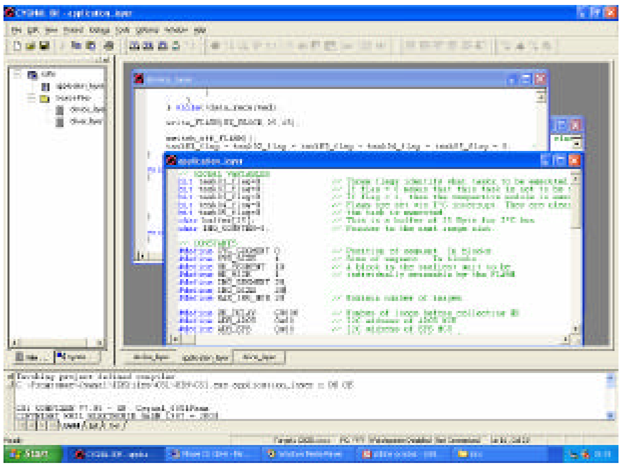


Figure 3.3: An IDE screenshot

3.3 Design and Development Activities

Figure 3.4 depictures the process that will eventually lead to the complete program code of the CDHS. To start with, the environmental parameters of the CDHS must be defined. This includes the idea on how the software shall be structured, i.e. the number and purpose of hierarchic levels. The flowchart design furthermore needs information on how the memory is organized and structured. Finally, the system bus design will provide with details on how to exchange commands and data with other subsystems.

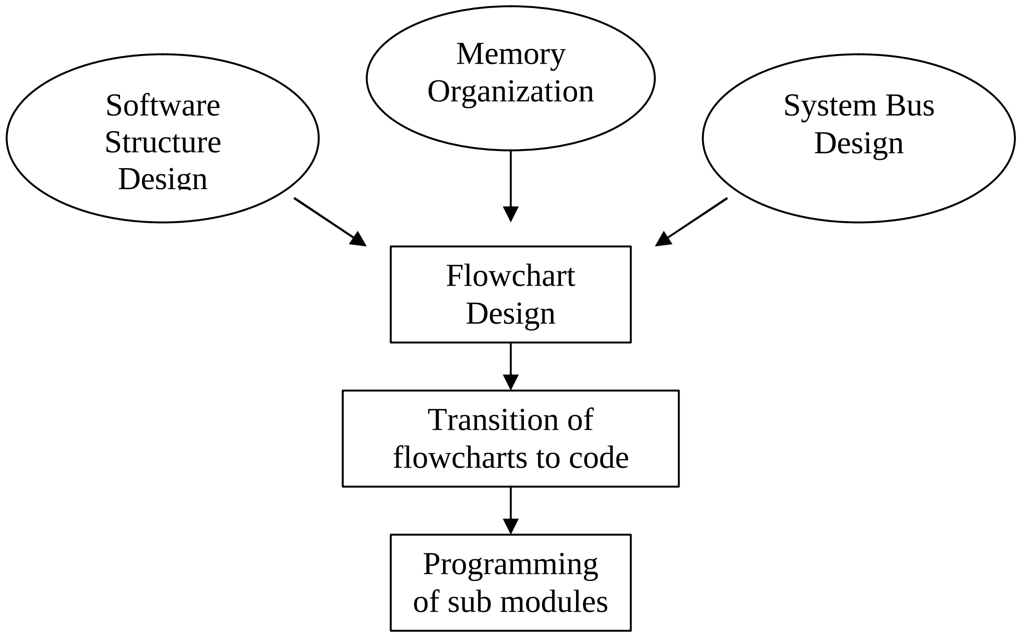

Software Structure Design
Memory Organization
System Bus Design
Flowchart Design
Transition of flowcharts to code
Programming of sub modules

Figure 3.4: The software design and development activities

3.3.1 Software Structure

The design of the program code is carried out in a strict top-down approach. Intentionally, not much attention had to be wasted on the technical details of the devices implemented on the board, but rather the software functions and tasks were put in the center of concerns.

By doing so, an abstractive and therefore transferable flight software code can be realized. Figure 3.5 illustrates the software layer design for the flight software. The application layer covers the main program code written in ‘C’. Also all other levels are in ‘C’ but they make us of the integrated standard library which is hardware specific to the 8051. The device interface layer is the middle layer between the application layer and the low level drivers and acts as logical representation of the hardware. Thus, changing the hardware (respectively parts of it) can be done, based on an appropriate modification of this layer’s modules.

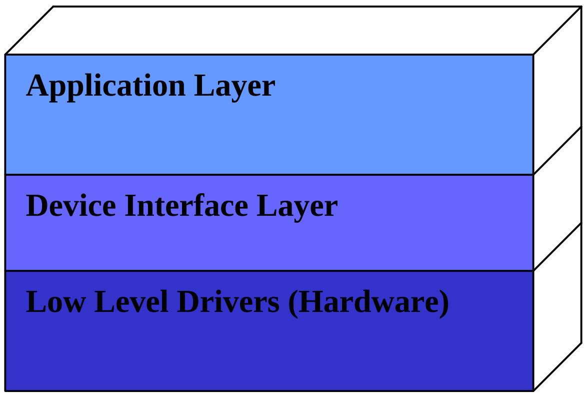


Figure 3.5: The software layer design

3.3.2 Memory Organization

There are three distinguishable types of data that are going to be stored in the mass memory unit of the CDHS. Those are the payload data (images), the housekeeping data (information from various sensors and subsystems) and the system data (status information of the CDHS). Even so the system information will be sent to ground together with the housekeeping data and as such could be referred to as the 'housekeeping data' of the CDHS it has a more important objective then just giving status information of the system. In fact, the system information holds such essential values as the image pointer for example, which indicates the next free slot for when a new image has to be stored. More information on that can be found in the documented program code.

The logical memory configuration, that means the way the memory is being addressed in software, is based on the hardware solutions. The bus width of the device has a big influence on how a single byte can be addressed. The Compass-1 MCU is an 8051 derivate and as such it uses 8 bit bus structure. Thus, only eight lines (8 bit allows 256 combinations) are available for addressing the Flash memory, which in this case is 16Mbyte. The solution here is to write to the bus several times sequentially (in cycles), latching it into the address buffer of the device (which is done automatically) and thus increasing the number of addressable bytes.

The mechanism explained above works for the Flash device that is being implemented as the mass storage memory on the CDHS board. It uses three cycles for address latching. The first eight bytes refer to the byte position in the page, whereas the following two cycles determine the block (as in figure 2.15).

Figure 3.6 illustrates the configuration, which is a layer stack of blocks, with each block made of by a number of pages. And each page is made up by a number of bytes.

In figure 3.7 a closer look is taken to the configuration of the image slots. It can also be seen how the pages and blocks approach contribute to the clearness of the memory arrangement.

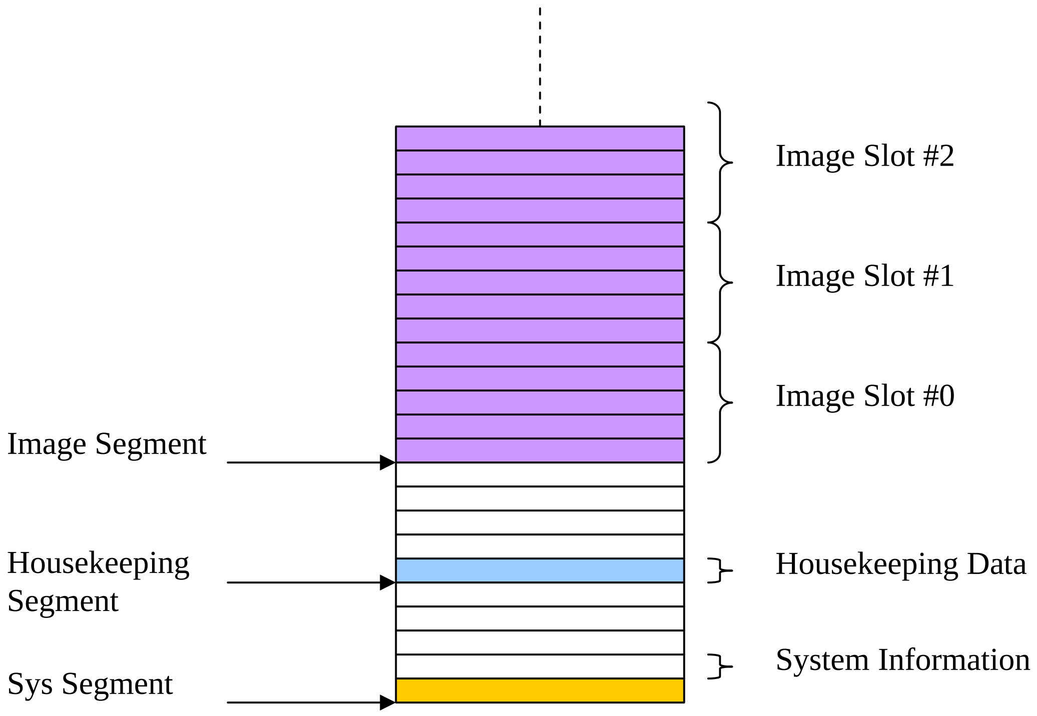


Figure 3.6: Memory organization

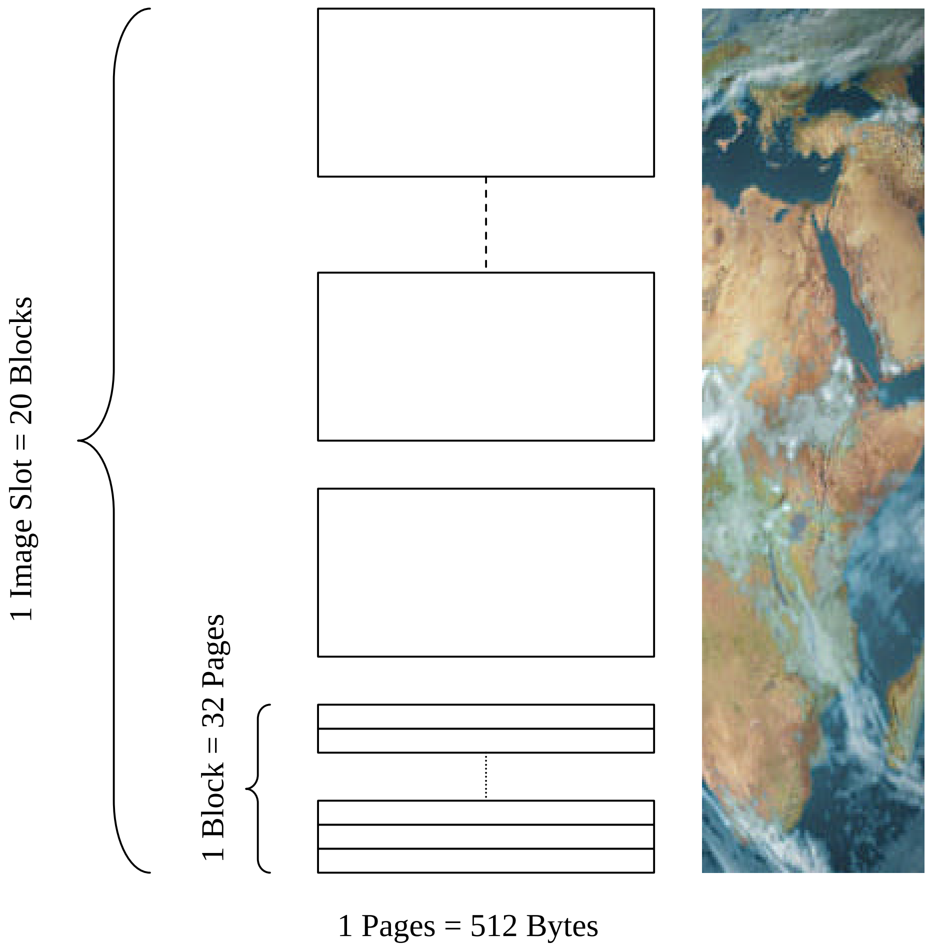


Figure 3.7: The image slot configuration

3.3.3 The System Bus: Command and Data Traffic

Subsystems communicate with each other via the system bus. The chosen bus concept is the I²C bus from Philips [7]. It is a multi-master bus, which means that every bus participant can be a Master, just by initiating a transfer. The bus uses only two lines, one for data and one for clock to synchronize transfers. The transfer rate is gradually up to 100kbit/s.

All subsystems that require the exchange of data/commands via the system bus (those are the ADCS, COM, CDHS and EPS) have an MCU that supports the I²C standard by hardware connected to the system bus. The initial idea to connect the majority of devices to the system bus (such as the various sensors) was eventually discarded, since it would increase potential error sources.

This is the major drawback for that bus. Since all members are wired-AND connected, a failure where one device pulls the bus lines to ground permanently will demolish bus communication. So, other devices than the subsystem MCU's shall be avoided to be plugged to the bus to minimize this risk.

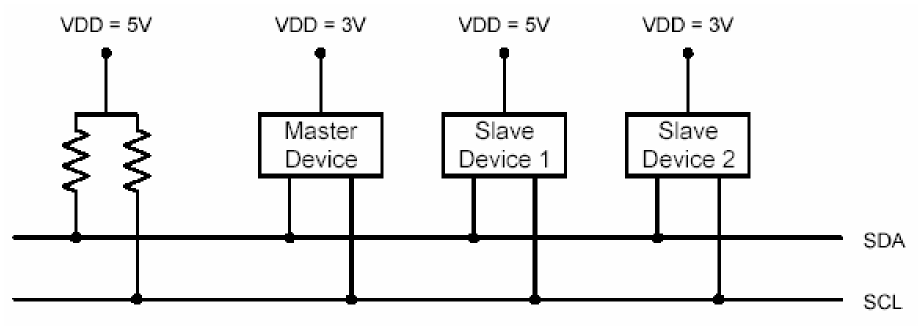


Figure 3.8: I²C is a multi-master bus that supports devices running at different voltage supply levels.

Communication Modes

The chosen I²C bus supports a total of four modes of communication. For convenience the CDHS will be restricted to make use of two modes only. Those are Master-Transmitter mode and the Slave-Receiver mode.

- Master-Transmitter: An MCU initiates a transfer by writing on the bus. It automatically becomes the Master for this transfer and transmits the data in packets. After each data packet sent, it waits for the Slave to acknowledge it.
- Slave-Receiver: Any component that is addressed by another device via the bus will become the Slave for this transfer and receives the data. Each single data packet has to be acknowledged by the Slave, otherwise the transfer terminates.

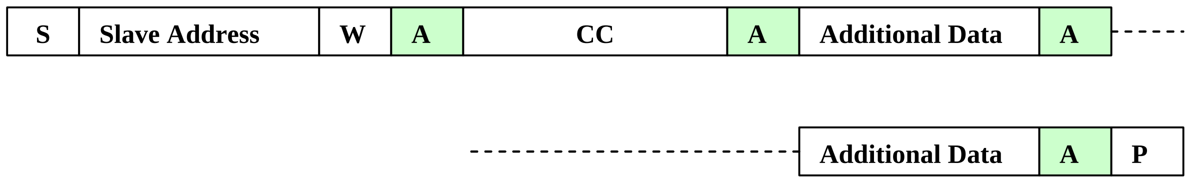
Protocol Format

The Master initiates the transfer by first claiming the bus when it's free. This is done by writing the 7-bit address of the Slave (the device to be addressed) on the bus. All data packets written on the bus have to be one byte long (8 bit), thus the lowest significant bit (LSB) of the first packet is used to determine if it is a read or write transfer. Since we only use the Master-Transmitter mode, it is always '0' for 'write'.

The slave address is a 7 bit unique code for each device. The address with seven zeros (‘0000000’) triggers a general call to all devices. Each MCU most be programmed if it shall respond to this general call or not. For Compass-1 there was no need to make use of this feature.

When the Slave acknowledges, the Master can continue with the transmission of the next byte. If there is a ‘not-acknowledge’ on the bus or no acknowledge at all after some time then the transfer could not be initiated and the bus is freed again.

The next byte that the Master will put on the system bus in case of an established communication with a Slave is the ‘command code’. The command code is an approach to structure the system bus communication. It is a dedicated protocol for Compass-1 on top of the I²C bus protocol.



S = START
P = STOP
A = ACKNOWLEDGE
W = WRITE
CC = command code

Figure 3.9: The protocol format for Compass-1

Command Codes

A dedicated protocol has been elaborated to cater the needs of the Compass-1 command and data exchange. It is shown in the above figure 3.9. The second byte to be transmitted on the bus contains the so-called command code. This is a data byte that informs the Slave about what it shall do and how much additional data the Master will send (if at all). Theoretically 256 commands are possible with this method. Since only a fraction is actually needed, the command codes can be spaced widely from each other. By doing so, the likelihood of a bitflip changing a valid command into another valid command is much less. Since the command always also identifies which two participants are involved, it turns out to be a highly effective and secure technique. The table of command codes can be found in the appendix.

Address of Devices

Each participant in the bus system is given a unique address. This might be done by the manufacturer or can be adjusted manually in hardware or even by software. There is also a general call address that can be used to attend to all participants of the bus. This might be helpful if the address of a specific component is not known or unidentifiable. Since the command codes are unique for each subsystem, the COM MCU for example will not carry out a command that is meant for the ADCS MCU. Nevertheless, the general calling address feature needed not to be used for the Compass-1 spacecraft.

Starting a Communication

The initialization of the bus communication (MCU to MCU) is an active event. That means that it is implemented at some parts of the program code and proceeds according to the specification given above. Thus the Master ‘knows’ at what stage the bus communication takes place from his side. In order to not confuse the MCU by interruptions, the interrupt service routine (ISR) is disabled during that time.

This approach is straight-forward, with a Master sending command/data to a Slave. But what about the cases the Master expects data from the Slave (for example it asked for house-keeping information from its sensors)? First the Master sends the respective command to the Slave to get data. Then the Master enters a polling loop that waits until the Slave reacted with the correct command code and is then able to receive the data from the bus. Now the former Master becomes the Slave and vice versa. During polling by the Master the Slave might be busy collecting the data, but for all this time the bus is free and can be used by other participants. It should be noted again that during this time the ISR is disabled, otherwise the MCU would respond to other requests on the bus and this would corrupt the data transfer.

Receiving the Communication

When a Master correctly addresses another participant, who is not busy communicating via the bus at this time (e.g. ADCS MCU addresses CDHS MCU), the Slave MCU suspends its current activity and executes the interrupt service routine. It evaluates the command code and does the necessary actions. When finished, it returns back to the code line where it was just when the interrupt occurred.

The ISR (also called interrupt handler) of the CDHS will trigger status flags in the memory of the MCU that will cause the task manager of the main program module to run its respective sub-module.

3.3.4 The Task Manager

There is a number of software functions (tasks) assigned to the CDHS to be carried out during mission operation. The task of collecting housekeeping data is autonomous and done on a periodical time basis. The other tasks are triggered via commands sent through the system bus. The origins of those commands so far are either the ADCS subsystem or the COM subsystem that relays the commands coming from the ground station. From the previous phase study, five different tasks evolved that are necessary to accomplish the mission objectives. They are listed in the table below, together with the corresponding status flag and the respective module name.

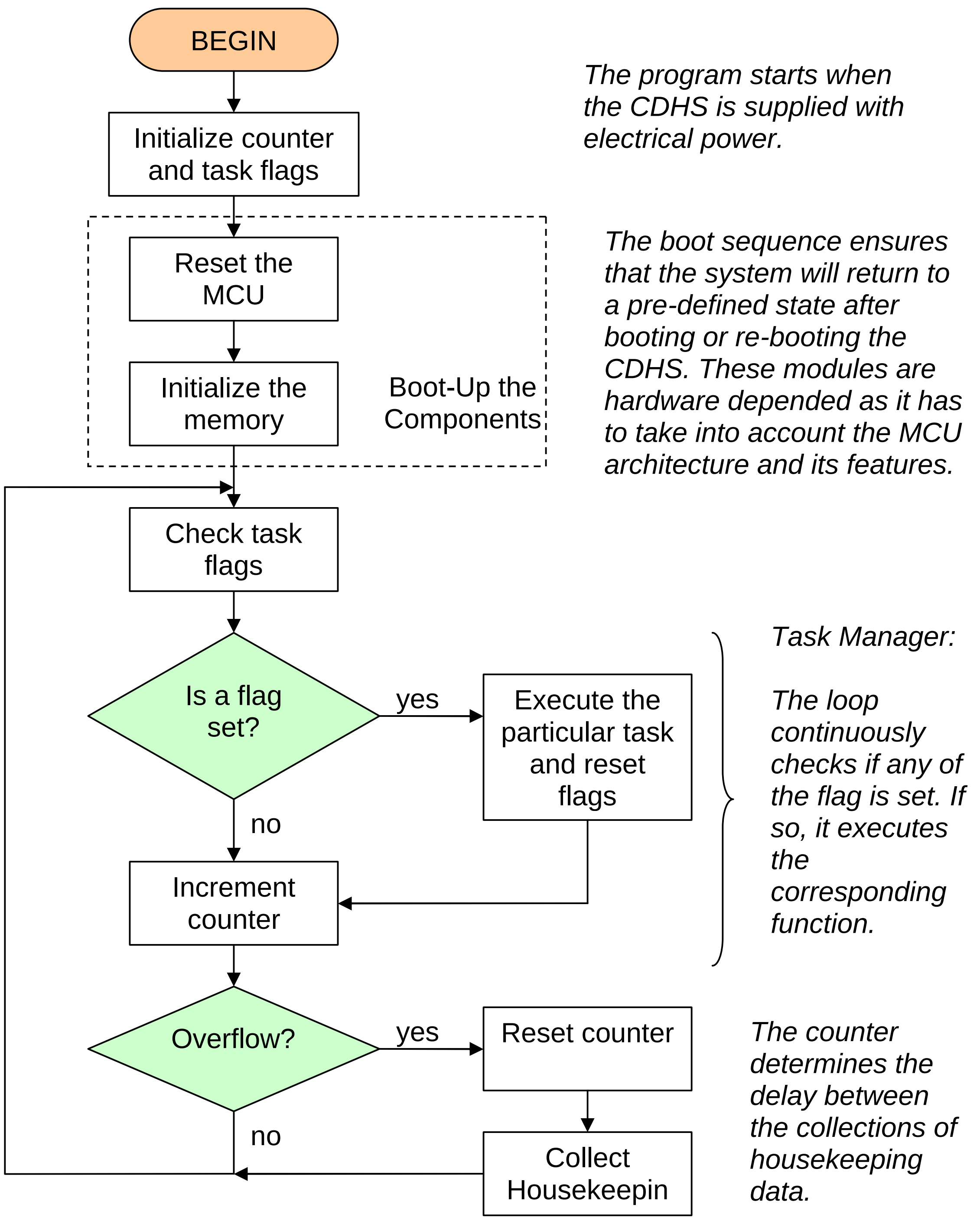
Variable	**Function**	**Description**
flag_update_adcs	func_update_adcs()	Update ADCS parameter
flag_send_hk	func_send_hk()	Send HK to ground
flag_send_img	func_send_img()	Send stored image to ground
flag_make_img	func_make_img()	Capture and send image
flag_run_adcs	flag_run_adcs()	Switch ADCS on/off

As said, the flags will indicate whether a task shall be carried out or not. Once the task is completed all flags will be cleared in order to go back into a defined state.

3.3.5 Flowchart Design

In the following the flowchart diagrams for the top and the middle layer of the software structure are elaborated. The corresponding code is listed in the appendix (which is the transition of the flowchart into program lines). The lowest level consists of modules which are also included in the appendix.

Top Level – Application Layer



The program starts when the CDHS is supplied with electrical power.

The boot sequence ensures that the system will return to a pre-defined state after booting or re-booting the CDHS. These modules are hardware depended as it has to take into account the MCU architecture and its features.

Task Manager:

The loop continuously checks if any of the flag is set. If so, it executes the corresponding function.

The counter determines the delay between the collections of housekeeping data.

The starting point labeled BEGIN is entered each time the CDHS gets switched on or re-booted respectively. A re-boot is triggered by either the internal watchdog timer that forces the MCU to restart when it has not cleared the timer flag within a set time frame. Another way to boot the CDHS is obviously to switch off and on the power supply, which can be done by the EPS (for power-saving or in case of a latch-up). As soon as the MCU and the other devices are connected to the power supply, they will go into an initial state, which is mainly configured by the hardware wiring of their pins and their logic levels. The initial state (right after the BEGIN block in the flowchart) of the chosen C8051F123 MCU from Silicon Laboratories in consistency with the engineered PCB is as follows:

- internal oscillator runs with 24.5MHz
- system clock is 3MHz
- watchdog timer is activated and its timeout is set to ~350ms
- program execution starts at 0x0000

Although the other devices on board the CDHS show specific start-up characteristics, only the MCU will need to run a boot sequence, as it is in control of the other devices. Their system state will depend on the arrangements set within this code. The boot sequence stands at the beginning of the program code so that it will be executed first.

When finished with the boot sequence, the MCU enters (and remains) in the infinite main loop.

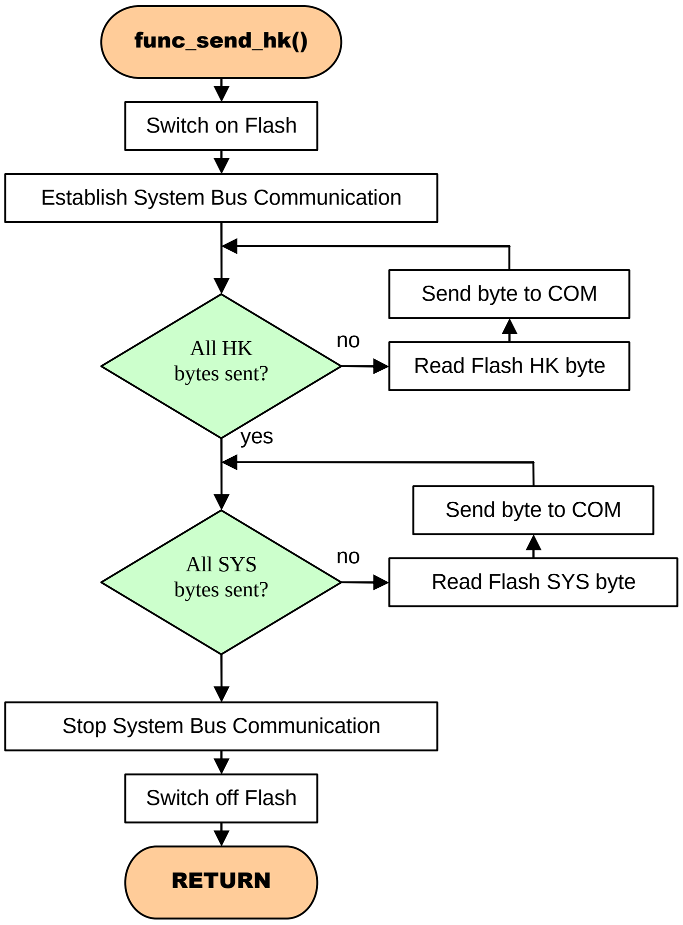
It continuously has to reset the watchdog timer in order not to trigger a forced re-boot, checks the task manager and collects housekeeping data on a pre-defined delay basis set by a value for the internal counter. The task manager keeps track of the commands to be carried out by the system. It is more a scheduler rather than a resource managing device. Principally it works on a first-come-first-serve basis. The task manager ensures that each task is given full attention and that no other tasks can interfere within its execution. Thus peak power operations, with many subsystems involved are avoided.

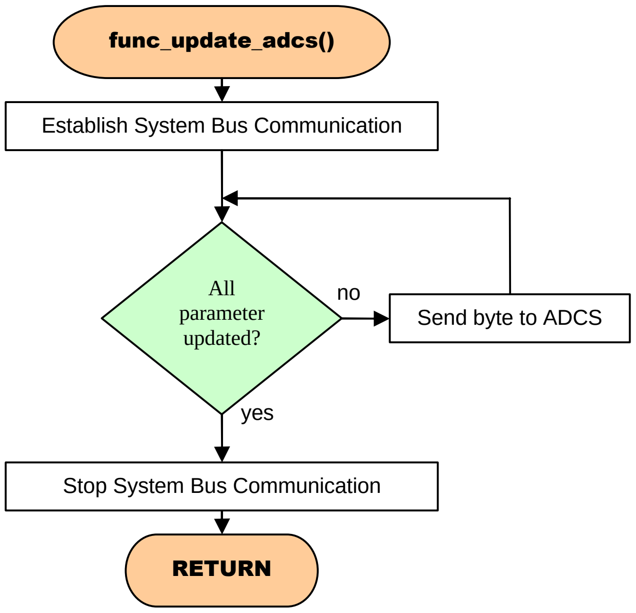
Middle Level – Device Layer

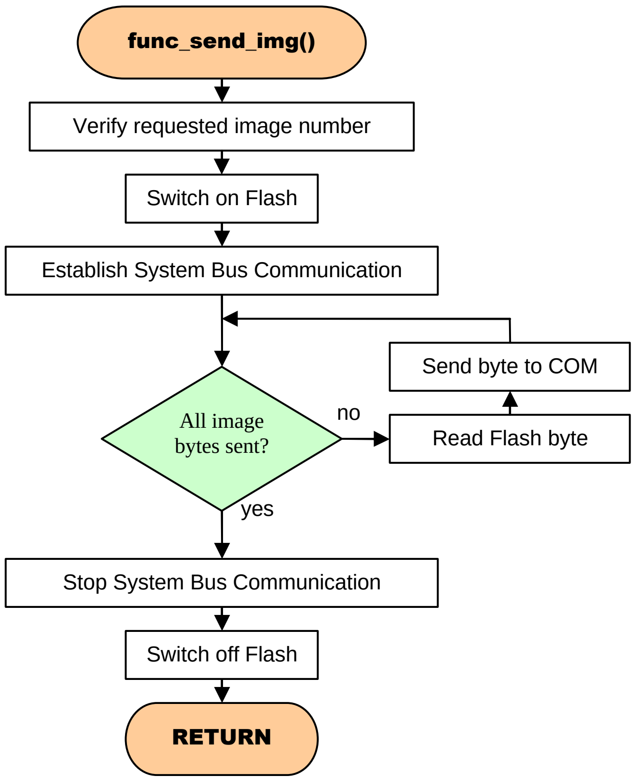
The middle layer translates the functions of the application layer into device-specific functions. They are still to be kept abstract in a way, as that the devices could be replaced by other models of the same product family with same functions. Not what happens inside the chip or component is of interest, but what goes in and what comes out!

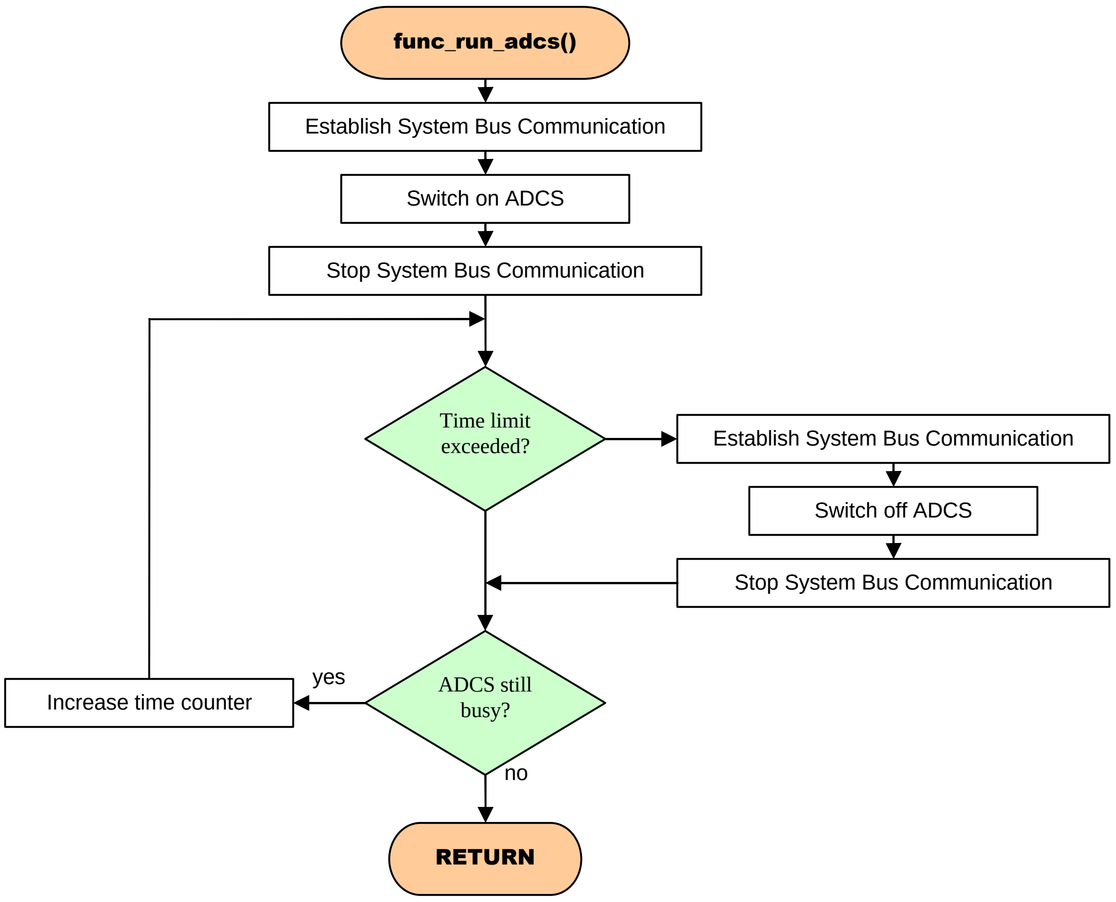
For the six functions that appear in the application layer mode (refer to the appendix) the following flowcharts have been generated. After that, the flowcharts were translated into code. Those functions are:

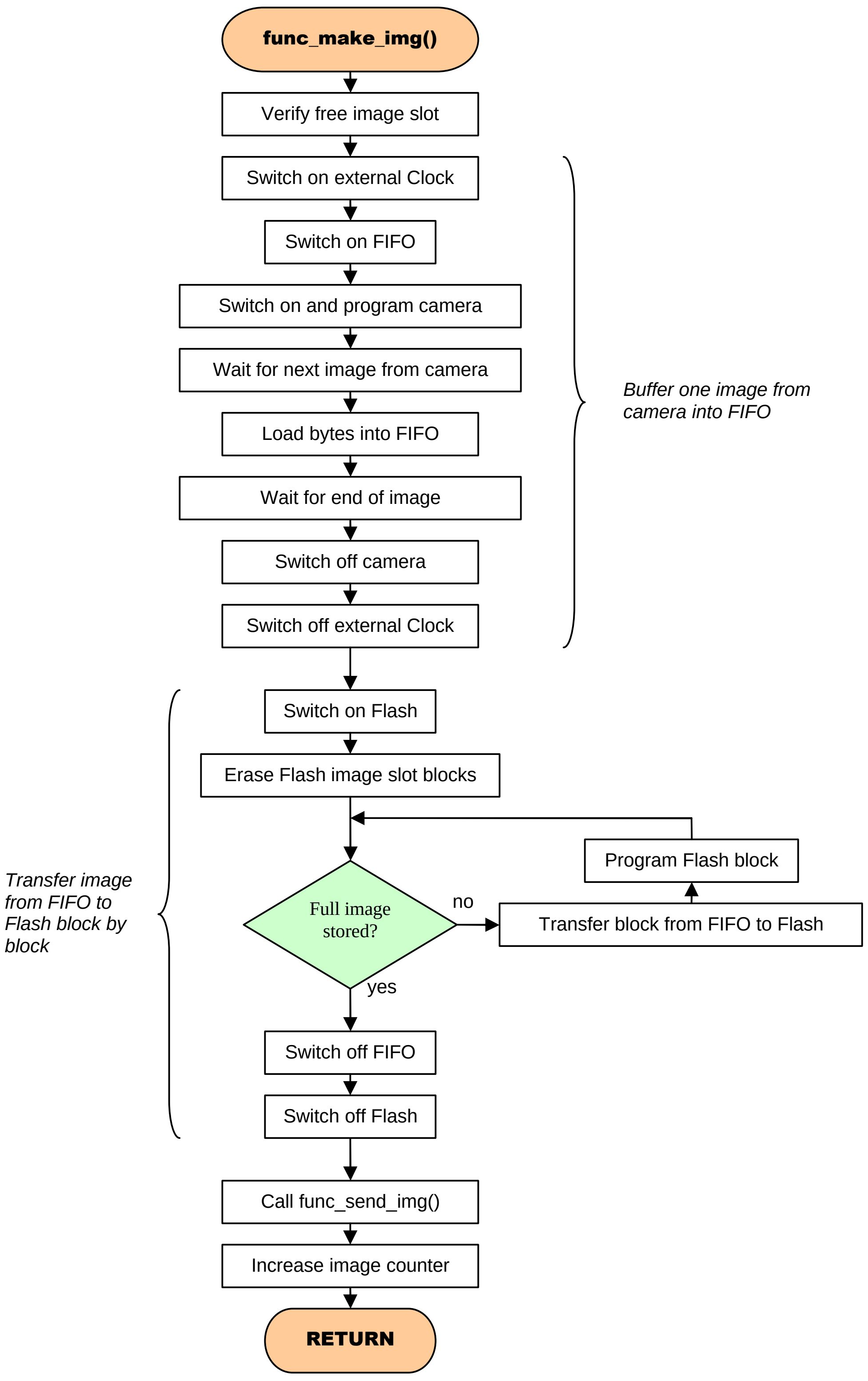
➢ func_send_hk()	Send the stored housekeeping & system data to ground
➢ func_update_adcs()	Update the ADCS settings
➢ func_send_img()	Send an image to ground
➢ func_make_img()	Capture an image and then send to ground
➢ func_run_adcs()	Switch on/off the ADCS
➢ func_collect_hk()	Gather new housekeeping data

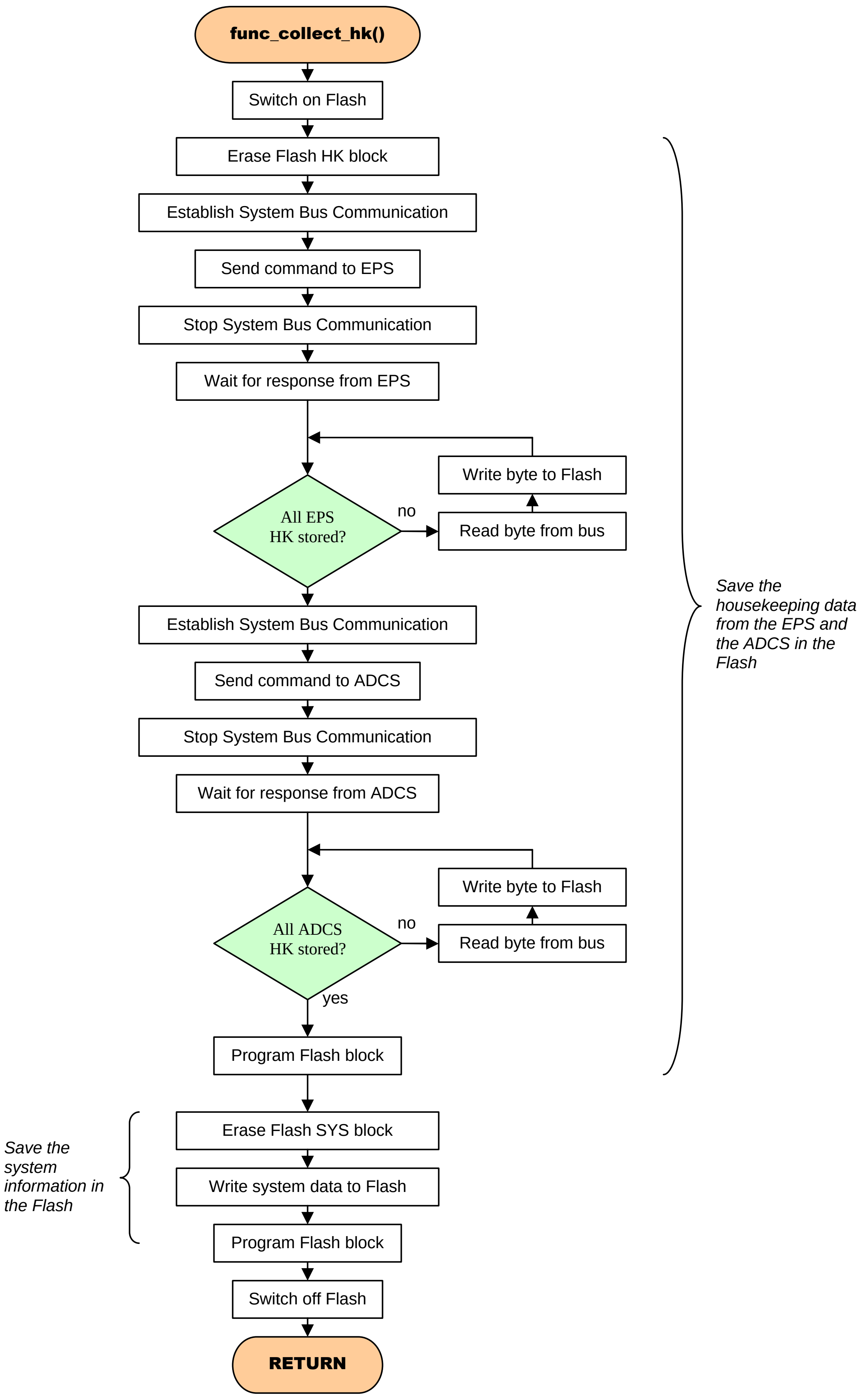

func_send_hk()
Switch on Flash
Establish System Bus Communication
All HK bytes sent?
no
Read Flash HK byte
Send byte to COM
yes
All SYS bytes sent?
no
Read Flash SYS byte
Send byte to COM
Stop System Bus Communication
Switch off Flash
RETURN


func_update_adcs()
Establish System Bus Communication
All parameter updated?
no
Send byte to ADCS
yes
Stop System Bus Communication
RETURN


func_send_img()
Verify requested image number
Switch on Flash
Establish System Bus Communication
All image bytes sent?
no
Read Flash byte
Send byte to COM
yes
Stop System Bus Communication
Switch off Flash
RETURN


func_run_adcs()
Establish System Bus Communication
Switch on ADCS
Stop System Bus Communication
Time limit exceeded?
Establish System Bus Communication
Switch off ADCS
Stop System Bus Communication
ADCS still busy?
yes
Increase time counter
no
RETURN


func_make_img()
Verify free image slot
Switch on external Clock
Switch on FIFO
Switch on and program camera
Wait for next image from camera
Load bytes into FIFO
Wait for end of image
Switch off camera
Switch off external Clock
Buffer one image from camera into FIFO
Switch on Flash
Erase Flash image slot blocks
Program Flash block
Transfer image from FIFO to Flash block by block
Full image stored?
no
Transfer block from FIFO to Flash
yes
Switch off FIFO
Switch off Flash
Call func_send_img()
Increase image counter
RETURN


func_collect_hk()
Switch on Flash
Erase Flash HK block
Establish System Bus Communication
Send command to EPS
Stop System Bus Communication
Wait for response from EPS
All EPS HK stored?
no
Read byte from bus
Write byte to Flash
Establish System Bus Communication
Send command to ADCS
Stop System Bus Communication
Wait for response from ADCS
All ADCS HK stored?
no
Read byte from bus
Write byte to Flash
yes
Program Flash block
Save the housekeeping data from the EPS and the ADCS in the Flash
Erase Flash SYS block
Write system data to Flash
Program Flash block
Save the system information in the Flash
Switch off Flash
RETURN

3. Budgets

Each of the seven subsystems of Compass-1 has been assigned with numbers for the mass, the power consumption and the costs that specified the available limit of that resource. No subsystem shall exceed those limits; the better case would be to stay below as much as possible. The budgets had to be handled very stringent because for a student picosatellite project, all those resources are very constrained.

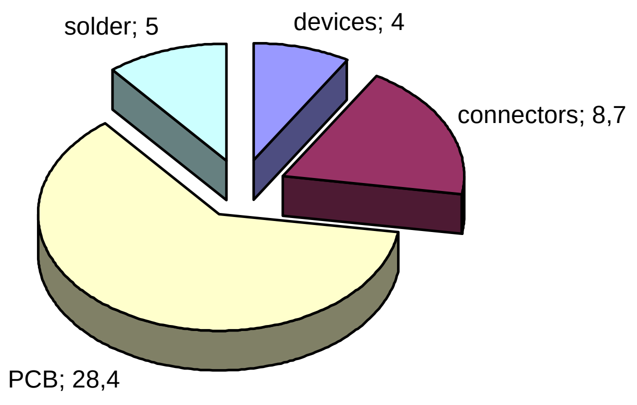
3.1 Power

A CubeSat has a strict regulation on available power. Having in mind an average of 1Watt that is constantly offered for the total system, the CDHS has to be satisfied with only a fraction of it, i.e. an average of 60mW. The following table lists the electrical devices on the board, their maximum consumption and the estimated fraction of time (based on one orbit) that they are active. It was assumed that the Satellite is accessed from ground two times each orbit and always captures and transmits an image to ground.

Identifier	Item	Definition	Maximimum Power (mW)	Operation Time (%)	Average Power (mW)
U1	MCU	C8051F123	6,00	100	6,00
U2	Flash	K9F2808UIC	60,00	22	13,33
U3	Clock Osc.	MC100EL1648D	95,00	6	5,28
U4	Vol. Reg 3V3	ZLDO330	3,15	11	0,35
U5	FIFO	IDT72V2111	165,00	11	18,33
U6	NAND	CD4011BCM	0,03	100	0,03
U7	Vol. Reg 2V5	ZXCL250	0,15	6	0,01
Q1	Transistor	MMBT2369LT1	9,00	6	0,50
	losses		5,00	100	5,00
	margin		10,00	100	10,00
	TOTAL		**347,33**		**58,83**

3.2 Mass

The CDHS board was constrained to a maximum weight of 70g, which includes the board itself, the assembled devices and connectors. The total weight, which was measured for the EM board yielded to a mass of 46g only. The figure shows how this value is composed by the different component groups.



3.3 Costs

The earlier estimations of the costs for the CDHS board produced an uncertain value. The EM production was in-house and therefore free of charge. The components however had to be ordered and paid. The problem here was that most of the formerly selected up-to-date ICs were primarily intended to be used in mass market applications and therefore only delivered to OEM customers that order huge quantities. Fortunately many suppliers showed themselves being very co-operative and gave out small numbers as samples. This helped to reduce costs massively. Summarizing, the majority of devices as well as the Protel license had been donations, which contributed to the astonishing cost reduction for the CDHS board.

4. Risk Analysis

The prototype design of the CDHS board, which will compose the engineering model of the Compass-1 satellite together with the other subsystem prototype boards, was done with best knowledge and expertise of the involved students at that time. During the design and development, the understanding of the chosen subsystem area increased enormously. This was and is one of the key objectives of such a hands-on project. Reviewing the decisions and selections that were done several months ago, optimizations would be possible in terms of design modifications or by using other components. But we have to stick to the design in order to keep the deadlines and to proceed with the project. This is a lesson worth to learn, as the industry with its long-term spacecraft projects of 5-10 years are facing such issues even more.

On the other hand, aspects that have critical impact on the mission success could be found with the prototype model, which were not so obvious from the beginning. Those problems where found by the help of a Failure Mode and Effects Analysis (FMEA) for the hardware and software. The risk assessment can be done based on this analysis and the suggested countermeasures have to be implemented in the further design of the succeeding board model.

The FMEA introduces specific parameters to support the identification of critical problems. Those are:

D	Critical characteristic which may effect safety, compliance with CubeSat/launcher regulations, or require special controls.
SEV	Severity rating (1 to 10)
OCC	Occurrence frequency (1 to 10)
DET	Detection Rating (1 to 10)
RPN	Risk Priority Number (1 to 1000) ; RPN = (SEV) x (OCC) x (DET) A 1000 rating implies a certain failure that is hazardous and harmful A 1 rating is a failure that is highly unlikely and unimportant Ratings above 100 will occur Rating below 30 are reasonable

Please refer to the appendix for the Severity, Occurrence, and Detection Criteria for FMEA.

Item and Function	Potential Failure Mode	Potential Effects of Failure	D	SEV	Potential Cause(s) of Failure	OCC	Detection Method & Quality Controls	DET	RPN	Recommended Actions
MCU / internal Flash program storage	Corruption of program code	Program execution will always hang-up at the corrupted code line	y	7	Bitflip due to radiation	6	Radiation tests and/or reports	6	**252**	Substitute internal Flash with PROM or implement external PROM for code storage
Flash / data storage	Corruption of data	Pixel errors in images, wrong data in housekeeping and system information	n	4	Bitflip due to radiation	5	“	6	**120**	No action intended. The image validation is on ground and checked for error occurrence
I²C / system bus	Complete failure	Subsystems cannot exchange commands/data, the satellite will not receive commands from ground	y	8	A components fails and draws the bus permanent to signal	2	Test with bus pulled to ground	1	**16**	Connect as few components to the bus as feasible. No sensors etc. EPS can switch off power
	Data corruptions	Wrong command codes and/or additional data	n	5	Bitflips due to radiation	2	Tests with ‘wrong’ command codes	6	**60**	Check spacing of command codes, in such as that they have enough distance
PCB / signal tracks	No signal	Devices cannot be controlled	y	8	Signal interference, broken tracks	3	Intensive structural testing. Application of launch loads	4	**96**	Possibly widen the tracks. Change routing of tracks. Use different board material
Camera / signal	No clock signal	No image output	n	6	External oscillator works at 5V, camera at 2V5	3	Test with EM board	1	**18**	Exchange oscillator
MCU / internal oscillator	Variations in clock cycle	Wrong Flash timing, data lost	n	5	Wide temperature changes	4	Test at extreme low and high temperatures	3	**60**	Program Flash timing for worst case values (+/- 25%)
Software / system bus communication	Hangs-up	No further operation, watchdog timer re-boots MCU	y	5	The addressed Slave device does not respond on the bus as expected	5	Test the CDHS with other subsystems attached or simulated	5	**125**	Add counter into polling loops to quit after a time-out

Item and Function	**Potential Failure Mode**	**Potential Effects of Failure**	**D**	**SEV**	**Potential Cause(s) of Failure**	**OCC**	**Detection Method & Quality Controls**	**DET**	**RPN**	**Recommended Actions**
MCU	Complete failure	CDHS inoperable, satellite cannot fulfill mission objectives	y	9	Burn-out	1	Radiation Test, current overdose	7	**63**	EPS has to switch off power supply to the subsystem for a few seconds, then switch on again
Flash	Complete failure	No data storage, loss of data	n	6	Burn-out	1	Radiation Test, current overdose	3	**18**	No action planned
FIFO	Complete failure	Loss of image	n	6	Burn-out	1	Radiation Test, current overdose	3	**18**	No action planned
NAND	Complete failure	Loss of image	n	6	Burn-out	1	Radiation Test, current overdose	3	**18**	No action planned
External Oscillator	Complete failure	No image	n	6	Burn-out	1	Radiation Test, current overdose	3	**18**	No action planned
Voltage Regulator 2V5	Complete failure	No image	n	6	Burn-out	1	Radiation Test, current overdose	3	**18**	No action planned
Voltage Regulator 3V3	Complete failure	No image	n	6	Burn-out	1	Radiation Test, current overdose	3	**18**	No action planned
All ICs	Complete failure	Device draws extensive current, inoperable	y	8	Latch-up	2	Radiation Test, current overdose	8	**128**	EPS has to switch off power supply to the subsystem for a few seconds, then switch on again

5. Conclusion

A hardware and software solution for the CDHS engineering board was designed and finally developed. The board is ready for testing and debugging. So far, the CDHS will fully work based on the paper work. But as the past has shown, there is always a discrepancy between theory and reality, such as variations in the datasheets and the real hardware. The conducted risk analysis is a method to identify as much potential failures as possible. Yet, more problems could show up, that would perhaps not have been taken into account just by looking through the documentations. Hence, intensive testing becomes essential for any project. From the results of the tests, modifications will be implemented to the prototype and it will eventually lead to the final model.

The result is a product that complies with the mission operation requirements and can serve as basic framework for future endeavors, leaving enough room for possible modifications and optimizations, in particular in terms of software.

One of the mission goals of Compass-1 is to educate the involved students and to give them better understanding of system engineering. Long before launch, this goal has already been accomplished.

6. Annex

6.1 Bill of Material (BOM)

The BOM lists the components that are assembled on the PCB of the CDHS board.

Identifier	Item	Definition	Footprint	Temp Range
U1	MCU	C8051F123	TGFP64A	industrial
U2	Flash	K9F2808UIC	TSOP48	industrial
U3	Clock Osc.	MC100EL1648D	751-06	industrial
U4	Vol. Reg 3V3	ZLDO330	SO-G8/P1.53	industrial
U5	FIFO	IDT72V2111	TGFP64A	industrial
U6	NAND	CD4011BCM	M14A	industrial
U7	Vol. Reg 2V5	ZXCL250	SOT23-5	industrial
Q1	Transistor	MMBT2369LT1	318-08	-55 .. +150
JP1	JTAG		HDR2X5	industrial
JP2	Camera Connector			-25 .. +85
J1	Board Connector			-25 .. +85
J2	Board Connector			-25 .. +85
J3	Board Connector			-25 .. +85
J4	Board Connector			-25 .. +85
J5	Board Connector			-25 .. +85
J6	Board Connector			-25 .. +85
L1	Inductor		CC4532-1812	industrial
C1	Capacitor	100nF	603	industrial
C2	Capacitor	100nF	603	industrial
C3	Capacitor	100nF	603	industrial
C4	Capacitor	100nF	603	industrial
C5	Capacitor	100nF	603	industrial
C6	Capacitor	100nF	603	industrial
C7	Capacitor	100nF	603	industrial
C8	Capacitor	100nF	603	industrial
C9	Capacitor	100nF	603	industrial
C10	Capacitor	180pF	603	industrial
C11	Capacitor	10µF	603	industrial
C12	Capacitor	100µF	603	industrial
C13	Capacitor	10nF	603	industrial
C14	Capacitor	10pF	603	industrial
C15	Capacitor	1µF	1206	industrial
C16	Capacitor	1µF	1206	industrial
C17	Capacitor	2.2µF	805	industrial
R1	Resistor	2.2K	603	industrial
R2	Resistor	2.2K	603	industrial
R3	Resistor	4.75K	603	industrial
R4	Resistor	1K	603	industrial
R5	Resistor	10K	603	industrial
R6	Resistor	10K	603	industrial
R7	Resistor	10K	603	industrial
R8	Resistor	10K	603	industrial
R9	Resistor	10K	603	industrial
R10	Resistor	10K	603	industrial
R11	Resistor	10K	603	industrial
R12	Resistor	10K	603	industrial
R13	Resistor	10K	603	industrial
R14	Resistor	10K	603	industrial
R15	Resistor	10K	603	industrial
R16	Resistor	200	603	industrial

6.2 Application Layer Code

```
/*****************************************************************
 APPLICATION LAYER
 This file contains the code implementation of the highest level
 of the flowchart for the CDHS of the COMPASS-1 satellite.
****************************************************************/

#include <C8051F123.h>       // this is the file containing MCU specific definitions
#include <globals.h>         // the global variables and definitions
#include <driver_layer.c>    // lowest level (hardware interface) layer
#include <device_layer.c>    // middle layer


/////////////////////////////////////////////////////////////////
//                          main()                             //
//                                                             //
//      This is the main program, which essentially consists of //
//      an endless loop. Inititally, the MCU runs a boot code   //
//      that sets all pins into pre-defined conditions. Then in //
//      the loop, it is continously checked if any of the flags //
//      is set, which would identify that this task shall be    //
//      carried out. The flags are triggered from extern (other //
//      subsystems) through the interrupt service routine (ISR) //
//      of the I²C system bus. The only expection of this is the//
//      collection of housekeeping data. This is done on a      //
//      periodically basis, whenever the counter has elapsed.   //
//                                                             //
void main(void) /////////////////////////////////////////////////
{
       // initialize the counter to zero
       unsigned int counter = 0;

       // at beginning set all task flags to zero
       flag_update_adcs      = OFF;
       flag_send_hk          = OFF;
       flag_send_img  = OFF;
       flag_make_img         = OFF;
       flag_run_adcs         = OFF;


       // now the boot code
       reset();              // boot-up the components, which puts MCU into defined state
       init_memory();        // load the system information from the FLASH


       // this is the infinite loop
       while(1)
       {
              reset_wdt();   // resets the watchdog timer to prevent forced re-boot

              // if any of the task flags is set then call the corresponding module
              // note that during execution of the modules no interruption by IRQ is allowed
              if (flag_update_adcs)        { IRQ = OFF; func_update_adcs();     IRQ = ON; }
              if (flag_send_hk)            { IRQ = OFF; func_send_hk();         IRQ = ON; }
              if (flag_send_img)           { IRQ = OFF; func_send_img();        IRQ = ON; }
              if (flag_make_img)           { IRQ = OFF; func_make_img();        IRQ = ON; }
              if (flag_run_adcs)           { IRQ = OFF; func_run_adcs();        IRQ = ON; }

              reset_wdt();          // reset the watchdog timer
              counter++;            // increase the counter

              // check if the counter exceeds the pre-set time period
              if (counter >= HK_COLLECTION_DELAY)
              {
                     counter = 0;          // reset the counter to zero
                     IRQ = OFF;            // don't allow interruption
                     func_collect_hk();    // call function to collect housekeeping data
                     IRQ = ON;             // interrupts on
              }

       }
}
// end of main()
/////////////////////////////////////////////////////////////////
```

6.3 The Command Codes

Each command code is specific. It specifies the two involved participants and what is happening. Command codes that have one byte are used to trigger (activate / deactivate) certain functions of a subsystem or to signal their state. In other cases they are used to transfer data over the bus and to assure that it reaches the correct destination.

The encoding of the command codes is based on eight bit syntax, thus allowing a total of 256 different commands. The spacing between command codes are chosen in such a way, as that they differ for each two subsystem combinations in two bits. This is done to reduce the effects of bitflips. An occurrence of two or more bitflips at a time is especially unlikely and can thus be neglected.

From	To	CC	Add. data	Description
CDHS	**ADCS**	0x11	-	Request housekeeping data
ADCS	**CDHS**	0x22	13 Bytes	Send housekeeping data
ADCS	**CDHS**	0x33	-	Request for activation
CDHS	**ADCS**	0x44	-	ADCS on
CDHS	**ADCS**	0x55	-	ADCS off
ADCS	**CDHS**	0x66	-	Urgent Request for activation
CDHS	**ADCS**	0x77	3 Bytes	Update ADCS parameter (for slew)
ADCS	**CDHS**	0x88	-	ADCS control finished
CDHS	**EPS**	0x12	-	Request housekeeping data
EPS	**CDHS**	0x23	25 Bytes	Send housekeeping data
CDHS	**COM**	0x13	256 Bytes	Send housekeeping to ground
CDHS	**COM**	0x24	(640x480)	Send image to ground
COM	**CDHS**	0x35	-	Request for image capturing
COM	**CDHS**	0x46	1Byte	Request for image number X
COM	**CDHS**	0x57	3 Byte	Send parameter to ADCS
COM	**CDHS**	0x68	-	Request housekeeping data
COM	**CDHS**	0x79	1 Byte	Camera exposure settings

6.4 Device Slave Addresses

The slave addresses are the 7-bit unique identification for each participant at the I²C system bus. The table can only be completed with the input from the other subsystems. It is very important that all other systems maintain the same list of addresses.

Subsystem	Device	Address (hex)
ALL	General Calling Address	0
PAYLOAD	CAM	42
ADCS	MCU	TBD
CDHS	MCU	TBD
EPS	MCU	TBD
COM	TNC-TX	TBD
COM	TNC-RX	TBD

6.5 Severity, Occurrence, and Detection Criteria for FMEA

Ranking	Effect	Criteria: Severity of Effect
1	None	No effect
2	Very Minor	Very minor effect on product or system performance.
3	Minor	Minor effect on product or system performance.
4	Low	Small effect on product performance. The product does not require repair.
5	Moderate	Moderate effect on product performance. The product requires repair.
6	Significant	Product performance is degraded. Comfort or convenience functions may not operate.
7	Major	Product performance is severely affected but functions. The system may not be operable.
8	Extreme	Product is inoperable with loss of primary function. The system is inoperable.
9	Serious	Failure involves hazardous outcomes and / or noncompliance with govt. regulations or standards.
10	Hazardous	Failure is hazardous, and occurs without warning. It suspends operation of the system an/or involves noncompliance with govt. regulations

Ranking	Possible Failure Rates	Probability of Failure
1	$\leq 1 \times 10^{-6}$	Nearly Impossible
2	1×10^{-5}	Remote
3	1×10^{-4}	Low
4	4×10^{-4}	Relatively Low
5	2×10^{-3}	Moderate
6	1×10^{-2}	Moderately High
7	4×10^{-2}	High
8	0.2	Repeated Failures
9	0.33	Very High
10	≥ 0.55	Extremely High: Failure Almost Inevitable

Ranking	Detection Probability
1	Almost Certain Detection
2	Very High Chance of Detection
3	High Probability of Detection
4	Moderately High Chance of Detection
5	Moderate Chance of Detection
6	Low Probability of Detection
7	Very Low Probability of Detection
8	Remote Chance of Detection
9	Very Remote Chance of Detection
10	Absolute Uncertainty – No Control

6.6 Abbreviations

ADCS	attitude determination and control system
BOM	bill of materials
CDHS	command and data handling system
COM	communication system
COTS	commercial off-the-shelf
EM	engineering model
EPS	electrical power system
FM	flight model
FMEA	failure mode and effects analysis
HK	housekeeping data
IDE	integrated development environment
ISR	interrupt service routine
MCU	micro control unit
OS	operating system
PCB	printed circuit board
SEE	single event effect
SEL	single event latch-up
SEU	single event upset
SMD	surface mount device
TCS	thermal control system
TID	total ionizing dose
WDT	watchdog timer

6.7 References

[1] Scholz, A. (2004) *Phase B Study of CDHS for COMPASS-1*. www.raumfahrt.fh-aachen.de

[2] Twiggs, B. and Puig-Suari, J. (2003). *CUBESAT Design Specifications Document, Rev. VIII.* http://cubesat.calpoly.edu/

[3] Marks, L and Caterina, J. A. (2000) *Printed Circuit Assembly Design*. McGraw-Hill, New York, USA

[4] MRC Microelectronics () *Radiation Hardening Microelectronics*. http://www.mrcmicroe.com/Radiation_Hardening.htm

[5] Shirvani, P. P. (2003) *COTS Technology & Issues – Space Environments*. 44th Meeting of IFIP Working Group. Montery, USA

[6] Some, R. () *Radiation Models and Hardware Design*. JPL Caltech, USA

[7] Philips Semiconductor (2000) *The I²C-Bus Specification*. http://www.semiconductors.philips.com/i2c

[8] Silicon Laboratories (2003) *C8051F120/1/2/3/4/5/6/7-DS12*. www.silabs.com

[9] Samsung (2003) *K9F2808U0C FLASH MEMORY*. www.samsung.com

[10] IDT (2001) *IDT72V2111 SUPERSYNC FIFO*. www.idt.com

[11] Philips (1998) *74LVC00A Quad 2-input NAND gate*. www.semiconductors.philips.com

[12] ON Semiconductors (2004) *MC100EL1648*. http://onsemi.com

[13] Zetex (2002) *ZXCL250 Series*. www.zetex.com

[14] Zetex (1996) *ZLDO330 Series*. www.zetex.com

[15] Keil Software (2001) *Cx51 Compiler User's Guide*. www.keil.com

[16] ECSS-E-10-04A (2000) *Space Environment*. ESA-ESTEC, Noordwijk, The Netherlands

[17] Miller, G. H. (2004) *Microcomputer Engineering*. Prentice Hall, New Jersey, USA

[18] Schildt, H. (2000) *C: The Complete Reference, Fourth Edition*. McGraw-Hill, Berkeley CA, USA