NTNU
Det skapende universitet

Grensesnitt for IR-kamera i studentsatellitten NUTS

Espen Meland

Master i elektronikk

Innlevert:	juli 2013
Hovedveileder:	Bjørn B. Larsen, IET
Medveileder:	Roger Birkeland, IET

Norges teknisk-naturvitenskapelige universitet
Institutt for elektronikk og telekommunikasjon

Master Thesis

INTERFACE FOR IR-CAMERA IN STUDENT SATELLITE NUTS

Espen Meland

Abstract

The NUTS test satellite plans to include an IR camera as its main payload to observe gravity waves in the atmosphere. The payload module of the satellite has perform both image enhancement and compression to obtain images that have a good SNR while also being small enough in size to be able to transferred down to an earth station in a reasonable amount of time. This places some demand on the payload module. It not only needs to acquire and store images, it needs to perform image processing on them as they are acquired in order to not have to dedicate a large amount of storage space and resources to simply storing image series before working on them.

Several possible camera interfaces were examined, and evaluated for use with either a FPGA or a microcontroller. It was found that the Camera Link interface was preferable when working with a FPGA, while a direct integration solution would work best with a microcontroller.

The payload module will also need to be able to take commands from, and transmit image data to, other parts of the satellite, and it was found that it would be sufficient for the module to act as a I2C slave on the back plane bus.

FPGAs and microprocessors were compared for their fitness in being used as the processing module in the payload, and it was found that the FPGA was in general better suited for this task.

Finally, an example implementation was designed, and a FPGA designed was written in verilog.

Sammendrag

NTNU test satellitt planlegger å inkludere et IR camera som hovednyttelast, med formål å ta bilde av tyngdebølger i atmosfæren. Nyttelastmodulen i satellitten må utføre både bildebehandling og bildekomprimering for å oppnå bilder som både har god nok SNR, men samtidig er små nok til å kunne sende til en mottakerstajon på jorda i en fornuftig tid. Dette setter noen krav til nyttelastmodulen. I tillegg til å måtte hente bilder fra kameraet og lagre disse må den behandle bilder etterhvert som de kommer inn slik at man unngår å bruke mye lagringsplass og ressurser på å lagre bildeserier før de behandles.

Flere kameragrensesnitt ble undersøkt og evaluert med tanke på bruk med FPGA eller mikrokontroller. Det ble funnet at Camera Link er det foretrukne grensesnittet når man bruker FPGA, mens direkte integrasjon var best når man bruker mikrokontroller.

Nyttelastemodulen må også kunne ta ordre fra, og sende bildedata til, andre deler av satellitten. Det ble funnet at det holder å ha nyttelastmodulen som en I2C slave på bakplansbussen.

FPGA og mikrokontroller ble sammenlignet i forhold til deres egnethet som prosesseringsenheten i nyttelastmodulen, og det ble funnet at FPGA generelt var mer egnet til denne typen oppgave.

Det ble til slutt gjort et design av en eksempelimplementering, som og et FPGA design ble skrevet i verilog.

Problem description

(Translated from Norwegian)

INTERFACE FOR IR-CAMERA IN STUDENT SATELLITE NUTS

The student will examine how an IR-camera can be integrated into the student satellite NUTS. The Main computer in NUTS is a microcontroller that is not capable of processing the raw images from the camera. Depending on which camera is chosen, the camera interface will likely be either USB or Cameralink. The student will come up with one or more design suggestions that will be able to receive images from the camera, perform image processing, and store these to a flash memory for later processing. The interface can for example be built using a FPGA. Considering it will be put in a satellite, there will be some constraints and demands on the solution. The electronics must fit on a board that is 66x123x5mm^3 or smaller. The power consumption is also important, the interface should draw as little power as possible, and it is especially important that it can either go into hibernation or shut down. The interface will need to both control the camera and collect the image data.

To create a datapoint, several pixels in the camera needs to be combined (for example four pixels become one). A datapoint has to be built from a number of single frames from the camera, so that the "integration time" becomes about one second. It is the result of this integration that will be stored as a datapoint in flash. The pictures will be stored in a raw, uncompressed format.

Original project goal text in norwegian:

GRENSESNITT FOR IR-KAMERA I STUDENTSATELLITTEN NUTS

Studenten skal undersøke korleis eit industrielt IR-kamera kan koblast til elektronikk i studentsatellitten NUTS. Hovuddatamaskina i NUTS er ein mikrokontroller som ikkje vil kunne handsame rå bildedata frå kameraet. Avhengig av kva kameratype som vert endeleg valt, vil truleg USB eller CamLink vere aktuelle grensesnitt. Studenten skal føreslå ei eller fleire løysingar til eit design til eit grensesnitt som kan motta bilde frå kameraet, behandle desse og lagre bilde frå kameraet til eit flashminne for seinare prossessering. Grensesnittet kan tildømes byggast ved hjelp av ein FPGA. Sidan dette skal inn i ein satellitt, set dette nokre avgrensingar og krav til løysinga. Elektronikken må få plass på eit kort som er maksiumum 66 x 123 x 5 mm^3. I tillegg er straumtrekket viktig; grensesnittet bør bruke minst mogleg straum, og spesielt viktig er det at det kan gå i dvale eller slåast av. Grensesnittet må kunne både styre kameraet og hente bildedata.

For å lage eit datamålepunkt må fleire pixlar i kameraet binnast (til dømes fire px blir ein px). Eit slikt datamålepunkt må byggast opp av fleire enkeltframes frå kameraet, slik at total "integrasjonstid" blir ca eit sekund. Det er resultatet av denne integreringa som skal lagrast som eit datapunkt til flash. Bilda skal lagrast i eit rått, ukomprimert format.

Contents

List of Figures

List of Tables

List of acronyms

AD Analog/digital

ADCS Attitude determination and control system

CMOS Complimentary metal oxide semiconductor

CSP CubeSat space protocol

DPCM Differential pulse code modulation

DSP Digital signal processor

ECC Error correcting code

EPS Electrical power system

FPGA Field programmable gate array

HDL Hardware description language

IC Integrated circuit

IP Intellectual property

IR Infrared

I/O Input/Output

I2C Inter Integrated Circuit

JTAG Joint Test Action Group

LE Logical element

LUT Lookup table

LVDS Low voltage differential signaling

NUTS Norwegian University of Science and Technology

OBC Onboard computer

RAM Random access memory

RTL Register transfer level

SEE Single event effect

SEL Single event latchup

SET Single event transient

SEU Single event upset

SNR Signal-to-noise ratio

SPI Serial peripheral interface

SRAM Static RAM

SWIR Short wave infrared

TMR Triple modular redundancy

TTL Transistor-transistor logic

UART Universal asynchronous receiver transmitter

USB Universal serial bus

1. Introduction

1.1. NUTS student satellite

The NUTS test satellite is a project supported and organized by the department for electronics and telecommunications at NTNU. The goal of the project is to design, build test and launch a double cubesat by 2014. The work will mostly be performed by students at NTNU as either projects or as thesises.

The payload of the satellite is an IR-camera, which will be used to capture gravity waves in the atmosphere. It will also have an experimental wireless internal bus, which may be used to transmit data between modules at high speed.[Bir11]

1.2. Previous work

This thesis builds in part on the theoretical groundwork performed by Marianne Bakken in her masters thesis, "Signal Processing for Communicating Gravity Wave Images from the NTNU Test Satellite" and Snorre Rønnings project work "Optimizing an Infrared Camera for Observing Atmospheric Gravity Waves from a Cubesat Platform". These works lay out the requirements of both the camera in the satellite and the methods by which an image with a good enough SNR ratio can be obtained.

1.3. Aim of thesis

The aim of this thesis is to map out the requirements and constraints of the payload module in the NUTS test satellite from a digital design standpoint, and to come up with a suggested design solution. It will attempt to analyze some scenarios that will affect the requirements of the payload module, and how these can be practically overcome. These design solutions should fulfill the requirements of the module, while being balanced against the constraints that the module is placed under.

1.4. Outline

The first section of this thesis will lay the theoretical groundwork for what the payload module should contain based on the facts at hand at the time of writing this thesis.

chapter 2 gives an overview of the larger system that the payload will fit into.

chapter 3 gives an overview on the application of the camera and how this affects the payload.

chapter 4 is a summary of the most common interfaces for SWIR cameras, and how these can be implemented in an embedded system.

chapter 5 takes a look at how the camera will communicate with the rest of the satellite.

chapter 5 examines what requiremements the image acquisition and processing imposes on the payload in a couple of different scenarios.

chapter 7 is a summary of the effects of space radiation on electronics and how these effects can be mitigated.

chapter 8 takes a look at what is requires of the central processing itself, and gives suggestions on how these could be implemented.

The second part of this thesis is an example design based on a specific case.

chapter 9 lays out the case, and discusses how the design best can be made to fit this case. A processing module is chosen.

chapter 10 contains a description of the FPGA design to be used on the processing module.

chapter 11 are the final remarks and conclusions of this thesis.

Part I.

Theory

2. NUTS test satellite

2.1. Introduction

This chapter will give a short overview of the NUTS test satellite mission and systems, to give context to how the work of this thesis will fit into the larger system.

2.2. Mission statement

Copied from [Bir11]:

> "The NUTS aims to design, develop, test, launch and operat a double Cubesat by 2014. Students from different curiums will do the largest part of the work, supported by the project managment and technical staff. The work will be performed as part of the students project. and theses. We have chosen our design to be generic and modular, so the satellite bus can support different pay loads. As paylad for the first satellite, an IR-camera will be implemented, in addition to a wireless internal databus.
>
> Recruitment and education of skillful students will be a main part of the project goals. Through hands-on experience, the students will be able to master different skills needed in their job after graduation."

2.3. Overview

The satellite consists (currently) of five seperate modules that perform their tasks autonomously. These are:

- **Communication system**

Responsible for transmitting and receiving radio signals to and from ground stations on earth

- **Attitude determination and control system (ADCS)**

Determines what direction the satellite is pointing, and uses magnetic coils to adjust it to the desired direction.

- **Electrical power system**

Conditions power collected from solar cells to the batteries, and provides power to the rest of the satellite.

- **Onboard data handling**

The central processing unit of the satellite, responsible for keeping the state of the satellite and issuing commands to other systems.

- **Payload (camera)**

The IR camera module, responsible for controlling the camera, and grabbing and processing images.

Fig. 2.1 shows a general diagram of the satellite.

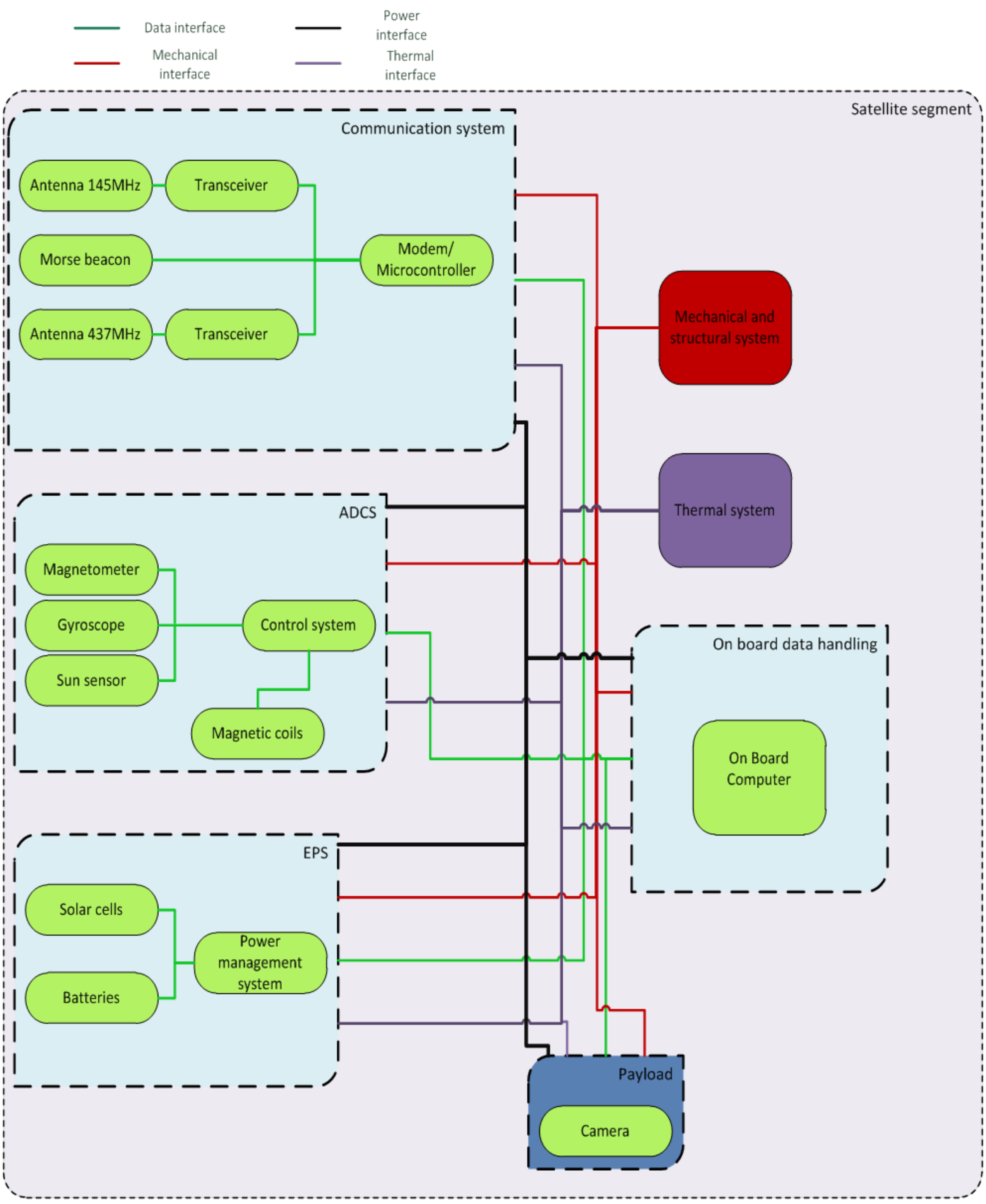

Figure 2.1.: System overview (from [nut])

As mentioned in the mission statement, the overall design of the satellite aims to be modular. In practice this means that the different parts (modules) of the satellite are connected by a backplane, which distributes power from a power module and routes data traffic between the modules through a common wired bus. Each module fits into the backplane using a standard connector.

Most info from this chapter comes from [BG13], where a more complete overview can be found.

3. Background on camera application

3.1. Overview

The main payload of NUTS student satellite is an IR-camera, which will be used to capture gravity waves in the atmosphere. This chapter will provide background on the application of this camera and how this will affect the requirements of any processing module on the payload.

3.2. Gravity waves

"Gravity wave" is a general term for waves that occur either in the meeting between two medium (for instance two liquids with different densities) or at the surface of a medium (ocean waves for example). When a medium (water for example) is displaced (for instance by wind), gravity will work to restore equilibrium, which will create oscillations (waves) as equilibrium is restored.

Atmoshperic gravity waves happen between different "layers" of air in the atmosphere, where lower layers of air gets displaced into higher layers (passing over mountains for instance) and oscillate up and down before equilibrium is restored by gravity. Since these air layers hold different temperatures, these waves can be captured by infrared cameras, allowing you to "see" the lower air travelling in and out of the higher air layer.[Hab] More information on gravity waves can be found in [Røn12] and [Bak12].

Fig. 3.1 illustrates atmospheric gravity waves making an impression on the ocean surface. This is an example of gravity waves being observed indirectly.

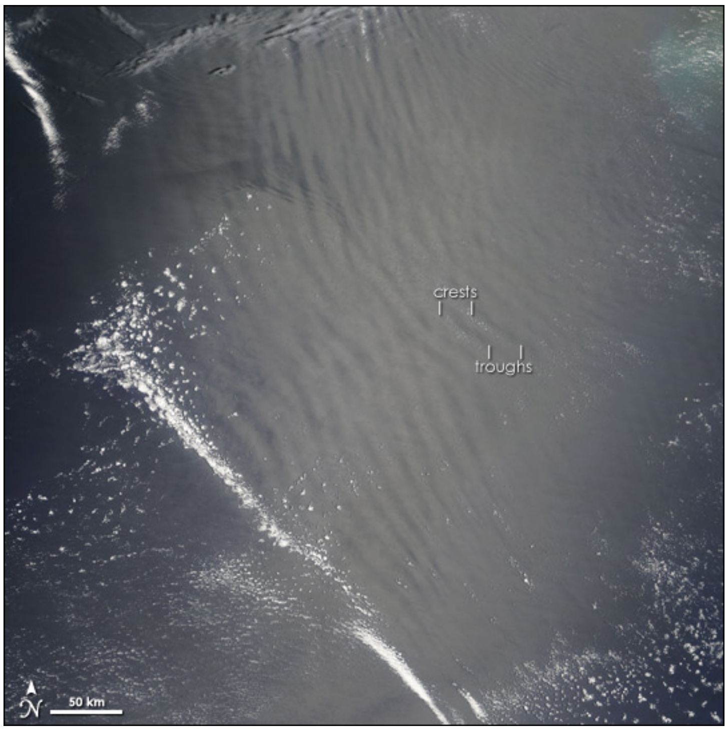

Figure 3.1.: Image of atmospheric gravity wave impression on the ocean surface (from [JSMRRT])

3.3. How the camera is used

To obtain a datapoint, the camera will take a series of images that will then be combined to a single image using motion compensation and image averaging. The resulting image will be compressed to take up less transfer time to the ground station. After it is compressed, the module will store the image until it ready to be received used by the OBC[Bak12].

Based on the preliminary results from [Bak12], the downlink capacity of the satellite amounts to a maximum of 45 per day on average (with a 256x256 resolution and 8 bit pixel depth). If all capacity is used to send images, this amounts to the imaging system being used every 32 minutes on average in the long term.

3.3.1. Overview

To get a proper image to capture the gravity waves, some form of image enhancment needs to be performed, as a single image will be subject to motion blur and noise. To do this, several images will be taken and averaged to obtain a good SNR. Background subtraction will also be performed. As the satellite is in motion while the pictures are taken, it might also be necessary to compensate for the motion blur that occurs over the integration time of each image.

As these images end up being rather large to transmit from the satellite through the limited bandwith of the radio link, compression will also need to be performed. Seen in Fig. 3.2 is the suggested image processing system from Bakkens thesis:

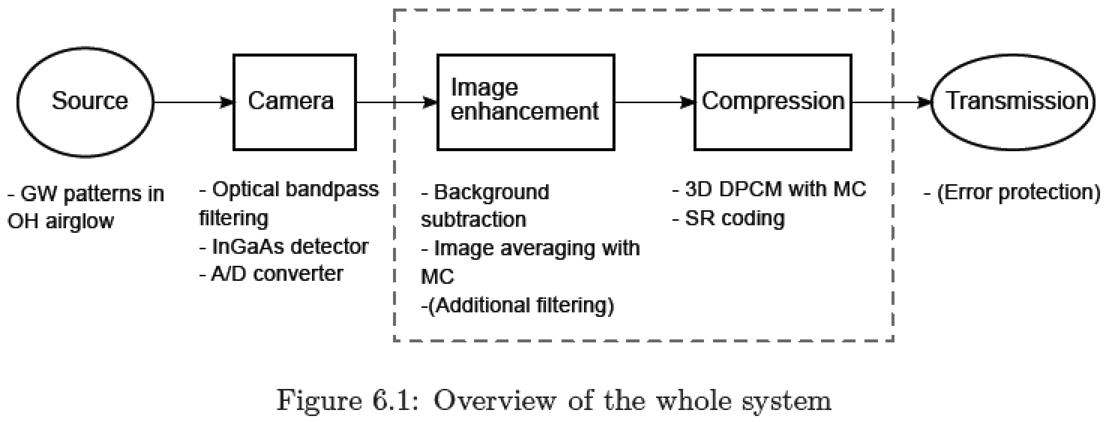

Figure 3.2.: Conceptual system, from Bakken's thesis[Bak12]

To limt the scope of this thesis, only the image averaging part of this will be considered in an implementation.

3.3.2. Background subtraction

Dark current in a CMOS camera will produce a fixed noise pattern on any images produced. The simplest way to remove this from the images is to obtain an image with the shutter closed, and use the noise imprint from this image to subtract from any images taken with the camera.

3.3.3. Image averaging

When producing an image of a faint phenomenon, a long integration time is required to get an image with acceptable signal-to-noise ratio. The usual way of doing this is to take a single image with a long shutter time. The problem with doing this in the satellite is that this would result in motion blur from the motion of the satellite itself. To avoid this, the camera will instead take several pictures in a series and average them to obtain a single image with good SNR. This avoids the problem with motion blur, but introduces a new one: figuring out how much and how a picture has shifted since the last one was taken.

3.3.4. Motion compensation

The motion of the satellite between images has to be compensated for, as mentioned above. Depending on the type of motion, this can be a potentially complex problem in terms of finding how images in the series should overlap. There is also the problem of detecting the motion of the satellite and the image, but that won't be explored in this thesis. [Bak12] presents some suggested algorithms for finding a match between images.

There are two kinds of motion between the camera and the object of interest (gravity waves in the atmosphere); the sideways movement of the satellite along its orbit, and the rotational movement of the satellite around its center of gravity.

Sideways movement is not avoidable, but may not prove a problem as long as it does not move too fast. It is also easy to compensate for, as images only need to be overlapped where they have a common subject.

Rotational movement is far trickier to compensate for, and will not be touched on in this thesis besides acknowledging that it exists as a potential problem.

4. Potential camera interfaces

4.1. Overview

When considering what camera to use in the satellite, ease and practicality of integration depends mostly on what interface the camera uses for control and readout of images. Interfaces vary in what physical link layer they use, and in how much physical space they require. The end result should in any case to obtain a frame from the camera, which is the part of the design this chapter will deal with.

The three most common ways of interfacing with cameras (USB, Camera Link, and direct integration) will be looked at and evaluated for their complexity and space usage. It will also be examined generally how they could be connected to a processing module

4.2. USB

4.2.1. Bus specification

USB (Universal Serial Bus) is a very common interface in peripheral devices designed to hook up to a PC. It is a four-wire, serial, single-master multiple-slave bus. In a typical USB application a “host” (typically a PC) will control one or more “devices”. Depending on the version of USB used (low speed, full speed, high speed, and super speed) the bus has a bandwidth of between 1.5Mbit/s (USB 1.0) to 480 Mbit/s (2.0) to 5 Gbit/s (3.0). [usb11, usb00]

4.2.2. Link to processing module

Making use of the USB interface will most likely require an external IC to convert to an inter-IC bus such as UART or SPI (which usually already exists, or can be easily implemented on a FPGA), or to some sort of parallel bus. A big problem is that since the camera is a USB “device”, the image acquisition has to act as a USB host. This leads to an increase in complexity and difficulty of implementation.

4.2.3. Implementation suggestions

As shown in [AA11], a USB 2.0 interface can be implemented using a CY7C67300 chip acting as a host controller. While they have chosen to use an embedded micro-controller to acquire the image data from the USB controller, it can also be connected to a simpler FPGA design as the USB controller has an onboard microcontroller that should be able to .

4.3. Camera Link

4.3.1. Camera Link specification

Camera Link is a standard for image transmission from cameras to frame grabbers based on the channel link physical link layer. It transmits parallel pixel data by using LVDS, a high speed serial link. It has three configurations; base (24 data bits wide), medium (48 data bits), and full (72 bits). The base configuration contains 4 LVDS channels for image data, 1 for clock, 4 for camera configuration, and 2 for transmitting and receiving serial communication. The other configurations add 4 channels for data and 1 for clock each. The receiving design will need to deserialize and sort the data.

The possible clock speeds for this kind of communication is between 20 MHz and 85 MHz, achieving data rates of between 2.04 Gbit/s (base configuration) to 5.44GBit/s (full) at 85 MHz. On each clock cycle it transmits one pixel regardless of the bit size of each pixel. In addition it transmit three bits; frame valid, line valid, and data valid. Frame and line valid

To configure and communicate with the camera the standard provides four control inputs and two serial communication lines to the camera. The serial communication uses the RS232 standard with 1 start bit, 1 stop bit, no parity and no handshaking settings.

Fig. 4.1 illustrates the physical layer of the Camera Link standard.

[cam00, cama, camb]

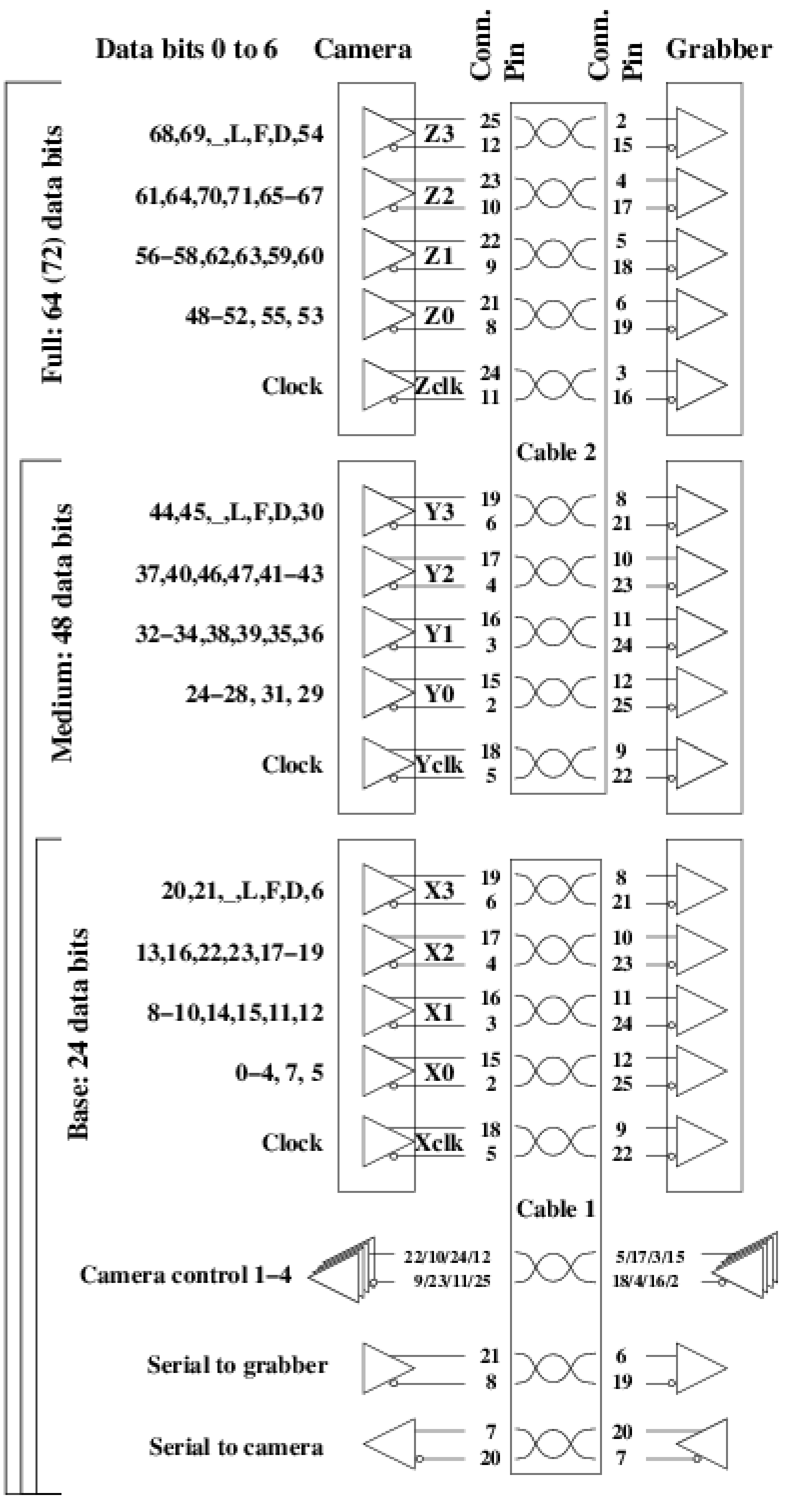

Figure 4.1.: Illustration of camera link layer

4.3.2. Link to processing module

The Camera Link specification is based upon using the National Semiconductor Channel Link SerDes series of chips, so using a receiver chip would give you a parallel TTL output to deal with. If using a FPGA with LVDS capability however , this deserializer could more easily be implemented on the FPGA itself (as has been done in [WGLC13]).

4.4. Direct integration

A common option among small cameras is direct integration with a pin-out version. This means communicating directly to the camera on an IC to IC level with a standard TTL parallel output. As there is no standard for readout among cameras however, coming up with a standard solution for this kind of interface is not really possible. It should however be pointed out that this is the preferred interface if using a microcontroller, as it needs no converting or go-betweens like USB and Camera Link.

5. Payload to satellite interface

5.1. Overview

All modules in the NUTS test satellite connect to each through the backplane. This chapter will review how to connnect to the backplane and what is required to communicate with the rest of the satellite.

5.2. The backplane

The backplane provides the power and common communication interface between the modules in the satellite. For power there is a 3.3V and a 5V line. The backplane uses a I2C bus to communicate between modules. The "active" modules use the CubeSat Space protocol to communicate[RB13, Bru11]. Fig. 5.1 shows how the module interface is logically connected while Tab. 5.1 lists the functionality of the backplane connections.

Name	Width	Function
3V3_MODULEn	1	5 V line from the EPS
5V_MODULEn	1	3.3 V line from the EPS
I2Cn	2	I2C bus lines
IDn	3	Module ID
DBGn	4	JTAG programming and debug line
RESETn	1	Active low module reset

Table 5.1.: Backplane connections

5.3. Requirements

The module will need to be able to take commands from the master modules in the satellite, and to be able to transmit the images from storage within the module to the a receiving module in another part of the satellite. To do this it will have able to implement a I2C bus interface. Depending on whether it needs to act as a master on the bus, it might need to be able to implement CSP. It will in addition need

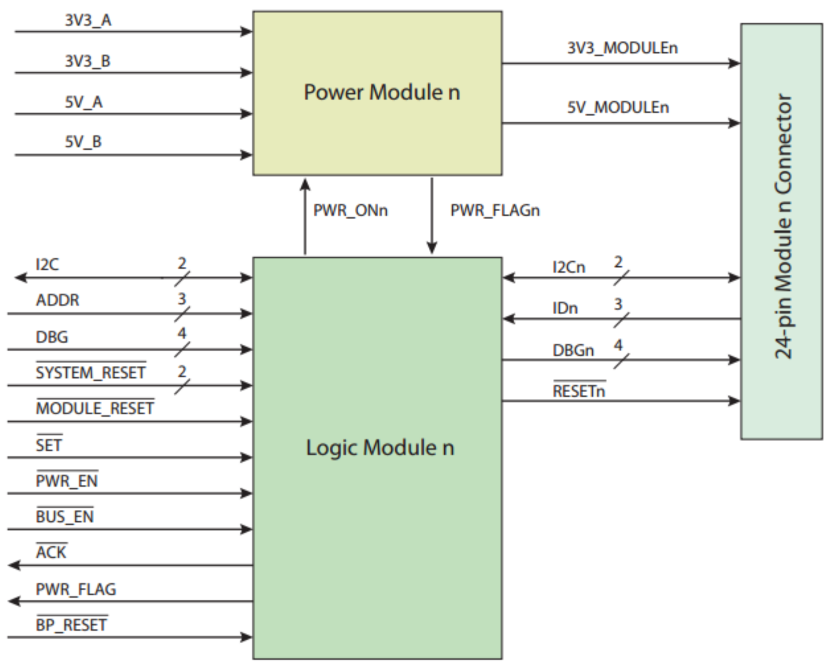

Figure 5.1.: Backplane logic unit (from[Bru11])

to provide a ID to the logic module. Whether it provides a JTAG interface to the backplane depends on what processing module is chosen.

5.3.1. I2C

I2C stands for "Inter IC", and is commonly used for communication between integrated circuits situated on the same printed circuit board. It is a half duplex, serial, two-wire, multi-master bus which can achieve data rates up to 400 kB/s. The bus implemented on the backplane has been proven to be capable to run at this rate, although this will likely not be achieveable in the fully integrated satellite. Two big advantages of I2C bus is that it has automatic arbitration between masters, and low power consumption. The big drawback is the low bandwidth of the bus. [i2c00, Vol11]

The I2C bus is common enough that most microcontroller has I2C ready to use. It can also be relatively simply be implemented on an FPGA. As the payload module should act independently of other modules without requiring to talk to other parts of the satellite, it should be sufficient to implement this module as a slave. This also avoids the overhead of using CSP, as the master module simply addresses the payload as a slave on the bus.

5.4. Module command interface

As the module will act mostly independently, it does not need to have a very extensive command set. The data it needs to operate is when it needs to start a image series, if it should go into and out of hibernation, and a command to abort imaging in progress. In addition it needs to be able to report its status, so other modules can determine whether it is ready to transmit new image data. This can be accomplished by having a status register that can be read by a master.

Tab. 5.2describes a preliminary set of commands the module should implement.

Command	Description
BEGIN	Begin image acquisition
HIBERNATE	Go into low power mode
ABORT	Abort image acquisiton in progress

Table 5.2.: Commands

5.5. Image transmission bus

While the backplane provides a ready bus to use when transmitting images to the data handling part of the satellite, the limited bitrate of this bus (max 400 kB/s at best) might be insufficient to use as a image data bus. Not only will this act as a bottleneck in how often images can be taken, it will also take a hold of the primary communication bus of the satellite for an inordinate amount of time, preventing other modules from talking to each other.

The solution then could be using a separate bus that connects directly to the OBC, using for instance SPI (which has much higher potential data rates), or the experimental internal wireless bus. For this reason it can be interesting to have the payload module be part of the wireless bus, in case images can be offloaded of the payload using this bus.

6. Data rates and storage

6.1. Overview

As an image series is captured, it would be advantageous to perform the background subtraction and image averaging as the pictures are acquired from the camera, as this saves on storage resources both in the processing module and external storage. This chapter gives an approximation of what throughput and storage space is required, based on reasonable assumptions of image format and mode of usage.

6.2. Image acquisition and storage

6.2.1. Assumptions

To limit the range that needs to be considered, the max resolution, framerate and pixel resolution (AD conversion resolution) from the XSW-640 specifications has been used. This camera was considered for use in the satellite, but was discarded for cost reasons.

Max resolution	640x512
AD resolution	14 bits
Max framerate	50 Hz
Integration time	1 s

Table 6.1.: XSW-640 specification[xsw]

When talking about “full resolution” and “full framerate” later, this refers to these specifications. As these are well above the requirements listed in previous works, using these figures are considered the ceiling of what is relevant to consider.[Bak12, Røn12] 14 bits resolution per pixel is mostly standard in these kinds of cameras, so this is what will be used to estimate picture size.

6.2.2. Image and series size data

160x128	280
320x256	1120
640x512	4480

Table 6.2.: Size per image vs image size [Kbits]

Tab. 6.2 gives the size per image and consequently the minimum size of the memory needed contain one image series when using simultaneous averaging.

	10 Hz	20 Hz	30 Hz	40 Hz	50 Hz
160x128	2800	5600	11200	22400	44800
320x256	11200	22400	44800	89600	179200
640x512	44800	89600	179200	358400	716800

Table 6.3.: Total size of image series as a function of series size vs resolution [Kbits]

Tab. 6.3 lists the total size an image series, and signifies the amount of storage needed to store an image series without simultaneous averaging

6.2.3. Bus frequency and data rate

	10 Hz	25 Hz	50 Hz
160x128	204 800	512 000	1 024 000
320x256	819 200	2 048 000	4 096 000
640x512	3 276 800	8 192 000	16 384 000

Table 6.4.: Pixels per second, Framerate vs. Image size

Tab. 6.4 are the number of pixels needed to transfer per second, effectively translating minimum number of camera bus transactions assuming one pixel is sent per transaction.

Tab. 6.5 represents how many store operations would be needed per second to store incoming data.

Tab. 6.6 represents the amount of data the processing module would need to work through to process an image series.

Word size [bits]	Operations pr second	Time [ns]
8	32,768,000	30.5
16	16,384,000	70

Table 6.5.: Time window per word stored vs. word size (full resolution, full framerate)

10 Hz	25 Hz	50 Hz
44800	112000	224000

Table 6.6.: XSW full resolution throughput [Kbits]

Up to ... samples	Accumulator bit size
4	16
8	17
16	18
32	19
64	20

Table 6.7.: Accumulator bit size vs number of samples

To accommodate overflow when accumulating the pixel values, the accumulators need to be of bigger length than the input values. Tab. 6.7 gives an overview of how big an accumulator needs to be vs how many images need to be added together. Additional bit requirements are calculated:

$log_2(NumbeOfSamples) = AdditionalBitsNeeded$ (rounded up)

6.2.4. Minimum transfer I2C transfer time

Resolution	Transfer time [s]
160x128	0.82
320x256	3.2768
640x512	13.1

Table 6.8.: I2C transfer time for raw images

Tab. 6.8 gives an overview of how long it would take to transfer an image from the payload to another part of the satellite. This assumes each pixel was sent as two transmissions and the transmission went a constant 400 kB/s from start until end.

6.2.5. Example series of images

Below are some samples of different configuration of image resolution and framerate, and what demands are placed on the system in these cases. The minimum required memory is images are averaged as they are acquired. The minimum system frequency means the minimum frequency needed to store and load a pixel per bus cycle, assuming each pixel can be stored and loaded in one cycle. Accumulator bit size means how big the pixel value accumulator needs to be to avoid overflow.

6.2.5.1. "Worst Case": 640x512, 50 Hz

Minimum system frequency: 32,768,000 Hz

Minimum required memory: 4480 Kbits

Accumulator bit size: 20 bits

6.2.5.2. "Average case": 320x256, 25 Hz

Minimum system frequency: 32,768,000 Hz

Minimum required memory: 1120 Kbits

Accumulator bit size: 19 bits

6.2.5.3. "Optimistic case": 160x128, 10 Hz

Minimum system frequency: 32,768,000

Minimum required memory: 280 Kbits

Accumulator bit size: 19 bits

7. The space environment

7.1. Overview

The space environments offers extra challenges in designing electronic devices compared to earth based devices. The biggest difference lies in the extra destructive radiation that results from not being protected by the earths atmosphere, which is made worse by the fact that the device will be inaccessible once operational in space. This means that not only is it important that the design is thoroughly verified to be correct, it also needs to have the ability to self-correct any error that can be induced by space radiation (or other kinds of interference). This chapter will review the different effects resulting from space radiation, and go over some techniques to mitigate and correct the errors can result in digital circuits from these.

7.2. SEE

SEE, or Single Event Effects, is the umbrella under which the effects of space radiation on the function of digital circuits lie. They are called single event effects as they are assumed to be random and non-reoccurring (in the same sense that lightning those not strike the same place twice). When operating in the space environments SEEs are all but unavoidable, and will (according to Murphy's law) most likely strike where they are the most destructive. The three most important kinds of SEEs are SET (Single Event Transient), SEU (Single Event Upset), and SEL (Single Event Latch-up). [nas96][Pet11]

7.2.1. Single Event Upset

Single event upsets occur when a high energy particle causes a memory (for example a flip-flop) to change state, either through direct radiation of the memory circuit or via a SET.

7.2.2. Single Event Transient

Single event transients happen when a high energy particle causes influences the output of a logical gate. If this change is stored in memory, it becomes a SEU.

7.2.3. Single Event Latch-up

A single event catchup happen if a CMOS transistor is hit in such a way that the current generated will activate a “parasitic” transistor, in essence causing a short in the transistor structure. While this can be corrected by cycling the power, the short circuit current may already have caused irreversible damage to the circuit. [lat]

7.3. Mitigation

There are several techniques that can be employed both on the hardware and the software side of the design that will lessen the potential impact a SEE can have. What techniques will be used where will should be evaluated on the criticality of the part that it will be employed on, and should be balanced against the likeliness of failure.

7.3.1. Choosing rad-hard components

The first line of defense against radiation is choosing appropriate techology and devices. Some types of electronics are more sensitive to radiation, where for instance a smaller technology node (90 nm vs 45 nm) has a smaller treshold for the amount of energy needed to cause a SEU. Some manufacturers offer so-called “rad-hard” devices which mean that they have been designed with less sensitivity to radiation in mind. An example of this are the RTAX FPGAs from Microsemi, which are specifically created to survive in a space environment[rta].

7.3.2. Triple modular redundancy

Triple modular redundancy is the practice of having three redundant modules performing the same task, and having the result be decided by majority vote. This can be employed on every level of the system, from having three identical systems performing the same function, down to having specific parts of a FPGA design or memory. It can also be used as a software technique bu performing the same operation(s) three times and comparing the results. The drawback of using this technique is a threefold increase in required space and effect when dealing with hardware, while a software application will potentially take three times as long to complete.

A variation on this technique is having three modules that are functionally identical but implemented differently. Using this technique you not only avoid errors from SEEs, but also from bugs and designer errors.

7.3.3. Memory scrubbing

By adding redundant information in the form of error correcting codes, SEUs in memory may be detected and corrected. This is especially valuable in volatile RAM, which is much more vulnerable than non-volatile memory such as FLASH[nas96].

7.3.4. Sanity checking

In software it is common to have some sort of checking whether results from a calculation is “reasonable”, i.e. if the result is within bounds of what the range of inputs could produce. This can also be applied when designing hardware to ensure that a SEU or SET does not propagate unchecked.

8. Processing module

8.1. Overview

As an independent module, the design has to:

1. Communicate with and acquire images from the camera
2. Perform image enhancement and compression
3. Store until ready to be received by the OBC
4. Take commands from and transmit images to the OBC

In short, it will have to function as a frame grabber and an image processing unit. The previous chapters have reviewed the different options for what the requirements of the system will be. This chapter will attempt to come up with suggestions for how to implement the module, and look at the drawbacks and benefits of the different solutions.

8.2. Review of requirements

Using the theoretical background established so far, it can be determined that the central processing module needs to be able to:

- Implement I2C slave interface
- Needs to be able to communicate with and configure camera through serial communication
- Perform image acquisition using a high speed data bus
- Be able to handle image processing in a reasonable amount of time

Ideally, it should also be able to:

- Be radiation tolerant
- Do image enhancement as images arrive
- Be able to enter low power mode

8.3. Data logistics

To conserve memory resources, it is preferable to keep an even data flow through the system. Looking at the system from a data flow perspective:

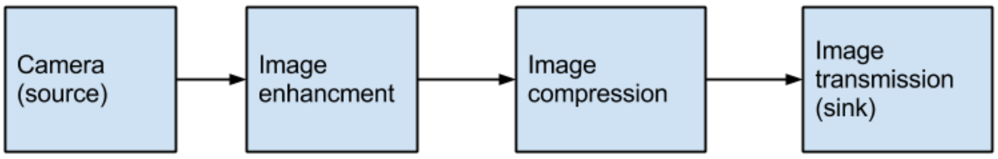

Figure 8.1.: Data flow in module

The same as described in[Bak12].

Expanding and adding in the known components:

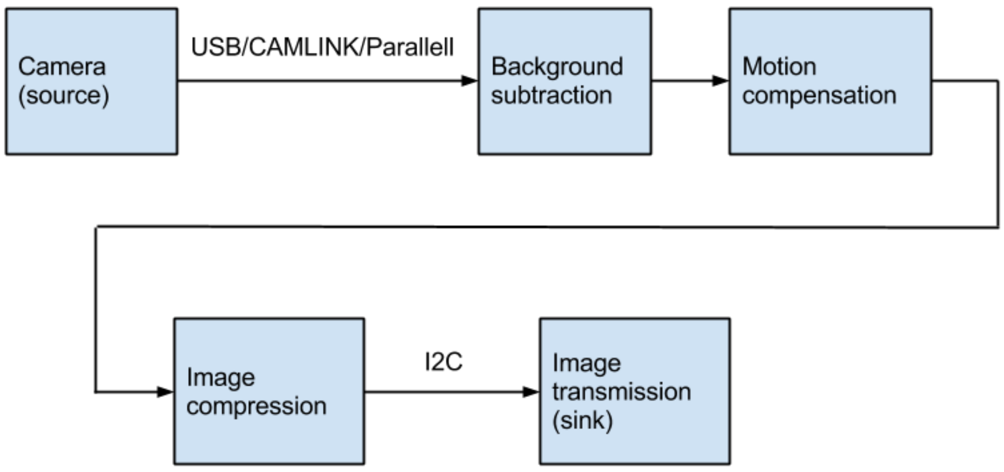

Figure 8.2.: Data flow 2

The goal then becomes to minimize bottlenecks that would require extra storage space. Not only does this save on resources, it also gives room for greater potential output by reducing the amount of time that needs to be between imaging series. As identified earlier, the I2C bus serves as a big bottleneck when it comes to freeing up space for the module, so a efficient pipeline will serve to free up more space to hold compressed images until they are ready to transmit. With this in mind, we look at the options available for processing.

8.4. Processing

Working with image processing, especially in real time, is an inherently computationally demanding task. Working with large arrays of data is often handled with specially designed circuits, or handled by special microcontrollers called Digital Signal Processors (DSP). A good alternative to the specially designed circuit is using FPGAs, especially when considering cost and flexibility.

The greatest difference between DSPs and FPGAs is that a DSP works serially, while a FPGA has the potential to perform truly parallel computation. A brief overview of drawbacks and benefits of using each are listed below.

8.4.1. FPGA

Using a FPGA gives a great advantage in terms of throughput, as several parts of the image processing can be performed simultaneously. It gives great flexibility in creating a design that suits the purpose of the system. This flexibility may even allow the whole module to be implemented on a single chip, which will give fewer points of failure. The module overall does not require a complex control structure, so a FPGA design can be implemented efficiently and in a modular fashion.

SRAM based FPGAs are vulnerable to SEEs, potentially causing the design to get corrupted. This will require some form of mitigation, which will result in a more complex design. It will also be harder to implement some sort of module-wide power saving solution, as the FPGA cannot power itself down.

8.4.2. DSP

DSPs have the advantage of having an architecture specifically geared towards signal processing. Most DSPs allows working on large sets of data using special instructions, and typically have some form of direct memory access to allow peripheral devices to directly store to memory. Serial communication such as I2C and UART come standard in most of these controllers, there is no need to implement a special solution for these. Practically all controllers have some sort of power saving mode, which would allow easy implementation of a hibernation mode for the module. This would also most likely allow lower power consumption.

DSPs are at an disadvantage performance-wise compared to a FPGA. As they are at heart microcontrollers, they can only work serially despite having powerful instructions. As there does not need to be complex control systems in the module, control structures in the program code will mostly be overhead. It would potentially have to run at a high clock speed to be able to load and store data for the image averaging process while receiving image data.

Part II.

Implementation

9. Example implementation

9.1. Overview

9.2. Assumptions

As the the specifications for the system are not fully ready, a number of assumptions have to be made. These will be based on information from the image processing theses and educated guessing based on my experience and research.

9.2.1. Framerate and resolution

Resolution: 320 x 256, close to and a bit more than what is recommended in [Bak12],[Røn12]

Number of samples: To avoid having to do actual division, the number of samples has been chosen as 16. In digital systems, right shifting a binary number four spaces is equivalent of dividing by 16.

9.2.2. Camera interface

The assumed interface is Camera link, as this is a common interface from my experience with looking at different IR cameras. It also allows a more efficient and easy implementation than USB or direct integration, and would be the preferable interface given the choice.

The transfer clock is assumed to be 20 MHz, as this is the lowest transfer clock supported by Channel Link chipsets while also surpassing the requirements of transfer bandwidth posed by the framerate and resolution assumptions.

Number of pixels * framerate = $320 * 256 * 14 * 16 = 1310720$ pixels per second.

9.2.3. Satellite interface

It is assumed that the payload can be accessed as a I2C slave.

Spec	Value
Framerate	16 Hz
Image resolution	320x256
Camera interface	Camera link
Satellite interface	I2C slave

Table 9.1.: Specification assumptions

9.3. Processing module

The first decision to be made is on what the processing module will be, as this will determine how the rest of the system will be implemented. The major part is whether to use a FPGA or a DSP. As outlined in sec. 8.4.1 and sec. 8.4.2, FPGA has the advantage of greater throughput and parallel computation. Considering the amount of data that needs to be processed simultaneously while also storing and loading to external storage, using a DSP becomes risky. The processing module needs to do motion compensation, image acquisition and storage at the same time, which leaves a controller with little leeway unless running at a high clock frequency. It would also mean that the camera link interface would need to be deserialized using external chips, which introduces more points of failure into the system as well as increasing the size of the design. A FPGA on the other hand would not have any of these problems, and would have resources to spare for further processing without risking it interfering with with image acquisition and enhancement.

For these reasons, a FPGA will be used as the main processing module in the project.

9.4. Camera interface

9.4.1. Physical link

To physically connect to the FPGA, the output port from the camera needs to be linked to the circuit board via cable, which will connect into a port mounted on the circuit board. From the port mount, 16 LVDS lines need to be connected to LVDS compatible ports on the FPGA.

Care should be taken to avoid different trace lengths in the data and clock lines to avoid skew. These should also be kept physically separate from other parts of the board due to the high frequency signals.

For general layout design rules with regard to channel/camera link, refer to [cha].

9.4.2. FPGA camera link interface and camera configuration module

The module that is receiving from the camera will need to act as a deserializer, recovering clock from the clock line, using a PLL to multiply the frequency by 7, and using it to capture bits from each of the four data lines into shift registers. This in itself can be a challenge, as there is no common standard for clock alignment.

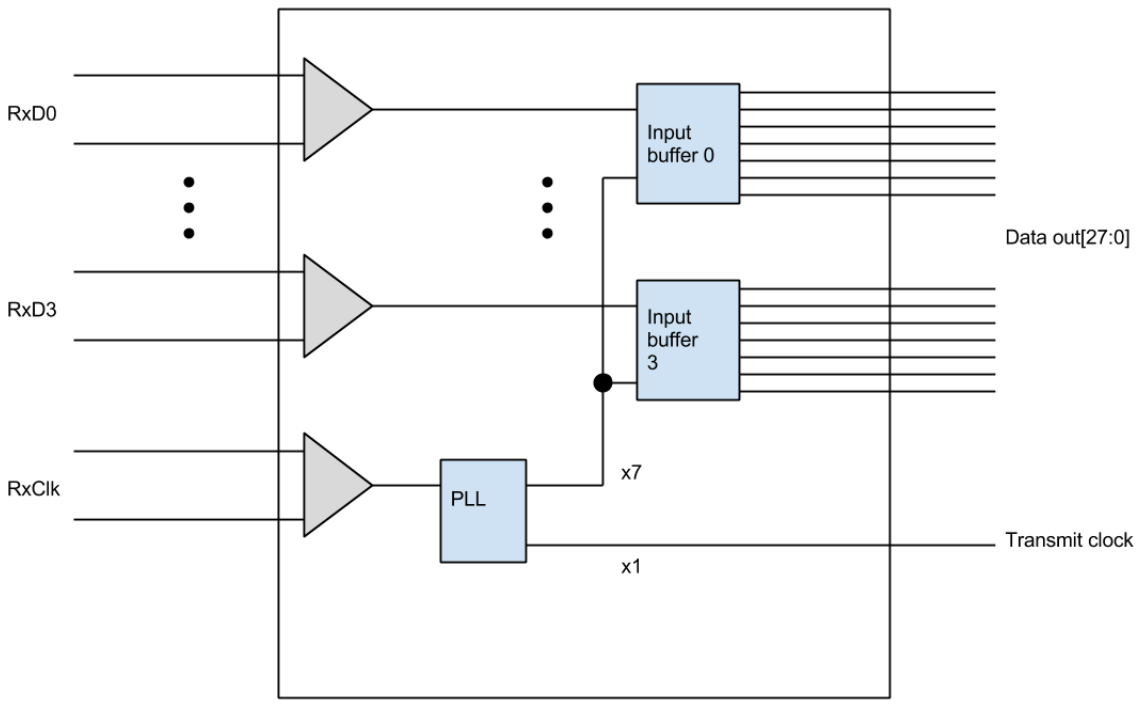

Figure 9.1.: Camera link deserializer

The four camera control signal lines have their usage defined by the camera producer, and will need to have their behavior defined accordingly as there does not seem to be any common practice.

The serial communication line can be implemented as a RS232 compatible serial communication device. Finding a simple UART module that can be interfaced to a camera configuration state machine should be sufficient to build a mostly self sufficient camera configuration design module.

The settings for serial communication are one start bit, one stop bit, no parity and no handshaking.

[cama]

9.5. Image enhancement and storage

9.5.1. Image averaging

Each transmission from the camera will give 18 bits of data, of which three are control signals and 14 are pixel data. As the receiving module is done storing these temporarily, the next part of the chain should start working immediately in averaging this pixel data with the values of the other images in the series, and then store the new average onto temporary storage. The data valid bit should be sampled to be certain that there is actual image data that is being received, as the transmissions will be padded out with dummy data between each image.

The averaging is performed by accumulating the pixel values per pixel. As pixel data is received from the camera, it is added into an accumulator for each pixel. To accommodate the potential extra bit length, $log_2(16) = 4$ extra bits will be needed. Referring toTab. 6.7, the accumulators and the adder need to be 18 bits. The result is then obtained by shifting the result of each 4 places to the right to divide by 16 when a picture series is finished. This can be accomplished by just loading the leftmost 14 bits of the 18 bits from memory before storing them again, or just transmitting the leftmost 14 bits.

9.5.2. Motion compensation

Assuming the displacement from picture to picture is known in terms of pixels, the images can be aligned by adding or subtracting from the address of the memory. This means that for each image, the address for the memory needs to be incremented by one to reflect the change in overlap (assuming the pixel data is loaded from left to right, top to bottom).

While the mechanics of the compensation of more complex motion is the same (adding or subtracting from the accumulator address), finding the number to shift by is more difficult, and may need to be done on a pixel by pixel basis (for instance when considering rotational motion). If the difference can be found as a vector however, it is simply a matter of deconstructing the vector and translating it to the equivalent pixel position. Pixels that do not “overlap” can be discarded.

This implementation assumes no compensation is needed.

9.5.3. Storage

To preserve memory in the FPGA while processing, the images will be stored in external memory. This will be accomplished using SRAM, as it provides good speed and low power consumption, and as the memory requirements are not very great. This also reduces the complexity of the memory controller, as the memory does not need to be clocked. To accommodate the 18 bit accumulators, two SRAM chips will be used in parallel, one with a 16-bit word size and one with a 8-bit word size, combining together to get a 24-bit word size memory. To contain a single image this memory needs to have at room for at least 81920 words, rounding up to 128K word memory size.

The memory controller will need to keep track of memory depth and storing and retrieving pixel data to be processed. It will also need to provide data to be transmitted to the I2C control module as needed.

9.5.3.1. Image channel out of module

To transmit a raw image over the I2C bus will in the best case take 3,2768 seconds (Tab. 6.8). In this implementation it is assumed that this is not a problem, but should it be necessary to perform "burst" imaging, more memory would need to be added to hold more than one image series.

9.6. Satellite interface

The command interface can be implemented by building a serial interface that receives write/read command and register address, and receives/transmits the appropriate data. To keep things simple there will be three registers available to the master in the transaction: one writable command register, one readable status register, and one data register to read out image data.

The command register will be hooked directly up to the main control module to simplify the logic.

The status register will contain information about the state of the processing module, as well as the state of the image data store.

The data register will contain new image data as long as it is available, and will automatically load new data when it is read.

sec. 10.3 contains the full command and status list.

9.7. Top level control

The top level control module is the module responsible for keeping track of the status of the payload module, acting on commands received by the I2C module, and providing control signals to the other parts of the design.

9.8. SEU mitigation

The FPGA should either have some sort of ECC checking on the configuration memory to prevent a SEU upset from corrupting the design, or have a design memory that is hard against SEU (such as FLASH memory).

In the FPGA design the parts that are critical to continued operation, such as control and status, will be triplicated using TMR on a design module level. In practice this means the I2C interface and main control.

The data acquisition is not considered critical enough in this instance to warrant full TMR on a device or design level, as a SEU in the datapath will not interrupt continued function of the payload. If a part of the image processing becomes faulty as a result of a SEU, it can be reset by the main control.

In the event of a failure of the payload, the biggest possible consequence for the rest of the FPGA is that it starts putting out random signals on the I2C bus. This can be corrected by resetting the payload and reconfiguring the FPGA.

9.9. Power saving

As the payload control is contained in a single device, putting the entire module in a complete shutdown state is not possible without adding some sort controller module external to the FPGA. The FPGA main control in the design will instead clock gate parts when a HIBERNATE command is issued. Depending on the camera it can also be relevant to power gate the power supply to the camera.

9.10. Clock domains

To be able to store the result of one transmission and load a new one for the accumulator before a new transmission arrives, the main clock for the system is twice the transmission clock, meaning the system clock is 40 MHz. This allows for a simplified store/load mechanism, as a store/load will be performed each clock cycle until accumulation is done.

The output of the deserializer will be double buffered to prevent metastability when crossing clock domains.

If the system needs to run at a higher clock frequency, the memory controller will need to start sampling the transmission clock to determine whether new data has arrived.

9.11. Choosing FPGA

9.11.1. Requirements

The two critical requirements to consider with an FPGA is how many resources the design will use and what kind of interfaces will be used.

9.11.1.1. Resources

As the design consists of three parts, each part can be generally looked at separately. While this does not necessarily give a very accurate assessment, it will provide some ground to estimate how much will be required. These figures will be found by looking at previous results from others.

The I2C slave implemented in [sc] uses around 70 LUTs when synthesized for a Lattice semiconductor device. To give headroom, this will be rounded up to 100 LUTs.

The camera link interface in [Mic] lists the typical resource usage as 280 LEs for a Camera Link base receiver when synthesized for a Altera device. This will be approximated to 300 LUTs, as LEs are the most basic logic block in Altera devices and contain one four input LUT.

The control structures, such as top level control state machine do not take up significant resources, and 50 or so LUTs should be more than sufficient.

The image averaging and memory controller process will need to keep track of and control the memory of the processing module. 100 LUTs has been reserved for this part.

To accommodate further addition to the design (such as a compression scheme), the resource requirement is doubled to account for the possibility of resource consuming encoding schemes.

All in all:

Part	LUTs
I2C	100
Camlink receiver	300
Memory and averaging	100
Control	50
Total (x2)	1100

Table 9.2.: Resource estimation

9.11.1.2. I/O interfaces and pins

The parts of the design that require I/O are the I2C module, the Camera Link interface, and the memory controller.

I2C requires two in/outputs, each with a tri-state.

The Camera Link interface requires 5 LVDS inputs for data and clock, 4 outputs for camera control, and one input and output for serial communication. This totals up to 11 lines, or 22 pins as it is a differential interface.

The memory controller will need to communicate with a 24-bit memory, using a 32-bit address. Together with various control signals, this amounts to about 60 I/O pins.

9.11.2. Priorities

The preferred qualities that will be looked at when making a selection, in no particular order, are

- Low power
- Relatively low cost
- Radiation hardness
- Built in SEE mitigation
- Easy to use/familiar tools
- Ready to use IP cores

9.11.3. FPGA comparison

Three FPGAs for this implementation has been considered, each from a different vendor; Xilinx, Altera and Microsemi. Each of these satisfy the approximate requirements of the design, and are approximately equal in terms of core statistics and price. A thing to note about the core statistics is that no vendor use a common unit for resource, so the closest comparable stat has been selected for comparison. The definition of the units are defined below each table.

9.11.3.1. Altera: Cyclone IV

1 Logic element = 1 4-input LUT + 1 flip-flop

Benefits:

- Easy and familiar tool set

Logic Elements	6,272
Embedded Memory (Kbits)	270
Approximate price	20 USD[alt]

Table 9.3.: Core statistics for Cyclone IV EP4CE6[cyc]

- Hard configuration SEE mitigation

Drawbacks:

- Not the strongest in any category

9.11.3.2. Xilinx: Spartan 6

Logic cells	9,152
Flip-flops	11,440
Block RAM (Kbits)	576
Approximate price	20 USD[avn]

Table 9.4.: Core statistics for Spartan 6 XC6SL9

1,6 logic cells = 1 6-input LUT (supposedly to account for LUTs being 6-input instead of 4)

Benefits:

- Best value in terms of resources vs price
- Soft SEU mitigation
- One of the major vendors, easy to find support

Drawbacks:

- No hard SEU mitigation.

9.11.3.3. Microsemi: ProAsic3

Versatiles	6,144
RAM (KBits)	36
Approximate price	20 USD[avn]

Table 9.5.: Core statistics for ProAsic3 A3P250[pro13]

1 versatile = 1 3-input LUT OR 1 flip-flop

Benefits:

- Built in FLASH configuration

The major advantage of the Proasic3 is using built in FLASH memory as configuration memory. This makes the configuration immune to SEUs, and does not require external configuration storage and programming. It also allows for a much faster start up.

Drawbacks:

- Small vendor, hard to find support
- Worst value in terms of resources vs price

9.11.4. FPGA selection

The FPGA selected for this implementation is the Altera Cyclone IV EP4CE6, chosen ultimately for the familiarity with the tools and the device family, allowing faster integration and implementation. It also has the advantage of the automatic SEU mitigation in configuration memory. While the Proasic3 has better SEU mitigation overall, the payload module is not critical for the for the overall working of the satellite and therefore this is not considered the highest priority. Better to spend some extra resources triplicating control structures in the Cyclone IV, especially since the Proasic3 has roughly half the resources of the other FPGAs. While the payload might fail eventually, it can be restarted or reconfigured without general loss of functionality in the satellite.

9.12. Other components

9.12.1. SRAM

As the design will run on a 40 MHz clock, the memory will need to have access times less than $\frac{1}{40MHz} = 25$ns. Otherwise there are no other special requirements, but having low power consumption is a plus. This implementation is based on using the CY7C1011 (16-bit 128K) and CYC1019 (8-bit 128K)

In reality it would probably be just as well to use what is on hand, as long as the access times are sufficiently short and the physical interface in the FPGA is adjusted.

9.12.2. Configuration hardware

To load the configuration data into the FPGA some sort of external programmer is needed. To make things simple this implementation uses the EPCS4, a serial configuration device made by Altera specifically for programming Altera FPGAs using Active Serial configuration. It contains a 4-MBit FLASH memory, with the design configuration file taking up approximately 2,944,088 bits at maximum uncompressed.

9.13. Space requirements

Device	Part	Package	Size [mm]
FPGA	EP4CE6	144-pin EQFP	22 x 22
Configuration controller	EPCS4	8-pin SOIC	6 x 4.9
8 bit SRAM	CY7C1019DV33	32-pin TSOP	20.95 x 11.76
16 bit SRAM	CYC71011DV33	44-pin TSOP	18.5 x 12

Table 9.6.: Device sizes

Total required area with the packages chosen is approximately 50 x 40 mm, well below the constraint of 66 x 123 mm. In addition, all of the devices come in smaller packages, so this area can be reduced if necessary.

9.14. Cost estimate

Device	Part	Cost [USD]
FPGA	EP4CE6	17.93
Configuration controller	EPCS4	13
8 bit SRAM	CY7C1019DV33	1.42
16 bit SRAM	CYC71011DV33	2.54
Total		$\approx$ **36**

Table 9.7.: Device costs

The cost estimate in Tab. 9.7 is based on prices listed on [alt, avn], and is simply meant as a rough guide. The main point is that device cost for the module is quite low.

10. FPGA design

A FPGA design has been planned and partially implemented. This section will give an overview of this design.

10.1. Data flow and control overview

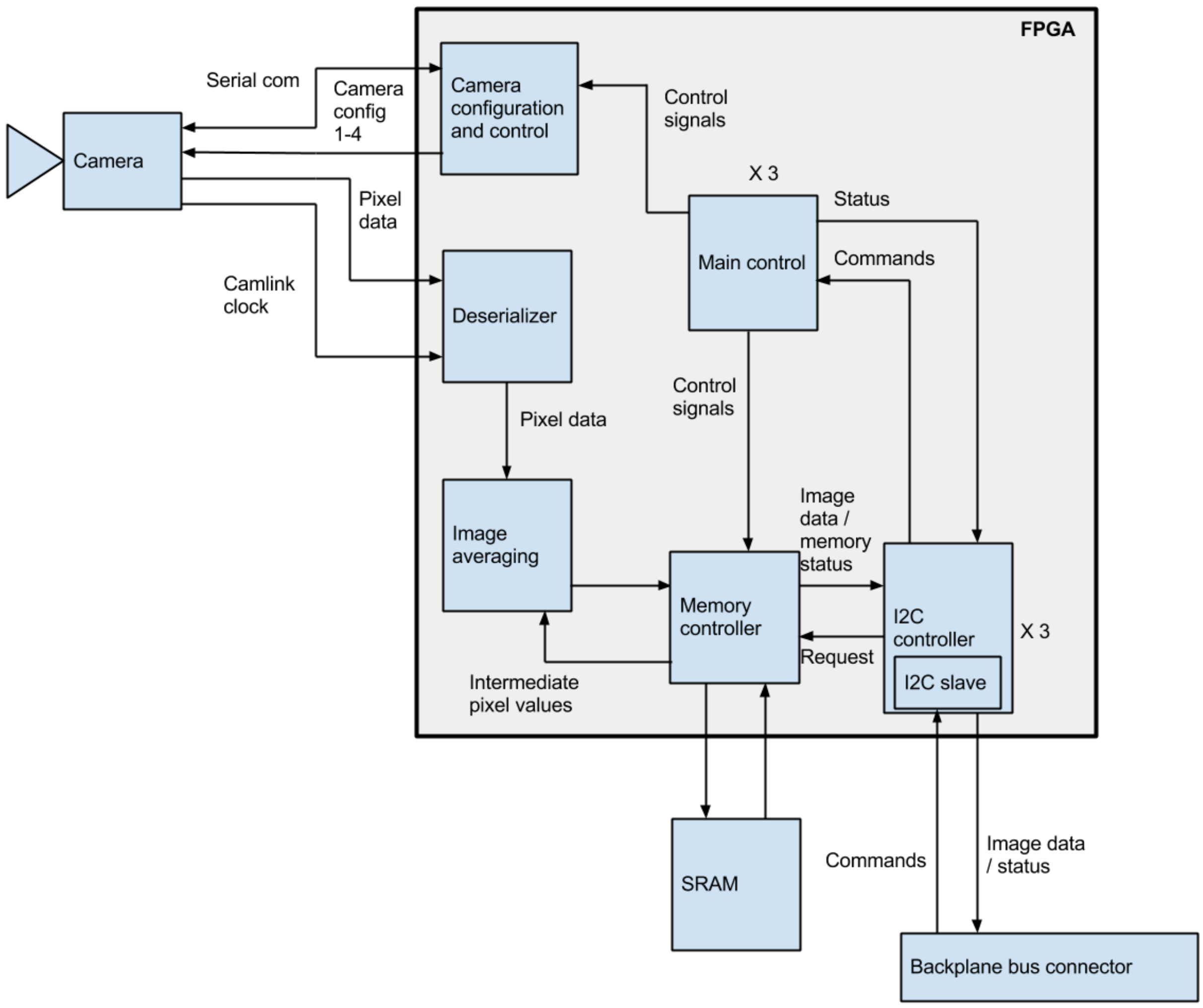

Figure 10.1.: Diagram of data flow and control

This graph gives a rough overlook of how the different design parts should interact.

Typical use case from power on to data transmitted:

1. Design loaded, all modules reset
2. Camera configured by camera configuration module, signals main control
3. I2C receives "Begin imaging command" from I2C master, signals camera config and memory controller to start
4. Deserializer starts receiving data, puts out to accumulator every transfer clock
5. Memory module stores and loads accumulated data to combine image series
6. Once the image series is finished, memory module signals main control module
7. I2C receives read command from I2C master and starts reading out data

10.2. Design Modules

10.2.1. I2C controller and slave interface

Name	Input/output	Size	Function
clk	input	1	Clock input
nreset	input	1	Active low reset
status	input	3	Status output from main control
data_in	input	16	Data from memory module
mem_state	input	5	Memory state from memory controller
req	output	1	Request new data from memory module
command	output	3	Output from command register to main control
sda	inout	1	I2C data line
scl	input	1	I2C clock line

Table 10.1.: Interface for I2C controller

The I2C control module consists of an I2C physical interface and a control module responsible for loading in data from the memory module.

The I2C physical interface is a modified version of an I2C slave design by [Fie]. The parts that are modified are the serial interface (to signal that the output buffer has been loaded) and the register interface (to implement custom read and write registers).

It contains three communication registers: command, status and data. The command register is takes any command from the a master module and sends control signals directly to the main controller.

10.2.2. Main control

Name	Input/output	Size	Function
clk	input	1	Clock input
nreset	input	1	Active low reset
avg_done	input	1	“Averaging finished” signal from memory module
data_in	input	1	Data from memory module
command	input	3	Command input from I2C module
config_done	input	1	“Configuration of camera done” signal from camera control module
mem_status	input	5	Memory status from memory module
cam_config	output	1	“Configure camera” signal to camera control module
start_im	output	1	“Start imaging and averaging” signal to camera and memory control
enable	output	1	Enable signal to allow clock gating unneeded parts of the design
sreset	output	1	Subsystem reset to allow resetting other modules
status	output	3	I2C data line (connected to pin)

Table 10.2.: Interface for main controller

The main controller is the module that is responsible for keeping the state of the design and providing control signals to the other design modules.

10.2.2.1. State machine diagram and description

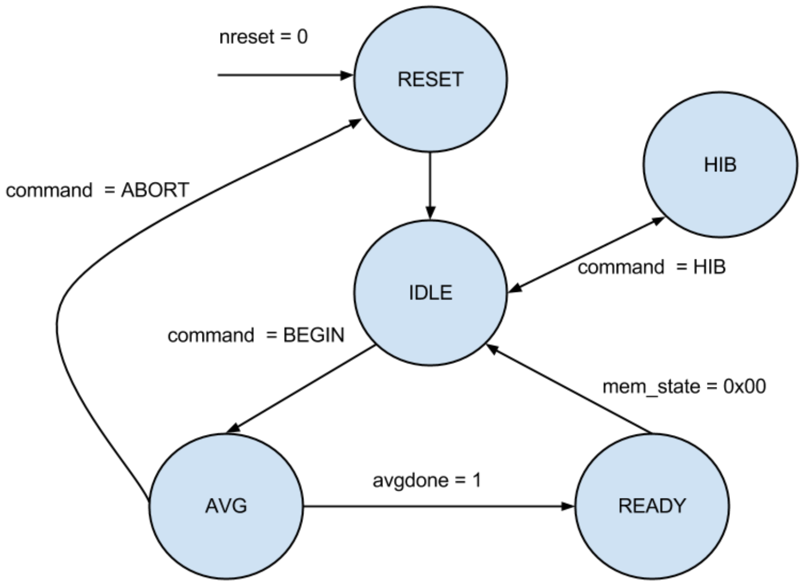

Figure 10.2.: Main control state machine

State name	Function
RESET	Reset state. Sets default values for control outputs
IDLE	Waiting for command
AVG	Image averaging in process
READY	Image averaging done, ready to transmit
HIB	Hibernation, unnecessary modules clock gated

Table 10.3.: Main control state description

10.2.3. Camera configuration and control

Since the configuration interface beyond the physical layer is specific to each camera, no functionality has been implemented for this part of the design beyond a simple control signal interface.

Name	Input/output	Size	Function	I/O standard
clk	input	1	Clock input	
sreset	input	1	Active low reset	
start_im	input	1	“Start imaging” signal from main control	
config_done	output	1	“Configuration done” signal to main control	
UART_Rx	input	1	Input serial communication pin from camera	LVDS
UART_Tx	output	1	Output serial communication pin to camera	LVDS
Camera control 4	output	1	Camera control signal 4	LVDS
Camera control 3	output	1	Camera control signal 3	LVDS
Camera control 2	output	1	Camera control signal 2	LVDS
Camera control 1	output	1	Camera control signal 1	LVDS

Table 10.4.: Interface for Camera configuration controller

10.2.4. Deserializer

Name	Input/output	Size	Function	I/O standard
Rx3	input	1	Cameralink data line 3	LVDS
Rx2	input	1	Cameralink data line 2	LVDS
Rx1	input	1	Cameralink data line 1	LVDS
Rx0	input	1	Cameralink data line 0	LVDS
RxClk	input	1	Cameralink transmit clock line	LVDS
data	output	28	Parallel data output from deserializer	
tr_clk	output	1	Transmit clock output (20 MHz)	

Table 10.5.: Interface for deserializer

The deserializer has been realized with the ALTLVDS_RX megafunction, using 4 channels, 7:1 deserialization factor and 20 MHz input clock settings. The output data is then double buffered at 40 MHz to prevent metastable outputs when crossing to the 40 MHz clock domain.

10.2.5. Memory and image accumulation controller

The memory controller both acts as controller module for the accumulation process, and as the memory manager for the system. In this combined process the controller keeps track of how much data has been acquired from the camera using an address

Name	Input/output	Size	Function
clk	input	1	Clock input
sreset	input	1	Active low reset
start_img	input	3	“Start imaging” signal from main control
data_in	input	18	Data from accumulator
req	input	1	Request for data from I2C module
mem_status	output	3	Memory status output
ram_data_load	input	24	Input line from RAM physical interface
ram_data_store	output	24	Output line to RAM physical interface
address	output	32	Address line to RAM physical interface
read	output	1	Write signal to RAM physical interface
write	output	1	Read signal to RAM physical interface

Table 10.6.: Interface for memory controller

and iteration counter, and signaling the main controller when the process is finished. It then waits for the I2C controller to request data to transmit. The dividing by 16 part of the averaging process is accomplished by only loading the fourteen upper bits of the accumulated pixel values when sending them to the I2C controller.

10.2.6. Image data accumulator and bit untangler

Name	Input/output	Size	Function
clk	input	1	Clock input
previous_value	input	18	Accumulated value from RAM
new_value	input	14	New value to be added to accumulator
result	output	18	Result of accumulation

Table 10.7.: Interface for accumulator

This is simply an 18 bit adder, receiving new pixel data from the camera and adding it to the accumulated value from RAM. The result is then stored by the memory controller

Since the bit values come out from the deserializer out of order[cama], this part simply rewires them back in order.

10.2.7. Module connection diagram

As the RTL diagrams are quite large, they have been relegated to the appendix.

Name	Input/output	Size	Function
data_in	input	28	Data from deserializer
data_out	output	14	Data arranged from MSB to LSB
d_val	output	1	Data valid
f_val	output	1	Frame valid
l_val	output	1	Line Valid

Table 10.8.: Interface for bit untangler

10.3. Command interface and data readout

The module interface is implemented as an I2C slave. This means that whenever image data needs to be transmitted to another part of the satellite, the receiving module needs to take control in the transaction.

10.3.1. Writing

To write to the command register, send a write command to the device address, and transmit a register address. While there is no address for the command register, the I2C protocol still requires an address to be transmitted. As such it doesn't matter what register address is given when writing. A list of commands is given below in Tab. 10.9.

Command	Value	Meaning
NOP	0x0	No operation. Default value for command register
BEGIN	0x1	Begin averaging operation
HIBERNATE	0x2	Design goes into or out of power saving mode. Only possible in IDLE/HIB
ABORT	0x3	Aborts current operation

Table 10.9.: Command list

10.3.2. Reading

When reading, there are two registers that can be addressed: the status register and the data register. The status register contains the state of the main controller and the state of the memory.

The three MSB of the status register contain the main controller state, while the lower five contain the state of the memory as seen in Tab. 10.11.

Register	Address	Meaning
STATUS	0x0	Status register
DATA	0x1	Image series data

Table 10.10.: Register list

Status2	Status1	Status0	Memory4	Memory3	Memory2	Memory1	Memory0

Table 10.11.: Status register

State	Value	Meaning
31+	0x1f	31 or more words left to transmit in memory
30-0	0x1e - 0x00	Number of words left to transmit in memory

Table 10.12.: Memory state list

State	Value	Meaning
IDLE	0x0	Idling
AVG	0x1	In averaging operation
READY	0x2	Ready to transmit data
HIB	0x3	In power saving mode

Table 10.13.: Status list

10.4. Power conservation

The main control module has an “enable” output control signal that is intended to clock gate the camera configuration and control module and the memory control module when the main control is in hibernation state. This functionality has not been implemented however.

10.5. SEU mitigation

The main controller module and the I2C controller will be triplicated using the TMR scheme, creating three equal design modules with same outputs and voting on the output.

Other modules that fail should have error detection that allows the main controller to reset them when a SEU has been detected. This has not yet been designed.

10.6. Current state of implementation and considerations for further work

Currently, the design is at the main module interface and behavior stage. Most has been implemented in verilog, but remains to be fully verified.

10.6.1. Current resource use

Synthesiszing the current design in Quartus for the EP4CE6 shows a resource usage of 476 LEs, about 8% of device resources.

10.6.2. Main control

Fully implemented, but not verified.

10.6.3. Memory controller

Contains a working state machine for loading and storing every other cycle, and a higher level state machine for keeping track of memory address and image series progress. Not yet verified.

10.6.4. Deserializer, accumulator, bit untangler

Fully implemented, not verified.

10.6.5. I2C serial interface and controller

Fully implemented, not verified.

10.6.6. Not yet implemented

- TMR scheme for main control and I2C
- SEU detection in memory controller
- Using “enable” signals to clock gate
- Behavior for camera configuration module, as well as the UART module.
- Sampling FVALID signal from deserializer to confirm valid data

11. Summary, conclusion and further work

11.1. Summary

To implement a SWIR camera and perform image processing boils down to a three step problem: how to acquire images from the camera to the processing module, how to do the image processing, and how to provide a interface to control the processing and readout of the processed images.

The acquisition part consists of implementing a physical bus such as USB, Cameralink or a parallel TTL interface. As the configuration and control of a camera is both unique to each camera and not well documented, the problem of how to set up and communicate with a generic camera has not been approached beyond how to implement the physical bus. It has, however, been determined that the Camera Link bus is most congruent with the aim of implementing a framegrabber in a parallel processing system such as a FPGA. A parallel interface is preferred if using a microcontroller or DSP, as these can more easily treated as a memory mapped device.

The main processing module would either need to be an actual processor, such as a microcontroller or digital signal processor, or a complex digital circuit that could be programmed onto an FPGA. Performing image processing is in general a highly parallel problem. While not strictly required for any part of the design in this thesis, it was determined that a FPGA is most suited to perform this kind of processing by its nature as a programmable circuit. Other benefits include being able to add processing tasks without worrying about image acquisition being interfered with.

Connecting the payload to the backplane of the satellite requires an I2C bus interface, and it was determined that the simplest and most effective way to provide a command interface is to implement this interface as a slave device, requiring a master on the bus to to send commands and control readout. This reduces complexity of the controller in the FPGA, and reduces the amount of masters trying to take up time on the bus.

Implementing image averaging in an FPGA (creating a datapoint) is solved by accumulating pixel data per pixel, and right shifting the result to divide. This approach requires that a power of two number of sample images (in the example implementation this is 16), but is the fastest, simplest and most accurate way. The accumulation process works by loading accumulated pixel data from memory, and adding it with new pixel data as it arrives. This means it is not necessary to keep a whole image series in memory before averaging, saving memory space.

11.2. Conclusion

This thesis has examined how a SWIR camera can be integrated into the NUTS test satellite. It has looked at what would be required in terms of digital system components, interfaces and processing, and how to compensate for the space environment. It has also looked upon the effects motion blur and compensation on image acquisition. Finally it has laid out a design concept and an example design for a specific case, and partially implemented the HDL code in Verilog for the processing module. This design concept is fault-tolerant, low cost, within the constraints of the specification, and provides a simple and efficient way of acquiring images from the the IR-camera.

11.3. Further work

While most of the example implementation should be usable for the final use case, it needs to be finished and verified before it is ready to be used in the satellite. It would likely also need to be modified to fit the specification of the actual camera and use case.

Image compression using 3D DPCM and SR needs to be implemented, and the memory module and main control needs to be modified to accommodate this. It might be necessary or of interest to separate the image averaging process from the memory controller to provide a common bus for the different processes.

No physical board layout has been done.

Power needs to be estimated, and the design might need to be modified to reduce power consumption.

Bibliography

[AA11] M. Singh A. Errandani E. Châtelet A. Doumar T. Elfouly A. Abdaoui, K. Gurram. Video acquisition between usb 2.0 cnos camera and embedded fpga system. 2011.

[alt] Altera homepage store.

[avn] Avnet express.

[Bak12] Marianne Bakken. Signal processing for communicating gravity wave images ntnu test satellite. Master's thesis, NTNU, 2012.

[BG13] Roger Birkeland and Odd Gutteberg. Overview of nuts. In *2nd IAA Conference on University Satellite Missions and CubeSat Workshop, Rome*, 2013.

[Bir11] Roger Birkeland. Nuts-1 mission statement. 2011.

[Bru11] Dewald De Bruyn. Power distribution and conditioning for a small student satellite design of the nuts backplane & eps module. Master's thesis, NTNU, 2011.

[cama] The camera link camera interface. Accessed 08.05.2013.

[camb] Camera link techonology brief.

[cam00] Specifications of the camera link interface standard for digital cameras and frame grabbers, October 2000.

[cha] National semiconductor channel link design guide.

[cyc] *Cyclone IV device handbook.*

[Fie] Steve Fielding. i2cslave design. opencores.

[Hab] Jeff Haby. Gravity wave defined. http://www.theweatherprediction.com/habyhints/64/ (accessed 14.05.2013).

[i2c00] The i2c-bus specification version 2.1, 2000.

[JSMRRT] NASA-GSFC Jeff Schmaltz MODIS Rapid Response Team.

[lat] Microelectronics 1 notes supplement.

[Mic] Mictronix. Mictronix camera link ip core.

[nas96] Space radiation effects on electronic components in low-earth orbit, April 1996.

[nut] The satellite - system overview (ntnu test satellite webpage). http://nuts.cubesat.no/the-satellite (accessed 13.05.2013).

[Pet11] Edward Petersen. *Single Event Effects in Aerospace, first edition.* John Wiley & Sons, Inc., 2011.

[pro13] *ProASIC3 Flash Family FPGAs.* Microsemi, 2013.

[RB13] Odd Gutteberg Roger Birkeland. Overview of the nuts cubesat project. 2013.

[Røn12] Snorre Stavik Rønning. Optimizing an infrared camera for observing atmospheric gravity waves from a cubesat platform. 2012.

[rta] Rtax product page.

[sc] Lattice semiconductor corporation. I2c slave/peripheral reference design rd1054.

[usb00] Universal serial bus 2.0 specification, April 2000.

[usb11] Universal serial bus 3.0 specification, June 2011.

[Vol11] Marius Lind Volstad. Internal data bus of a small student satellite. Master's thesis, NTNU, 2011.

[WGLC13] Gao Kai Wu Xiaolan Wang Gang. Liu Cai, Wang Shigang. Implementation of camera link interface on virtex-5 fpga. In *Research Journal of Applied Sciences, Engineering and Technology.* 2013.

[xsw] Xsw-640: High resolution cooled swir infrared module (xsw-640 product page). Accessed 30.04.2013.

A. The appendix

There are two parts to this appendix: An overview of the HDL files included with thesis, and an overview of the RTL diagrams included. As these are too large to be included without looking like a mess, they have been included seperately.

B. Example design verilog code

Following is a list of verilog files included with the thesis, and a short description of their content.

B.1. Top level instantiation

B.1.1. top_level_module.v

This module instantiates and connects all the other modules of the system. If all the design files are loaded into a project in quartus with this file as the top module, the whoel system will be instantiated.

B.2. I2C

B.2.1. I2C_ctrl.v

The top level control module of the I2C part of the design, instantiates the output register loader module and the I2C slave interface, and connects them together as well as connecting the input and output registers to the rest of the design.

B.2.2. I2CSlave.v

Connects the register interface and the serial interface.

B.2.3. I2CSlave_define.v

Definitions for configuring the I2C slave to your purpose.

B.2.4. register_loading.v

Loads image data from the memory controller to the output data register.

B.2.5. registerInterface.v

Loads from, and sends data to the serial interface to and from the registers.

B.2.6. serialInterface

The serialises output data and deserializes input data to and from the I2C line.

B.3. Camera Link and accumulator

B.3.1. camlink.v

A 7:1 deserializer generated with the ALTLVDS_RX megafunction.

B.3.2. bitUntangler.v

Puts the output bits from the deserializer in an order that is easier to deal with.

B.3.3. accumulator.v

A 18-bit adder that receives data from the deserializer and the memory controller, with the result being stored by the memory controller.

B.3.4. doublebuffer.v

Just a double buffer of the output of the deserializer to prevent metastable output.

B.3.5. topLevel.v

Instantiates the deserializer, accumulator, bituntangler and double buffer, and connects them together.

B.4. Main control

B.4.1. main_ctrl.v

Main controller of the design, takes commands and uses them to control the rest of the design.

B.5. Memory

B.5.1. memory_top.v

The top level instantiator of the memory controller and the physical interface.

B.5.2. memory_ctrl.v

The memory controller is responsible for keeping track of the memory state, and storing and loading values for the accumulator. It also provides ready image to the I2C module.

B.5.3. memphy.v

The physical interface that generates the appropriate output signals from the memory controller.

B.6. Camera config

B.6.1. camera_config_ctrl.v

Camera config has no specified behaviour yet. Just a generic state machine.

C. RTL diagrams

RTL diagrams to illustrate how the different modules are connected.

C.1. Top level instantiation: top_module.png

C.2. I2C control: I2C_ctrl_I2C_ctrl_inst.png

C.3. I2C slave: i2cSlave_i2C_slave_I.png

C.4. Camera Link and accumulator: topLevel_topLevel_inst.png

C.5. Memory:memory_top_memory_top_inst.png