NTNU

Norwegian University of
Science and Technology

Development of a prototype of a candidate camera payload

Jon Kalevi Oltedal

Master of Science in Electronics
Submission date: June 2016
Supervisor: Bjørn B. Larsen, IET
Co-supervisor: Roger Birkeland, IET
Amund Gjersvik, IET

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Summary

The second prototype for the NUTS camera module have been tested to confirm if changes made from the first prototype were successful. The first prototype suffered from noise issues when operating at the maximum clock frequency of 96MHz. This needed to be fixed for the MT9P031 image sensor to be usable in further designs. Debugging and testing using the camera prototype hardware and software proved that the prototype managed to produce noise free images with bright parts in the images. These results enable the use of the image sensor in further prototypes. Suggestion and discussion for the next prototype has also been done. The AT32UC3C MCU was chosen to be used on the prototype for the communication to the backplane of the satellite. Software and hardware logic requirements has also been provided. These suggestion were made so that further development can begin with a better start point.

Sammendrag

Den andre prototypen for NUTS kamera modulen blitt testet for bekrefte at forandringene gjort fra den frste prototypen var suksessfulle. Den frste prototypen led av sty problemer nr den opererte p maksimal frekvensen p 96 MHz. Dette mtte bli fikset for at MT9P031 bilde sensoren kan bli brukt i videre design. Debugging og testing med bruk av kamera prototypens hardware og software beviste at prototypen klarte produsere styfrie bilder med hvite deler i bildene. Disse resultatene gir mulighet til bruke bildesensoren i de neste prototypene. Anbefaling og diskusjon for den neste prototypen har ogs blitt utfrt. AT32UC3C mikrokontrolleren ble valgt til bli brukt p prototypen for kommunikasjon med bakplanet til satellitten. Software og hardware logikk krav har ogs blitt oppgitt. Disse anbefalingene ble gjort for at videre utvikling kan begynne fra et bedre startpunkt.

Contents

List of Figures

List of Tables

Abbreviations

MP	**M**ega**P**ixel
mW	**m**illi**W**att
mV	**m**illi**V**olt
FPS	**F**rames **P**er **S**econd
NUTS	**N**TN**U** **T**est **S**atellite
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
IR	**I**nfra**R**ed
OBC	**O**n **B**oard **C**omputer
PCB	**P**rinted **C**ircuit **B**oard
XPS	**X**ilinx **P**latform **S**tudio
IP	**I**intellectual **P**roperty
CAN	**C**Controller **A**rea **N**etwork
ASIC	**A**pplication **S**pecific **I**intergrated **C**ircuit
I^{2}C	**I**inter **I**integrated **C**ircuit
ISE	**I**ntegrated **S**ynthesis **E**nvironment
DMA	**D**irect **M**emory **A**ccess
UART	**U**niversal **A**synchronous **R**eceiver/**T**ransmitter
USB	**U**niversal **S**serial **B**us
SPI	**S**erial **P**eripheral **I**interface
FIFO	**F**irst **I**n **F**irst **O**ut
CCLK	**C**onfiguration **CL**oc**K**
DDR SDRAM	**D**ouble **D**ata **R**ate **S**ynchronous **R**andom **A**ccess **M**emory
AC	**A**lternating **C**urrent
DC	**D**irect **C**urrent
VDMA	**V**ideo **D**irect **M**emory **A**ccess

UCF	**U**ser **C**onstraint **F**ile
ROM	**R**ead **O**nly **M**emory
EPROM	**E**rasable **P**rogrammable **ROM**
EEPROM	**E**lectrical **EPROM**
SEE	**S**ingle **E**vent **E**ffects
GIMP	**G**NU **I**mage **M**anipulation **P**rogram
GUI	**G**raphical **U**ser **I**nterface
JTAG	**J**oint **T**est **A**ction **Group**

Chapter 1

Introduction

1.1 Problem Description

The current focus of the NUTS project is to finish the design and build hardware for an integrated engineering model. Through this project, the student should focus on the system design of the camera module in general, with particular focus on the hardware design, choice of components and aspects of high frequency layout.

The camera module must be designed to be reliable, as maintenance is impossible after launch. Challenges due to the space environment, such as temperature cycles, radiation environment and vacuum must be identified and discussed. In areas where mitigation of such problems is possible, solutions should be presented. Whether the solutions be implemented should be based on a cost/benefit analysis.

Key tasks for the student:

- Test of previously developed camera prototype
- Design complete schematic of the camera module, including imaging module, FPGA and MCU
- The module must be designed to be compatible with the NUTS satellite backplane
- Participate in the hardware/software design process of the camera module

1.2 Motivation

Images from the camera module is the planned payload of the NUTS satellite. Maintenance of hardware in space applications is impossible after launch, and fixing of hardware problems would require a new expensive launch. Therefore, all the modules must be designed to be as reliable as possible to secure the intended functionality for as long as possible for the system to accomplish its tasks.

1.3 NTNU Test Satelite - NUTS

The NUTS project is a student satellite project with the goal of developing a double CubeSat through volunteer and master's degree students at NTNU[4]. The main goal of the NUTS project is educate NTNU students in group project work and the design of space applications. The project started in September 2010, and was originally planned for launch in 2014. As of spring 2016, no new launch date has been set, and a new one will be set when the project is nearing completion. Even if the satellite still has not been launched, the project has completed one of its main goals of recruiting and educating students, and still continues to do so.

1.3.1 NUTS Payload

The NUTS satellite has several mission goals, such as establishing a two way communication and de-tumbling of the satellite, where the payload of the project is a camera. It was originally planned to use an IR-camera to observe gravity waves in the atmosphere. This payload idea was then later scrapped because of the required workload and the high cost of the technology. Other types of cameras, such as visual range cameras have been considered, until it was decided upon a commercial camera, to capture color images of the earth from space. A couple of cameras have been tested for the use on the satellite, and a possible candidate has been found and will be used if it can be proven to work reliable with a custom made PCB and interfaced with the NUTS satellite.

1.4 Scope of the thesis

In chapter 2, previous prototypes of the camera module is presented, with some theory around the functionality of the prototype system to understand the testing of the module, which appear in a later chapter. Some theory of problems with electronics in space is also presented, as well as some other relevant theory.

In chapter 3, the testing and debugging of the second iteration of the NUTS camera prototype design is presented. There were some difficulties with finding the problem of the prototype, as most of the logic and software used to operate the system where designed by previous students. The documentation for the used software was very limited, resulting in a lot more time than scheduled where used on understanding and debugging.

Chapter 4 revolves around the design choices and suggestions for the next iteration of the camera prototype. This involves proposing design alternatives in accordance to the requirements of the camera module, choosing the main components for the system, creating an overview of the required logic on the FPGA, and setting requirements for the software on the module's MCU. It was initially planned to make a full schematic design of the next camera payload as stated by the problem description. The scope was then changed to make some choices and suggestion for the next prototype. This decision was made because the second prototype of the unexpected time used to fully understand the functionality of the prototype system's logic and software.

Chapter 5 presents the results and discussion of the prototype testing. It also discusses the choices and suggestions for the next camera prototype. Some choices are presented in regards to challenges of the space environment.

At the end in chapter 6, all the work from this thesis is concluded. Future work to be done on the NUTS camera payload is also presented here.

Chapter 2

Background

2.1 Previous prototypes

Two previous prototypes of the NUTS payload module has been developed. The second, which is currently the latest fully developed prototype, was a redesign done to solve noise issues. The following three subsections presents the two prototypes and how the system is set up.

2.1.1 First prototype

The first NUTS camera prototype was developed by Thomas Hanssen Nornes in the autumn of 2014[5]. The work was based on finding a compatible camera sensor, optics and mount for space applications, and then interfacing it with a PCB layout. Although the design worked, it suffered from some noise issues in bright parts of the captured images. This problem would be very noticeable on images of the earth, as many images would contain bright clouds. The causes of these problems were not found, and for the custom camera module to be used in the final system, design changes would have to be made to prove that the module can operate reliably.

The noise in the bright parts of images, which can be seen in figure 2.1, appeared only with the normal operating speed at 96MHz. A temporary workaround to the problem was found by limiting the clock frequency of the sensor down to 72MHz. Although this modification is better than having visible noise in the images, a solution should be

found to get the module to work reliable at intended speeds. The noise was speculated to either be a result of ripples of 200 mV on the voltage supply to the image sensor, because of high-speed complications on the data signals from the image sensor, and/or because of the need for soldered wires on the PCB because of layout limitations of a two-layer board. A second prototype was then designed, with the goal to remove the noise issues, high speed complications, and other layout limitations.

FIGURE 2.1: Captured image from the first prototype, showing the visible noise in bright parts..

2.1.2 Second prototype

The second and current prototype was designed to solve the aforementioned problems of the first prototype. The new design was done in the Autumn of 2015 by Jon Oltedal[6]. The changes between the prototypes consisted mainly of some new power supply components and routing changes between the image sensor and development board pins. The TPS799 voltage regulators were chosen for their low output ripple characteristics of 29,4 μV_RMS, compared to 250 μV_RMS on the MAX883 used on the first prototype board. The routing changes were done to ensure that the 12 data bits with pixel values were sampled simultaneously. This was done by matching the lengths of the routes. Calculations of propagation delay concluded that this was most likely not the source

of the noise issues, but was kept in the design as there were plenty of room to do so without negative consequences.

As the software and hardware code previously used to read and store data from the sensor was quite extensive and with little documentation, there was not enough time to fully test the new board. On the other hand, some small design faults were found which would have to be fixed before testing of the module. The setup and full testing procedure of the second prototype can be seen in chapter 3, with its results listed and discussed in chapter 5.

2.1.3 Prototype setup

The first and second prototype system consists of a prototype board, development board, and the FPGA on the development board. Figure 2.2 shows the main components, and how they are connected to each other, with the arrows pointing to the slave modules.

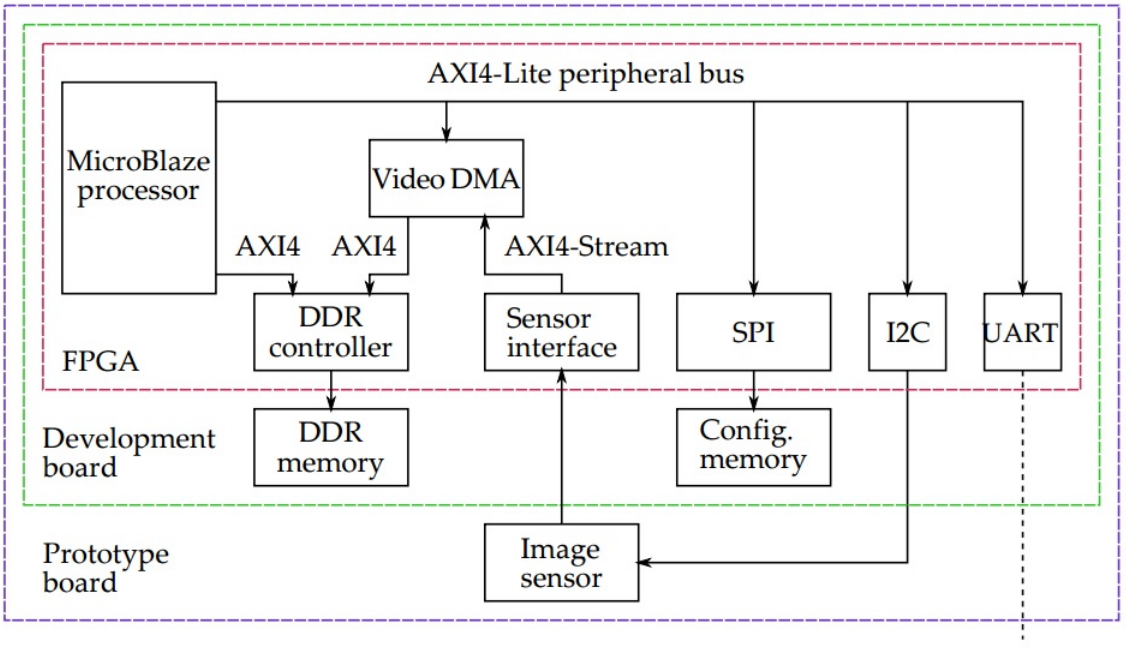

FIGURE 2.2: Block schematic of the first and second camera module prototypes[1].

The prototype board is a custom made PCB with the MT9P031 image sensor, voltage supply, and header pins to connect to the FPGA development board. The used development board is a Numato Lab Saturn - Spartan 6 FPGA development board with DDR SDRAM[3]. The memory is used to store the captured images, and the configurable memory is used to store software for the MicroBlaze processor to run.

The logic design of the on the FPGA consist of several modules; three communication blocks, and three blocks to control image sensor data and memory, as seen on figure 2.2. UART is used to communicate between Microblaze processor and the host computer. The processor uses it to recveive the software it needs to operate the camera interface, to receive information of which part of the software to execute, and to dump the image data to the host. The SPI block is used to store the executable software to the configurable memory. The last communication block, I^2C, is used to configure the image sensor. The processor receives commands from the UART, and uses the I2C block to write to the images sensor's registers. More about the blocks and the interconnecting buses is mentioned in section 2.1.5.

2.1.4 Software commands

The software for the camera prototype that runs on the microblaze processor builds up a command line interface by using the UART communication block. It starts up by resetting the image sensor before accepting commands via a serial terminal on the host computer. The different commands and their purpose are as follows:

Memory R/W A command used for reading/writing from/to the DDR memory located on the FPGA development kit. This is useful to for example read the memory locations of captured images, to confirm that a new image has been written before transferring it to the host. It can also be used to check the pixel values, as they are written to this memory.

Image sensor control Used to initiate reads and writes to the registers on the MT9P031 sensor, via the I^2C interface. The registers control the behaviour settings of the sensor. This is used to for example change the exposure time, or turn on test pattern output. The full list of registers can be found in the register reference document[7].

Frame capture A command used to initialize the sensor interface to capture and store images in the DDR memory.

Image download A command used to dump the image data from the DDR memory to the host computer. The argument for this command enables skipping data bits

for a faster transfer rate in exchange for lower quality. This is useful for when changing the focus of the lens, or to adjust the exposure time.

In addition to these software commands, some software to run on the host is also needed. The host needs to run a script to log the image data transferred over the UART with the image download command. It also needs to run a script to scan and extract the image data into an image file. The two scripts used for this has been developed by previous NUTS project members. The script used to log the terminal for image data can be found in Appendix A, while the script for scanning and extracting image data from logs is located in Appendix B.

2.1.5 Xilinx LogiCORE IP blocks

Xilinx Platform Studio (XPS) is a development program that can be used to build, connect and configure embedded processor-based systems for Xilinx FPGAs. It has a library of different available cores that can be inserted into a design. The connections between the blocks and the settings for the blocks are configured with a text file. Every block inside the FPGA on the prototype setup figure 2.2 is a configurable Xilinx LogiCORE IP, and has interconnected to form the prototype system. The different AXI4 buses between the IP blocks are also addded in the XPS software. AXI4 is a bus interface architecture developed by the ARM company. Connections are made with a simple GUI, and no further configuration is needed. The XPS design file for the prototype design where the block options is set, is attached in Appendix C.

2.1.5.1 Sensor interface block

The sensor interface block is a Video In to AXI4-Stream LogiCORE IP block. It takes data and clock input from a video source and outputs the data on an AXI4-Stream video protocol interface. It uses sync signals and/or blank signals from the same video source as inputs to show when active video is to be sampled. As seen in the following theory chapter about the MT9P031 image sensor, it outputs

2.2 Theory

The following sub section details some theory around how the image sensor is operated and some theory relevant to the test and design of the NUTS camera module.

2.2.1 MT9P031 digital Image sensor

The image sensor used for the camera module prototype is a 1/2.5-Inch 5 Mp CMOS Digital Image Sensor, the MT9P031 by ON Semiconductor[8]. It was chosen for the first prototype for satisfying the requirements[5] of high resolution (2592x1944), and a low power consumption (381mW at 14 FPS with full resolution). The settings of the image sensor, are controlled by writable registers[7]. The settings range from everything from exposure time, frame size, and test pattern outputs. The registers are set through an I^2C interface between the sensor chip and driver.

The sensor has three frame output modes; continuous, snapshot, and bulb. The continuous mode outputs frame data continuously and can be captured at any time. The snapshot and bulb modes outputs one frame, initiated by a trigger signal. The difference between the two modes is how exposure time is controlled. Snapshot uses is electronically controlled, as seen in the following sub-section, while bulb exposure time is ended by a second trigger signal. Two signals, Line valid and Frame valid, are available for the capturing interface to signal when a new line begins, and when the whole frame is finished. The output of the image sensor is in the form of RGB Bayer array. More about Bayer color arrays are presented in section 2.2.5.

2.2.1.1 Exposure time

One of the features of the MT9P031 image sensor is its superior low-light performance. With the sensors default configurations, even regular lamps appear too bright, causing images to become almost blank. This means tweaking of the exposure time will be very important to secure clear images. For the MT9P031, the exposure time EXP is given by equation 2.1 below,

$$EXP = SW \times {}^tROW - SO \times 2 \times {}^tPIXCLK \tag{2.1}$$

where:

$$
\begin{aligned}
&\text{SW} = \max(1, (2 \times 16 \times \text{Shutter\_Width\_Upper}) + \text{Shutter\_Width\_Lower}) \\
&\text{SO} = 208 \times (\text{Row\_Bin} + 1) + 98 + \min(\text{SD},\text{SDmax}) - 94 \\
&\text{SD} = \text{Shutter\_Delay} + 1 \\
&\text{SDmax} = 1232, \text{ if SW} < 3; \text{ 1504 otherwise} \\
&{}^{t}\text{PIXCLK} = 1/\text{F}, \text{ where F is the clock frequency.} \\
&{}^{t}\text{ROW} = 2 \times {}^{t}\text{PIXCLK} \times \\
&\qquad \max(((\text{W}/2) + \max(\text{HB}, \text{HB}_{\text{MIN}})), (41 + 346 \times (\text{Row\_Bin} + 1) + 99))
\end{aligned}
$$

SW stands for shutter width, and tells us how wide the rolling shutter is. The lower the shutter width, the lower the exposure. SO, shutter overhead, is a combination of row binning and shutter delay (SD). Binning combines the charge of nearby pixels, making for a faster capture speed in exchange for spatial resolution, which lowers the quality of the image[9]. The SD value is a negative adjustment, meaning increasing the SD value decreases the exposure time. ^{t}PIXCLK is the time of one period on the image sensor clock. ^{t}ROW is the time period between the first pixel outputs of two rows, where W is the image width, and horizontal blanking (HB) is a delay added after each row. Every variable in the expression above has its own register[7]. Thus, to change the exposure time, one would have to change the value of one or more of the registers.

2.2.2 Harmonics in square signals

Square waves are used in digital systems to signal ones and zeros. In the frequency domain, square waves are seen as a combination of sine signals with odd harmonics of the fundamental frequency. The fundamental frequency has the same period as the period of the square wave. The frequency of the following harmonics increases with the fundamental frequency for each harmonic, but since a square wave skips every other harmonic as it consist of only the odd harmonics[10]. Higher harmonics also have a lower amplitude. Figure 2.3 visualizes how a square wave is generated from four sine waves, and also shows how they are calculated. The more harmonics that are included in the square wave, the more closer to a perfect square wave it will be.

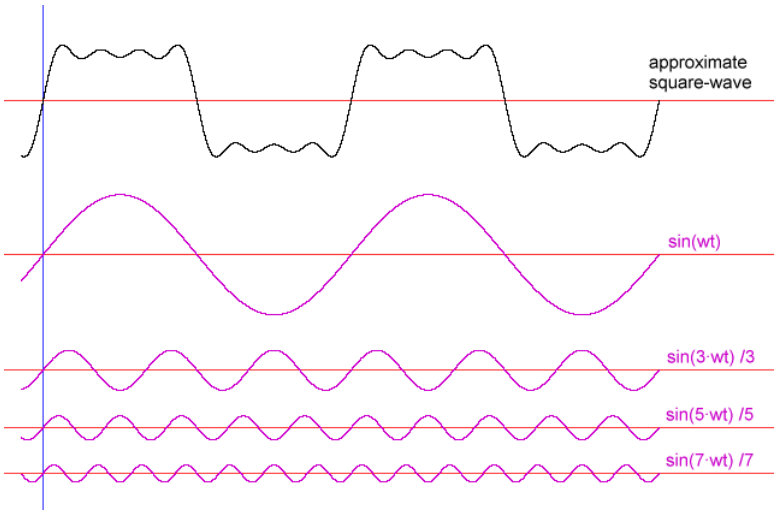

FIGURE 2.3: Square wave constructed from the first, third, fifth, and seventh harmonics.

2.2.3 NUTS Backplane and module compatibility

The NUTS satellite module system consists of a backplane providing power and communication, with eight slots for the satellite's sub modules, where two of the slots provide master functionalities. The dimensions for the sub-module circuit boards has been set to be 92×90 mm, with the backplane connector situated on the edge of the 90 mm side. Every sub-module needs to be within these limits to be compatible with the backplane and frame of the satellite. An early version of the cube satellite's backplane with the OBC connected can be seen in figure

FIGURE 2.4: Square wave constructed from the first, third, fifth, and seventh harmonics.

The backplane with the connected modules will be fitted into an a cube with dimensions of 10×10×20 cm. The outside of the cube frame will be fitted with solar panels on every side except of the bottom. This gives space for the camera module to be situated on the bottom of the cube on the end with no solar panels, so that the camera lens can peak through the bottom.

2.2.4 Electronics in space

The difficulty with using electronics in space applications, is the exposure to hazardous space radiation[11]. When electronic systems are hit with radiation, there is a chance for it to trigger faults called Single Event Effects (SEE). These effects can range from a brief voltage spike at a node, to the breakdown in the gate oxide of a transistor.

2.2.5 Bayer color filter array

Most digital image sensor consists of a grid of red green and blue colored pixels, called a Bayer color filter array[2]. This filter array is depicted in figure 2.5. Because the human eye is more sensitive to green light than both red and blue, the bayer array is constructed to mimic this effect with twice as many green pixels than red and blue. This also makes image appear less noisy with a finer detail than it would have with equal amount of pixels for each color. A side effect is that images appear with a green tint. This effect can be lessened, and more "natural" colors can be produced with a concept called color correction. The image editing software GNU Image Manipulation Program (GIMP) provides the ability to perform manual color correction through its GUI.

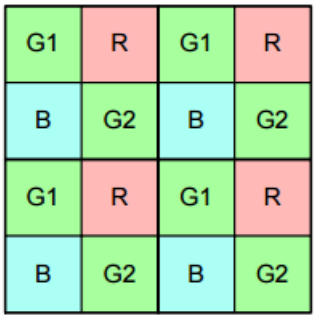

FIGURE 2.5: Red, green and blue pixels forming a Bayer color filter array[2]

.

Chapter 3

Testing and debugging of prototype

As mentioned in background chapter 2.1.2, a second prototype has been previously designed, and in need of testing. More time than planned went into the debugging of the prototype, as the problem was not discovered until internal probing of the FPGA with the Xilinx Platform Cable USB was available. As the prototype used a lot of software and hardware from previous students, finding the fault in the system was difficult. The possible locations of the fault were speculated to be found in one or more of the following parts:

- PCB pins and routing
- Image sensor
- Software running on the microblaze processor
- Sensor interface and Video DMA on the FPGA

This chapter will go through the testing and debugging process of the second prototype.

3.1 PCB and data signals

3.1.1 FPGA header pins

As mentioned in section 2.1.2, some of the chosen pins between the FPGA-w development kit and the PCB prototype were unusable. For the prototype to function properly, these pins had to be resoldered to usable neighbouring pins. Figure 3.1 shows the pins between the boards.

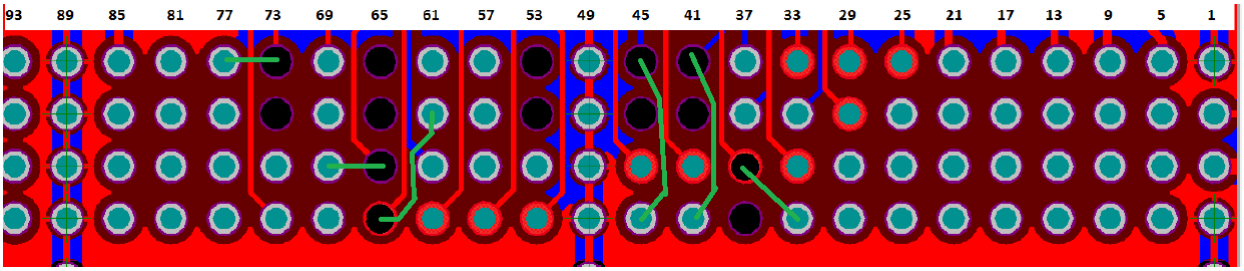

FIGURE 3.1: Pin connections between the FPGA development kit header and developed prototype board. Pins are numbered top to bottom, starting in the top right corner, in accordance to the development kit user guide[3].

The blue and red lines show which pins are connected to the imaging sensor. The pins that marked black are not connected to the FPGA, and can not be used as I/O. This led to the rewiring to neighboring pins, visualized by the green wires. The resoldered wire from pin 68 to 62 was chosen, because the pin 62 connects to a global clock buffer. It is recommended by Xilinx to use global clock buffers on every clock, to ensure correct timing results[12]. However, during the testing process, the pixel data signal D0 (Pin 68) and clock signal PIXCLK (Pin 67) were mistakenly swapped. This meant that the clock signal PIXCLK was not connected to a global clock buffer, as recommended. The Xilinx ISE tool reports an error when a clock signal uses a regular I/O port. Since the clock signal is only connected to the sensor interface block, it was concluded that there would be no timing issues for the signal. If there are to be two or more blocks with this particular clock signal as input, the pin will have to be soldered to an appropriate neighboring pin. A workaround for this was to set the constraint $CLOCK\_DEDICATED\_ROUTE$ to false for the clock signal. This forces the routing algorithm of the development Xilinx ISE to ignore the clock aspects of the signal, and routes it as it would a normal signal. Constraints are instantiated in the user constraint file (UCF) in the ISE software. The

constraint types used for the prototype system is I/O placement and voltage, and some timing constraints for the clocks. This UCF can be found in appendix D

3.1.2 Power supply

As mentioned in chapter 2.1.1, the voltage regulator on the first prototype suffered from large spikes of 200 mV peak to peak on the output. A case study on the use of the MT9P031 image sensor in space applications, stated that the best noise performance is achieved with a maximum peak to peak of 10 mV[13]. The second prototype was designed with voltage regulators with a lower noise output[6]. The voltage ripple on the 2.8V power supply is shown in figure 3.2. As seen, the measured noise averaged at about 50 mV.

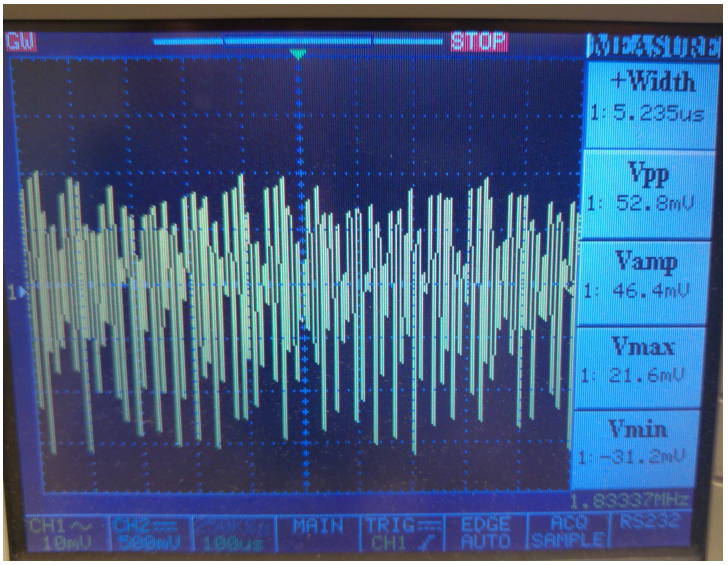

FIGURE 3.2: Image showing the ripple on the voltage supply to the image sensor.

3.1.3 Clock signals

During debugging, the clock signal was one of the speculated reasons for the system not working properly. Figure 3.3 shows the 96 MHz clock signal from the image sensor to the development board, as sampled by a 150MHZ digital oscilloscope . Note that the figure shows only the AC component of signal. The DC peaks of the signal was measured to be 1.76 V, which is within the typical value for the image sensor's output[8]. The clock signal was then cross-checked with a different 300MHz oscilloscope, where the signal had an extra peak when transitioning from high to low. By looking at the theory from 2.2.2,

the deformed clock signals was concluded to stem from the inability of the oscilloscopes to capture more than one or two harmonics from the signal. As square waves have only odd harmonics, the harmonics of a square wave of 96MHz can be calculated by adding 96 MHz with 96*2 Mhz, giving the first three harmonics with frequencies of 96MHz, 288 MHz, and 480 MHz. This corresponds to the 150 MHz displaying almost a perfect sine wave of 96 MHz, as it can't capture the higher frequency harmonics.

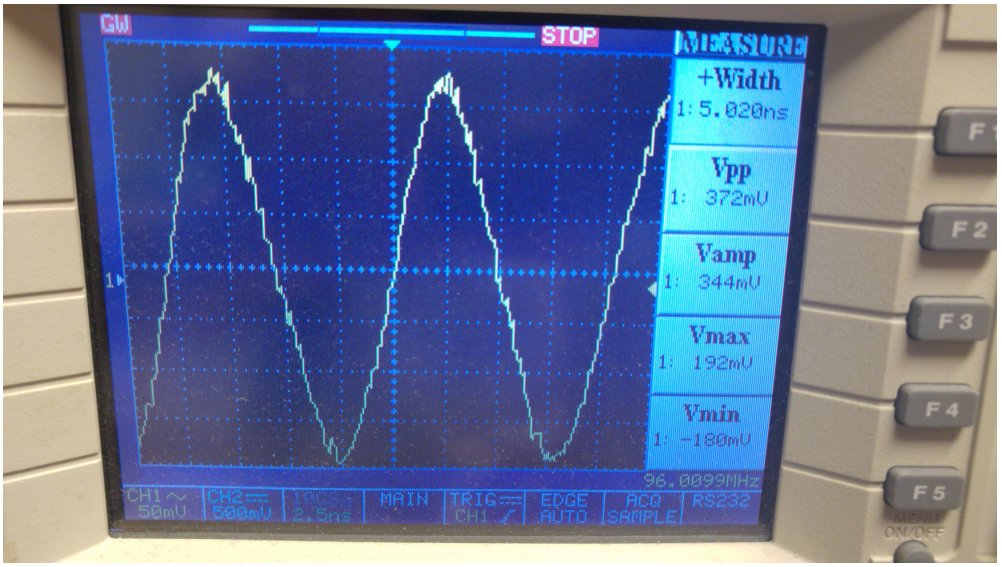

FIGURE 3.3: Screenshot of the 96MHz clock signal from the image sensor to the development kit.

3.2 Testing of image sensor

The testing of the image sensor were done by using the software commands listed in the background chapter 2.1.4. Configuration of the sensor parameters worked as intended, as the written registers gave the written value when read. When the first image capturing test was done, the software froze during the writing of the frame. The frame capture command was cancelled, and the memory location of the frame that was supposed to be captured had not been written to. There were found to be two possible reasons for the problem, either the sensor interface did not receive the correct signals, or the VDMA block could not write the data it received to memory.

The output data and control signals from the image sensor was probed with the continuous output mode mentioned in theory chapter 2.2.1. The line and frame signals were output with constant periods, corresponding to the datasheet. The data signals

where not possible to probe correctly with a 150 MHz oscilloscope for the same reasons mentioned in chapter 3.1.3. However, it was noted that the data signals were reacting to light changes by using a flashlight on the sensor. By changing the output of the sensor to a test pattern, the sensor seems to output correct and clear signals, as seen in figure 3.4.

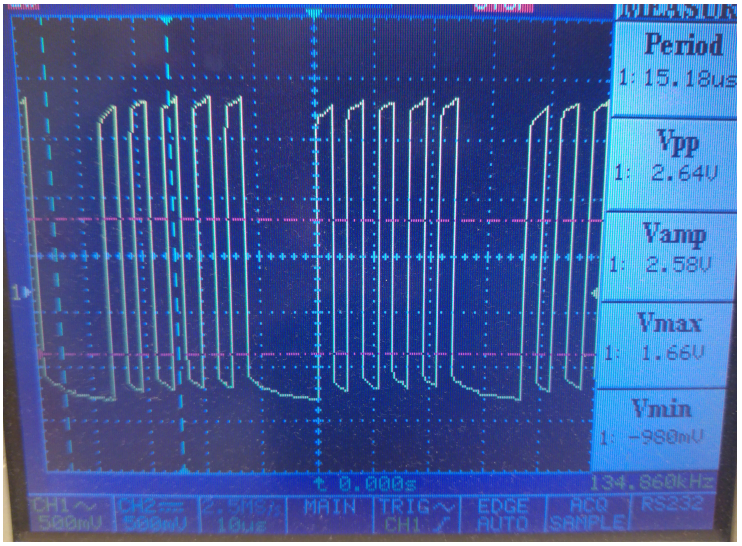

FIGURE 3.4: Screenshot of one of the data outputs during a test pattern output.

To find if the problem really was the image sensor or the FPGA logic, probing of the signals inside the FPGA needed to be done. Xilinx has its own hardware tool, Chipscope, and cable, Platform Cable USB II, to probe the internal signals of their FPGAs and display a waveform of the probed signals[14]. The platform Cable USB II accesses the FPGA internals via the JTAG debugging interface. The following section goes through the aspects of the Chipscope debugging.

3.3 Chipscope debugging

The Xilinx ISE tool provides software for easy use of the Chipscope tool. It works by automatically inserting cores to the design, and the user only needs to choose which signals to sample[14]. With the use of Chipscope, it was immediately discovered that no data was sent between the sensor interface block VDMA blocks, seen in figure 2.2. The signals to the sensor interface was then probed with the Chipscope tool, were the problem was discovered. During the re-soldering of the prototype board pins in chapter 3.1.1, the data signal D0 and the clock signal PIXCLK had been mistakenly swapped in

the UCF. This means that the sensor interface block had a non-clock signal as a clock input. With this problem fixed, the prototype managed to capture images.

3.4 Exposure and lens adjustment

To get the best quality on captured images, the exposure time has to be adjusted so that no part of the images are too dark or too bright. The lens on the camera also needs to be adjusted in order to get the correct focus of the scenery to be imaged. Both the lens and exposure time were adjusted by trial and error. The exposure time can be changed by changing one or more of the variables in equation 2.1.

With the default settings of the camera, images in normal daylight produce a fully blank image, meaning the exposure time has to be decreased. By using the equation, we can decrease the exposure time by either decreasing SW and/or ^{t}ROW, or increasing SO and/or ^{t}PIXCLK. ^{t}PIXCLK is already at its maximum speed of 96MHz and therefore can't be increased more. ^{t}ROW can only be decreased by decreasing the width W, as HB and Row_Bin can only be increased from the default values. Shrinking the width of the image is not an option, as the largest resolution possible is wanted. This leaves the options of either decreasing SW, and/or increasing SD. It was decided to only adjust the SW, as it had a large enough interval to influence the, and to only have to write one register between each try.

Figures 3.5 through 3.7 shows images taken with three different values for SW, resulting in three different exposure times. Note that the images are of lower quality, because the image download command from chapter 2.1.4 were run with the argument for bit skipping This was done so that the images could be captured and downloaded one by one with a faster download speed, and more easily compared together instead of waiting for the full quality images. As mentioned in chapter 2.2.1, the image sensor uses RGB Bayer output gives images a green tint. By using equation 2.1 and the default values, found in the datasheet of the sensor[8], for every variable except SW, we end up with equation 3.1.

$$EXP = SW \times 36.8s - 2219.46ns \tag{3.1}$$

FIGURE 3.5: Image captured with a SW of 50

FIGURE 3.6: Image captured with a SW of 30

FIGURE 3.7: Image captured with a SW of 20

Using the above equation the resulting exposure time for figures 3.5 through 3.7 are 1.838 ms, 1.1 ms, and 0.734 ms respectively. The differences between these exposure times are clearly visible in the images. Withe the exposure time of 1.1 ms giving the best results in normal daylight.

Chapter 4

Further development

Previous prototypes have only consisted of an image sensor with power supply connected to an FPGA development kit. The next iteration of the prototype will be a complete design containing all the parts of the module on a single PCB. This chapter focuses on furthering the development of the camera prototype. Initially, this thesis was planned to contain the full schematic design of a new prototype. However, because more time than planned was spent debugging and testing the previous prototype, it was decided to make design choices for the new prototype without detailed schematic design. This chapter describes the design choices made for the next iteration of the camera payload prototype, in accordance to the requirements and tasks of the camera module.

4.1 Requirements

To give a good overview of the requirements and implementations for every module of the NUTS satellite, a standardized requirement and design implementation lists has been established. This makes it easy for every project member to check what the other modules are required to do, and how they are to be solved. It is also useful when making design choices that depend on other modules, for example choosing components for receiving and sending information via the backplane. The set requirements for the camera module can be seen in table 4.1.

The syntax of the requirement list consists of four different codes. R0X states which system-level the requirement refers to, which in the case of the camera module is R05:

Test the payload camera. The second code, CAM, refers to which module the requirement is entrusted. The third code refers to which main requirement is addressed, for example CPR for compression of images requirements. The last code consisting of only numbers, states the number of the sub-requirement of the given main requirement.

Table 4.1: Functional requirements of the camera module

ID CAM = Camera	Specification
R05-CAM-COM-001	**COM = Internal Communication Bus** Must be able to communicate with the other sub systems using the backplane
R05-CAM-COM-002	Must be able to capture image on request
R05-CAM-COM-003	Must be able to send images to the OBC on request
R05-CAM-COM-004	Must be able to change image sensor parameters on request
R05-CAM-CPR-001	**CPR = Compression of images** Must to be able to read images from the image sensor and compress them to reduce file size
R05-CAM-CPR-002	Must be able to produce thumbnails
R05-CAM-CPR-003	Must be able to produce histograms of pixel values
R05-CAM-CPR-004	Must be able to detect and not process unwanted images (Pictures of space or the sun)
R05-CAM-CPR-005	Must be able to make gamma corrections on captured images
R05-CAM-IMG-001	**IMG = Storing of images** Must be able to store compressed images to local memory
R05-CAM-IMG-002	Must be able to retrieve images from local memory
R05-CAM-REP-001	**REP = Reprogramming** The compression logic should be able to be reprogramed in flight

As seen in the table, the module has four main requirements, along with some sub requirements. The first requirement, Internal Communication Bus (COM), refers to the subsystems communication to the OBC via the backplane. The compression of images (CPR) requirements tells us how the module has to handle compression of captured images. The last two requirements, Storing of images and reprogramming, tells us that images are to be stored on the camera module, and that the module has to be designed so that it can be reprogrammed.

4.2 Camera subsystem design overview

By following the given requirements for the subsystem, a complete block schematic of the module has been proposed, and can be seen in figure 4.1. The following sections will go through every part of the figure, and explain how they will fulfill requirements, as well as component and design choices made for the next iteration of the prototype.

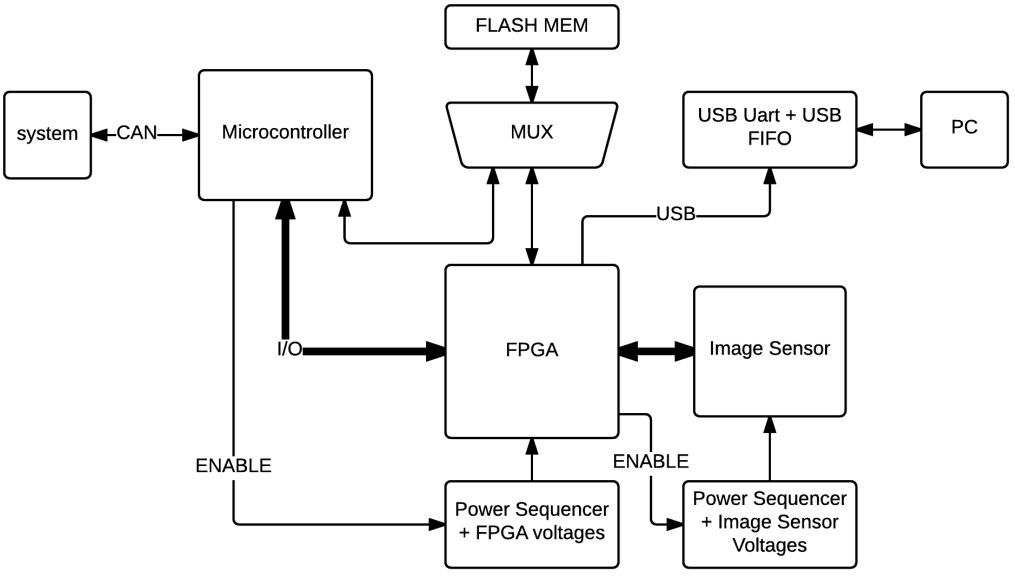

FIGURE 4.1: Overview of the NUTS camera module system.

4.3 Microcontroller

The purpose of the microcontroller (MCU) in the camera module, is to carry out orders received from the OBC via the backplane. It is meant to fulfill the internal communication bus requirements listed in chapter 4.1. There were several reasons for the decision to use a separate MCU for communication instead of only using the FPGA. One reason was that the payload module will, for the most parts of the satellites lifespan, stay idle. This can be utilized by using a MCU with very efficient sleep mode to save power, while the FPGA can be powered off when not in use. Another reason was that the satellite communicates between modules via the CAN bus interface. As software for communication over the CAN bus will be developed for the OBC, a lot of time will be saved on reusing previously developed code. In addition, it also negates the need to implement the CAN logic on the FPGA.

4.3.1 Required MCU software

The MCU needs to handle all requests from the OBC. The main functions this entails are the following

Write sensor register: Send a write command with register and value to the FPGA I^2C interface.

Configure FPGA: Read the image file from the flash memory, then transfer it to the FPGA for configuring. The Slave Serial configuration mode in figure 4.2 (a) is suggested to be implemented, as it requires few pins to operate, and will most likely not need the speedup of using parallel bins as in (c) in the same figure.

Send file to OBC: Read image or histogram file from flash memory, then transfer it to the OBC.

Automatic exposure time adjustment: Initiate a series of image captures with histograms. Configure the exposure time according to the histogram data.

4.3.2 MCU characteristics and suggestion

The MCU has no special requirements needed to perform its tasks. The software is will mainly be relaying data between the OBC and the camera module's flash memory and FPGA. This means that it does not need to have a very high storage room for application data. Although the MCU will be used to transfer large image files, and a possible large FPGA configuration file. Therefore, the chosen MCU needs more than low end operating frequency to faster boot the FPGA and to not occupy the CAN bus on the backplane. When the camera module is not capturing or processing image data, the MCU will go to sleep mode. The AT32UC3C MCU is planned to be used on the OBC of the NUTS satellite, and will therefore also be suggested for the camera module, for the benefits of reusable software. Because of these reasons, the suggestion of MCU will be decided by evaluating sleep efficiency mode, operating frequency, and reusability of code previously developed by the NUTS project. Table 4.2 below shows three suggested MCUs. The suggested MCUs and characteristics are taken from previous work by the NUTS project, where the choice for the OBC MCU were made[15].

TABLE 4.2: Comparison for suggested MCUs

Characteristics	MSP430	AT32UC3C	SAML21
Frequency [MHz]	24	66	48
Power consumption sleep [mA]	0.32−285	31−100	1.2−185.5
reusable NUTS software	No	Yes	No

The only downside with the AT32UC3C is the lower end of the power consumption. It has however a lower upper end of power consumption, meaning it can still potentially outperform the other two. Additionally, a report done by NASA states overly stringent requirements can complicate software. There is no reason to set very strict constraints on the sleep power consumption, as it will be the only current drain on the module when the module is idling. As long as it is moderately low, it will have little impact on the total current drain of the satellite. With the higher frequency and the reusable software, the AT32UC3C is clearly the best suggestion.

4.4 FPGA logic

The FPGA on the next iteration of the prototype will have the same tasks as on the current prototype, as well as some additional ones. The main tasks are capturing, compressing, and writing image files to memory. In addition to this, it will need a configuration interface from the MCU for programming purposes. The choice of FPGA, will not be discussed in this thesis, as another member of the NUTS project is discussing the decision.

4.4.1 Compression logic

The compressing logic is the main task for the camera module FPGA. The captured images will be compressed to the JPEG2000 image format. This format is used for its ability to retain image quality despite for bit errors[5]. In addition to compressing images, it will also perform gamma adjustment, color transformation, and histogram computation. The histogram will be used to show the pixel values. This is useful to know when the camera is pointing toward empty space or the sun, as all the values will be either white or black. By only transferring the histogram file, precious satellite bandwidth is saved from useless black or white pictures. The compression logic is currently being worked on by another student from the NUTS project.

4.4.2 Capturing and storing logic

Until now, Xilinx logiCORE blocks have been used to capture image data and store it. Even if a Xilinx FPGA will be used or not on the next prototype, a custom capture and storing logic should be implemented in hardware. The Xilinx blocks are all purpose blocks and more customizable. Specialize custom made blocks can be made smaller, and possibly to consume less power as well. Some register should be included between the capturing interface and the compression logic, so that the external memory is not needed during compression. This is, however, dependant on the available space on the FPGA.

4.4.3 I^2C interface

The I^2C interface needs to be ported from software to the FPGA. As there should not be a soft-processor running on the FPGA. The commands for configuring the image sensor will be sent from the MCU to the FPGA, and can be sent directly to the I^2C block for immediate write to the image sensor.

4.4.4 Configuration interface

he MCU will also be used to configure the FPGA. As discussed later in chapter 4.7, the USB programming and debugging block will not be included in the final iteration of the prototype. Because FPGAs lose their configuration after being powered down, they need some external memory to boot from, or a a processor or MCU to write the configuration to the FPGA. Using the Spartan 6 FPGA from the prototype development kit as an example, it can be programmed in three different modes by a processor or MCU. Either slave serial mode, JTAG, or slave SelectMAP mode[16]. The setup for these configuration modes can be seen in figure 4.2. If the FPGA were to be configured to read from the memory as a master, a separate ROM chip or an SPI Serial flash interface between the FPGA and flash memory would have to be added to the system, as noted in the Spartan 6 user guide[16].

To choice of which of the three configuration modes above, is dependent on the on the size of the configuration file, and available pins on the MCU and FPGA. For simplicity,

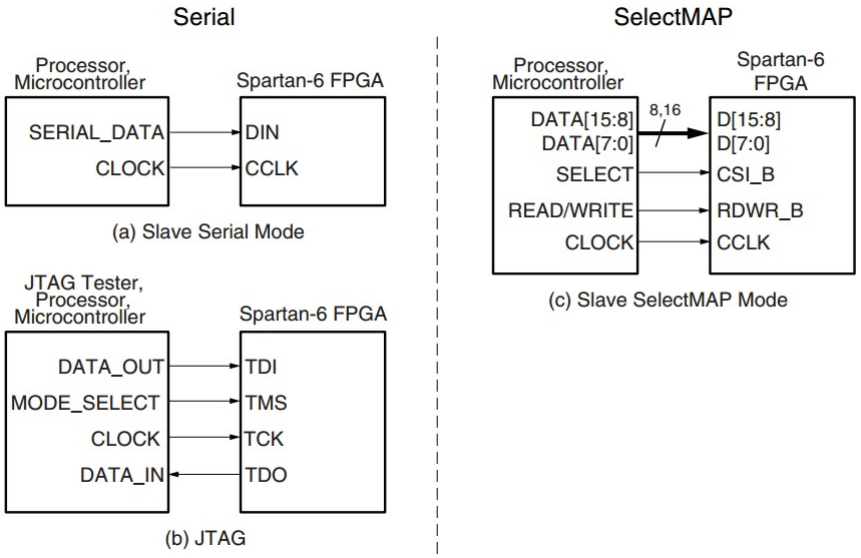

FIGURE 4.2: Setup for the configuration modes for the Spartan 6 FPGA, with the FPGA as slave.

and to save PCB area and I/O pins on both the MCU and FPGA, the slave serial mode should be chosen. If, on the other hand, the configuration file is very large, the SelectMAP can be considered for parallel data bus instead of serial. This is however unlikely, as the size of the bit file for prototype design is at only 1.6 MB. With the configuration clock (CCLK) of up to 80 MHz on the Spartan 6, transferring 1.6 MB will only take about 20 ms. Even though the configuration file for the full design will be larger, it will most likely still be sufficient with the serial interface.

4.5 Flash memory

Flash memory were chosen over other types of non-volatile memories such as EPROM and EEPROM, because of the need of large storage capacity for the image files. They are also shock-resistant and power-economic, which are important characteristics for space applications[17]. The MUX in figure 4.1 is needed to prevent simultaneous reads and writes between the MCU and FPGA.

The main task for the flash memory, is to store captured images before, during, and after compression. This fulfills the two requirements concerning storing of images. In addition to these two requirements, it will also be used in reprogramming of the FPGA, which is the last point on the requirement list in table 4.1. There are two reasons for

having the possibility of reprogramming the FPGA logic after launch. The first reason being having the ability to modify the FPGA logic, either to add other functionalities, or to improve on the existing logic in regards to resources or speed. The second, and most likely scenario for the use of FPGA logic reprogramming, is in cases of logic faults, either from an undiscovered bug, or from bit flips as caused by cosmic radiation, as mentioned in section 2.2.4 .

For the flash memory, SD-Card is recommended. This is mainly because of the easy to use SPI interface. More research has to be done to estimate how reliable it performs in space environment.

4.6 Power sequencers

As seen on figure 4.1, two power sequencers are needed on the camera module; one for the FPGA, and one for the image sensor. The power sequencer for the image sensor is present on the second prototype. The output ripple from the sequencer, as mentioned in chapter 3.1.2, has a peak to peak value of 50 mV, while a case study recommends a maximum of 10 mV peak to peak for best noise performance[13]. However, as discussed in the result chapter 5, no noise was present in the captured images. This means it is not necessary to modify the power sequencer, unless it is very certain that a possible modification can decrease the ripple effect. The power sequencer to the FPGA switch connected to the backplane, which will be triggered by the MCU.

4.7 Debugging interface

The USB UART + USB FIFO block is there to serve as a high-speed link for debug purposes. In addition to the USB interface, if a Xilinx FPGA is chosen for the next prototype, a JTAG interface should be added in addition to the USB interface. This gives access to the valuable debugging properties of the Chipscope tool, as is evident in section 3.3.

The FT2232H Dial high Speed USB to Multipurpose UART/FIFO integrated circuit, which was used on the prototype development board[3], is perfect for the debugging interface, as it provides two separate channels which can be configured to nine different

types of communication interfaces[18]. This gives a good opportunity to use both USB UART, USB FIFO and JTAG.

Chapter 5

Results and Discussion

5.1 Second prototype results

The second prototype succeeded its goal of noise free images with the normal operating speed of 96 MHz. Figure 5.1 shows a full quality and color corrected image captured by the prototype. The color correction was necessary because of the Bayer color array of the image sensor, and was done manually through the GIMP software. Comparing the white parts of this figure, and the figures in chapters 3.4 and 2.1.1, it can be seen that the brightness noise that was present in the first prototype has been eliminated. The ripple voltage on the image sensor's power supply has been decreased by 75% from 200 mV on the first prototype to 50mV on the second. The high ripple noise of the first prototype might have been the source of the brightness noise, but was not confirmed. It is a possible source of the issue, therefore future prototypes should keep the ripple at 50 mV or less. The length matching of the data wires on the second prototype design[6] should also be kept on further installments. This should be done to mitigate complication that may occur with high frequency layout that was discussed in the paper for the design of the second prototype.

With the brightness noise gone, the MT9P031 image sensor has been proved to be able to capture clear high resolution images with a custom designed PCB interface. Some dark spots on the upper right side of images appeared on every captured image, and can be seen in figure 5.1. These spots were most likely some dust stuck under the lens, or

in worst case the image sensor has been damaged. Either way, it should disappear by cleaning under the lens or changing to a new sensor chip.

FIGURE 5.1: Color corrected image captured by the second prototype.

5.2 Development discussion

The AT32UC3C MCU has been proposed to be used on the camera module because of its reusable code from the OBC, as well as its potential for low power sleep mode and goof clock speed. The main software functions needed to be implemented on the OBC has also been listed.

For the FPGA, the main logic blocks functionalities needed for have been explained. The slave serial configuration mode for the FPGA has been suggested to use, because of it simple needs of only two pins. This frees up MCU pins for other uses, as well as lessens the amount of traces needed on the. However, this is only applicable if the Spartan 6 FPGA is chosen to use, or the other chosen FPGA has the same configuration mode.

SD-card flash memory was suggested as external memory because of its simplicity. Further research should be done on the use of SD-card in space environment to confirm that it is usable for the intended purpose.

As mentioned in above in the prototype testing results, the power sequencer should only be changed if it is with utmost certainty that the new regulator can provide less than 50 mV peak to peak ripple. Worsening the ripple may bring back the brightness noise issues if the power supply was the cause.

For the debugging interface, the FT23232H is a excellent candidate to handle the debugging interface because of the possibility to configure it to multiple different interfaces. It is important to implement the JTAG interface if a Xilinx FPGA is to be used, to enable the internal debugging with Chipscope.

Chapter 6

Conclusion

During the course of this project, the previously developed camera prototype has been debuged and tested. This needed to be done to be able to confirm whether or not the chosen image sensor, MT9P031, was a valid candidate or not for the NUTS camera module. Through tedious debugging without much success, the problem was ultimately found when the Chipscope tools were available. The tests could then be issued on the camera prototype. By adjusting for exposure and using a color correction tool, a high resolution, noiseless image was produced. This allows for the image sensor to be part of further on the NUTS camera module.

The setup and suggestion for the next iteration helps the next iteration of the prototype to be developed. This was done instead of fully developing the next prototype, becuase of the time drain from debugging the previous prototype. The suggestions made provides basic knowledge of how the whole system is inteded to be developed.

6.1 Future work

None of the modules on the NUTS satellite are completely finished, so there is always more work to be done for the next students on the project. Further work needed to be done on the camera module can be divided into three main parts; Designing the full PCB, implementing the software for the MCU on the module, and develop the full hardware logic needed for the FPGA.

Appendix A

Terminal logger

This software is used to log incoming image data through the USB. The QT toolkit software is used to run this code.

```
#include <QtSerialPort/QSerialPort>
#include <QTextStream>
#include <QCoreApplication>
#include <QStringList>
#include <QFile>

QT_USE_NAMESPACE

#define PROGRESS_DOT_INTERVAL 16*1024
#define PROGRESS_TEXT_INTERVAL PROGRESS_DOT_INTERVAL*64

int main(int argc, char *argv[])
{
        QCoreApplication coreApplication(argc, argv);
        int argumentCount = QCoreApplication::arguments().size();
        QStringList argumentList = QCoreApplication::arguments();

        QTextStream standardout(stdout);

        if (argumentCount != 5) {
                standardout << QString("Usage: %1 <serialportname> <baudrate> <
    scriptfile> <logfilename>").arg(argumentList.first()) << endl;
                return 1;
        }

        QSerialPort serialPort;
        QString serialPortName = argumentList.at(1);
        serialPort.setPortName(serialPortName);
```

```
    int serialPortBaudRate = argumentList.at(2).toInt();
    serialPort.setBaudRate(serialPortBaudRate);

    // open files used for input and output, as well as the serial port
    QString scriptFileName = argumentList.at(3);
    QFile scriptFile(scriptFileName);
    if (!scriptFile.open(QIODevice::ReadOnly)) {
            standardout << QString("Failed to open script file for reading:
%1").arg(scriptFile.errorString()) << endl;
            return 1;
    }
    QByteArray script = scriptFile.readAll();

    QString logFileName = argumentList.at(4);
    QFile logFile(logFileName);
    if (!logFile.open(QIODevice::WriteOnly)) {
            standardout << QString("Failed to open log file for writing: %1")
.arg(logFile.errorString()) << endl;
            return 1;
    }
    QDataStream logStream(&logFile);

    if (!serialPort.open(QIODevice::ReadWrite)) {
            standardout << QString("Failed to open port %1: %2").arg(
serialPortName).arg(serialPort.errorString()) << endl;
            return 1;
    }

    // send the input data, then log all output to file
    standardout << "Executing script:" << endl << script << endl;
    serialPort.write(script);

    QByteArray readData = serialPort.readAll();
    int lastProgressDot = 0;
    int lastProgressText = 0;
    while (serialPort.waitForReadyRead(250)) {
            readData.append(serialPort.readAll());
            int nowsize = readData.size();
            if (nowsize > lastProgressDot + PROGRESS_DOT_INTERVAL) {
                    standardout << '.' << flush;
                    lastProgressDot += PROGRESS_DOT_INTERVAL;
            }
            if (nowsize > lastProgressText + PROGRESS_TEXT_INTERVAL) {
                    standardout << " " << lastProgressDot/1024 << "K" << endl
 << flush;
                    lastProgressText += PROGRESS_TEXT_INTERVAL;
            }
```

```
        }

        standardout << endl;

        if (serialPort.error() == QSerialPort::ReadError) {
                standardout << QString("Failed to read from port %1: %2").arg(
    serialPortName).arg(serialPort.errorString()) << endl;
                return 1;
        } else if (serialPort.error() == QSerialPort::TimeoutError && readData.
    isEmpty()) {
                standardout << QString("No data was received").arg(serialPortName
    ) << endl;
                return 1;
        }

        standardout << QString("Total: %1 bytes received and saved to %2.").arg(
    readData.size()).arg(logFileName) << endl;

        logStream.writeBytes(readData.constData(), readData.size());

        standardout << '\a'; // ring system bell

        return 0;
}
```

Appendix B

Extract image data

```
import os, sys
from datetime import datetime
from struct import *
from array import array
import numpy as np
from netpbmfile import imsave
from raw2ppm import raw2ppm

if len(sys.argv) < 2:
        print 'Not enough arguments. Specify log file name.'
        sys.exit()

if not os.path.isfile(sys.argv[1]):
        print 'Log file not found'
        sys.exit()

logfilename = sys.argv[1]

fdata = open(logfilename, "rb").read()

start = fdata.find('####')
end = fdata.find('****')
if start == -1 or end == -1:
    print 'Could not find start or end marker for size'
    sys.exit()

start = start+4 # skip leading ####
parameters = fdata[start:end].split(':') # split parameters separated by :
if len(parameters) != 3:
        print 'Not 3 parameters: ' + str(parameters)
        sys.exit()
address = parameters[0].lower()
```

```
sizestr = parameters[1].lower()
skipstr = parameters[2].lower()
size = int(sizestr, 16)
skip = int(skipstr, 16)
start = end+4
end = start + size

data = fdata[start:end]
filename = datetime.now().strftime('%Y-%m-%dT%H-%M-%S.%f') + ' '+ address +'-' +
    sizestr + '.dat'

udata = array("H")
udata.fromstring(data)
npdata = np.array(udata, dtype=np.uint16)
#np.set_printoptions(edgeitems = 5)
print npdata.shape
npdata = npdata.reshape((1944/skip,-1))
print npdata.shape
print npdata

imsave(filename + '.pgm', npdata, maxval=2**12 -1)
npdata.tofile(open(filename, 'wb'))
raw2ppm(filename + '.pgm', filename + '.ppm')
```

Appendix C

Xilinx logiCORE IP code

```
# ##############################################################################
# Created by Base System Builder Wizard for Xilinx EDK 14.5 Build EDK_P.58f
# Fri Oct 10 23:37:56 2014
# Target Board:  Numato Lab Saturn_LX45 Rev 3.0
# Family:    spartan6
# Device:    xc6slx45
# Package:   csg324
# Speed Grade:  -2
# ##############################################################################
 PARAMETER VERSION = 2.1.0


 PORT rzq = rzq, DIR = IO
 PORT mcbx_dram_we_n = mcbx_dram_we_n, DIR = O
 PORT mcbx_dram_udqs = mcbx_dram_udqs, DIR = IO
 PORT mcbx_dram_udm = mcbx_dram_udm, DIR = O
 PORT mcbx_dram_ras_n = mcbx_dram_ras_n, DIR = O
 PORT mcbx_dram_ldm = mcbx_dram_ldm, DIR = O
 PORT mcbx_dram_dqs = mcbx_dram_dqs, DIR = IO
 PORT mcbx_dram_dq = mcbx_dram_dq, DIR = IO, VEC = [15:0]
 PORT mcbx_dram_clk_n = mcbx_dram_clk_n, DIR = O, SIGIS = CLK
 PORT mcbx_dram_clk = mcbx_dram_clk, DIR = O, SIGIS = CLK
 PORT mcbx_dram_cke = mcbx_dram_cke, DIR = O
 PORT mcbx_dram_cas_n = mcbx_dram_cas_n, DIR = O
 PORT mcbx_dram_ba = mcbx_dram_ba, DIR = O, VEC = [1:0]
 PORT mcbx_dram_addr = mcbx_dram_addr, DIR = O, VEC = [12:0]
 PORT RESET = RESET, DIR = I, SIGIS = RST, RST_POLARITY = 1
 PORT FT2232_UART_SOUT = FT2232_UART_SOUT, DIR = O
 PORT FT2232_UART_SIN = FT2232_UART_SIN, DIR = I
 PORT CLK_100MHZ = CLK_100MHZ, DIR = I, SIGIS = CLK, CLK_FREQ = 100000000
 PORT sensor_pixclk = v_vid_in_axi4s_0_vid_in_clk, DIR = I, SIGIS = CLK
```

```
 PORT sensor_de = v_vid_in_axi4s_0_vid_de, DIR = I
 PORT sensor_vblank = v_vid_in_axi4s_0_vid_vblank, DIR = I
 PORT sensor_hblank = v_vid_in_axi4s_0_vid_hblank, DIR = I
 PORT sensor_data = v_vid_in_axi4s_0_vid_data, DIR = I, VEC = [11:0]
 PORT sensor_extclk = clock_generator_0_CLKOUT3, DIR = O, SIGIS = CLK, CLK_FREQ =
     96000000
 PORT axi_spi_0_SCK_pin = axi_spi_0_SCK, DIR = IO
 PORT axi_spi_0_MISO_pin = axi_spi_0_MISO, DIR = IO
 PORT axi_spi_0_MOSI_pin = axi_spi_0_MOSI, DIR = IO
 PORT axi_spi_0_SS_pin = axi_spi_0_SS, DIR = IO
 PORT iic_gpo = axi_iic_0_Gpo, DIR = O, VEC = [7:0]
 PORT iic_sda = axi_iic_0_Sda, DIR = IO
 PORT iic_scl = axi_iic_0_Scl, DIR = IO


BEGIN proc_sys_reset
 PARAMETER INSTANCE = proc_sys_reset_0
 PARAMETER HW_VER = 3.00.a
 PARAMETER C_EXT_RESET_HIGH = 1
 PORT MB_Debug_Sys_Rst = proc_sys_reset_0_MB_Debug_Sys_Rst
 PORT Dcm_locked = proc_sys_reset_0_Dcm_locked
 PORT MB_Reset = proc_sys_reset_0_MB_Reset
 PORT Slowest_sync_clk = clk_100_0000MHzPLL0
 PORT Interconnect_aresetn = proc_sys_reset_0_Interconnect_aresetn
 PORT Ext_Reset_In = RESET
 PORT BUS_STRUCT_RESET = proc_sys_reset_0_BUS_STRUCT_RESET
 PORT Peripheral_aresetn = proc_sys_reset_0_Peripheral_aresetn
 PORT Peripheral_Reset = proc_sys_reset_0_Peripheral_Reset
END

BEGIN lmb_v10
 PARAMETER INSTANCE = microblaze_0_ilmb
 PARAMETER HW_VER = 2.00.b
 PORT SYS_RST = proc_sys_reset_0_BUS_STRUCT_RESET
 PORT LMB_CLK = clk_100_0000MHzPLL0
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = microblaze_0_i_bram_ctrl
 PARAMETER HW_VER = 3.10.c
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00001fff
 BUS_INTERFACE SLMB = microblaze_0_ilmb
 BUS_INTERFACE BRAM_PORT = microblaze_0_i_bram_ctrl_2_microblaze_0_bram_block
END

BEGIN lmb_v10
 PARAMETER INSTANCE = microblaze_0_dlmb
```

```
 PARAMETER HW_VER = 2.00.b
 PORT SYS_RST = proc_sys_reset_0_BUS_STRUCT_RESET
 PORT LMB_CLK = clk_100_0000MHzPLL0
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = microblaze_0_d_bram_ctrl
 PARAMETER HW_VER = 3.10.c
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00001fff
 BUS_INTERFACE SLMB = microblaze_0_dlmb
 BUS_INTERFACE BRAM_PORT = microblaze_0_d_bram_ctrl_2_microblaze_0_bram_block
END

BEGIN bram_block
 PARAMETER INSTANCE = microblaze_0_bram_block
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = microblaze_0_i_bram_ctrl_2_microblaze_0_bram_block
 BUS_INTERFACE PORTB = microblaze_0_d_bram_ctrl_2_microblaze_0_bram_block
END

BEGIN microblaze
 PARAMETER INSTANCE = microblaze_0
 PARAMETER HW_VER = 8.50.a
 PARAMETER C_INTERCONNECT = 2
 PARAMETER C_USE_BARREL = 1
 PARAMETER C_USE_FPU = 1
 PARAMETER C_DEBUG_ENABLED = 1
 PARAMETER C_ICACHE_BASEADDR = 0xa4000000
 PARAMETER C_ICACHE_HIGHADDR = 0xa7ffffff
 PARAMETER C_USE_ICACHE = 1
 PARAMETER C_CACHE_BYTE_SIZE = 8192
 PARAMETER C_ICACHE_ALWAYS_USED = 1
 PARAMETER C_DCACHE_BASEADDR = 0xa4000000
 PARAMETER C_DCACHE_HIGHADDR = 0xa7ffffff
 PARAMETER C_USE_DCACHE = 1
 PARAMETER C_DCACHE_BYTE_SIZE = 8192
 PARAMETER C_DCACHE_ALWAYS_USED = 1
 BUS_INTERFACE ILMB = microblaze_0_ilmb
 BUS_INTERFACE DLMB = microblaze_0_dlmb
 BUS_INTERFACE M_AXI_DP = axi4lite_0
 BUS_INTERFACE M_AXI_DC = axi4_0
 BUS_INTERFACE M_AXI_IC = axi4_0
 BUS_INTERFACE DEBUG = microblaze_0_debug
 PORT MB_RESET = proc_sys_reset_0_MB_Reset
 PORT CLK = clk_100_0000MHzPLL0
END
```

```
BEGIN mdm
 PARAMETER INSTANCE = debug_module
 PARAMETER HW_VER = 2.10.a
 PARAMETER C_INTERCONNECT = 2
 PARAMETER C_USE_UART = 1
 PARAMETER C_BASEADDR = 0x41400000
 PARAMETER C_HIGHADDR = 0x4140ffff
 BUS_INTERFACE S_AXI = axi4lite_0
 BUS_INTERFACE MBDEBUG_0 = microblaze_0_debug
 PORT Debug_SYS_Rst = proc_sys_reset_0_MB_Debug_Sys_Rst
 PORT S_AXI_ACLK = clk_100_0000MHzPLL0
END

BEGIN clock_generator
 PARAMETER INSTANCE = clock_generator_0
 PARAMETER HW_VER = 4.03.a
 PARAMETER C_CLKIN_FREQ = 100000000
 PARAMETER C_CLKOUT0_FREQ = 400000000
 PARAMETER C_CLKOUT0_GROUP = PLL0
 PARAMETER C_CLKOUT0_BUF = FALSE
 PARAMETER C_CLKOUT1_FREQ = 400000000
 PARAMETER C_CLKOUT1_PHASE = 180
 PARAMETER C_CLKOUT1_GROUP = PLL0
 PARAMETER C_CLKOUT1_BUF = FALSE
 PARAMETER C_CLKOUT2_FREQ = 100000000
 PARAMETER C_CLKOUT2_GROUP = PLL0
 PARAMETER C_CLKOUT3_FREQ = 48000000
 PARAMETER C_CLKOUT3_BUF = TRUE
 PORT LOCKED = proc_sys_reset_0_Dcm_locked
 PORT CLKOUT2 = clk_100_0000MHzPLL0
 PORT RST = RESET
 PORT CLKOUT0 = clk_400_0000MHzPLL0_nobuf
 PORT CLKOUT1 = clk_400_0000MHz180PLL0_nobuf
 PORT CLKIN = CLK_100MHZ
 PORT CLKOUT3 = clock_generator_0_CLKOUT3
END

BEGIN axi_interconnect
 PARAMETER INSTANCE = axi4lite_0
 PARAMETER HW_VER = 1.06.a
 PARAMETER C_INTERCONNECT_CONNECTIVITY_MODE = 0
 PORT INTERCONNECT_ARESETN = proc_sys_reset_0_Interconnect_aresetn
 PORT INTERCONNECT_ACLK = clk_100_0000MHzPLL0
END

BEGIN axi_interconnect
 PARAMETER INSTANCE = axi4_0
 PARAMETER HW_VER = 1.06.a
```

```
 PORT interconnect_aclk = clk_100_0000MHzPLL0
 PORT INTERCONNECT_ARESETN = proc_sys_reset_0_Interconnect_aresetn
END

BEGIN axi_s6_ddrx
 PARAMETER INSTANCE = LPDDR
 PARAMETER HW_VER = 1.06.a
 PARAMETER C_MCB_RZQ_LOC = N4
 PARAMETER C_MCB_ZIO_LOC = NOT_SET
 PARAMETER C_MEM_TYPE = MDDR
 PARAMETER C_MEM_PARTNO = MT46H32M16XXXX-5
 PARAMETER C_MEM_BANKADDR_WIDTH = 2
 PARAMETER C_MEM_NUM_COL_BITS = 10
 PARAMETER C_SKIP_IN_TERM_CAL = 1
 PARAMETER C_S0_AXI_ENABLE = 1
 PARAMETER C_INTERCONNECT_S0_AXI_MASTERS = microblaze_0.M_AXI_DC & microblaze_0.
    M_AXI_IC & axi_vdma_0.M_AXI_S2MM
 PARAMETER C_MEM_DDR2_RTT = 50OHMS
 PARAMETER C_S0_AXI_STRICT_COHERENCY = 0
 PARAMETER C_INTERCONNECT_S0_AXI_AW_REGISTER = 8
 PARAMETER C_INTERCONNECT_S0_AXI_AR_REGISTER = 8
 PARAMETER C_INTERCONNECT_S0_AXI_W_REGISTER = 8
 PARAMETER C_INTERCONNECT_S0_AXI_R_REGISTER = 8
 PARAMETER C_INTERCONNECT_S0_AXI_B_REGISTER = 8
 PARAMETER C_S0_AXI_BASEADDR = 0xa4000000
 PARAMETER C_S0_AXI_HIGHADDR = 0xa7ffffff
 BUS_INTERFACE S0_AXI = axi4_0
 PORT rzq = rzq
 PORT s0_axi_aclk = clk_100_0000MHzPLL0
 PORT ui_clk = clk_100_0000MHzPLL0
 PORT mcbx_dram_we_n = mcbx_dram_we_n
 PORT mcbx_dram_udqs = mcbx_dram_udqs
 PORT mcbx_dram_udm = mcbx_dram_udm
 PORT mcbx_dram_ras_n = mcbx_dram_ras_n
 PORT mcbx_dram_ldm = mcbx_dram_ldm
 PORT mcbx_dram_dqs = mcbx_dram_dqs
 PORT mcbx_dram_dq = mcbx_dram_dq
 PORT mcbx_dram_clk_n = mcbx_dram_clk_n
 PORT mcbx_dram_clk = mcbx_dram_clk
 PORT mcbx_dram_cke = mcbx_dram_cke
 PORT mcbx_dram_cas_n = mcbx_dram_cas_n
 PORT mcbx_dram_ba = mcbx_dram_ba
 PORT mcbx_dram_addr = mcbx_dram_addr
 PORT sysclk_2x = clk_400_0000MHzPLL0_nobuf
 PORT sysclk_2x_180 = clk_400_0000MHz180PLL0_nobuf
 PORT SYS_RST = proc_sys_reset_0_BUS_STRUCT_RESET
 PORT PLL_LOCK = proc_sys_reset_0_Dcm_locked
END
```

```
BEGIN axi_uartlite
 PARAMETER INSTANCE = FT2232_UART
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_BAUDRATE = 921600
 PARAMETER C_DATA_BITS = 8
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_ODD_PARITY = 1
 PARAMETER C_BASEADDR = 0x40600000
 PARAMETER C_HIGHADDR = 0x4060ffff
 BUS_INTERFACE S_AXI = axi4lite_0
 PORT S_AXI_ACLK = clk_100_0000MHzPLL0
 PORT TX = FT2232_UART_SOUT
 PORT RX = FT2232_UART_SIN
END

BEGIN v_vid_in_axi4s
 PARAMETER INSTANCE = v_vid_in_axi4s_0
 PARAMETER HW_VER = 2.01.a
 PARAMETER C_M_AXIS_VIDEO_DATA_WIDTH = 12
 PARAMETER C_M_AXIS_VIDEO_FORMAT = 12
 PARAMETER RAM_ADDR_BITS = 5
 PARAMETER HYSTERESIS_LEVEL = 8
 BUS_INTERFACE M_AXIS_VIDEO = v_vid_in_axi4s_0_M_AXIS_VIDEO
 PORT aclk = clk_100_0000MHzPLL0
 PORT vid_in_clk = v_vid_in_axi4s_0_vid_in_clk
 PORT rst = proc_sys_reset_0_Peripheral_Reset
 PORT aresetn = proc_sys_reset_0_Peripheral_aresetn
 PORT vid_data = v_vid_in_axi4s_0_vid_data
 PORT aclken = net_vcc
 PORT axis_enable = net_vcc
 PORT vid_de = v_vid_in_axi4s_0_vid_de
 PORT vid_vblank = v_vid_in_axi4s_0_vid_vblank
 PORT vid_hblank = v_vid_in_axi4s_0_vid_hblank
END

BEGIN axi_vdma
 PARAMETER INSTANCE = axi_vdma_0
 PARAMETER HW_VER = 5.04.a
 PARAMETER C_NUM_FSTORES = 1
 PARAMETER C_DYNAMIC_RESOLUTION = 0
 PARAMETER C_INCLUDE_MM2S = 0
 PARAMETER C_S_AXIS_S2MM_TDATA_WIDTH = 16
 PARAMETER C_S2MM_MAX_BURST_LENGTH = 16
 PARAMETER C_ENABLE_VIDPRMTR_READS = 1
 PARAMETER C_BASEADDR = 0x7e200000
 PARAMETER C_HIGHADDR = 0x7e20ffff
 PARAMETER C_S2MM_LINEBUFFER_DEPTH = 128
```

```
 PARAMETER C_INCLUDE_S2MM_DRE = 1
 BUS_INTERFACE S_AXI_LITE = axi4lite_0
 BUS_INTERFACE M_AXI_S2MM = axi4_0
 BUS_INTERFACE S_AXIS_S2MM = v_vid_in_axi4s_0_M_AXIS_VIDEO
 PORT s_axi_lite_aclk = clk_100_0000MHzPLL0
 PORT m_axi_s2mm_aclk = clk_100_0000MHzPLL0
 PORT s_axis_s2mm_aclk = clk_100_0000MHzPLL0
END

BEGIN axi_spi
 PARAMETER INSTANCE = axi_spi_0
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_FIFO_EXIST = 0
 PARAMETER C_SCK_RATIO = 2
 PARAMETER C_BASEADDR = 0x40a00000
 PARAMETER C_HIGHADDR = 0x40a0ffff
 BUS_INTERFACE S_AXI = axi4lite_0
 PORT S_AXI_ACLK = clk_100_0000MHzPLL0
 PORT SCK = axi_spi_0_SCK
 PORT MISO = axi_spi_0_MISO
 PORT MOSI = axi_spi_0_MOSI
 PORT SS = axi_spi_0_SS
END

BEGIN axi_iic
 PARAMETER INSTANCE = axi_iic_0
 PARAMETER HW_VER = 1.02.a
 PARAMETER C_GPO_WIDTH = 8
 PARAMETER C_BASEADDR = 0x40800000
 PARAMETER C_HIGHADDR = 0x4080ffff
 BUS_INTERFACE S_AXI = axi4lite_0
 PORT S_AXI_ACLK = clk_100_0000MHzPLL0
 PORT Gpo = axi_iic_0_Gpo
 PORT Sda = axi_iic_0_Sda
 PORT Scl = axi_iic_0_Scl
END
```

Appendix D

User Constraint File

```
NET "clk_100mhz" LOC = V10 | IOSTANDARD = LVCMOS33;
TIMESPEC TS_CLK = PERIOD "clk_100mhz" 100 MHz HIGH 50%;

NET "nreset" IOSTANDARD = LVTTL | LOC = G13 | PULLUP; # external pullup, used for
     button

NET "uart_in" LOC = L17 | IOSTANDARD = LVCMOS33;
NET "uart_out" LOC = L18 | IOSTANDARD = LVCMOS33;

NET "iic_sda" LOC = B12 | IOSTANDARD = I2C;
NET "iic_scl" LOC = D11  | IOSTANDARD = I2C;

NET "pixclk" LOC = D9 | IOSTANDARD = LVCMOS33; #resoldered
NET "pixclk" PERIOD = 96 MHz HIGH 50 %;


NET "extclk" LOC = A9 | IOSTANDARD = LVCMOS33;
NET "extclk" PERIOD = 96 MHz HIGH 50 %;

NET "frame_valid" LOC = A13 | IOSTANDARD = LVCMOS33;
NET "line_valid" LOC = B14 | IOSTANDARD = LVCMOS33;

NET "data[3]" LOC = B11 | IOSTANDARD = LVCMOS33;
NET "data[2]" LOC = G9 | IOSTANDARD = LVCMOS33;
NET "data[1]" LOC = B9 | IOSTANDARD = LVCMOS33;
NET "data[0]" LOC = B8 | IOSTANDARD = LVCMOS33;
NET "data[4]" LOC = C11 | IOSTANDARD = LVCMOS33;
NET "data[5]" LOC = A12 | IOSTANDARD = LVCMOS33;
NET "data[6]" LOC = F13 | IOSTANDARD = LVCMOS33;
NET "data[7]" LOC = E13 | IOSTANDARD = LVCMOS33;
NET "data[10]" LOC = A15 | IOSTANDARD = LVCMOS33;
NET "data[11]" LOC = C18 | IOSTANDARD = LVCMOS33;
```

```
NET "data[9]" LOC = C15 | IOSTANDARD = LVCMOS33;
NET "data[8]" LOC = A14 | IOSTANDARD = LVCMOS33;

#strobe input at A7, not currently used (Not verified, spring 2015)
NET "gpo[0]" LOC = H16 | IOSTANDARD = LVCMOS33; #led
#NET "gpo[1]" LOC = C7 | IOSTANDARD = LVCMOS33; # standby_bar, external pullup
#NET "gpo[2]" LOC = D8 | IOSTANDARD = LVCMOS33; # oe_bar, external pullup
NET "gpo[1]" LOC = C7 | IOSTANDARD = LVCMOS33; # standby_bar, external pullup
NET "gpo[2]" LOC = F9 | IOSTANDARD = LVCMOS33; # oe_bar, external pullup,  unused
     pin, might not work
NET "gpo[3]" LOC = A3 | IOSTANDARD = LVCMOS33; # img_enable, external pulldown
NET "gpo[4]" LOC = G14 | IOSTANDARD = LVCMOS33; #led
NET "gpo[5]" LOC = F16 | IOSTANDARD = LVCMOS33; #Led
NET "gpo[6]" LOC = B4 | IOSTANDARD = LVCMOS33; #changed from LED4 to test pin


NET "oe_bar" LOC = C8 | IOSTANDARD = LVCMOS33; # always low

NET "axi_spi_0_SCK_pin" LOC = R15  |  IOSTANDARD = LVCMOS33;
NET "axi_spi_0_SS_pin" LOC = V3  |  IOSTANDARD = LVCMOS33;
NET "axi_spi_0_MOSI_pin" LOC = T13  |  IOSTANDARD = LVCMOS33;
NET "axi_spi_0_MISO_pin" LOC = R13  |  IOSTANDARD = LVCMOS33;
```

Bibliography

[1] A. Bertheussen. Digital processing system for a cubesat camera. 2014. URL http://nuts.cubesat.no/upload/2015/03/06/digital_processing_for_cubesat_camera_andreas_bertheussen.pdf.

[2] ON Semiconductor. Image sensor color correction. Rev. 3, 2015. URL http://www.onsemi.com/pub_link/Collateral/TND6114-D.PDF.

[3] Numato Lab. Saturn spartan 6 fpga development board user guide. Rev. 9, 2011. URL http://community.numato.com/api/productdata/assets/downloads/fpga/saturn/SaturnSpartan6ModuleV9.pdf.

[4] C. J. Hawthorn, K. P. Weber, and R. E. Scholten. Littrow configuration tunable external cavity diode laser with fixed direction output beam. *Review of Scientific Instruments*, 72(12):4477–4479, December 2011. URL http://nuts.cubesat.no/upload/2012/01/20/nuts-1_mission.pdf.

[5] T. H. Nornes. Prototype design for cubesat camera. 2014. URL http://nuts.cubesat.no/upload/2015/03/06/prototype_design_for_cubesat_by_thomas_hanssen_nornes.pdf.

[6] J. K. Oltedal. Review of the hardware description of the camera module prototype for ntnu test satellite (nuts). 2015. URL http://nuts.cubesat.no/upload/2016/03/12/prosjektoppgave_ferdig_oltedal.pdf.

[7] Aptina imaging. Mt9p031 register reference. Rev. A, 2011. URL http://dl.btc.pl/kamami_wa/mt9p031_rr.pdf.

[8] ON Semiconductor. 1/2.5-inch 5 mp cmos digital image sensor. Rev. J, 2015. URL http://www.onsemi.com/pub_link/Collateral/MT9P031-D.PDF.

[9] M. Parmarac J. Farrella, M. Okinchab and B. Wanellac. Using visible snr (vsnr) to compare image quality of pixel binning and digital resizing. Technical report, Stanford University, 2010. URL http://scien.stanford.edu/jfsite/Papers/ImageCapture/vSNR_PixelBinningSPIE_111809.pdf.

[10] S. W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 2nd edition, 199. URL http://ft-sipil.unila.ac.id/dbooks/The%20Scientist%20and%20Engineer's%20Guide%20to%20Digital%20Signal%20Process.pdf.

[11] Mayeul Marcadella. Improvement in the reliability of a bi-processing unit satellite subject to radiation-induced bit-flips. Technical report, NTNU, 2014. URL http://daim.idi.ntnu.no/masteroppgaver/011/11553/masteroppgave.pdf.

[12] Xilinx. 7 series fpgas clocking resources. V. 1.11.2, 2015. URL http://www.xilinx.com/support/documentation/user_guides/ug472_7Series_Clocking.pdf.

[13] D. Sinclair J. Enright and K. C. Fernando. Cots detector for nanosatellite star trackers: A case study. Technical report, Ryerson University, 2011. URL http://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=1162&context=smallsat.

[14] Xilinx. Chipscope pro 11.4 software and cores. V. 11.4, 2009. URL http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/chipscope_pro_sw_cores_ug029.pdf.

[15] M. A. Normann. Hardware review of an on board controller for a cubesat. 2015. URL http://nuts.cubesat.no/upload/2015/03/06/prototype_design_for_cubesat_by_thomas_hanssen_nornes.pdf.

[16] Xilinx. Spartan-6 fpga configuration user guide. V. 2.8, 2015. URL http://www.xilinx.com/support/documentation/user_guides/ug380.pdf.

[17] M. Fabiano P. Prinetto M. Caramia, S. D. Carlo. Flash-memories in space applications: Trends and challenges. Technical report, Thales Alenia Space, and Politecnico di Torino, 2009. URL http://porto.polito.it/2296440/2/2009_EWDTS_Flash_AuthorVersion.pdf.

[18] FTDI Chip. Future technology devices international ltd ft2232h. V. 2.3, 2016. URL http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT2232H.pdf.