# AAUSAT6 Camera solution

an image capturing unit for use in a cubesat


P4 project report

Group 413

Aalborg University
Electronic Engineering & IT
Frederiks Bajersvej 7
DK-9000 Aalborg



This report is compiled in LaTeX, originally developed by Leslie Lamport, based on Donald Knuth's TeX. The main text is written in *Computer Modern* pt 11, designed by Donald Knuth. Flowcharts and diagrams are made using Microsoft Visio.

**Institute of Electronic Systems**
Fredrik Bajers Vej 7
DK-9220 Aalborg Ø

**AALBORG UNIVERSITY**
STUDENT REPORT

**Title:**
AAUSAT6 camera solution

**Abstract:**
This project is going to be awesome!

**Theme:**
Digital Systems Design

**Project Period:**
P4: 2. February 2015 - 27. May 2015

**Project Group:**
15gr412

**Participants:**
Amalie Vistoft Petersen
Mikkel Krogh Simonsen
Rasmus Gundorff Sæderup
Simon Bjerre Krogh
Thomas Kær Juel Jørgensen
Thomas Rasmussen

**Supervisor:**
Jens Dalsgaard Nielsen

**Copies:** 8

**Page Numbers:** ??

**Date of Completion:**
September 7, 2015

# Contents

# Todo list

# Preface

Something awesome about a satellite and its story to achieve an epic camera payload.

Aalborg Universitet, 17. december 2014


Amalie "Chewie" Vistoft Petersen
apet13@student.aau.dk

Mikkel 'Hulk' Krogh Simonsen
mksi13@student.aau.dk

Rasmus 'Onion Knight' Gundorff Sæderup
rsader13@student.aau.dk

Simon 'Nightmare' Bjerre Krogh
skrogh13@student.aau.dk

Thomas 'T-bone' Kær Juel Jørgensen
tkjj13@student.aau.dk

Thomas "Godlike" Rasmussen
trasm12@student.aau.dk

# Part I

# Preanalysis

# 1 | Design considerations

In this section, the system will be designed with a top-down approach, starting with a use-case description of the overall functionalities of the system. Hereafter, the constraints set by the satellite, with regards to physical limitations, downlink speed etc. is considered. Based on the use-case descriptions and the constraints, it is possible to make an overall system design, describing the systems building blocks. The requirements for the system will also be listed.

## 1.1 Use case design

Before the payload can be designed, a few design considerations has to be made, in order to establish what the payload must be able to do. This will be done by means of a UML use case diagram, see Figure 1.1.

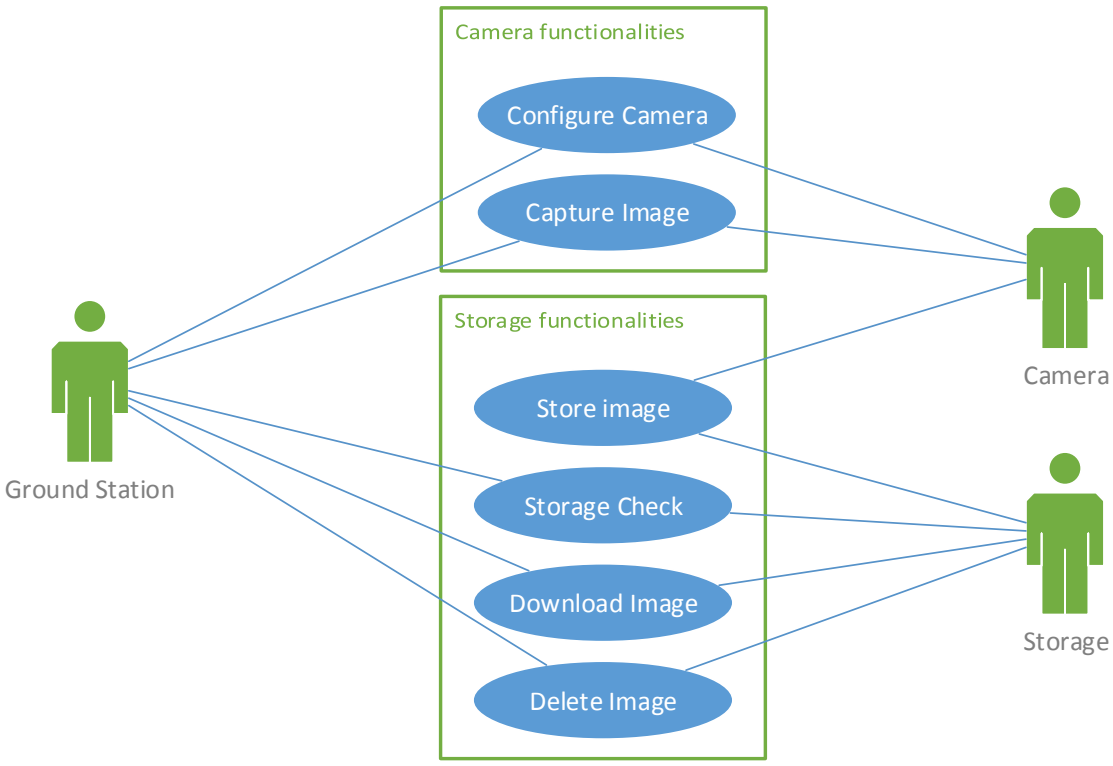



**Figure 1.1:** A usecase diagram showing the overall functionalities of the system

As previously mentioned, the main purpose of the payload is to automatically capture images with a interval set by Ground Station, store them on board the satellite, and be able to transmit them to the ground, so the images can be viewed.

The ground stations function is to set up the payload. Therefore, all the functionalities of the payload are to be controlled from the ground station. This includes configuring the camera, i.e. change settings, and downloading images from the satellite.

The ground station should also be able to delete images and do a storage check of the satellite, so the storage space can be managed manually to prevent overload.
Also, being able to remotely capture an image from the ground station should be possible, if it is wished to take an images during a pass.

## 1.2 Payload constraints

Before the building blocks of the system can be established, some considerations has to be done with respect to the limitations set by the platform the system is to be implemented in, i.e. a cubesat.

To minimize the amount of data to be sent from the satellite to the ground station, it is decided that the demosaicing will be performed on the ground station. This is because the demosaicing procedure produces 3 times as much data as the original image, which is not ideal when downlink time has to be considered.
The images must also be saved in a commonly used file format, so they can be seen and used by other users who do not have any special software available that might otherwise be needed to view the image. Such formats could be JPEG og BMP images. Here, BMP will be chosen since this format is much easier to save images as than JPEG, which can be seen in Section **??**: **??**. However, the actual conversion of the raw image data to a BMP file will not be done in this project.

Since the payload is to be implemented in a cubesat having a size of minimum 1U, the size of the entire satellite must not be greater than $10 \times 10 \times 11$ cm. see Section **??**: **??**. A maximum payload height of 4 cm has been established as a reasonable and realistic size, when considering the printed circuit board (PCB) height of the other subsystems. Also, since the satellite has to fit the frame, solar panels etc. the PCBs cannot be larger than $87 \times 87$ mm in size, as can be seen in the schematics/board layouts of the AAUSAT PCBs.

Because the satellite has a power budget of 1.5 watts on average, see Section **??**: **??**, the payload must not use more power than this, since the satellite will then be drained for all power. Also, there should also be enough power to supply the other subsystems. The exact power consumption of the subsystems is not known, but a rough estimate is that half the power of the satellite goes to the payload and the other half to the rest of the subsystems. The exact power consumption is not critical, since the EPS will shut down the payload if the battery voltage drops below a certain voltage. Also, since the 1.5 Watts available on the satellite is an average power, a spontaneously higher power consumption of the payload is allowed, since it will only be on some of the time, thus averaging down below the allowed power consumption threshold. Nonetheless, a low power consumption is desired, to ensure the payload to be used for as long as possible per battery charge.

As investigated in Section **??**: **??**, the downlink speed sets a limitation on how big the image

files should be, to be able to download them in a reasonable amount of time. Here, it is decided that a reasonable download time is 2 days. This can be achieved either by taking small photos or compressing the images. Since it is not desired to decrease image quality, a lossless compression scheme is to be used.
However, it would be very impractical to wait so long to download every image before it is known if the image actually contains useful image data or is just a "black frame"[1]. This can be solved by rendering small resolution previews of all the images. These thumbnails can then be sent to the ground station, where the operator can choose which images to download in full resolution. Assuming an 8-bit black and white preview, with a size of 100 × 100 pixels, a pass duration of 10 minutes and a downlink speed of 9600 bps, 24 previews per pass should theoretically be obtainable, as can be calculated from Equation 1.1. Based on these ideal conditions, it is estimated that in reality, the ability to download at least 20 previews per pass is both a sufficient and realistic amount.

$$\frac{\text{baud rate} \cdot \text{pass length}}{\text{preview size} \cdot \text{bit depth} \cdot \text{data ratio}} = \frac{\text{previews}}{\text{pass}} \tag{1.1}$$

Where:

| | |
|---|---|
| Baud rate | is the transfer ratio [bit/s] |
| pass length | is the duration of a pass [s] |
| preview size | is the resolution of the preview [pixels] |
| bit depth | is the number of bits per pixel [bits/pixel] |
| Data ratio | is the ratio between data and package size in CSP, i.e. 84/250 [-] |

To further improve the amount of useful previews being downloaded per pass, the payload could automatically discard the blackframes so they do not have to be downloaded.

It would be preferable to make a time lapse video of an entire orbit. This could be done by capturing an image each second for a full orbit, and then stitch these together to form a movie. This would however set a requirement to the storage capabilities of the system so it can store these images.
Assuming a 5 megapixel image, with 8 bits per pixel (since the demosaicing is not done on the spacecraft), one image would take up 5 MB of storage space. Since an orbit takes about 100 minutes, see Section **??**: **??**, this would mean that 6000 images has to be stored, summing to a total of 30 GB. There should of course also be room to store low-resolution previews of these images, but these would only take up 60 MB, assuming an 8-bit black and white preview, with a size of 100 × 100 pixels, which is negligible.

Based on the analysis performed in Section **??**: **??** and Section **??**: **??**, it is possible to consider the optimum object area that the camera photographs, as well as the necessary pixel density, also known as spatial resolution. This decision is however left up to the designers of AAUSAT6, since it depends on the desired object size which is not known at this time. It also depends on the orbit altitude, which is also not known. When these factors are known, the field of view

[1] A blackframe is defined as a frame containing a certain percent of only black pixels

and spatial resolution can be calculated based on the graphs in Section **??**: **??** and Section **??**: **??**. The field of view and spatial resolution can be changed as wished, by choosing a suitable lens and sensor. The choice of lens and sensor that will be made in this project is therefore not critical, since it can be changed if necessary, as long as the interface to the camera unit is identical.

Finally, due to financial restrictions, the PCBs used will not be of space grade.

## 1.3 Requirements

Based on the preanalysis, the considerations and system overview, see Section 1.2: Payload constraints, it is possible to set up the requirements for the system.

The requirements are highlighted with **bold**, and the reason behind each requirement is listed underneath each requirement.

1. **The payload shall be able to capture an image and store it on onboard memory**

   The purpose of the payload is to capture images as well as store it for further processing, e.g. compression and downlink to ground station, see Section 1.1: Use case design.

2. **The payloads dimensions should not exceed 87x87x50 mm**

   This is the PCB size used in the current AAUSATs, see Section 1.2: Payload constraints

3. **The payload shall be able to communicate with other AAUSAT subsystems using CAN and CSP**

   Since the subsystems communicate via CAN and CSP, the payload should support these interfaces, see Section **??**: **??**.

4. **The power consumption of the payload shall not exceed 0.75 watt on average**[2]

   The payload should not use more than half of the available power on the satellite on average, see Section 1.2: Payload constraints

5. **The payload shall be able to send an image from storage through the communication (COM), to the ground station where it can be saved as an image**

   Captured images should be viewable to the operator, see Section 1.2: Payload constraints

6. **From the ground station, it shall be possible to change the workings of the payload, in terms of resolution and when images should be captured**

   It should be possible to change settings depending how and when images should be captured, see Section **??**: **??**

7. **The payload shall be able to compress the image lossless such that it can be downloaded from space in less than 2 days**

   Downloading an image should be done in a reasonable amount of time without degrading image quality, Section 1.2: Payload constraints

[2] Meaning the average power consumption measured during the entire mission, i.e. the time when the system is off is taken into account

8. **The payload must be able to detect black frames[3] and not store these**

   Images not containing any useful information should not be stored, see Section 1.2: Payload constraints

9. **At least 10 previews must be able to be downloaded from space per pass**

   The payload must be able to render and downlink small resolution previews of the captured images, as per Section 1.2: Payload constraints.

10. **From the ground station, it shall be possible to request previews and full resolution images as well as remote capture an image**

    Since the ground station acts as the main interface between the operator and the payload, it should be possible to download previews and full resolution images through it. The operator shall also be able to manually remote capture an image, see Section 1.1: Use case design.

11. **From the ground station, it shall be possible to delete pictures and view on-board memory statistics**

    It is important that the memory can be managed manually from the ground station in case of memory issues that the payload cannot solve automatically, see Section 1.2: Payload constraints

12. **The payload must have at least 30 GB of storage space**

    To ensure that images from a full pass can be stored in on-board memory, see Section 1.2: Payload constraints

[3] A blackframe is defined as a frame containing a certain percent of only black pixels - "a shot in the dark", so to speak

# Part II

# Test & conclusion

# Bibliography

Omnivision (2009). Ov5647 datasheet. http://www.seeedstudio.com/wiki/images/3/3c/Ov5647_full.pdf. Downloaded February 13 2015.

# Appendix

## A | Reverse engineering the Raspberry Pi camera

The Raspberry Pi camera is for the project, as it is readily available at the university. This means it will be easy and quick to get a new camera, if it should stop working.

Unfortunately, there is currently no complete documentation available for the camera, so most of the register setup must be done by reverse engineering the camera and the interaction between the Raspberry Pi and the camera.

The schematic for the camera module is not being provided by the Raspberry Pi Foundation, so the first step is to determine the pinout of the camera. Most of it can be found in the schematic for the Raspberry Pi, see Figure 1.2

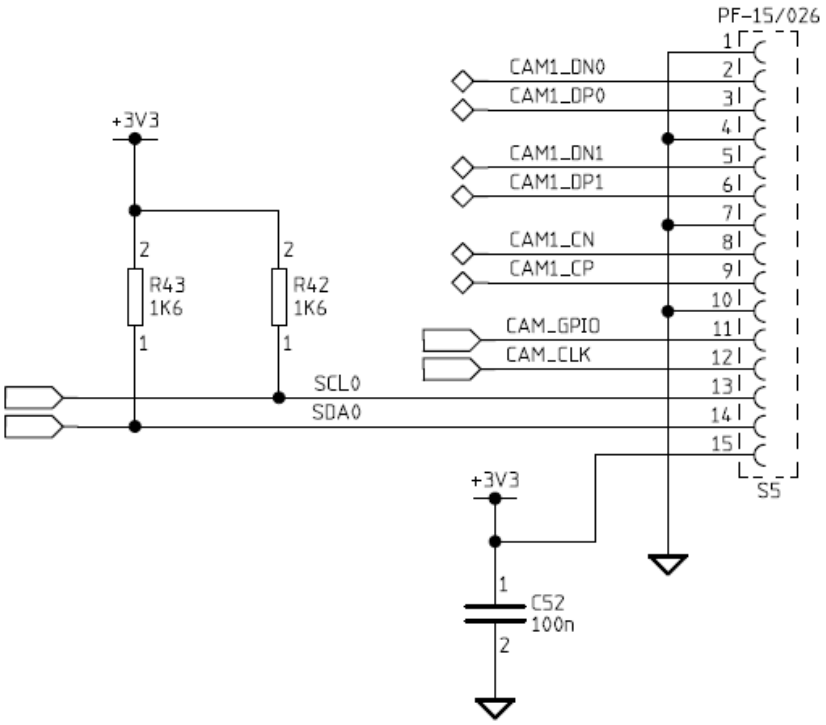



**Figure 1.2:** Pinout of the Raspberry Pi camera connector [source: http://www.rs-online.com/designspark/electronics/knowledge-item/r-pi-ffc-connectors]

The majority of these connections can be recognized from the CSI-2 specifications, and a preliminary table can be put together of the camera pinout. Only two connections remains unknown, see Table 1.1.

| Pin number | Name | Description |
|---|---|---|
| 1, 4, 7 and 10 | GND | Supply Ground |
| 2 | CAM1_DN0 | D-PHY Differential Data Lane 0 |
| 3 | CAM1_DP0 | |
| 5 | CAM1_DN1 | D-PHY Differential Data Lane 1 |
| 6 | CAM1_DP1 | |
| 8 | CAM1_CN | D-PHY Differential Clock |
| 9 | CAM1_CP | |
| 11 | CAM_GPIO | Undocumented |
| 12 | CAM_CLK | Undocumented |
| 13 | SCL0 | I²C bus |
| 14 | SDA0 | |
| 15 | +3V3 | Supply 3.3V |

**Table 1.1:** Camera pinout

As the pinout is now known, a breakout board can be made to connect between the camera and the Raspberry Pi. The board provides easy access to the high speed data lines via BNC connectors, and pinheaders for listening on the I²C bus.

The PCB can be seen on Figure 1.3.

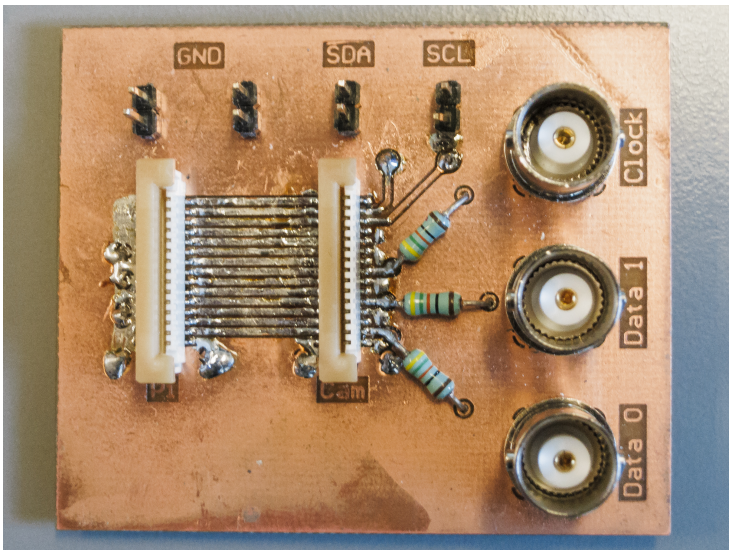



**Figure 1.3:** Breakout board for monitoring camera signals.

Now step two can begin, the actual reverse engineering. The Raspberry Pi will be commanded to take pictures in various resolution, while the I²C bus is monitored with an oscilloscope, a Rigol

DS1102E. This oscilloscope provides a "Deep memory"-Mode, where up to a million samples can be captured, and afterwards saved as a .csv file. The test setup can be seen on Figure 1.4

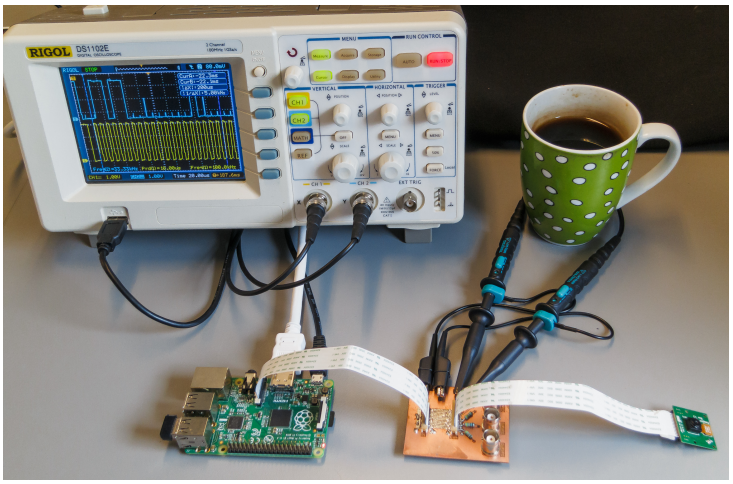
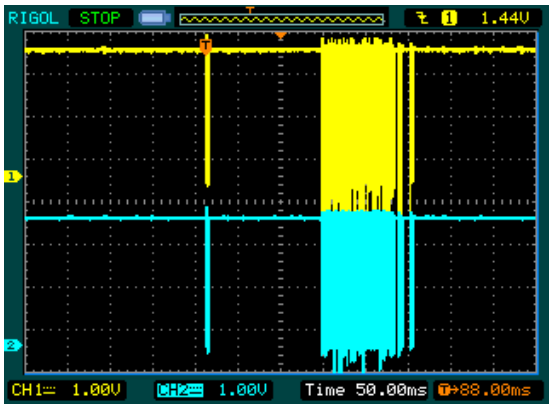


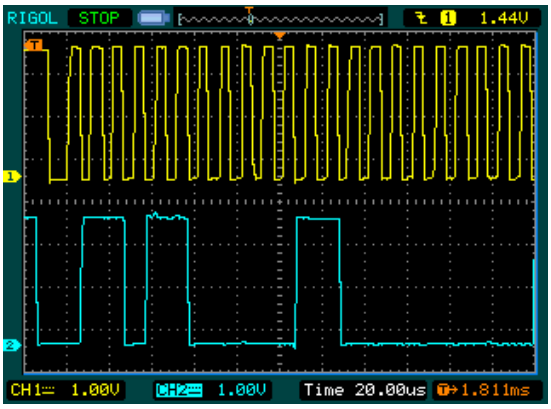

**Figure 1.4:** Test setup for monitoring the I²C bus

A screenshot of a capture can be seen on Figure 1.5a. As seen on the screenshot, the Raspberry Pi sends a few short data burst, followed by a long burst. The short bursts contains very little data, and are easily decoded manually, by watching the waveforms.

A zoomed in view of the initial short burst can be seen on Figure 1.5b.




**(a)** Captured I²C data




**(b)** Zoomed in short burst

**Figure 1.5:** Image captures containing I²C burst from the setup on Figure 1.4

Decoding a couple of transfers in this data results in the following bit sequences:

| **Binary:** | 011 0110 | 0 | 0 | 0000 0001 | 0 | 0000 0000 | 0 | 0000 0000 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| **Hex:** | 0x36 | 0 | 0 | 0x01 | 0 | 0x00 | 0 | 0x00 | 0 |
| **Description:** | address | write | ack | data | ack | data | ack | data | ack |

**Translation:** "Write 0x01, 0x00 and 0x00 to I²C address 0x36"

| **Binary:** | 011 0110 | 0 | 0 | 0000 0001 | 0 | 0000 0000 | 0 |
|---|---|---|---|---|---|---|---|
| **Hex:** | 0x36 | 0 | 0 | 0x01 | 0 | 0x00 | 0 |
| **Description:** | address | write | ack | data | ack | data | ack |

**Translation:** "Write 0x01 and 0x00 to I²C address 0x36"

| **Binary:** | 011 0110 | 1 | 0 | 0000 0000 | 1 |
|---|---|---|---|---|---|
| **Hex:** | 0x36 | 1 | 0 | 0x00 | 1 |
| **Description:** | address | read | ack | data | ack |

**Translation:** "Read from I²C address 0x36" (0x00 is read from the slave)

| **Binary:** | 011 0110 | 0 | 0 | 0000 0001 | 0 | 0000 0011 | 0 | 0000 0001 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| **Hex:** | 0x36 | 0 | 0 | 0x01 | 0 | 0x03 | 0 | 0x01 | 0 |
| **Description:** | address | write | ack | data | ack | data | ack | data | ack |

**Translation:** "Write 0x01, 0x03 and 0x01 to I²C address 0x36"

According to [Omnivision, 2009, p. 77] I²C slave address of the camera is 0x6C for writes, and 0x6D for reads. As the I²C standard calls for 7 bit addresses, followed by a read/write bit, looking at the provided addresses in binary gives the answer why the decoded address is different from what the datasheet claims:

| **Hex:** | 0x6C | 0x6D |
|---|---|---|
| **Binary:** | 0110 1100 | 0110 1101 |

The datasheet is simply interpreting the read/write bit as part of the address.

According to [Omnivision, 2009, p. 23], a software sleep can be commanded, by writing 0x00

to register address 0x0100. This matches the first transfer. The first two bytes are the register address, and the last is the data to write.

Starting the configuration with a sleep command makes sense - there is no need for the camera to be active until it has been fully configured. The next two transfers reads back the written data from the last transfer. Why this is done is unknown, but it might be for making sure the camera is working correctly.

Now that the transfer format, has been undestood, it is clear that the last of the four transfers must be "Open register 0x0103, and write 0x01". According to the datasheet, this is the command to reset the sensor, again something which makes sense to do during initialization, to make sure all registers are in a well known state.

The long databurst now needs to be decoded. Because of the large amount of data, this will need to be automated. The oscilloscope can save the captured data to a Comma Separated Values (csv) file.

Now Sigrok, an open source signal analysis software suite will be used (`http://www.sigrok.org`). This package contains a tool for reading a .csv file from a logic analyser, and finding I²C packets. As the data has been captured with an oscilloscope, and not a logic analyzer, the data will need to be reformatted in a way sigrok understands.

The oscilloscope saves the captured data as the voltages of the two channels, together with a timestamp. Sigrok expects only 1's and 0's. The conversion from voltages to binary values is easily done using Matlab. The data is imported, and the clock and data channels identified, see Figure 1.6

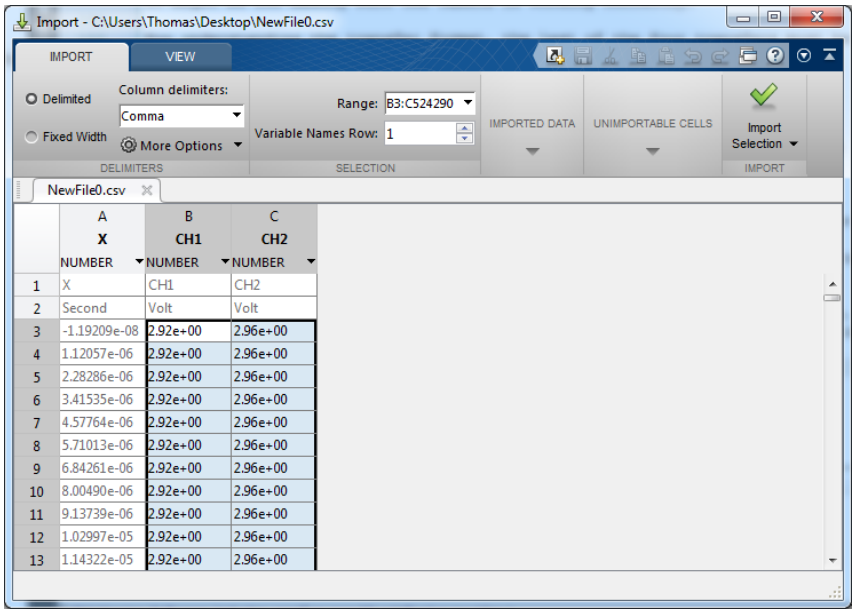



**Figure 1.6:** Importing oscilloscope data.

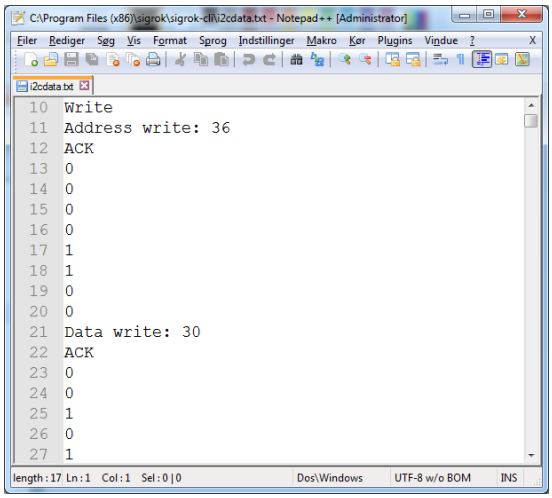
The voltages can now be converted to 1's and 0's, and saved as a new .csv file using the following commands:

**Listing 1.1:** Matlab commands

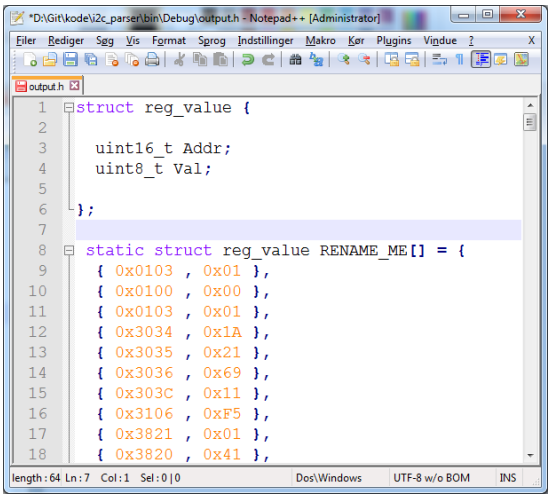
```
CH1(CH1<2.7)=0
CH2(CH2<2.7)=0
CH1(CH1>2.6)=1
CH2(CH2>2.6)=1
DATA = [CH1, CH2]
csvwrite('i2cdata.csv',DATA)
```

The generated .csv file can now be processed by Sigrok. This will analyse the data and create a text file with the I²C transfers, see Figure 1.7a.

Now that the actual I²C data has been obtained, it has to be converted to a format that can be easily included in the microcontroller software. To do this, a small C program is written, that takes the sigrok data in and outputs a C header file, containing an array with the data to be written to the camera, see Figure 1.7b




**(a)** Sigrok output data




**(b)** Generated C header file

**Figure 1.7:** Screenshots describing output data and the generated C header file

The above procedure with the decoding of I²C packages and generation of registers based on these is repeated for the relevant image resolutions, to get the register set for each. An example of such a register set can be seen in Table 1.2.

| Register Address | Description | Default data | Configured data |
|---|---|---|---|
| 0x3034 | SC_CMMN_PLL_CTRL0:<br><br>Bit[6:4]: pll_charge_pump<br>Bit[3:0]: mipi_bit_mode<br>0000: 8 bit mode<br>0001: 10 bit mode<br>Others: Reserved to future use | 0x1A | 0x1A |

**Table 1.2:** Example of a register description from the datasheet. Note that the Default configuration is "Reserved to future use", which is rather odd.

With the register setups at hand, the camera can now be powered on without the Raspberry Pi. During the testing, an MSP430 MCU development board is used to send the I²C data, as it is a simple architecture to get up and running, and is configured for 3.3V operation just like the camera. An Arduino could have been used, but this would require I²C level shifters, as the Arduino uses 5V logic levels.

As an initial test, the sleep command is sent to the camera and then read back. Unfortunately, the camera ignores the I²C transfer, and appears completely dead.

Probing around the camera board with the oscilloscope, it turns out that the oscillator providing the camera clock is not running. This must be caused by one of the two undocumented pins on the camera board. It is presumed that the CAM_CLK pin could be responsible for activating the oscillator, and pulling this pin to a logic high, in fact makes in oscillator starts up. At the same time an LED on the camera board lights up.

With the oscillator now running, the sleep command is sent to the camera again, again without result. According to [Omnivision, 2009, p. 23: reset], using a hardware reset pin is recommended, even though the camera supports software reset.

As it is very likely that the camera board designers have read the same section, it is almost certain that the last pin (CAM_GPIO) controls this. Pulling this pin to a logic high, the camera starts responding to I²C commands.
Sending one of the captured data bursts causes the CSI bus to become active and start streaming data. The camera is now ready for connecting to the MIPI bridge in the FPGA.

# B | Fibonacci proof using induction

The Fibonacci sequence is defined as:

$$F_i = F_{i-1} + F_{i-2} \qquad \text{for} \quad i \geq 2 \tag{1.2}$$
$$F_1 = 1, F_2 = 1 \tag{1.3}$$

We want to prove that:

$$\sum_{i=1}^{n} F_i = F_{n+2} - 1 \tag{1.4}$$

**Basis step:**

We want to show that our statement holds for the smallest i, namely i = 2:

$$F_1 + F_2 \stackrel{?}{=} F_{2+2} - 1 = F_4 - 1 \tag{1.5}$$
$$F_1 + F_2 \stackrel{?}{=} (F_3 + F_2) - 1 \tag{1.6}$$
$$F_1 + F_2 \stackrel{?}{=} ((F_2 + F_2) + F_2) - 1 \tag{1.7}$$
$$F_1 + F_2 \stackrel{?}{=} 1 + 1 + 1 - 1 \tag{1.8}$$
$$2 = 2 \tag{1.9}$$
$$\tag{1.10}$$

**Induction step:** Now that we have shown that the statement holds for n, we want to show that the statement holds for $n + 1$, by replacing $n$ with $n + 1$:

$$\sum_{i=1}^{n+1} F_i \stackrel{?}{=} F_{(n+1)+2} - 1 \tag{1.11}$$
$$F_1 + F_2 + F_3 + ... + F_n + F_{n+1} \stackrel{?}{=} F_{n+3} - 1 \tag{1.12}$$
$$F_1 + F_2 + F_3 + ... + F_n + F_{n+1} \stackrel{?}{=} F_{n+2} + F_{n+1} - 1 \tag{1.13}$$
$$\left(\sum_{i=1}^{n} F_i\right) + F_{n+1} \stackrel{?}{=} F_{n+2} + F_{n+1} - 1 \tag{1.14}$$

Now, we can use what we have proven in the basis step, to reduce this:

$$\left(\sum_{i=1}^{n} F_i\right) + F_{n+1} \stackrel{?}{=} F_{n+2} + F_{n+1} - 1 \tag{1.15}$$
$$F_{n+1} = F_{n+1} \tag{1.16}$$

$$Q.E.D$$

Since this is true, the original statement must therefore also be true, and the proof is complete.