CCD Controller for Counting Photons
User Guide

Revision 3.2.10

every photon counts

Contents

List of Tables

List of Figures

List of Acronyms

ADC Analog to Digital Converter

ADU Analog to Digital Unit

AFE Analogic Front-end

CCCP CCD Controller for Counting Photons

CCD Charge Coupled Device

DAC Digital to Analog Converter

DNS Domain Name Server

DSP Digital Signal Processor

EM Electron Multiplying

FIFO First-in First-out

FPGA Field-programmable Gate Array

HV High Voltage

LSB Least significant byte

MSB Most significant byte

PCB Printed Circuit Board

PLL Phase Locked Loop

SERDES Serialized-Deserializer

SRAM Static Random Access Memory

TCP Transmission Control Protocol

Chapter 1

Introduction

This guide presents the version 3 of the CCD Controller for Counting Photons (CCCP), a CCD controller optimized for use with an EMCCD. It is intended for users whom would like to use a CCCP to drive a CCD.

CCCP allows arbitrary waveforms to be generated and used as clocks for a CCD. It can also generate a high voltage clock for the EM register of an EMCCD.

Chapter 2 explains the different components of the CCCP.
Chapter 3 lays out the philosophy of operation of the controller.
Chapter 4 presents *cccpComm* for communication and configuration of CCCP.
Chapter 5 presents *cccpServer*, an application linking *cccpComm* to *cccpView*.
Chapter 6 presents *cccpView*, a tool to display and/or record images collected.
Chapter 7 describes in depth the CCCP's technical aspects.
Chapter 8 describes the controller's mechanical housing.
Chapter 9 explains the terms of the warranty.
As outlined in Section 2.1.6, the controller can be used in either virtual or real VSS mode. Switching between the two modes requires the replacement of some resistors on the controller board. The controller is shipped with a virtual VSS configuration, which implies that the VSS pins of the CCD are expected to be connected to ground. Should a real VSS operation be required, please contact Nüvü Camēras.

Chapter 2

Hardware Overview

2.1 Controller board

This section contains an overview of hardware components of the controller board of CCCP. The controller board is one of two boards needed for CCCP, the other one being the power supply board covered in Section 2.3. This board is connected to the CCD interface through the connector covered in Section 7.5. It can clock the CCD with 14 phases (see Section 2.1.5), it can feed the CCD interface with 13 biases (see Section 2.1.6), it can clock the EM registers of an EMCCD with up to two high voltage clocks (see Section 7.2) and it can digitize two different CCD video outputs.

2.1.1 FPGA

The on-board Virtex-6 FPGA comprises mainly of the sequencer module (see Section 2.1.2), the CDS module (see Section 2.1.3) and an embedded microprocessor used to communicate with the host computer and to configure the rest of the other modules of the controller board.

The FPGA is automatically booted and configured from a flash memory on each start-up. The embedded processor's firmware is also loaded from a flash memory after the FPGA has been configured.

2.1.2 Sequencer

The sequencer module, instantiated in the FPGA, is the synchronous master of all of the clocks generated by the controller. It controls what pattern (see Section 3.1) to execute and when, as well as many features of the controller. It comprises of the following:

- 14 memories, one for each of the 14 phases that hold the data for pattern generation plus one memory for digital signals generation. The sequencer allows for up to 20,000 steps to be defined for the patterns.
- The logic to execute the steps from the memories adequately as stated by the sequences (see Section 3.2).
- The logic to execute the sequences adequately as stated by the ROS (see Section 3.3).

- The logic for handling the controller's external signals outputs (arm, fire, shutter) and input (trigger) with the following options:
  - Configurable signal polarity.
  - Internal or external trigger, with the possibility of controlling the exposure time through an external trigger.
  - Possibility of controlling an external shutter through the shutter signal and compensating for the external shutter's opening and closing delays.
- Capacity to generate the clock used by the controller's ADC and the data acquisition logic of the FPGA for the digitization of the EMCCD's outputs.

It is configured by the host computer through the embedded FPGA processor by using *cccpComm* and the serial commands listed in the serial commands section (2.2). Its main feature is to generate the addresses necessary to execute the blank, pre-expose and expose patterns as well as the sequences patterns. The generated addresses will fetch the right data from the FPGA memories to output the patterns. This feature is explained in more details in Section 3. In short, the sequencer handles the execution of all the synchronous items of a read-out sequence, an overview of that procedure is given in Figure 2.1.

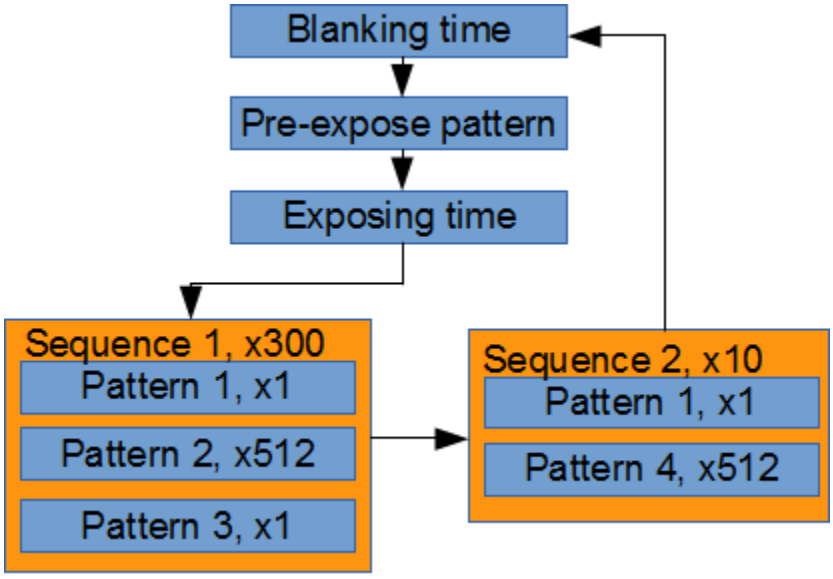

Figure 2.1: Sequencer steps execution.

The sequencer can be configured to execute a ROS on an external TTL trigger input (through the external trigger signal) or in response to the **re** serial command (internal trigger). When using the external trigger functionality, the sequencer can be configured to either define the exposure time with the trigger signal or to just define the beginning of the exposure time. More details about the external signals can be found at the Section 3.

2.1.3 CDS

Correlated double sampling (CDS) is a technique for quantifying sensor values which have a floating reset level followed by a signal level. First, the reset level is measured as a reference and then, the signal is measured to capture signal strength. By then computing the difference between the first and the second measurement, the result will be the sensor's output value and an accurate representation of the signal level with respect to the reference level. For a Charge-Coupled-Device (CCD), this represents the amount of photons converted to electrons in each pixel. In addition, this technique exhibits additional advantages such as DC or low-frequency signal removal and reset-noise reduction.

CCD output waveform

The Charge-Coupled-Device (CCD) analog output is composed of three parts :

- A reset pulse which is undesired in the pixel calculation,
- A reference level which is the reset level for each pixel,
- And a signal level which determines the intensity of the pixel.

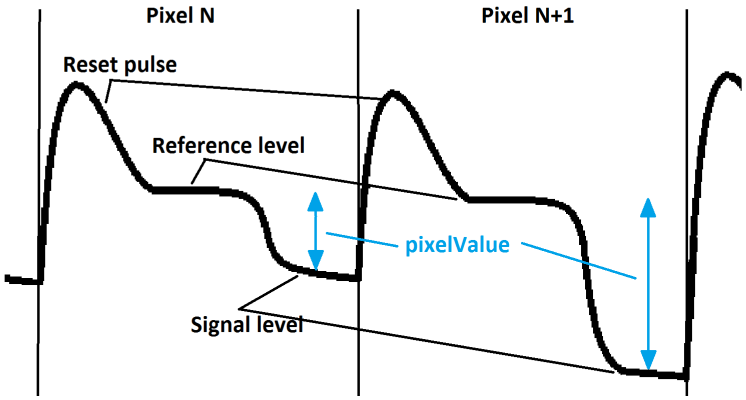

Figure 2.2: CCD output waveform.

Implementation and synchronization

In the CCCP-3 controller, the CDS algorithm is computed digitally by VHDL modules in the FPGA; a schematic of its location on the CCCP-3 board is given in Figure 2.3. In the CDS module, the processed equation is the following :

$$\text{pixelValue} = \frac{\sum(P_i.D_i)}{\text{divisor}} + \text{offset}$$

where :

i is the position of the sample in the pixel

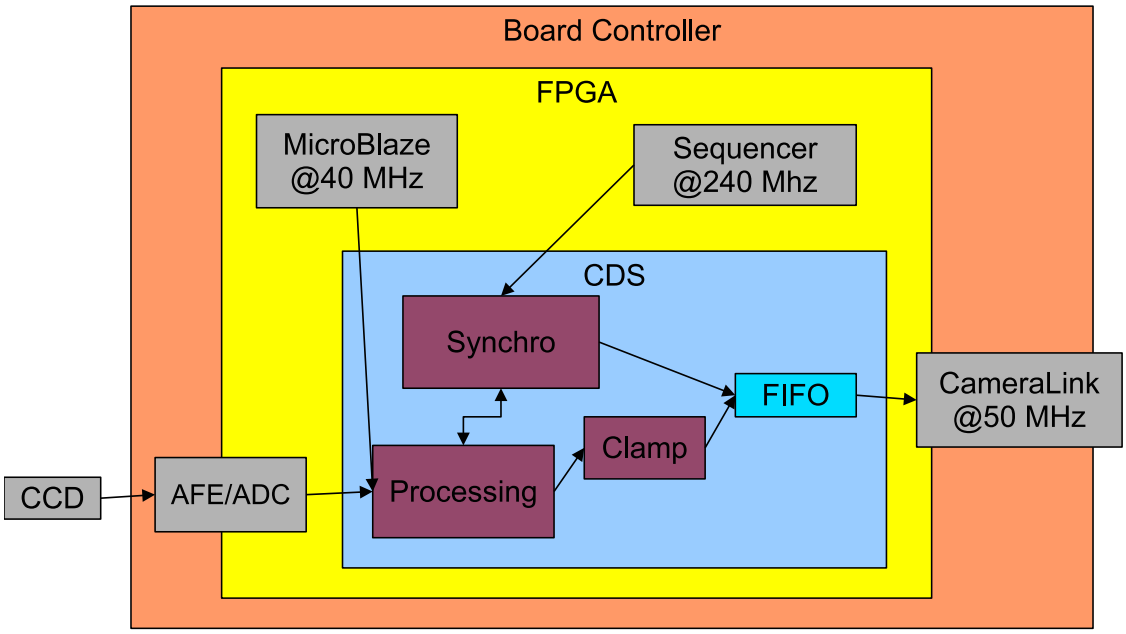

Figure 2.3: CDS module environment.

P_i is the weight of the sample i $\begin{cases} > 0 & \text{when } i \text{ is a sample of the reference level} \\ = 0 & \text{when } i \text{ is in the reset pulse or in the transition state or in the transition state.} \\ < 0 & \text{when } i \text{ is a sample of the signal level} \end{cases}$

D_i is the value of the sample i

$\mathrm{divisor}$ is the divisor applied to the pixel

offset is the offset from zero of all values in the image

and for processing purposes, the module uses memory to set weights for the CDS samples and arithmetic cores to implement the equation, as modeled in Figure 2.4.

As indicated, the module's memory stores the weights P_i of each pixel; the first weight P_0 is stored at the address 0 and will be used to multiply the first sample D_0 of each incoming pixel and so on for each sample i. The maximum number of weights is determined by the pixel rate frequency : the lower the pixel rate frequency is, the higher the number of samples per pixel will be. For example, with a pixel rate frequency of 100 kHz and with a sample clock frequency at 120 MHz, the number of samples per pixel will be 1,200. Consequently, the BRAM depth has been set to 2048, which gives a minimum pixel rate of 59 kHz. Weights are also used to make the subtraction between the reference level and the signal level: all weights associated with a sample of the reference level should be positive and all weights associated with a sample of the signal level should be negative. All weights associated with samples captured in a transition state or in a reset pulse should be equal to zero.

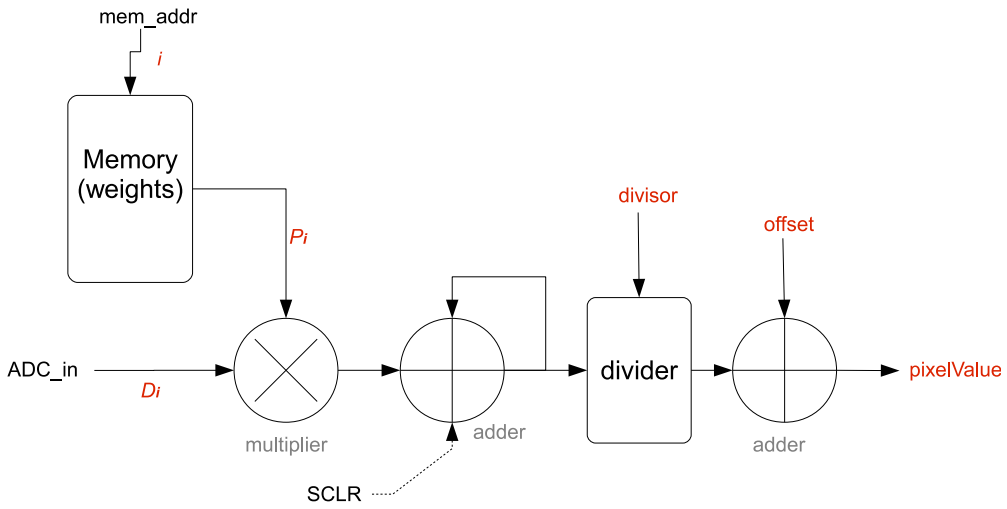

Figure 2.4: CDS processing algorithm implementation.

To synchronize the algorithm with the beginning of a pixel, the sequencer feeds the CDS module with three synchronization signals : FVAL (*frame valid*), LVAL (*line valid*) and DVAL (*data valid*). These signals are used by the CDS to align the processing part with the incoming samples. When a FVAL or a LVAL toggles, the position of the sample *i* in a pixel is reset to 0. If there is no skipped sample set by the *cdsskip -n* function, the next valid data (FVAL, LVAL and DVAL active) will be the first sample of the first pixel in the line. If there are *n* skipped samples, the beginning of the pixel will be the $(n+1)^{th}$ valid sample. These signals are thus extremely important and should be correctly set in the readout sequences. FVAL has to be cleared before the beginning of a new frame and LVAL should be cleared before a new line. Moreover, to ensure the right alignment of the incoming ADC data with those synchronization signals, the operating frequency of the ADC should be a divider of 240 MHz which is the operating frequency of the sequencer.

Once the pixel value is calculated, a clamp module verifies if the data is negative or if it exceeds 65535. If it is negative, the output data will be set to 0 and if it is superior to 65535, the output value will be clamped at 65535. After this verification, the data and three synchronization signals are sent to the CameraLink serializer through a FIFO.

CDS configuration

The easiest way to configure the CDS module is to use the following three serial commands :

- **cdsskip**, defined on page 24
- **cdsweightsbuild**, defined on page 25
- **cdsoffset**, defined on page 24

The function **cdsweightsbuild** calculates automatically the weights to apply to each sample. It sets positive weights for samples which belong to the reference level, negative weights for samples which belong to the signal level and all others weights are set to zero. If the number of reference sample is equal to the

number of signal samples then all reference weights will be set to 1 and all signal weights will be set to −1. If the number of reference samples and signal samples are unbalanced the function will set the weights by calculating the smallest common multiplier between these two numbers and divide it by the number of associated samples. For example, if there are 3 reference samples and 2 signal samples per pixel then the weights for reference samples will be $\frac{6}{3} = 2$ and the weights for signal samples will be $\frac{6}{2} = 3$. It also calculates the divisor to apply to the pixel. If the number of reference samples is equal to the number of signal samples then the divisor will be equal to the reference weights multiplied by the number of reference samples. If it is not equal then the divisor will be the smallest common multiplier calculated for the weights construction.

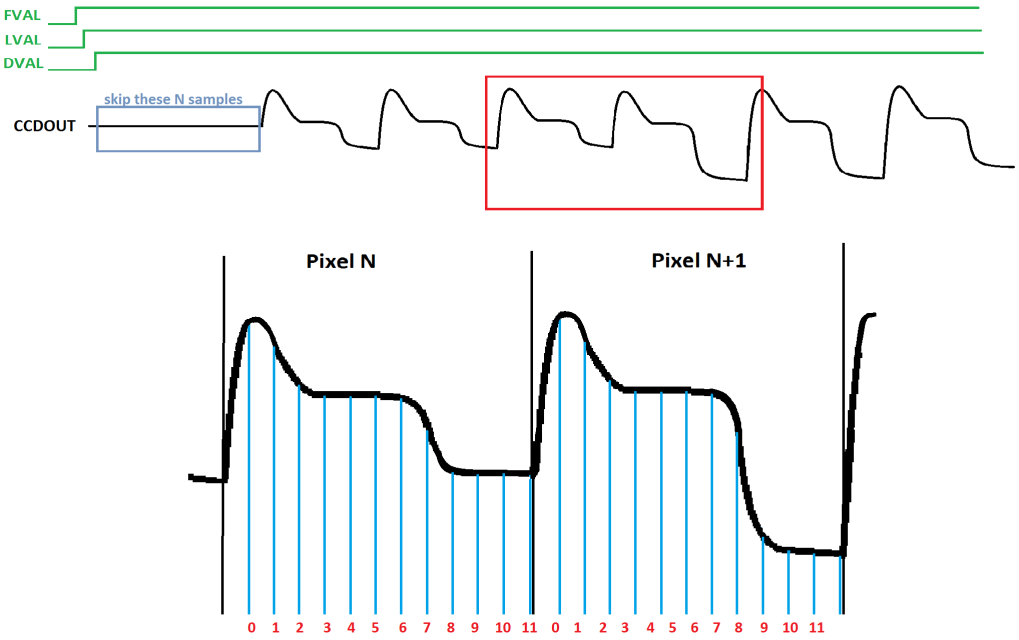

Figure 2.5: Example of a CCD output sampling.

In the example shown in Figure 2.5, there are *N* samples to skip at the beginning of the line and 12 samples per pixel. Amongst these 12 samples, 4 belong to the reference level (3 to 6) and 4 others belong to the signal level (8 to 11); all others have to be ignored in the processing. Given this sampling, their is no guarantee that the smallest pixel value generated by the CDS will be greater than zero, depending upon how the floating reference level evolves during readout. To ensure unsigned pixel values an offset *x* may be applied to the values generated by the CDS. The settings for this example are thus :

- **cdsskip** *N*
- **cdsweightsbuild 12 3 4 1 4**
- **cdsoffset** *x*

The number of reference samples is equal to the number of signal samples and thus the reference weights will be set to 1 and signal weights will be set to −1. The divisor will be set to 4.

Table 2.1: Configuration of the CDS weights for the example shown in the figure 2.5

i	0	1	2	3	4	5	6	7	8	9	10	11
P_i	0	0	0	1	1	1	1	0	−1	−1	−1	−1
divisor	4											

To configure the CDS with more fine control, it is possible to adjust the weight of each sample. Instead of setting all weights automatically, it is possible to overwrite manually each weight in the BRAM with the **cdsweights** serial command, defined on page 25.

Pass-through mode

The pass-through mode disables the CDS algorithm and sends all valid data (FVAL, LVAL and DVAL active) to the frame grabber through the CameraLink interface. In order for the frame grabber to detect a new frame or a new line, the CDS in pass-through mode also sends data when the signals FVAL and LVAL toggle. This mode is useful for debugging or while an ADC test pattern is in progress. This mode is nonetheless limited by the CameraLink output rate which is 50 MHz.

2.1.4 The CameraLink interface

The controller outputs video data through a CameraLink interface. The serial communication is also embedded through the CameraLink interface. For very specific cases, the timing delay for the 4 CameraLink data streams can be adjusted by using the **cldelay** serial command, defined on page 34.

2.1.5 Synchronous items

The synchronous items of the controller board are those that can be changed in sync with the read-out of the CCD. They are the following:

- 14 BRAM–DAC chains. These chains allow the data stored in the memory of the sequencer to be transformed into low power analog clock signals. The DACs have 14-bit resolution.
- 14 power amplifiers. These power amplifiers take the low power generated by the DACs and amplify them into the range of −15 to +15 volts at high current (350 mA sustained). The bandwidth of these power amplifiers are adjusted to produce either the horizontal or vertical clocks of the CCD. The bandwidth of the power amplifiers are presented in Table 2.3.
- 1 or 2 high voltage clocks. This resonant high voltage clock is used to clock the EM register of an EMCCD.
- 1 ADC (16 bits). This ADC is used to digitize the output signal of a CCD at a 16-bit resolution and at a sampling rate of 120 MSPS. This ADC has 4 inputs, as such it is possible to easily switch between two outputs of a CCD with the **ads5263channel** serial command, defined on page 18. It is an ADS5263 from Analog Devices. Its sampling clock is driven by the sequencer module. Usually, when driving an EMCCD, the EM output is connected to the ADC channel 1 and the conventional output is connected to the ADC channel 4.

The synchronous signals are summarized in Table 2.2. The phase's address and number are also provided.

Table 2.2: Description of the synchronous signals generated by the detector board

Signal name	Address	Description
Phase 0	0 @[0:13]	Analog 14 bits signal
Phase 1	1 @[0:13]	Analog 14 bits signal
Phase 2	2 @[0:13]	Analog 14 bits signal
Phase 3	3 @[0:13]	Analog 14 bits signal
Phase 4	4 @[0:13]	Analog 14 bits signal
Phase 5	5 @[0:13]	Analog 14 bits signal
Phase 6	6 @[0:13]	Analog 14 bits signal
Phase 7	7 @[0:13]	Analog 14 bits signal
Phase 8	8 @[0:13]	Analog 14 bits signal
Phase 9	9 @[0:13]	Analog 14 bits signal
Phase 10	10 @[0:13]	Analog 14 bits signal
Phase 11	11 @[0:13]	Analog 14 bits signal
Phase 12	12 @[0:13]	Analog 14 bits signal
Phase 13	13 @[0:13]	Analog 14 bits signal
Phase 14	14 @[0:3]	(HV_CLK0): Digital 4 bits data for the serializer driving one of the HV clocks, giving about 1-ns resolution. The low level of the high voltage clock is adjustable from 0 to 20 volts by the bias at address 10. The high level of the high voltage clock is adjustable from 25 to 50 volts by the biases at address 11. The frequency of the resulting serialized digital signal will be the frequency of the HV clock. **When the HV clock is not in use, this digital signal must be held high (value of 1)**[1].
Phase 14	14 @[4:7]	(HV_CLK1): Digital 4 bits data for the serializer driving one of the HV clocks, giving about 1-ns resolution. The low level of the high voltage clock is adjustable from 0 to 20 volts by the bias at address 12. The high level of the high voltage clock is adjustable from 25 to 50 volts by the biases at address 4. The frequency of the resulting serialized digital signal will be the frequency of the HV clock. **When the HV clock is not in use, this digital signal must be held high (value of 1)**[2].
DVAL	14 @8	Digital data valid signal for the frame grabber.
LVAL	14 @9	Digital line valid signal for the frame grabber.
FVAL	14 @10	Digital frame valid signal for the frame grabber.
ADC_SCLK_SYNC	14 @11	ADC sync signal, assert this signal for one sequencer step at the very begining of a ROS to synchronise the ADC sampling clock with the ROS.
PS_RESET	14 @15	Power supply sync signal, see Section 2.3 on how to use this signal.

[1] HV_CLK0 is not available by default, although it could be used as a second HV clock should the user wish it, please contact Nüvü Camēras should the user want a second HV clock.

[2] HV_CLK1 is the default HV clock (more information on this clock at Section 7).

The vertical clocks of a CCD typically have a high capacitance ($>$ 1 nF). The horizontal clocks typically have a lower capacitance ($<$ 300pF). They are usually switched at a higher frequency than the vertical clocks. The high dV/dt and the high bandwidth can cause ringing of the amplifiers when driving a light capacitive load or when the propagation delay on the line exceeds $\sim$ 2 ns. In order to avoid this effect, serial resistances have been placed between the output of the amplifier and the output pin, the placed serial resistances are shown in Table 2.2. See Section 7.6 for information about the effects of the propagation delay, serial resistance and self-inductance of the transmission line from the power amplifier to the CCD pin.

The configuration of the power amplifiers is summarized in Table 2.3. The propagation delay between the amplifier output and the edge-mount connector are shown in the last column and are approximate values. More details on the power amplifiers are given in Section 7.3.

Table 2.3: Configuration of the power amplifiers

Phase	Serial resistance (Ω)[1]	Bandwidth (MHz)[2]	$\mathbf{t_{pd}}$[3] (ps)
Phase0	0	40	125
Phase1	0	40	47
Phase2	0	40	98
Phase3	0	40	34
Phase4	0	40	116
Phase5	0	40	62
Phase6	0	40	79
Phase7	0	40	46
Phase8	0	40	111
Phase9	0	40	36
Phase10	0	40	97
Phase11	0	40	69
Phase12	0	40	104
Phase13	0	40	74

[1] As soldered on the controller board.
[2] The bandwidth is given for the -3dB frequency response when driving a resistive load. When driving a capacitive load, which is the case for a CCD pin, the effect of the RC network made with the serial resistance lowers the effective bandwidth. If a higher bandwidth is required, please contact Nüvü Camēras.
[3] The propagation delay of the PCB trace from the power amplifier to the edge-mount connector of the detector board

Table 2.4: Description of the asynchronous signals generated by the controller board

Signal name	Address	Description
Bias 0	0	14 bits bias typically selectable between 0 and 25 V
Bias 1	1	14 bits bias typically selectable between 0 and 25 V
Bias 2	2	14 bits bias typically selectable between 0 and 25 V
Bias 3	3	14 bits bias typically selectable between 0 and 25 V

Table 2.4 – Continued

Signal name	Address	Description
Bias 4	4	14 bits bias controlling the high level of the HV_CLK1 high voltage clock (typically between 25 and 50 V).
Bias 5	5	14 bits bias selectable between 0 and 25 V
Bias 6	6	14 bits bias selectable between 0 and 15 V
Bias 7	7	14 bits bias selectable between 0 and 15 V
Bias 8	8	14 bits bias selectable between −10 and 0 V
Bias 9	9	14 bits bias selectable between −10 and 0 V
Bias 10	10	14 bits bias controlling the low level of the HV_CLK0 high voltage clock[1].
Bias 11	11	14 bits bias controlling the high level of the HV_CLK0 high voltage clock[1].
Bias 12	12	14 bits bias controlling the low level of the HV_CLK1 high voltage clock (typically between 0 and 20 V)
Bias 13	13	14 bits bias selectable between 0 and 30 V.
Bias 14	14	14 bits bias selectable between 0 and 30 V.
Bias 15	15	14 bits bias setting the value of VSS. This bias is adjustable between 0 and 10 V. When the controller board is configured for real VSS operation, this bias controls the VSS output. When the controller board is configured for virtual VSS operation, this bias controls the offset on all clocks and biases.

[1] HV_CLK0 is not available by default, although it could be used as a second HV clock should the user wish it, please contact Nüvü Camēras should the user want a second HV clock.

2.1.6 Asynchronous items

The asynchronous items are those that can be changed at any time, regardless of the state of the read-out. They are the following:

- 5 biases (0–25 V). These biases are adjustable between 0 and 25 volts at a resolution of 14 bits. These biases are at addresses 0, 1, 2, 3 and 5.
- 2 biases (0–30 V). These biases are adjustable between 0 and 30 volts at a resolution of 14 bits. These biases are at addresses 13 and 14.
- 2 biases (0–15 V). These biases are adjustable between 0 and 15 volts at a resolution of 14 bits. These biases are at addresses 6 and 7.
- 2 biases (−10–0 V). These biases are adjustable between −10 and 0 volts at a resolution of 14 bits. These biases are at addresses 8 and 9
- 4 high voltage biases. These biases are reserved to control the high voltage clock used to clock the EM register of an EMCCD. Two biases control the HV_CLK0[1], low level at address 10 and high level at address 11. Two biases control the HV_CLK1, low level at address 12 and high level at address 4.

- 1 VSS bias. This bias can either be used as a real VSS or as a virtual VSS. This depends on the configuration of the controller board. In real VSS mode, this bias controls a high power amplifier to drive the substrate of the CCD. In virtual VSS mode, the substrate of the CCD must be tied to the ground, and the VSS bias offsets all the clocks and other biases. This bias is at address 15.
- ADC gain and channel.

[1] HV_CLK0 is not available by default, although it could be used as a second HV clock should the user wish it, please contact Nüvü Camēras should the user want a second HV clock.

The biases can be changed with the **sb** and **sbnl** serial commands (see section 2.2.4) or directly through *cccpComm*'s GUI, while the ADC gain and channel can be changed with, respectively, the **ads5263gain** and **ads5263channel** serial commands (see section 2.2.1).

2.2 Serial commands

This section lists all the serial commands the user may wish to use in order to have full control of the CCD's driving parameters and full control of the CCD's output acquisition parameters. The output information of each function lists the literal output that will be sent by the controller following the use of that function. Each command will send an `OK` response following it's successful execution or an `ERROR` if an error occurred during its use. Other commands exist but are not listed here, they should not be used by the user since they are made to be called directly by *cccpComm*.

The data digitization subsection lists all commands which are connected to the digitization process of the CCD's outputs.

The CDS commands subsection lists all commands used to make a useful averaging of the CCD pixels.

The sequencer commands subsection lists all commands which affect the driving of the CCD and the control of the external signals.

2.2.1 Data digitization commands

ads5263channel

Description:

- Selects one of the 4 ADS5263 output as the main channel for video data. Channel 2 and 4 are unused by default.

Parameters:

- *Channel*: Selects the channel on which to set the gain. When the controller is shipped, valid values are 1 for the CCD0 output, 3 for the CCD1 output. The channels 2 and 4 are unused. *This parameter is optional.*

Note: If no parameters are supplied, the function will return the currently active channel.

Output:

- `Channel: ch`
  Channel configuration the controller is in, where **ch** is the channel number.
- If an invalid channel is given an error will be reported.

ads5263clk

Description:

- For the FPGA to be synchronous with the ADC, the FPGA supplies it with a clock. The ADC then supplies the FPGA with a feedback clock that is used for the data acquisition. This function sets the parameters for the ADC clock generation and thus allows the user to generate different types of clocks for the EMCCD data acquisition.
- This is done through 4-bit serialization of data. The 4-bit serialization is done at each sequencer clock cycle, the sequencer is clocked at about 240 Mhz, which yields an effective time resolution of about 4.17 ns.
- The maximum sampling rate is 120 MSPS.
- **Note 1:** The sequencer frequency (about 240 MHz) should be divisible by the sampling rate when the user wishes to use the CDS averaging functionality since the sequencer generates signals for data acquisition and processing (DVAL, LVAL and FVAL).
- **Note 2:** The controller board is optimized to operate the data acquisition at its maximum sampling rate of 120 MSPS, the user should not attempt to use a lower sampling rate except in some very specific cases. The user should issue the command from the first example below to achieve a sampling rate of 120 MSPS.

Parameters:

- *Length*: Defines the length of the ADC clock generation pattern. Valid values are defined by these rules:
  - Pattern length must be lower or equal to 124.
  - Pattern length must be higher or equal to 4.
  - Pattern length needs to be divisible by four.
- *Frequency*: Number of periods in the pattern. Valid values are defined by these rules:
  - The frequency must be lower than the pattern length.
  - The frequency must be divisible by the pattern length.
- *Duty*: Defines the duty of a ADC clock period in the pattern, must be lower or equal to 124.
- *Delay*: Delay the start of the first active period in the pattern, must be lower or equal to 124.

Output:

- The output may change depending on the firmware version and should not be taken as a stable API.
- If an error is reported, invalid parameters may have been supplied or the FPGA failed to sync itself with the ADC.

Examples:

- `ads5263clk 8 1 4 4`: Would create the following 8-bits pattern 0000 1111. That pattern would be serialized by a 4-bit serializer, the resulting output would be a clock with half the frequency of the sequencer. Since this frequency is 240MHz, the result is data acquisition at 120 MSPS.

- `ads5263clk 12 2 3 3`: This invalid command would create the following 12-bits pattern 0001 1100 0111 and result in a sampling rate of 160 MSPS. Since it goes above the maximum sampling rate of 120 MSPS this command will return an error.

- `ads5263clk 12 1 6 2`: Would create the following 12-bits pattern 0011 1111 0000 and result in a sampling rate of 80 MSPS.

- `ads5263clk 20 2 5 0`: Would create the following 20-bits pattern 1111 1000 0011 1110 0000 and result in a sampling rate of 96 MSPS.

Note: Theoretically you could generate many types of ADC clocking configurations with this command. However the maximum supported sampling rate is 120 MSPS and not every configuration can work, for example a configuration resulting in a clock with a very low or very high duty cycle may not work. If the controller fails to sync itself with the ADC for a given configuration, the user may try the following to sync the controller with the ADC successfully:

- Try adjusting the input delay configuration with `ads5263iodelay`.

- Try adjusting the resulting clock's duty cycle by adjusting the `ads5263clk` command parameters.

ads5263gain

Description:

- Sets the gain of the 16 bits ADC. This allows to add a gain up to 12 dB right after the digitization of the EMCCD's outputs.

Parameters:

- *Channel*: Selects the channel on which to set the gain. When the controller is shipped, valid values are 1 for the CCD0 output, 3 is the CCD1 output. The channels 0 and 2 are unused. *This parameter is optional.*

- *Gain*: Sets the ADC gain in dB for that channel. Valid values are between 0 to 12 in dB, floating values are not accepted. *This parameter is optional.*

Output:

- `Gain ch:G`
  Gain configuration where **ch** is the active ADC channel and **G** is the gain in dB.

- If an error is reported, invalid parameters have been supplied or failed to set the gain.

ads5263iodelay

Description:

- Fine adjustment of the global input data delay from the ADS5263 in the FPGA by increments of 78 ps, a maximum of 31 delay increments can be added.
- Needs to be called if the user wishes to change the sampling rate and the controller fails to sync. When the controller is shipped the default value is of 17 for a sampling rate of 120 MSPS.
- **Warning:** When you call this function, the default value will be overwritten by the value you provide.

Parameters:

- *Global*: The number of increments from which the ADC feedback clock will be delayed.

Output:

- `Setting delay to` **d**`, pivot:` **p**
  `Written`
  Delay configuration where **d** is the global delay and **p** is a deprecated value. **Written** is to indicate the default value was overwritten.

ads5263rb

Description:

- Reads consecutive values outputted by the ADC and output them in the console. This command can be used in conjunction with **ads5263tp** to validate that the data acquisition from the ADC is done correctly.

Parameters:

- *n*: The number of consecutive values minus 1 to read from the ADC and output in the console.

Output:

- **n**`:` **data** `d l f`
  Where **n** is the data number and **data** is the data value.

ads5263sync

Description:

- Syncs the ADS5263 ADC with the FPGA data acquisition logic. This command can be used in conjunction with **ads5263tp** and **ads5263rb** to validate that the data acquisition from the ADC is done correctly.
- This command does not need to be executed after a **ads5263clk** command if it automatically did the re-sync.

Parameters:

- *None*

Output:

- The output may change depending on the firmware version and should not be taken as a stable API.

ads5263tp

Description:

- If you want the ADC to output a test pattern, this command will set what test pattern the ADC will output continuously. This command can be used in conjunction with **ads5263sync** and **ads5263rb** to validate that the data acquisition from the ADC is done correctly.
- Does not need to be called before using **ads5263clk** automatic re-sync.
- Make sure the data isn't processed by the CDS when using a test pattern, the user should put the CDS in passthrough mode with the **cdspt** command.

Parameters:

- *Enable*: 1 will enable the ADC to output a test pattern. 0 will disable the test pattern functionality.
- *Pattern type*: What type of pattern the ADC will output:
  - 0: ADC will output a ramp that increases by 1 each ADC clock period.
  - 1: ADC will output a constant value.
  - 2: ADC will alternate between 2 constant values each clock period.
- *Data 0*: The constant value the ADC will output if asked to output a constant value. *This parameter is optional.*
- *Data 1*: The second constant value the ADC will take alternatively each ADC clock cycle if asked to alternate between two constant values. *This parameter is optional.*

Output:

- `Setting test pattern to` **en**
  `OK`
  Where **en** is the enable parameter.

svideobw

Description:

- Selects between the 4 available video bandwidth, typically each video output has its own video bandwidth.
- When switching the video channel (**ads5263channel** command) for another video output, then you may need to also change the video bandwidth as this function changes the bandwidth for both outputs.
- This is useful to reduce image noise when using the controller to get a lower pixel rate.

Parameters:

- *bw*: EM output (ADC channel 1):
  0 - 40MHz
  1 - 20MHz
  2 - 10MHz
  3 - 6MHz
  Conv output (ADC channel 3):
  0 - 6MHz
  1 - 3MHz
  2 - 500KHz
  3 - 6MHz

Output:

- `Video BW:` **bw**
  Where **bw** is the selected video bandwidth number.

2.2.2 CDS commands

Commands in this section control the configuration for averaging a pixel on multiple samples using the CDS module. The methodology is explained in Section 2.1.3.

cdsdivisor

Description:

- Manual override of the divisor value of the CDS.

Parameters:

- *Divisor*: Integer divisor value of the CDS. Valid values are from 1 to 65535. If nothing is given, this function will just give the current divisor value. *This parameter is optional.*

Output:

- `:` **div**
  `OK`
  Where **div** is the active divisor value.

cdsgain

Description:

- Manual override of the analog gain emulation by the CDS module.

Parameters:

- *Gain*: Integer gain value of the CDS. Valid values are from 1 to 8. If nothing is given, this function will just give the current CDS gain value.*This parameter is optional.*

Output:

- `Gain 1:` **G**
  `OK`
  Where **G** is the CDS gain.

cdsoffset

Description:

- Sets the CDS offset value.

Parameters:

- *Offset*: Integer offset value of the CDS. Valid values can be from 0 to 65535. If nothing is given, this function will just give the current CDS offset value.*This parameter is optional.*

Output:

- `CDS offset: O`
  `OK`
  Where **O** is the active CDS offset.

cdspt

Description:

- Sets the CDS in pass-through mode, effectively disabling or enabling the CDS module. See section 2.1.3 (CDS).
- The sampling rate is usually higher than the CameraLink data rate, as such the user should be aware that some data will be lost when enabling the pass-through mode.

Parameters:

- *Enable*: If set to anything but 0, will activate the pass-through mode of the CDS, effectively disabling it.

Output:

- `Pass through: enabled` or `Pass through: disabled`

cdssamples

Description:

- Sets the number of samples the CDS will use for its pixel averaging window.

Parameters:

- *n*: The number of samples that will be used for the averaging window.

cdsskip

Description:

- Sets the number of samples the CDS will skip in the beginning of a line.

Parameters:

- *n*: The number of samples the CDS will skip in the beginning of a line.

cdsweights

Description:

- Sets the weights of each sample in the averaging window. This function can be called multiples times but the n counter will hold its value in between calls.
- **cdsweightsstart** needs to be called prior to calling this function the first time and call **cdsweightsstop** when done building the weights.
- The maximum number of weights cannot go above 2048.

Parameters:

- *Weigth 0*: The weight of sample 0.
- *Weigth 1*: The weight of sample 1. ...
- *Weigth n*: The weight of sample n.

Output:

- **n : w**
  Where **n** is the sample number and **w** is the weight of the sample.

cdsweightsbuild

Description:

- Builds the weight configuration for averaging a pixel on multiple samples using the CDS module, the methodology is explained in the Section 2.1.3.
- **cdsdivisor**, **cdsweightsstart** and **cdsweightsstop** do not need to be called when using this function to build the weights.

Parameters:

- *n_samples*: The total number of samples used for the averaging window.
- *skip_reference*: The number of samples the CDS will ignore in the averaging window before averaging the reference level. It will set $skip\_reference$ weights to zero.
- *n_reference*: The number of samples the CDS will use for averaging the reference part after the skipped samples.
- *skip_signal*: The number of samples the CDS will ignore in the averaging window after the reference samples before averaging the signal level. It will set $skip\_signal$ weights to zero.
- *n_signal*: The number of samples the CDS will use for averaging the signal part after the skipped samples.

Output:

- `CDS samples: n`
  `CDS Weights:skip_rx0, n_refxweight_r, skip_sx0, n_sigxweight_s`
  `CDS divisor: div`
  `OK`
  Where **n** is the number of samples for the averaging window. The rest of the output is the weights configuration in order, **skip_r** is the number of weights multiplied by 0, **n_ref** is the number of weights multiplied by **weight_r**, **skip_s** is the number of weights multiplied by 0, **n_sig** is the number of weights multiplied by **weight_s** and **div** divides the sum of all the samples multiplied by their own weight.

Examples:

- `cdsweightsbuild 10 2 3 3 2`: This command would configure the CDS to average a pixel on 10 samples, using 3 reference samples and 2 signal samples. The weights for the 10 samples would be : 0,0,2,2,2,0,0,0,-3,-3 and the divisor value would be 6, the smallest common multiplier between 2 and 3.
- `cdsweightsbuild 100 10 20 20 50`: This command would configure the CDS to average a pixel on 100 samples, using 20 reference samples and 50 signal samples. The weights for the 100 samples would be 0 x 10, 5 x 20, 0 x 20, -2 x 50 and the divisor value would be 100, the least common multiplier between 20 and 50.
- `cdsweightsbuild 12 0 6 2 3`: This command would configure the CDS to average a pixel on 12 samples, using 6 reference samples and 3 signal samples. The last 3 sample weights are not defined and will be set to 0,

cdsweightsstart

Description:

- Starts the CDS weights configuration. Call this function before building the weights using **cdsweights**.

Parameters:

- *none*

cdsweightsstop

Description:

- Stops the CDS weights configuration. Call this function after building the weights using **cdsweights**.

Parameters:

- *none*

2.2.3 Sequencer commands

abort

Description:

- Aborts the read-out sequence, if running, and sets blanking state. The controller will go in blanking state immediately if it was in the expose or pre-expose state. If it was executing a ROS, it will wait for the ROS to finish its execution before going to the blank state.

Parameters:

- none

Output:

- If an error is reported, the controller is not working properly or the configuration data is invalid.

ds

Description:

- Dumps the readout sequence information.

Parameters:

- None

Output:

- The output may change depending on the firmware version and should not be taken as a stable API.

dsf

Description:

- Dumps all the information from the readout sequences stored into the flash memory.

Parameters:

- None

Output:

- The output may change from firmware version to firmware version and should not be taken as a stable API.

dsv

Description:

- Dumps all the values of the sequence variables. This is equivalent to calling **rsv** for all sequence variables.

Parameters:

- None

Output:

- For every sequence variable (0 to 31), the output of **rsv** is sent.

erasesequence

Description:

- Erases all sequences from the FPGA memory.

Parameters:

- *None*

ld

Description:

- Loads a readout sequence from flash memory to the FPGA memory.

Parameters:

- *n*: Number of the read-out sequence to load.

Output:

- The read-out sequence can output some text through its `msg` commands while loading.
- If an error was reported it may be because the specified ROS isn't present in the memory.

ls

Description:

- Lists the readout sequences stored in the flash memory.

Parameters:

- None

Output:

- **n: name length**
  For each read-out sequence stored into the flash memory, this line is output. **n** is the number of the read-out sequence. This number must be passed to **ld** in order to load the corresponding read-out sequence. **name** is the name given to the read-out sequence (its name in *cccpComm*). Spaces in the name are converted to underscores. **length** has a meaning that depends upon the firmware version.

rsrt

Description:

- Reads the active ROS running time and dumps it in in the console.

Parameters:

- None

Output:

- **t**
  Where **t** is the running time in ms.

rss

Description:

- Reads the sequencer state.

Parameters:

- None

Output:

- **state** (n)
  Where **state** is either `stopped`, `exposing`, `waiting` or `reading`. **n** is the amount of exposures left (-1 is returned for an unlimited amount of exposures).

rsv

Description:

- Reads a sequence variable.

Parameters:

- *n*: Index position of the sequence variable to read. Valid values are between 0 and 31.

Output:

- **n: value**
  `OK`
  Where **n** is the index position of the sequence variable and **value** is its value.

re

Description:

- Requests exposures. This will cause the read-out sequence to be executed a given number of times.

Parameters:

- *n*: The number of times to run the read-out sequence. A value of -1 makes the sequence run infinitely.

Output:

- If an error is reported, it may be because no ROS is loaded.

sap

Description:

- Changes the polarity of the arm external signal.

Parameters:

- *polarity*: 0 means the arm signal is active high, 1 means the arm signal is active low.

se

Description:

- Sets the exposing time of the read-out sequence.

Parameters:

- *t*: Time in ms to expose before running the read-out sequence. Valid values of *t* are between 0 and 1174852063.628728 ms. Floating values are accepted and the exposing time resolution is 4.17 ns. If this parameter is omitted, the exposing time will not be updated, but the effective exposing time will still be returned.

Output:

- **e**
  Where **e** is the effective exposing time set. It can vary slightly from the value requested. **e** is rounded to the nearest possible value of *t*.

sesm

Description:

- Sets the mode for the external shutter signal, when there's an external shutter present (see **sesp** command).

Parameters:

- *m*: External shutter mode, which may take one of the following values :
  - −2 : the external Shutter signal will be kept in the closed state (see **ssp** command).
  - 0 : automatic mode, such that the external Shutter signal is in the open state during exposing time and in the closed state the rest of the time.
  - 2 : the external Shutter signal will be kept in the open state (see **ssp** command).

Output:

- `OK`

sesmpw

Description:

- Sets the minimum pulse width for the fire, arm and shutter external signals. The minimum is 40 ns.

Parameters:

- *pw*: The minimum pulse width in millisecond.

Output:

- **pw**
  Where **pw** is the configured minimum pulse width.

sesp

Description:

- Sets the presence of an external shutter. Must be set to 1 when there is one. When an external shutter is present the sequencer will compensate for the opening and closing delays of the shutter when exposing, see Section 3 for more details. The external shutter delay must be adjusted to your external shutter with the **ssd** command.

Parameters:

- *en*: Must be set to one if there is an external shutter present.

Output:

- **en**
  Where **en** is the set value.

sfp

Description:

- Sets Fire output polarity. The fire output is active when the device is integrating (exposing).

Parameters:

- *p*: Polarity mode, which may take one of the following values :

  ≥ 0 : positive polarity; the Fire output will be high when exposing and low the rest of the time.
  > 0 : negative polarity; the Fire output will be low when exposing and high the rest of the time..

shvbw

Description:

- Selects between the two available high voltage clock bandwidths.

Parameters:

- *p*: 1 - For 10 MHz high voltage clock or lower.
  0 - For higher frequency high voltage clocks.

ssd

Description:

- Sets the shutter delay. The shutter delay will allow the shutter to open prior to starting the exposure and will allow the shutter to close prior to starting the read-out. The shutter delay will only be effective when the shutter is in automatic mode.

Parameters:

- *t*: Shutter delay in ms. Valid values of *t* are between 0 and 1174852063.628728 ms. Floating values are accepted. Delay resolution is 4.17 ns.

Output:

- **d**
  Where **d** is the effective shutter delay time set. It can vary slightly from the requested value. **d** is rounded to the nearest possible value of *t*.

ssm

Description:

- Sets the internal shutter mode.

Parameters:

- *m*: Shutter mode, which may take one of the following values :

  ≤ -2 : Internal shutter will always be closed.

  $-1, 0, 1$: Internal shutter is in automatic mode and will be opened during exposing time.

  ≥ 2 : Internal shutter will always be open.

ssp

Description:

- Changes the external Shutter signal polarity.

Parameters:

- *p*: Polarity mode, which may take one of the following values :

  ≥ 0 : positive polarity; the Shutter output will be high when the shutter is open and low the rest of the time.

  > 0 : negative polarity; the Shutter output will be low when the shutter is open and high the rest of the time.

ssv

Description:

- Sets the value of a sequence variable. The new value will be used on the next iteration of the read-out sequence.

Parameters:

- *n*: Index position of the sequence variable to set.
- *value*: Value of the sequence variable. Valid values are between 0 and 65535.

ssva

Description:

- Automatically sets multiple sequence variables at once. The variables will all be updated at once on the next iteration of the read-out sequence. If the sequence is running while the sequence variables are being updated, their new value will be effective only once they have all been updated.

Parameters:

- *n*: Amount of sequence variables to set. The update will start at the sequence variable number 0 up to sequence variable $n-1$.
- `\0`: The ASCII code 0 must be sent after *n*
- `binary data`: The binary data of the sequence variables values. The sequence variables are unsigned 16-bit values (between 0 and 65535) and they are sent LSB first.

stm

Description:

- Sets the trigger mode.

Parameters:

- *t*: Trigger mode, which may take one of the following values :

  0 : Internal trigger, exposure time controlled by the sequencer (**se** command).

  1 : External trigger, exposure time controlled by the sequencer (**se** command), triggers from low to high transition.

  −1 : External trigger, exposure time controlled by the sequencer (**se** command), triggers from high to low transition.

  2 : External trigger, exposure time is controlled by the trigger signal, triggers from low to high transition.

  −2 : External trigger, exposure time is controlled by the trigger signal, triggers from high to low transition.

- *count*: The number of exposures to execute for each external trigger signal with trigger mode 1 or −1; set this to −1 for an unlimited number of exposures, using **abort** to end acquisition.

Output:

- **t count**
  Where **t** is the set trigger mode and **count** is the set number of exposures.

sw

Description:

- Sets the blanking (waiting) time during which the sensor is flushed continuously before exposure begins, for the active ROS.

Parameters:

- *t*: Time in ms to wait before the exposing time of the read-out sequence. Valid values of *t* are between 0 and 2147483646 ms. Floating values are accepted. The blanking time resolution is 0.1 ms. If this parameter is omitted, the blanking time will not be updated, but the current blanking time will be returned. *This parameter is optional.*

Output:

- **wt**
  Where **wt** is the effective waiting time set. It can vary slightly from the requested value. **wt** is rounded to the nearest possible value of *t*.

2.2.4 Miscellaneous commands

cc

Description:

- Checksum check, this function will compare the checksum received in parameters with the one calculated by the firmware. Use this in conjunction with **cs** for proper checksum check with your user software.

Parameters:

- *checksum*: The checksum value that was calculated by the user software.

Output:

- On a mismatch between the two checksums, the firmware will return `ERROR`.

cldelay

Description:

- Allows to manually adjust the timing delay of each of the 5 individual CameraLink serialized video data lines with increments of about 78 ps, a maximum of 31 delay increments can be added.
- Default values are of 0 for the 5 delay parameters and are not overwritten when using this function. **Note:** CameraLink video data delays should be adjusted only in special cases. Normally the user should only use the default values.

Parameters:

- *Clock delay*: Integer value for the number of delay increments to be added on the CameraLink clock output. Valid values are from 0 to 31.
- *Data 0 delay*: Integer value for the number of delay increments to be added on the CameraLink data 0 output. Valid values are from 0 to 31.
- *Data 1 delay*: Integer value for the number of delay increments to be added on the CameraLink data 1 output. Valid values are from 0 to 31.
- *Data 2 delay*: Integer value for the number of delay increments to be added on the CameraLink data 2 output. Valid values are from 0 to 31.
- *Data 3 delay*: Integer value for the number of delay increments to be added on the CameraLink data 3 output. Valid values are from 0 to 31.

cs

Description:

- Checksum start, this function will reset the checksum value in the firmware. As such, every byte sent after this command will increase the checksum. Use this in conjunction with **cc** for proper checksum check with your user software.

Parameters:

- *none*

Output:

- This function will not return a `OK` when called.

d

Description:

- Dumps the status of the controller.

Parameters:

- none

Output:

- The output may change depending on firmware version and should not be taken as a stable API.

dbconfig

Description:

- Dumps the bias configuration. The bias configuration is stored in EEPROM in the controller and is updated when the **Update bias config in DSP** button is clicked in *cccpComm* (see Section 4.2.3).
- This is the same as if one **rbconfig** command would be issued for every bias.

Parameters:

- None

Output:

- For every bias (0 to 15), the output of **rbconfig** is sent.

echo

Description:

- Turns the serial echoing on or off.

Parameters:

- *val*: 0: turns serial echo off. 1: turns serial echo on.

Output:

- `Serial echo is now` **val**
  Where **val** is either `on` or `off`.

getconfig

Description:

- Dumps the controller configuration from the EEPROM.

Parameters:

- *None*

Output:

- The output may change depending on firmware version and should not be taken as a stable API.

gpsinfo

Description:

- Dumps information from the GPS module, if it is present. Please contact Nüvü Camērasshould the user want to add a gps module to the controller.

Parameters:

- *None*

Output:

- The gps data will be outputted in the following way:

  `Lat:` latitude
  `Lon:` longitude
  `Alt:` altitude
  `Speed:` speed
  `Track:` track
  `Quality:` quality
  `Sats:` lat
  `Hdop:` hdop

help

Description:

- This function will list all available commands to the user. The listed commands may include commands not listed in this document, these commands are used directly by *cccpComm* and should not be called by the user under normal circumstances.

Parameters:

- *None*

Output:

- The function will output a list of all the available commands

msg

Description:

- Sends a message through the serial port. This command is most useful when used in an action of a read-out sequence stored into the flash memory. The message will be sent over the serial port upon loading a sequence (with the `ld` command). This can be used to send information specific to a read-out sequence to the host computer. See Section 4.4.3 for more information about actions.

Parameters:

- *message*: The message to output to the serial port. The message can be up to 255 characters in length. Longer messages will be truncated.

Output:

- **message**
  Where **message** is identical to *message* passed as a parameter.

rb

Description:

- Read the bias at address *address*.

Parameters:

- *address*: Address of the bias to read. Valid values are between 0 and 15.

Output:

- **address: valueA valueV**
  The controller will respond with a string indicating the bias **address** that was read, its value (**ValueA**) in ADU, and its corresponding voltage value (**ValueV**) (assuming that the calibration data is valid. See Section 4.2.3 for the calibration data).

rbconfig

Description:

- Reads the configuration data of a bias. The bias configuration is stored in EEPROM in the controller and it is updated when the **Update bias config in DSP** button is clicked in *cccpComm* (see Section 4.2.3).

Parameters:

- *address*: Address of the bias in which the configuration data is to be read. Valid values are between 0 and 15.

Output:

- `address:` **default minimum maximum offset gain name**
  Where **address** is the address of the bias specified, **default** is its default value upon powering-on the controller, **minimum** is the minimum value that can be set, **maximum** is the maximum value that can be set, **offset** is the voltage value for a setting of 0, **gain** is the increment in volt for an increment of the value of 1 of the bias, and **name** is the name of the bias.

rpid

Description:

- Reads the status of the PID temperature control loops.

Parameters:

- None

Output:

- **n: err int out stp temp**
  Where **n** is the number of the PID, **err** is the last error calculated, **int** is the integral term accumulated, **out** is the last output command (from 0 to 65535), **stp** is the temperature set point (in ˚C) and **temp** is the last temperature read (in ˚C). The PID number 0 is the CCCP temperature control loop.

rrtd

Description:

- Reads the temperature value of one of the 4 RTDs.
- RTDs must be calibrated using **wrtdcal**.

Parameters:

- *i*: The number of the RTD from which to read the temperature. Valid values are from 0 to 3.

Output:

- `Temperature: T C`
  Where **T** is the temperature in Celsius read from the temperature detector.

rt

Description:

- Reads the temperature of multiple components.

Parameters:

- *i*: Temperature selection:
  - `0`: Controller temperature.
  - `1`: CCD temperature.

Output:

- **i: temp**
  Where **i** is the selected temperature and **temp** is its value in Celsius.

rtsp

Description:

- Reads the temperature set point of the camera components.

Parameters:

- *i*: Item selection:
  - `0`: Controller temperature set point.
  - `1`: CCD temperature set point.

Output:

- **i: temp**
  Where **i** is the selected item and **temp** is the set point.

rvfpga

Description:

- Reads different voltages on the controller.

Parameters:

- *i*: Item selection:
  - `1`: 1.8V Alim.
  - `2`: 2.8V Alim (for SRAM).
  - `3`: 3.3V Alim.
  - `4`: 5.0V Alim.
  - `5`: 15V Alim.
  - `6`: 30V Alim.

Output:

- **i: V**
  Where **i** is the selected item and **V** is the measured voltage.

sb

Description:

- Sets the value of a bias. The bias will be updated immediately. All pending bias updates (**sbnl**) are loaded synchronously.

Parameters:

- *address*: Address of the bias to set. Valid values are between 0 and 15.
- *value*: Value of the bias. Valid values of *value* are between 0 and 16383.

Output:

- **address: valueA (valueV `V`)**
  The controller will respond with a string indicating the bias **address** that was set, its value (bf valueA) in ADU, and its corresponding voltage value (bf valueV) (assuming that the calibration data is valid. See Section 4.2.3 for the calibration data).

sbnl

Description:

- Sets the value of a bias. The bias will not be updated immediately. It will be updated once a **sb** command is issued. Multiple **sbnl** commands can be issued before a **sb** command is issued.

Parameters:

- *address*: Address of the bias to set. Valid values are between 0 and 15.
- *value*: Value of the bias. Valid values of *value* are between 0 and 16383.

Output:

- **address: valueA (valueV V)**
  The controller will respond with a string indicating the bias **address** that was set, its value (bf valueA) in ADU, and its corresponding voltage value (bf valueV) (assuming that the calibration data is valid. See Section 4.2.3 for the calibration data).

settime

Description:

- Sets the internal camera time, images can be timestamped by the current time by using the **timestamp** command.

Parameters:

- *Year*: Year.
- *Month*: Month from 1 to 12.
- *Day*: Day from 1 to 31.
- *Hour*: Hour from 0 to 23.
- *Minute*: Minute from 0 to 59.
- *Second*: Second from 0 to 59.
- *uS*: uS from 0 to 999 999.

sysstat

Description:

- Dumps temperature information and fps. Useful to compile different statistics.

Parameters:

- *verbose*: If set to anything other than 0 will dump more information.

Output:

- The output will change depending on the controller type

timestamp

Description:

- Sets the timestamp parameters for the frames. The timestamping procedure essentially adds a line to the frame which contains the time information and gps information for the frame.

Parameters:

- *en*: Enables the timestamping if set to anything other than 0. If disabled no extra line will be added.
- *gps*: Enables the use of the time from the GPS if present.
- *pixel skip*: Sets the number of pixels that will be skipped in the extra line before the timestamp and gps data.

Output:

- **en gps skip valid present pps_ext_length**
  Where **en** is 1 if the timestamp was enabled, **gps** is 1 if the gps was enabled, **skip** is the set number of pixels to skip on the extra line, **valid** is 1 if the entered parameters are valid, **present** is 1 if there is a gps module present and **pps_ext_length** is the length of a PPS in data acquisition steps (data acquisition is usually done at 120 MSPS).

tsp

Description:

- Sets the temperature set point of a PID.

Parameters:

- *n*: The PID number to act upon.
- *d*: The new temperature to set, in $^{\circ}\mathrm{C}$. Floating point values are accepted.

version

Description:

- Dumps the serial number, the firmware version, the hardware version and the FPGA version of the controller, and the power supply version.

Parameters:

- None

Output:

- `Serial number:` **SN**
  `Firmware version:` **fw**
  `Hardware version:` **hw**
  `FPGA version:` **fpga**
  `Power supply version:` **ps**
  Where **SN** is the camera serial number, **fw** is the firmware version, **hw** is the hardware version, **fpga** is the FPGA version and **ps** is the power supply version.

wrtdcal

Description:

- Calibrates the resistance temperature detectors. Up to four temperature detector can be used. To have the correct temperature slope, the user needs to use a platinum RTD element with a temperature coefficient (TCR) of 0.00385 Ω/Ω/˚C and a base resistance of 100 Ω at 0 ˚C.

Parameters:

- *n*: The temperature detector number to act upon. Valid values range from 0 to 3.
- *gain*: The temperature detector gain, leave to 1 unless the user wishes to change the temperature slope.
- *offset*: The temperature detector offset in degrees. Adjust this value for each new RTD element.

Output:

- **n**`: gain:` **Gain**`: offset:` **Offset**
  Where **n** is the RTD number, **gain** is the temperature slope gain and **offset** is the temperature offset.

2.3 Power supply

The power supply of CCCP provides all the voltages for proper operation of the controller board. It generates +1.8 V, +2.5 V, +3.3 V, +5 V, −5 V, +15 V, −15 V, and +30 V. The external signals circuitry, the fan control circuitry and the temperature reading circuitry all reside on the power supply board. The power supply can operate either asynchronously or synchronously with the read-out of the CCD, although the user should operate it synchronously to reduce read-out noise. In order to synchronize the power supply with the read-out, the PS_RESET should be asserted at the very beginning of the read-out sequence. Once this has been done, the power supply will be synchronized with the beginning of the read-out sequence. Every time the read-out sequence is executed (every image), the power supply switching clock phase will be synchronized this way to reduce the noise in the resulting image.

Chapter 3

Architecture overview

CCCP uses a different architecture to that which a user of an SDSU (Leach) controller is accustomed to. The controller generates *readout sequences* that may be separated by a *blanking time* and/or an *exposing time* (or none). During the exposing and blanking times, the blanking and exposing patterns will run in loops (read below to learn more about patterns). If necessary, both blanking and exposing times can be set to 0 and/or controlled by the readout sequence itself. If the blanking and exposing times are set, they will be spent *before* the read-out sequence is executed. The blanking time will be spent first, then the exposing time and finally the readout sequence will be executed. If no blanking time is specified, the controller assumes that the time spent while executing the readout sequence is also an exposing time. That is, if the exposing time is 50 ms and the readout sequence has an execution time of 30 ms, on the second iteration of the readout sequence execution (for example, when multiple images are requested with the `re` command), an exposing time of 20 ms will be spent. When the controller is idle (the readout sequence is not running), the controller runs the blanking pattern (see Section 3.1). The blanking pattern must be manipulated with care. Please read Section 7.3 before running a blanking pattern to flush the CCD.

When the controller switches from the blanking pattern to the exposing pattern and from the exposing pattern to the read-out sequence, it does not ensure that the respective pattern is fully spent before going to the next state. If a blanking pattern has a 10-steps length and the exposing command is received when only 3 steps are done, the controller will not wait until the 10^{th} step of the blanking pattern is reached before switching to the exposing pattern. There is no idle state when switching from pattern to pattern. The exposing pattern will start immediately once the blanking time is finished.

A special pattern might be running for a pre-determined number of times between the blanking pattern and the exposing pattern. This allows, for example, to have a blanking sequence that clocks the vertical clocks at 500 kHz (to avoid damaging the controller when the blanking sequence is running for an extended period of time) and flush the CCD with a 2 MHz vertical clock just before the exposing pattern. This allows the CIC generated by the low frequency clock to be flushed before exposing the CCD. This special pattern, the *pre-expose pattern*, is defined in the read-out sequence tab of *cccpComm* (see Chapter 4). The first exposing time spent after a blanking period will be lenghtened by the run time of this pattern (rounded to the nearest 0.1 ms).

When the shutter is set in automatic mode (with **ssm**), the blanking time will be at least the shutter delay time (**ssd**). Thus, the shutter will start to open before the end of the blanking time. The shutter delay for

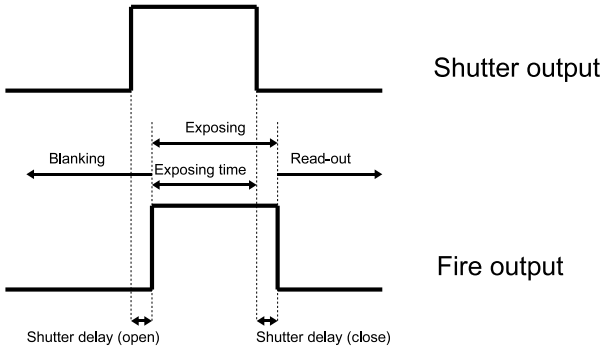

Figure 3.1: Shutter and Fire output in internal trigger mode.

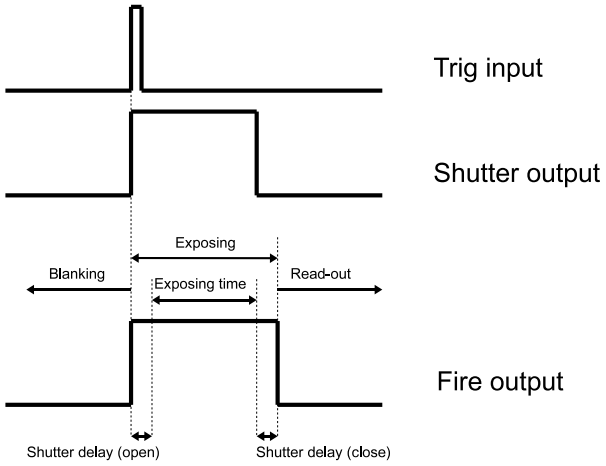

Figure 3.2: Shutter and Fire output in external trigger mode.

the opening will end at the end of the blanking time. The exposing time will be lengthened by the shutter delay time. The shutter will start to close at the end of the requested exposing time and the read-out will start once the shutter delay has expired. This behaviour is summarized in Figure 3.1.

When the trigger is an external source (as set with **stm**), the blanking time will be ignored. The controller will be in blanking mode until a trigger event is received. Once the trigger is received, the controller will switch immediately to exposing mode for the requested exposing time. If the shutter is in automatic mode, since it can not open before receiving the trigger event, the shutter will start opening immediately when the trigger event is received. The total exposing time will be lengthened by twice the shutter's delay to allow its opening and closing. This behaviour is summarized in Figure 3.2.

3.1 Patterns

Patterns are made of the data that are actually stored in the FPGA memories, a total of 20480 steps can hold in those memories. For each pattern, the waveforms of all phases will be defined using *cccpComm*. A pattern defines the value that the phases will have at any given time. The time step of the phases is the time steps of the sequencer at 240MHz. Patterns can be as small as 25 ns (6 steps) and as large as the memory can hold, except for the blank, pre-expose and expose patterns, which can have as little as 2 steps. However, all the patterns must fit into the FPGA memory dedicated to the patterns. Actually, this means that the sum of all patterns must fit into 20480 steps than can be held in the analog signals memories and the digital signals memory. Patterns define the output phases of the controller, but they also define some digital control signals of the controller. See Table 2.2 for information on the output phases and the digital signals of the controller.

3.2 Sequences

Sequences are built using repetitions of one or several patterns. For example, to readout a line of a 512x512 CCD, one vertical clock scheme and 512 horizontal clock schemes must be applied to the CCD, though only patterns for one vertical and one horizontal clock are needed. The sequence to read out one line of the CCD will repeat one time the vertical clock and 512 times the horizontal clock. The sequences are executed from the sequencer module of the FPGA.

3.3 Readout sequences

Readout sequences are built using repetitions of one or several sequences. In addition, a readout sequence defines three patterns: a blanking pattern, an exposing pattern and a pre-expose pattern.

The blanking and exposing patterns defined by the readout sequence are the patterns that will be executed in loop during the blanking and exposing times, respectively. The pre-expose pattern, if defined, will be run between a blanking period and an exposing period. If the controller is taking images continuously and no waiting time is defined, the pre-expose pattern will be run only once before the exposing time of the first image. The order in which everything is executed is schematized in Figure 2.1 in Section 2.1.2.

For example, to readout a CCD, a frame transfer of 512 vertical clock schemes must be executed, the sequence to readout a line must be executed 512 times as well. The frame transfer can be defined in a sequence and repeated once at the beginning of the readout sequence. Then, the line read sequence can be repeated 512 times.

3.4 Sequence variables

It is possible to modify the behaviour of a readout sequence without having to load a new one. The sequence variables allow the number of repetitions of the sequences and the patterns to be changed from one image to the next. There are 32 sequence variables that can be assigned to any repetition value of sequences and patterns. A same sequence variable can be assigned to more than one repetition value inside a readout sequence. The sequence variables are defined when building the sequences and readout sequences in *cccpComm* (see Chapter 4).

The sequence variables can be given names to disambiguate them. When names are given to the sequence variables, a **msg** command will be issued upon loading a readout sequence that will output the number of the sequence variable and its name. This feature can be used to set-up an acquisition computer so it can know which sequence variable controls a given parameter.

There is an interface in *cccpComm* to set the default values of the sequence variables but they may also be adjusted without reprogramming or updating patterns by means of the **ssv** and **ssva** commands.

3.5 Digitalization path

The EMCCD output data digitizing is done through a high performance ADC on the controller. Prior to the digitalization by the ADC, all analog video lines go through their own analog pre-amplification stage. This is done to cover as much of the dynamic range of the ADC as possible (see reference to ads5263.pdf). More details about the analog front-end is available at Section 7.1.

Chapter 4

cccpComm: The configuration application

An application is needed to generate the waveforms that will be stored in the FPGA's BRAMs.

The *cccpComm* application stores its general configuration (serial port, last opened file name, etc.) in a file `.cccpComm` stored in the user's home directory. In case something would go wrong and *cccpComm* would crash upon startup, delete this file before starting it up again.

The *cccpComm* application can store all of its data (pattern waveforms, calibration data, bias default values, etc.) in a `.cccp` file. CCCP is shipped with a default `.cccp` file that contains all the controller's configurations (phases and biases addresses) and calibration. The user should always start from this file to build patterns, sequences, and read-out sequences.

4.1 Setup

The *cccpComm* application communicates with CCCP through *cccpServer* (see Chapter 5) using the TCP/IP protocol. The *cccpServer*, in turn, communicates with CCCP through a serial link embedded in the CameraLink or GigE interface. In order to connect *cccpComm* to CCCP, CCCP must be connected, with either a CameraLink cable or a GigE compatible Ethernet cable, to a host computer that is running *cccpServer*. Next, *cccpComm* must connect to the host computer using the host computer's network address and *cccpServer*'s TCP port. Finally, *cccpComm* must indicate to *cccpServer* which frame grabber to use to communicate with CCCP. The next section explains the parameters to control this process.

4.2 Config tab

The **Config** tab allows the user to set-up their controller.

4.2.1 Misc tab

The **Misc** tab of the **Config** tab (Figure 4.1) allows one to set the parameters needed to communicate with CCCP. The **Host** parameter defines the network address of the host computer running *cccpServer* and the **TCP port** parameter defines the TCP port that *cccpServer* is listening to on that host computer. In the event

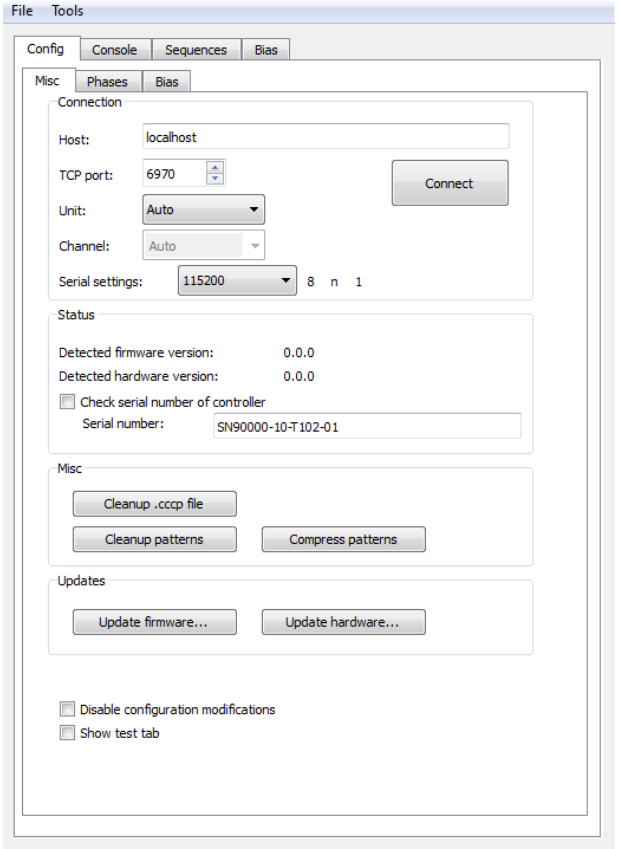

Figure 4.1: Screenshot of the Misc tab of the Config tab.

that *cccpComm* and *cccpServer* are running on the same computer, the address `localhost` can be used.

The **Unit** and **Channel** parameters define which device *cccpServer* should attempt to use. The Unit parameter defines which frame grabber or GigE port the desired camera is connected to. In the event that the selected frame grabber has more than one CameraLink interface, the **Channel** parameter can be used to specify which interface to use. Both of these parameters can be individually set to **Auto** to signal *cccpServer* to scan for an available CCCP.

Some of the serial port settings (parity, start, and stop bits) can only be changed by editing the `.cccpComm` file. However, these parameters should not be changed for a normal use of the controller. The baud rate can be changed with the **Serial** drop-down list. Changing the baud rate in *cccpComm* does not change the baud rate of the controller. The baud rate of the controller must first be changed with the **baudrate** serial command and then the baud rate of *cccpComm* should be set to the same value. Baud rates of 230400 and 460800 are available only under MacOS X and Linux and with some specific communication interfaces only.

The button **Cleanup config file** erases the content of the config file and the opened `.cccp` file and rewrites only the useful parameters.

The button **Cleanup patterns** looks for patterns that are not used by any readout sequence. For each unused pattern found, it asks the user if they want to delete it.

The button **Compress patterns** will simply put all the patterns contiguously into the memory (BRAMs located on the detector board) by changing their **Base address** parameters (see Section 4.4.1).

The button **Update firmware** allows an encrypted `.aes` firmware file to be opened and uploaded to the controller. When updating the firmware, follow the on-screen instructions. **Warning: Special care should be taken when updating the firmware, do not attempt to update the firmware without first contacting Nüvü Camēras.**

The check box **Disable configuration modifications** allows *cccpComm* to lock the phases and biases configuration of a file. The locked parameters for the phases are the type, address and bit offset. The locked parameter for the biases is the address. When this check box is checked for a given file, it is not possible to uncheck it.

The check box **Check serial number of controller** allows the user to lock a .cccp file to a particular controller using its serial number. It can be unchecked to disable the lock. This ensures the proper .cccp file is used with the proper controller.

4.2.2 Phases tab

The **Phases** tab of the **Config** tab (Figure 4.2) allows one to define which phases will be generated by the controller. It defines the address and type of all phases. Usually, the configuration of the phases is generated by Nüvü Camēras, and is provided in a `.cccp` file with the controller. When using this file, it is not possible to change the following parameters:

- Phase type;
- Phase address;
- Phase bit offset.

It is not recommended that the user build its own phase configuration. An inadequate phase configuration could lead to extreme DAC values being sent to the power amplifiers at a high frequency that would make them overheat and break (see Section 7.3). When using the default configuration supplied with the controller, it is not possible to add or remove phases. Whether or not all phases are needed to clock a CCD, all of them should be defined to make sure they are initialized to a default value.

The **Name** field is a name the user picks to identify this phase.

The **Type** field can be one of the following:

- Analogic 14 bits
- Digital low res (1 bit)
- Digital high res (4 bits)

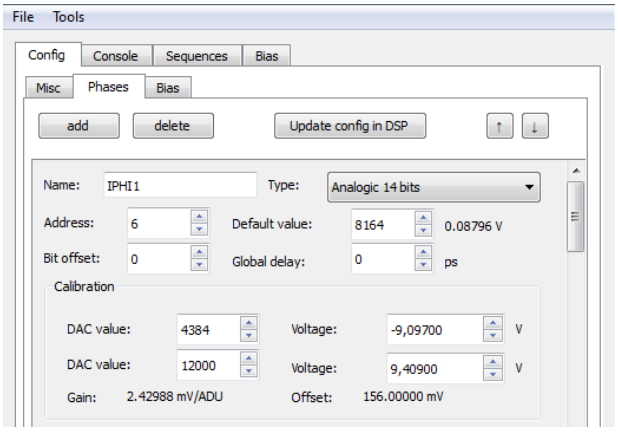

Figure 4.2: Screenshot of the Phases tab of the Config tab.

All phase types have about a 4.17 ns resolution, except the Digital high res that has about a 1.04 ns resolution.

The **Address** field specifies the address of the BRAM that is attached to this phase.

The **Bit offset** field specifies the least significant bit number of this phase. Both addresses and offsets are defined in Table 2.2.

The **Default value** is the value that will be applied to the phase's data when creating a new pattern. This is also the value the controller will write in the BRAM for this phase upon powering-up (once the **Update phase config in DSP** button is clicked).

The **Global delay** can be used to adjust the output delay of the phases by increments of 78 picoseconds up to a maximum of 2422 picoseconds. This value cannot be adjusted for digital low res signals as they are used internally.

The calibration data allows the user to calibrate an analog phase. This calibration data will be used to display accurate voltages when building patterns.

The order in which the phases appear can be changed by clicking on a phase and then clicking on the up and down arrow buttons to move it.

The button **Update config in DSP** allows the phase's configuration to be stored in the memory of the controller. This writes the default phase values in the controller. The controller will fill the BRAMs with these default values at power-on.

4.2.3 Bias tab

The **Bias** tab of the **Config** tab (Figure 4.3) allows one to define which biases will be generated by the controller. It defines the address of all biases. Usually, the configuration of the biases is generated by Nüvü

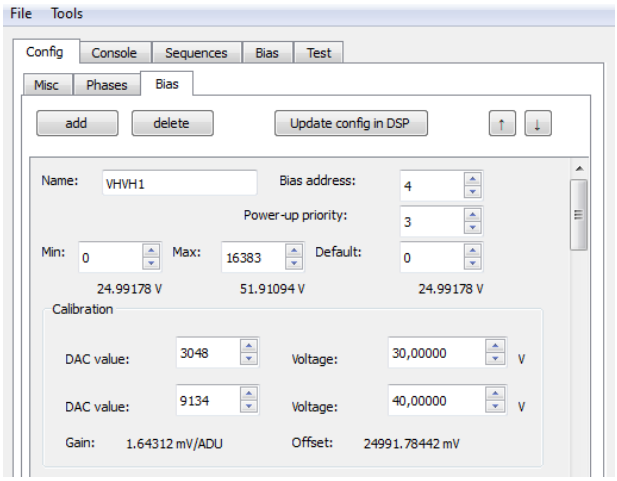

Figure 4.3: Screenshot of the Bias tab of the Config tab.

Camēras, and is provided in a `.cccp` file with the controller. When using this file, it is not possible to change the address parameter of the biases.

It is not recommended that the user build its own bias configuration. By using the configuration that comes with the controller, the user insures that every bias is given a safe default value.

The **Name** field is a name the user picks to identify this bias.

The **Bias address** field is the address of the DAC that will control this bias (see Table 2.4 for the addresses).

The **Min**, **Max**, and **Default** values are the minimum, maximum and default values that will be applied to this bias. Once the **Update bias config in DSP** is clicked, these values will be stored into the controller and be enforced. The **Min** value will define the absolute minimum value that can be applied to the bias. Requesting a bias update with a value lower than this will cause the **Min** value to be sent to the bias. The **Max** value will define the absolute maximum value that can be applied to the bias. Requesting a bias update with a value higher than this will cause the **Max** value to be sent to the bias. The **Default** value will be loaded at the controller start-up and when a new readout sequence is loaded (except for the readout sequences stored in flash memory, where it is the **Default** value that was set when storing the read-out sequence into the flash memory that will prevail). In order to update the default value of a DAC so that it is loaded at start-up, the **Update bias config in DSP** button must be clicked.

The calibration data allows the user to calibrate the biases. This calibration data will be used to display accurate voltages when changing a bias value (see Section 4.5). The calibration data will also be stored into the controller once the **Update bias config in DSP** is clicked. When changing a bias value with the **sb** or **sbnl** command, the voltage returned by the controller will be computed using the calibration data stored.

The order in which the biases appear can be changed by clicking on a bias to select it and then clicking on the up and down arrow buttons.

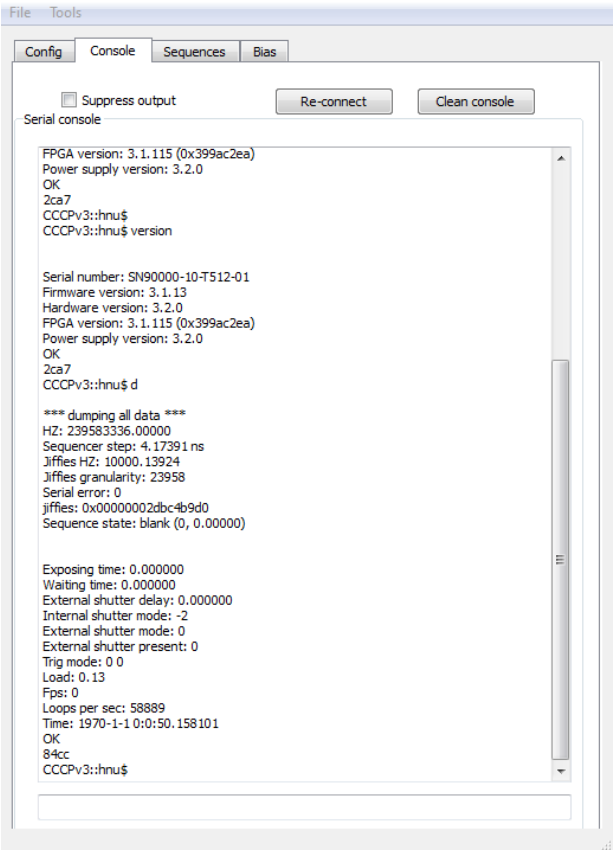

Figure 4.4: Screenshot of the Console tab.

4.3 Console tab

The console tab allows the user to send commands to the controller, such as **re**, **abort**, **se**, **sw**, **ls**, **ld**, or **d** for test purposes (see Section 2.2 for a list of available commands).

The data from the controller is displayed in the scrolling window. The user can send commands to the controller by typing them in the editing line at the bottom of the tab. Once the Return key is pressed, the command will be sent to the controller. The **Suppress output** combo box will silence the console, but commands will still be sent to the controller.

4.4 Sequences tab

The **Sequences** tab allows the user to build patterns, sequences, and readout sequences.

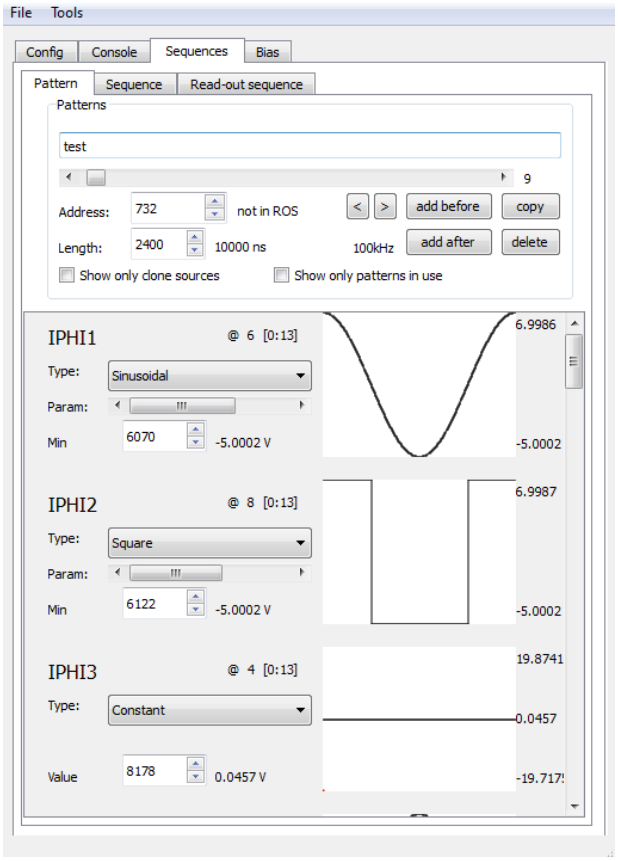

Figure 4.5: Screenshot of the Pattern tab of the Sequences tab.

4.4.1 Pattern tab

The **Pattern** tab (Figure 4.5) allows the user to build the patterns that will be used in the sequences.

New patterns can be created by clicking on the **Add before**, **Add after**, or **Copy** buttons. A visible pattern can be deleted by clicking on the **Delete** button. The pattern order can be changed by clicking on the arrows.

Once a new pattern is created (by means of **Add before** or **Add after** buttons), all the phases' data are initialized to the **Default** values of the phases, as defined in the **Phases** tab of the **Config** tab. The name of the pattern can be set to differentiate it from others.

The length of the pattern can be changed by typing a new value in the **Length** box. The length is expressed in time steps of about 4.17 ns. For example, a value of 1000 will cause the pattern to last about 4.17 microseconds. The minimum length for a blanking or exposing pattern is 2 steps. The minimum length for a pattern that is to be used in a read-out sequence is 6 steps.

The address of the pattern can be changed by typing a new value in the **Address** box. The address

must be between −1 and 245759. The special address −1 makes the pattern a *virtual* one, meaning that it can neither be used in a sequence nor as a blanking or an exposing pattern. A pattern with an address of −1 can only be used in a *merged* pattern or as a clone source for another pattern. Note that the address entered does not reflect the address where the pattern will be written in the controller when programming the patterns, that address is shown next to the **Address** box if the pattern is used in the currently active ROS.

For an analog phase, the phase type can be changed to:

- Constant. The phase will not change throughout this pattern. The constant value is specified by the **Value** parameter.
- Sinusoidal. The phase will take a sinusoidal shape. The parameters of this type of phase are the following:
    1. Min: The minimum value of the sinusoidal shape.
    2. Max: The maximum value of the sinusoidal shape.
    3. Delay: The delay (in ns) of the sinusoidal shape. Even though the time step is 4.17 ns, the delay is expressed in 0.2-ns time steps and the sinusoidal shape will be interpolated.
    4. Frequency: The frequency (repetitions) of the sinusoidal shape. For example, a value of 2 will cause the same sinusoidal shape to be repeated twice during this pattern.
    5. Start: The number of steps to truncate the sinusoidal shape at the beginning of the pattern. The value of the phase from step 0 to **Start**−1 will take the value of the phase at step **Start**.
    6. Stop: The amount of steps to truncate at the end of the pattern. The value of the phase from step **Length**−**Stop** to **Length**−1 will take the value of the phase at step **Length**−**Stop**.
- Square. The phase will take a square shape. The parameters of this type of phase are the following:
    1. Min: The minimum value of the square shape.
    2. Max: The maximum value of the square shape.
    3. Delay: The delay (in steps of 4.17 ns) of the square shape.
    4. Frequency: The frequency (repetitions) of the square shape. For example, a value of 2 will cause the same square shape to be repeated twice during this pattern.
    5. Duty: The duty cycle (in steps of 4.17 ns) of the square shape. The duty cycle is defined by the time the shape is at when it reaches the value defined by the **Max** parameter.
    6. Rise time: The time (in steps of 4.17 ns) the square shape will take to go from the **Min** to the **Max** value.
    7. Fall time: The time (in steps of 4.17 ns) the square shape will take to go from the **Max** to the **Min** value.
    8. Arb mods: This parameter will allow the user to arbitrarily change each step of the pattern, when enabled additional parameters will allow the user to change each step of the pattern.
- Trilevel. The phase will take a square shape and will have a plateau at an intermediate level. The parameters of this type of phase are the following:
    1. Min: The minimum value of the trilevel shape.
    2. Max: The maximum value of the trilevel shape.

3. Mid: The value of the intermediate plateau of the trilevel shape.
4. Delay: The delay (in steps of 4.17 ns) of the trilevel shape.
5. Frequency: The frequency (repetitions) of the trilevel shape. For example, a value of 2 will cause the same trilevel shape to be repeated twice during this pattern.
6. Duty max: The duty cycle (in steps of 4.17 ns) of the trilevel shape. The duty cycle is defined by the time the shape is at when it reaches the value defined by the **Max** or **Mid** parameters.
7. Duty mid: The duty cycle (in steps of 4.17 ns) of the intermediate level of trilevel shape. This duty cycle is defined by the time the shape is at when it reaches the value defined by the **Mid** parameter.
8. Rise time: The time (in steps of 4.17 ns) the square shape will take to go from the **Min** to the **Mid** value.
9. Fall time: The time (in steps of 4.17 ns) the square shape will take to go from the **Max** to the **Min** value.
10. Rise time mid: The time (in steps of 4.17 ns) the square shape will take to go from the **Mid** to the **Max** value.
11. Arb mods: When enabled, additional parameters will allow the user to change each step of the pattern.

- Arbitrary. The phase will take any shape that the user defines. The value of the phase can be hand-tuned at any time throughout the pattern. The parameters are the following:

  1. Delay: Delay by step increments of the defined arbitrary phase.
  2. Value #*N*: The value the phase will take at the N^{th} step.

- Clone. The phase will use the phase data defined in another pattern. The phase is defined in only one pattern and all others can clone the first one. If the source pattern is smaller than the destination pattern, the source pattern will be repeated the required number of times to fill the destination pattern. If the source pattern is larger than the destination pattern, it will be truncated. It is recommended to use a destination pattern that will allow a source pattern to be repeated an integer number of times (example: a 500 ns destination for a 100 ns source). The parameters for this type of phase are the following:

  1. Source: The pattern from which to clone the phase at the same address.
  2. Start: The number of steps to truncate the shape at the beginning of the pattern. The value of the phase from step 0 to **Start**−1 will take the value of the phase at step **Start**. Negative values can be entered to truncate the beginning of the pattern until the N^{th} minimum or maximum with the value at the N^{th} minimum or maximum. For the N^{th} maximum enter: $-2N+1$ (such that −1 gives the 1^{st} maximum, −3 the 2^{nd}, etc.) and for the N^{th} minimum enter: $-2N$ (such that −2 gives the 1^{st} minimum, −4 the 2^{nd}, etc.).
  3. Stop: The amount of steps to truncate the shape at the end of the pattern. The value of the phase from step **Length**−**Stop** to **Length**−1 will take the value of the phase at step **Length**−**Stop**. Negative values can be entered to truncate the end of the pattern from the N^{th} minimum or maximum with the value at the N^{th} minimum or maximum. For the N^{th} maximum enter: $-2N+1$ (such that −1 gives the 1^{st} maximum, −3 the 2^{nd}, etc.) and for the N^{th} minimum enter: $-2N$ (such that −2 gives the 1^{st} minimum, −4 the 2^{nd}, etc.).

- Clone min. The phase will use the minimum level defined in another pattern.

- Clone max. The phase will use the maximum level defined in another pattern.
- Merged. This phase type is special. When the type of one phase of a pattern is set to **Merged**, all the phases of this pattern become merged into one. A merged pattern is composed of the data of other patterns. The number of patterns composing the merged pattern is set as the first parameter and the patterns are then chosen as the remaining parameters. The length of a merged pattern is adjusted automatically as a function of the patterns composing the merged pattern.

For a digital phase, the phase type can be changed to:

- Constant. The phase will not change throughout this pattern. The constant value is specified by the **Value** parameter.
- Square. The phase will take a square shape. The parameters of this type of phase are the following:
  1. Min: The minimum value of the square shape.
  2. Max: The maximum value of the square shape.
  3. Delay: The delay (in steps of 4.17 ns) of the square shape.
  4. Frequency: The frequency (in repetitions) of the sinusoidal shape. For example, a value of 2 will cause the same sinusoidal shape to be repeated twice during this pattern.
  5. Duty: The duty cycle (in steps of 4.17 ns) of the square shape. The duty cycle is defined by the time the shape is at the value defined by the **Max** parameter.
- Arbitrary. The phase will take any shape that the user defines. This type of phase has the same amount of parameters as the length of its pattern. Thus, the value of the phase can be hand-tuned at any point in time throughout the pattern.
- Clone. The phase will use the phase data defined in another pattern. The phase is defined in only one pattern and all other patterns clone the first one. If the source pattern is smaller than the destination pattern, the source pattern will be repeated the required number of times to fill the destination pattern. If the source pattern is larger than the destination pattern, it will be truncated. It is recommended to use a destination pattern that will allow a source pattern to be repeated an integer number of times (example: a 500 ns destination for a 100 ns source).
- Merged. This is the same as the merged pattern of the analog phase.

The **Show only clone sources** combo box allows the user to see only the phases from this pattern that are used as clone sources in other patterns. This can be useful when using a base pattern with an address of −1 to build other patterns.

The **Show only patterns in use** combo box allow the user to see only the patterns that are used in the active ROS either as a source for a cloned pattern or directly.

4.4.2 Sequence tab

The **Sequence** tab (Figure 4.6) allows the user to build the sequences that will be used in the readout sequence.

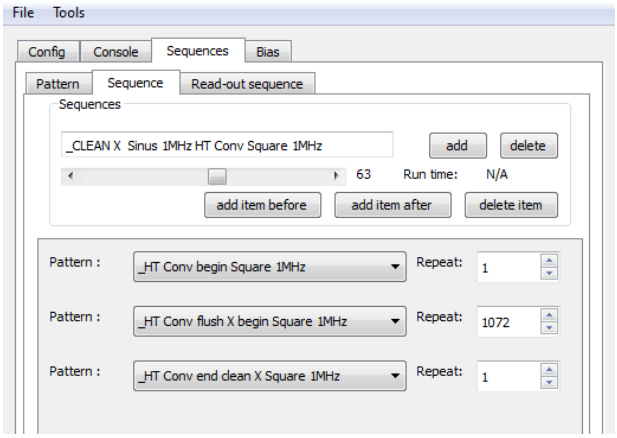

Figure 4.6: Screenshot of the Sequence tab of the Sequences tab.

The **Add** button is used to create a new sequence. The **Delete** button is used to delete the visible sequence. The name of the sequence can be set to differentiate it from others.

The sequence can be populated with patterns by clicking on the **Add item before** or **Add item after**. If an item of the sequence is selected (by clicking on it), the new item will be added before or after this item. The **Delete item** button will delete the selected item.

The pattern composing the item of the sequence can be selected with the **Pattern** combo box of the item. The patterns are identified by the name they have been given. Then, the number of repetitions of this pattern can be typed in the **Repeat** box. Valid values for repeat are between −32 and 65535. When a negative value is set, this means that a variable number of repetitions is to be used (known as a *sequence variable*, see Section 3.4). The negative number represents the index position of the variable to use (first variable: −1, fifth variable: −5 and so on).

4.4.3 Read-out sequence tab

The **Read-out sequence** tab (Figure 4.7) allows the user to build the readout sequences that will be used to readout a CCD.

The **add** button is used to create a new readout sequence. The **delete** button is used to delete the visible readout sequence. The name of the readout sequence can be set to differentiate it from others.

The pattern to be used while exposing can be set in the **Exposing pattern** combo box. The pattern to be used as a pre-expose pattern can be set in the **Pre-expose pattern** and the number of times to execute it can be set in the input box next to it, select **None** from the drop-down list if no pre-expose pattern is needed. The pattern to be used while blanking can be set from the **Blanking pattern** drop-down list.

The readout sequence can be populated with sequences by clicking on the **Add item before** or **Add item after**. If an item of the readout sequence is selected (by clicking on it), the new item will be added

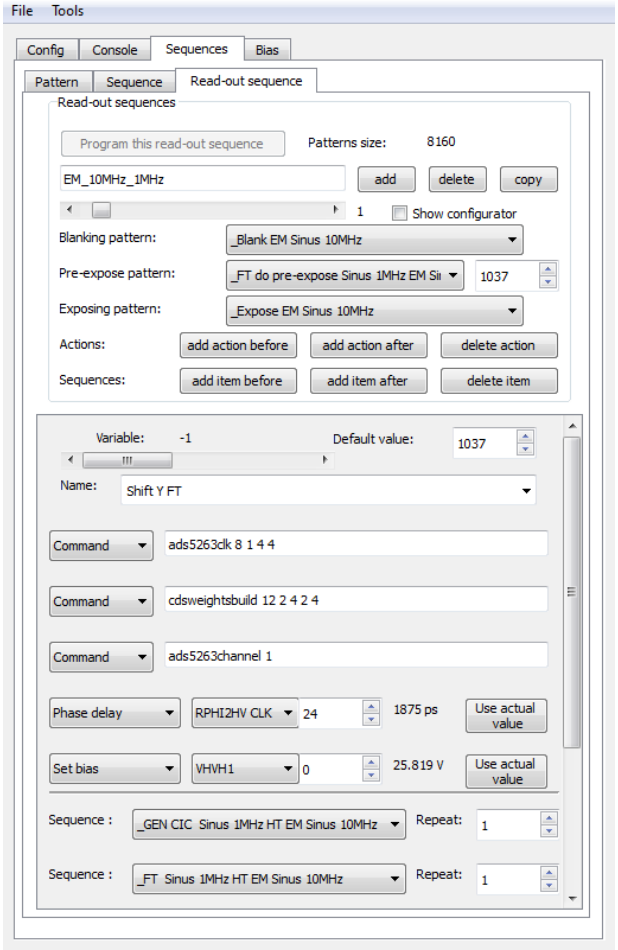

Figure 4.7: Screenshot of the Readout sequence tab of the Sequences tab.

before or after this item. The **Delete item** will delete the selected item.

The sequence composing the item of the readout sequence can be selected with the **Sequence** combo box of the item. The sequences are identified by the name they have been given. Then, the number of repetitions of this sequence can be typed in the **Repeat** box. Valid values for repeat are between −32 and 65535. When a negative value is set, this means that a variable number of repetitions is to be used (known as a *sequence variable*, see Section 3.4). The negative number represents the index position of the variable to use (−1 specifies the first variable; −5 specifies the fifth variable and so on).

All the sequence variables used in a readout sequence will appear at the top of the readout sequence. There, a default value can be assigned to the variables and a name can be given to them to facilitate their usage.

The readout sequence can also be populated with **actions** by clicking on the **Add action before** and **Add**

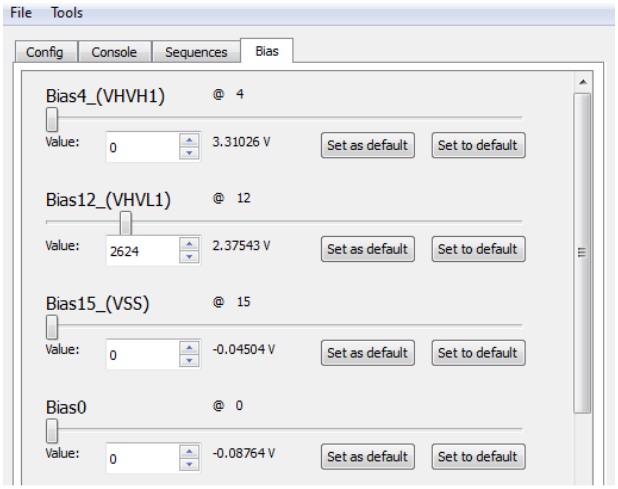

Figure 4.8: Screenshot of the Bias tab.

action after buttons. If an action is selected, it can be deleted by clicking on the **Delete action** button. The actions can be of three kinds: a bias value to set, a command to run or a delay to add to a phase. These actions will be executed when a readout sequence is loaded. Usually, these actions will have the following attributes, a **ads5263channel** command to set the appropriate ADC channel for this readout sequence, a **se** command to set an exposing time or a **msg** command to send some information to the host computer. The default values of the biases can be changed with the bias action. A bias that has no associated action will be assigned its default value, as defined in the **Bias** tab of the **Config** tab. A phase delay action can be executed on 4 bits digital signals and will essentially delay the output of the signal and override the default delay value set in the **Config** tab.

4.5 Bias tab

The **Bias** tab allows the user to set the value of the biases.

The value of the bias can either be set by dragging its slider or typing the value in the **Value** box. In the first case, the value of the bias is updated only when the slider is released. In the second case, the value of the bias is updated only when the Return button is pressed.

The **Set to default** button brings the bias back to its default value, as defined in the **Bias** tab of the **Config** tab (Section 4.2.3). When a new readout sequence is loaded, all biases are taken back to their default values, unless an action is specified for a given bias. The bias would then use the value defined by this action instead of its default value.

The **Set as default** button will set the current value as the default value, as would have been the case if the user had changed it in the **Config** tab.

4.6 Using it

Once the patterns, sequences, and readout sequences are built, one can start using them to clock a CCD.

There are two ways of loading the patterns and the readout sequence to the controller. One of them involves using *cccpComm* at all times. The other is made for using *cccpComm* only as a configuration application and then using a custom application for controlling the controller.

4.6.1 Loading patterns and readout sequences with cccpComm

First, the patterns must be programmed into the BRAMs of the controller board. This is done by clicking the **Program Patterns** button (*Ctrl-P* shortcut or clicking on the **Program Patterns** in the **Tools** menu) in the **Pattern** tab of the **Sequences** tab (Figure 4.5). This writes all the patterns defined in the **Pattern** tab of the **Sequences** tab.

Once the patterns are written into the controller board, a readout sequence can be loaded. This is accomplished by selecting a readout sequence in the **Readout sequence** tab of the **Sequences** tab. Then, by clicking the **Program this read-out sequence** button (*Ctrl-R* shortcut or clicking on the **Program read-out sequence** in the **Tools** menu), the readout sequence is loaded. When doing so, *cccpComm* does the following:

1. Aborts the running readout sequence, if there was one running;
2. Deletes all the sequences from the controller;
3. Loads all the sequences with their pattern start address, stop address and, repetition count into the controller;
4. Loads the sequences repetitions and order of execution into the controller;
5. Loads all the biases with their default values or with values specified by the actions;
6. Loads the exposing, pre-expose and blanking patterns addresses into the controller. Immediately, the blanking pattern starts to loop;
7. Executes all the commands specified by the actions in the order they appear in the readout sequence;
8. For each *sequence variable* used by the readout sequence, a **msg** command is issued with the following text: `VarN: name`, where `N` is the number of the *sequence variable* (0 for −1, 1 for −2, ...) and `name` is the name of this *sequence variable*, as given by the user. When changing the value of the *sequence variable*, the value of `N` has to be sent to the **ssv** command.

The user can then set the exposing and blanking times through the console (see Section 2.2 for a list of the available commands). The **re** serial command may be used to request the readout sequence be executed a given number of times. The **abort** serial command can be used to terminate repetition of the readout sequence and fall back to the blanking pattern.

In order to switch to another readout sequence, the user can simply select it in the **Read-out sequence** tab of the **Sequences** tab and click on **Program this read-out sequence**. Since all patterns are already loaded

into the controller, there is no need for re-loading them.

This method for controlling the controller is best for developing and testing the waveforms to send to the CCD. Once a readout sequence is running, a pattern can be modified and updated with the *Ctrl-U* shortcut (or by clicking on the **Update this pattern** in the **Tools** menu). When doing so, *cccpComm* does the following:

1. Aborts the running readout sequence;
2. Loads the active pattern's data into the controller. The active pattern is the one visible in the **Pattern** tab of the **Sequence** tab;
3. Issues a **start** command to re-start the readout sequence.

This allows pattern modifications to be made and tested quickly.

Synchronous items and asynchronous items distinction

Synchronous items of a ROS can only be changed in sync with the readout of the CCD, while asynchronous items can be changed anytime (see Section 2.1.5 and 2.1.6). When using *cccpComm* to load directly the patterns and the readout sequences into the controller the user should be aware of the following:

- A change to the patterns or to the sequences will be affected only when the user does the procedure explained in 4.6.1.
- A change to the exposing time or the blanking time can be done anytime through the **se** and **sw** serial commands, although it will take effect only after the current image has been acquired (the current ROS is done executing).
- The CDS can be configured asynchronously to the readout of the sequencer, but since the video signals generation is done in the ROS any adjustment to the number of samples per pixels (through **cdsweightsbuild** or **cdssamples** serial commands) needs an adjustment in the ROS (see Section 2.1.3).
- The biases are asynchronous items and can be changed at any time during the execution of the ROS.

4.6.2 Loading patterns and read-out sequences from the flash memory of the controller

CCCP has a flash memory that allows the patterns' data and readout sequences definitions to be stored. This allows the operation of the controller without resorting to the *cccpComm* application.

When clicking on the **Save config to flash** item of the **Tools** menu (*Ctrl-F* shortcut), *cccpComm* starts writing all of its configuration into the flash memory. This comprises:

- All patterns;
- All readout sequences with their respective actions and bias settings.

Once the readout sequences are written into the controller, they will be preserved even when the controller is powered off. A checksum is calculated on the contents of the flash memory once the writing process is finished and the resulting value is stored in EEPROM. Upon starting up, the controller again calculates a checksum of the flash memory. If the new checksum does not compare equal to the stored value, the controller will refuse to load any of the readout sequences from the flash memory as their data may have been corrupted.

A list of the available readout sequences stored into the controller is available by issuing the **ls** serial command. See Section 2.2.3 for a definition of the **ls** command.

In order to use a readout sequence, it is necessary to load it by issuing the **ld** serial command with the readout sequence number to load as its parameter. The **ld** command loads from the flash memory all (and only) the patterns that are needed for the execution of the readout sequence. Then, it does the same thing as when a readout sequence is loaded from *cccpComm* (see Section 4.6.1).

When issuing a **ld** command, it is possible that the controller outputs a message indicating that the flash memory is being checksumed. This happens when the background process that checksums the flash memory upon powering on the controller is not done reading the flash memory. Usually, if the **ld** command is issued 30 seconds after powering on the controller, the checksum background process should be terminated.

This method for storing the patterns and readout sequences into the controller allows the controller to be completely independent of the *cccpComm* application. Everything that is needed to run a readout sequence is kept into the controller. Once a readout sequence is loaded from the flash memory, the controller is exactly in the state it would have been if the readout sequence had been loaded with *cccpComm* (*Ctrl-P* and *Ctrl-R*). However, the update of the patterns (*Ctrl-U* in *cccpComm*) will not work.

Chapter 5

cccpServer: The access-management application

The *cccpServer* application is a server that acts as an intermediary between CCCP and the controlling applications, *cccpComm* (see Chapter 4) and *cccpView* (see Chapter 6). It functions as a server for the controlling applications by synchronising concurrent serial communications between them and CCCP along with managing framegrabbers and the acquisition and transfer of image data. Thus it is required, even if the controlling applications are used on the same computer system to which CCCP is connected. In addition, it allows the possibility of remote access for the controlling applications to any computer running *cccpServer* (with a connected CCCP) because they connect to it via a standard TCP socket.

5.1 Port Selection

The port selection area (Figure 5.1) allows you to specify on which **TCP port** the *cccpComm* and *cccpView* clients will connect to the server. Each of these applications connects to it's own respective port.

5.2 Client Status

The client status areas (Figure 5.2) display the current state of the TCP sockets. The **Client status** field indicates whether a given client (*cccpView* or *cccpComm*) is connected or disconnected.

5.3 Frame rate

The **Frames sent per second** (Figure 5.3) field indicates the current rate at which frames are sent to a connected instance of *cccpView*. The frame rate is only reported while *cccpView* is requesting images.

5.4 Server Messages

The server messages area (Figure 5.4) displays information about actions being preformed by the server and changes to the server's state.

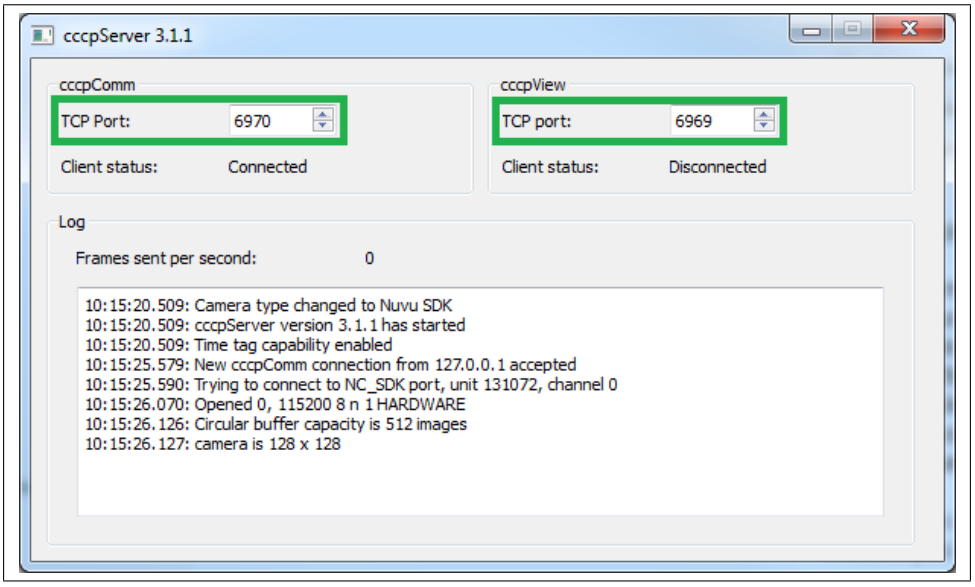

Figure 5.1: Screenshot of *cccpServer* indicating the Port Selection areas

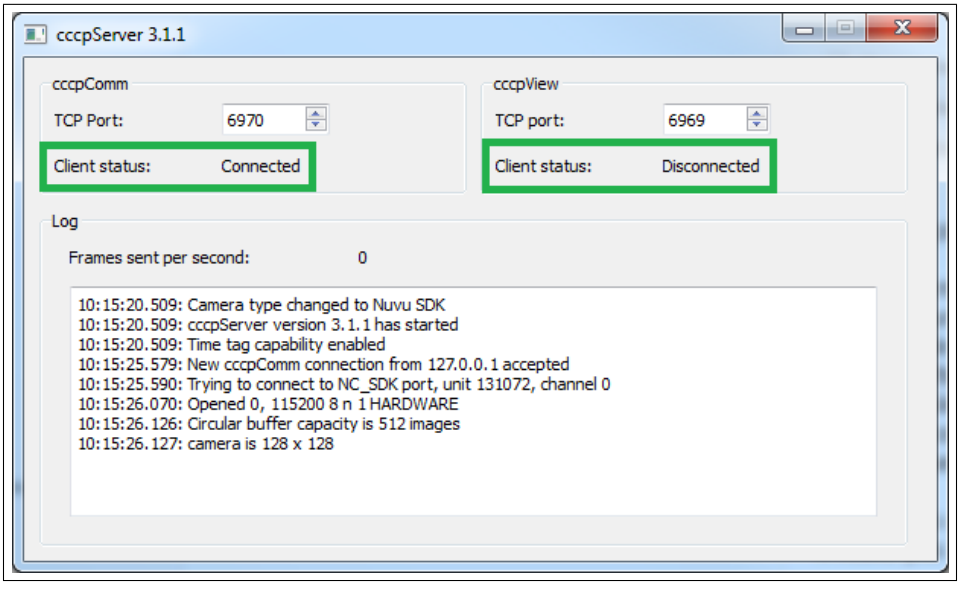

Figure 5.2: Screenshot of *cccpServer* indicating the Status areas

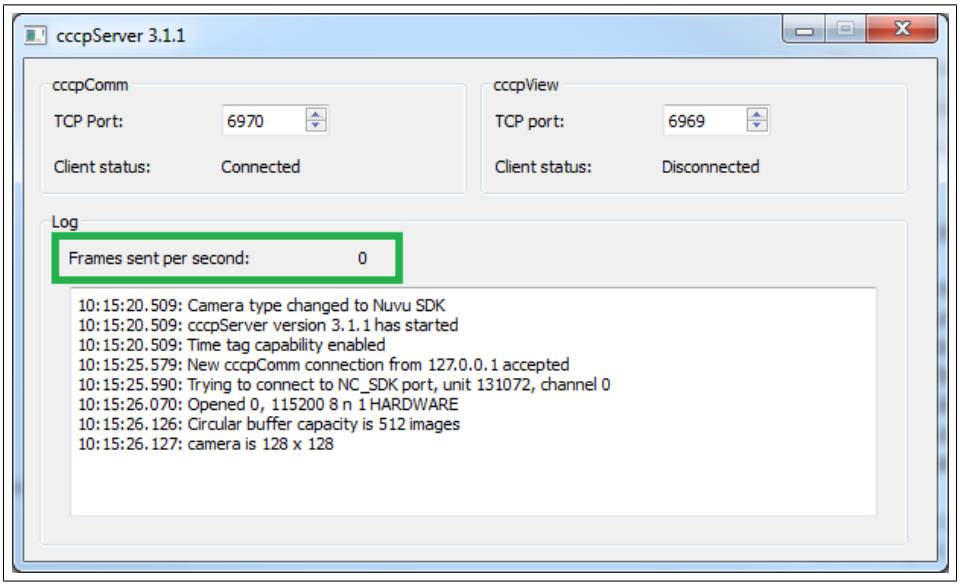

Figure 5.3: Screenshot of *cccpServer* indicating the frame rate field

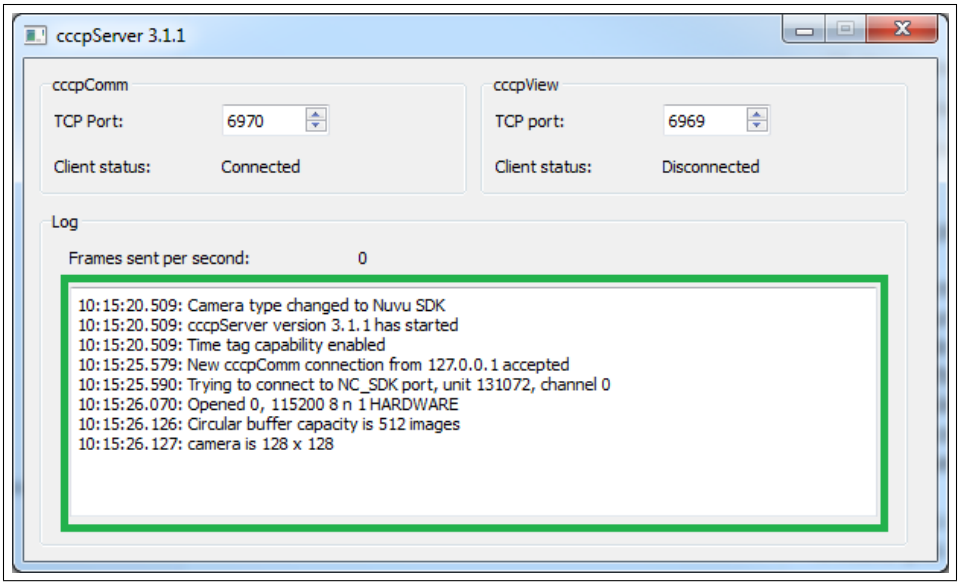

Figure 5.4: Screenshot of *cccpServer* indicating the Server Messages area

Chapter 6

cccpView: The image-display application

Images that are readout by CCCP and grabbed by the server can be displayed and/or recorded via *cccpView*. Once communication with CCCP has been established via *cccpServer* (see Chapter 5) and a readout mode has been programmed and loaded via *cccpComm* (see Chapter 4), *cccpView* can be used to connect to the server and request images from it, which will be captured by CCCP, grabbed by the server and then supplied to *cccpView*

Upon launching *cccpView*, the main window (see Section 6.1) will be shown. Here you may define the parameters of the TCP network connection to *cccpServer* and control the recording behaviour of *cccpView*. Also, a **Live image** window (see Section 6.2) is available to display supplied images and a **Live stats** window (see Section 6.3) to display the distribution of pixel values and provide certain statistical tools for analysis of the images.

To start viewing images supplied by *cccpServer* immediately, complete the **Server** tab in the main window (see Section 6.1.1), click the **Connect** button, click the **Go** button on the right and then select the **Show live image** checkbox just above it.

6.1 Main window

The main window (see Figure 6.1) controls the configuration of the network connection to the server, of basic camera properties and of the image acquisition and recording process (see Sections 6.1.1 and 6.1.2). In particular, in order for *cccpView* to correctly handle supplied images, you need to setup the expected size of the image buffers. For instances where the readout supports multiple taps simultaneously, there is also a tab that allows re-ordering of the columns of returned images.

In addition, the status of the controller and of the image acquisition pipeline are indicated (see Sections 6.1.4 and 6.1.5) and the **Live image** window may be activated via the **Show live image** checkbox at the bottom of the main window.

6.1.1 Server tab

Figure 6.1 shows the layout of the **Server** tab.

The **Connection** panel allows you to define the network settings used to connect to *cccpServer* (see Chapter 5) via the **Host** and the **TCP port** text boxes. A **Host** may be specified directly by IP or by hostname, if your DNS will recognise it, and the **TCP port** must be the same as that on which *cccpServer* will

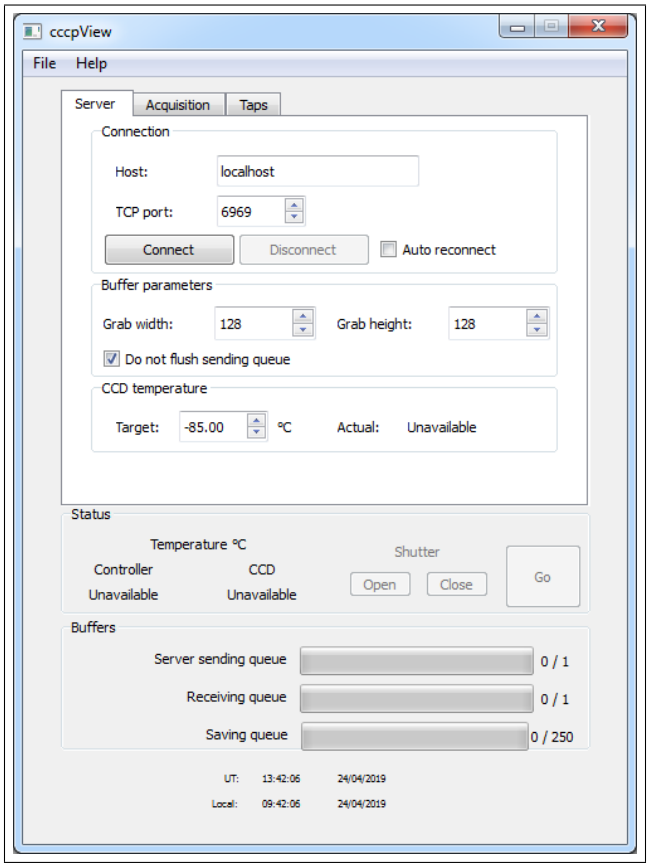

Figure 6.1: Screenshot of the *cccpView* main window with the **Server** tab selected

be listening (see Section 5.1). When these are set, clicking the **Connect** button allows *cccpView* to attempt to contact *cccpServer*, running on the appropriate host. You may drop the connection with *cccpServer* by clicking **Disconnect**. The **Auto reconnect** checkbox provides the option for *cccpView* to attempt to contact *cccpServer* on the given host and port immediately upon the next start-up.

The **Buffer parameters** panel provides the **Grab width** and **Grab height** spinboxes which allow you to specify the size of the image buffers handled and supplied by *cccpServer*: **Grab width** should correspond to the length of the image lines and **Grab height** should correspond to the number of image lines generated by CCCP during a readout sequence. Also present in this panel is the **Do not flush sending queue** checkbox which is selected by default. This ensures that all images acquired are passed from *cccpServer* to *cccpView* for display and/or writing to disk. However, if there is a large degree of latency in the connection with *cccpServer*, this may cause the server image buffer (see Section 6.1.5) to fill up. If you do not require every frame (e.g. you are not saving images and the CCCP frame rate is greater than the maximum refresh rate of your monitor or the **Max. frame rate** of the **Live image** window, see Section 6.2) you can deselect the **Do not flush sending queue** checkbox to allow the *cccpServer* image buffers to be flushed after each image has been received by *cccpView*.

Finally, if your hardware configuration supports this feature, the **CCD temperature** panel allows you to request and monitor the temperature of the sensor used by CCCP. The required temperature may be set

via the **Target** spinbox and the current temperature and status are presented next to it beside the **Actual** label.

6.1.2 Acquisition tab

Figure 6.2 shows the layout of the **Acquisition** tab.

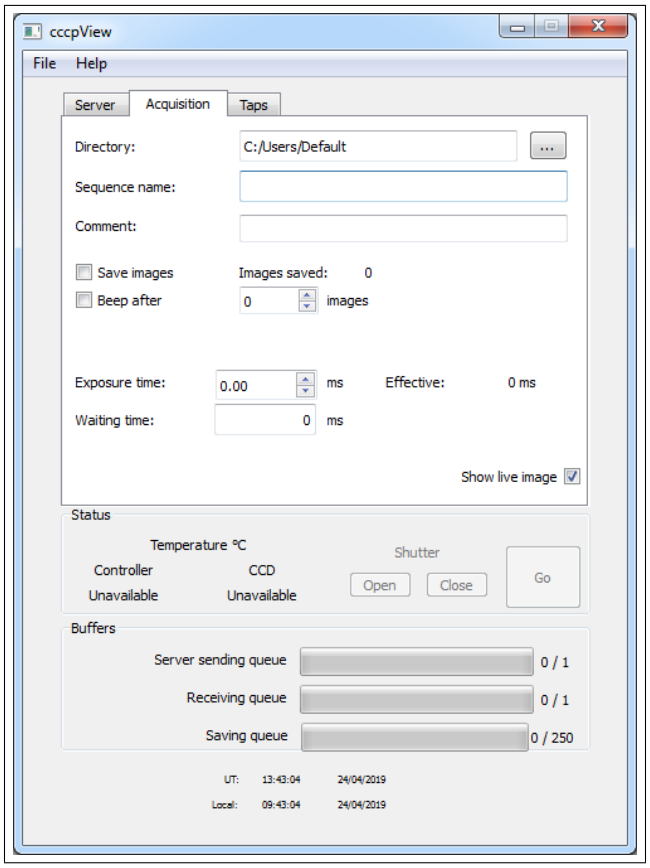

Figure 6.2: Screenshot of the *cccpView* main window with the **Acquisition** tab selected

The **Directory** textbox accepts a path to a directory in which new acquisition sequences will be saved, while the button next to it allows you to browse for an appropriate directory. The **Sequence name** textbox accepts the name of a directory that will be created in the acquisition directory (see Section 6.1.1) when images are saved from an acquisition sequence; the images will be saved in this new directory. If the sequence name has already been used, a warning window will ask if you want to delete the existing contents of the sequence name directory. The **Comment** textbox accepts a text string which will be inserted into the FITS file header of saved images. The text string will be contained in the FITS header field associated with the keyword `CCCP_CMT` and any **Comment** text string longer than 30 characters will be truncated.

The **Save images** checkbox, when selected, allows you to save all individual frames received from *cccpServer* as FITS format files. As noted above, the image files will be saved into the directory specified by the **Acquisition directory** and **Sequence name** textboxes. If the function is compatabile with your system, the **Beep**

after checkbox and spinbox will be available and allow you to request an audible warning at regular intervals during the acquisition sequence. If GNU Gzip is supported for your system, the **Compress images** checkbox will be available and, if selected, will cause the FITS image files to be compressed on saving.

NB: if compression is available and selected, the image files will be compressed as long as the buffer of the **Saving queue** (see Section 6.1.5) is less than one quarter full; if the buffer is too full, compression is bypassed to accelerate the pipeline.

The **Exposure time** textbox accepts a floating point value to be used as the duration of exposure requested from CCCP. Note that the actual effective exposure achieved cannot be less than the time required for the readout sequence of a single frame to be completed, unless a non-zero **Waiting time** is specified: the achieved exposure time is displayed next to the **Effective** label. This value is updated whenever an image is supplied by *cccpServer*. The **Waiting time** textbox accepts a floating point value to be used as the duration of the time spent flushing the CCD with a blanking pattern after the end of a readout sequence and before the beginning of the next exposure.

6.1.3 Taps tab

Figure 6.3 shows the layout of the **Taps** tab.

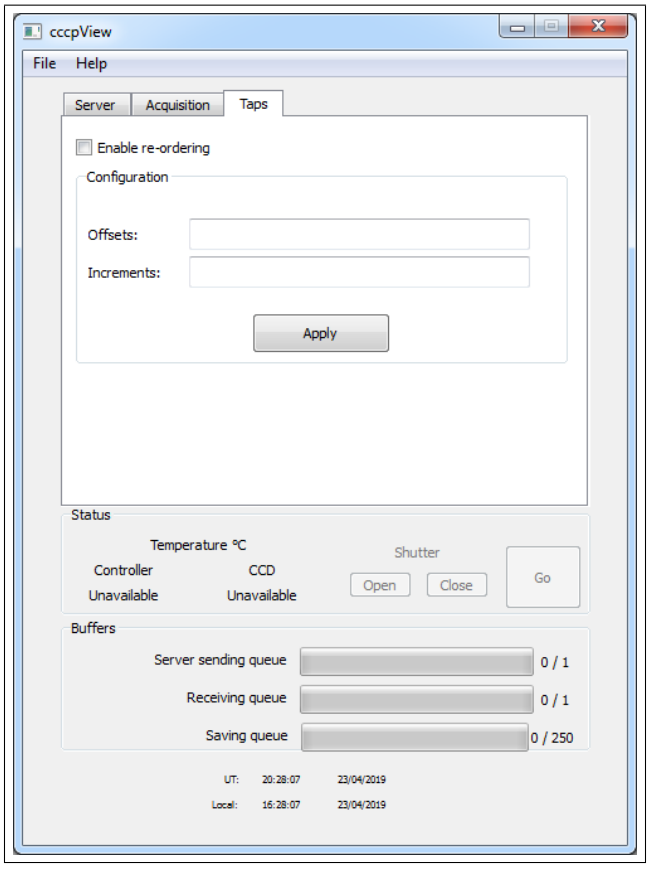

Figure 6.3: Screenshot of the *cccpView* main window with the **Taps** tab selected

This tab is intended for use when a readout sequence is defined such that multiple taps generate pixels simultaneously. This will result in each iteration of the sequence that generates a pixel from the CDS in fact causing one pixel for each active readout tap to be passed across the CameraLink channel. As a consequence, the grab width needs to be the product of the number of iterations of that sequence per line and the number of readout taps. Now, pixels from each of the readout taps are returned together whereas each readout tap generally operates on a distinct domain of the sensor. Thus an acquired frame does not present a contiguous image but rather the columns must be re-grouped to properly represent the distinct domains of the sensor.

In this tab you may define the order of the active readout taps in the **Configuration** panel before activating the re-grouping of columns with the **Enable re-ordering** checkbox. Pixel re-ordering is defined by identifying the offset in the final image where each contiguous domain of the sensor should begin (set in the **Offsets** textbox) and the increment in the column position in the raw image needed to reach the pixel appropriate to the next offset.

Values for the offsets and increments must be supplied as space-separated lists of integers; one each for each active readout tap. the **Apply** button may be used to refresh the re-ordering after modifying the configuration.

6.1.4 Status panel

The **Status** panel provides an overview of the status of CCCP and of its camera in terms of the **Temperature** (in degrees Celsius), the **Shutter** position and whether or not readout is in progress. The state of the displayed **Controller** and **CCD** temperatures may be indicated also :

Unavailable indicates that communication has not yet been established via *cccpServer* and CCCP cannot be polled for its temperature.

Stale indicates that the last attempt to poll CCCP for its temperature failed (e.g. if the serial communication channel was occupied) and the displayed value is from a previous polling attempt.

The **Shutter Open** and **Close** buttons allow the shutter state to be switched and readout of the CCD can be toggled via the **Go/Stop** button which controls whether *cccpView* is requesting images from CCCP (via *cccpServer*).

6.1.5 Buffers panel

The **Buffers** panel indicates the status of the buffers used in the image acquisition chain. The **Server sending queue** is maintained on the host running *cccpServer* and contains grabbed frames which have yet to be passed to *cccpView*. The **Receiving queue** contains frames which have been passed to *cccpView* but have yet to be processed for display. The **Saving queue** contains frames that have been queued for saving to disk.

NB: upon closing *cccpView*, if the **Saving queue** is not empty, the application will ask if you want to finish saving all images in the queue or discard them.

6.2 Live image window

Figure 6.4 shows the layout of the **Live image** window.

The **Live image** window displays images received from *cccpServer* in an image pane at the bottom of the window and provides a variety of functions to manipulate their appearance for improved viewing. In

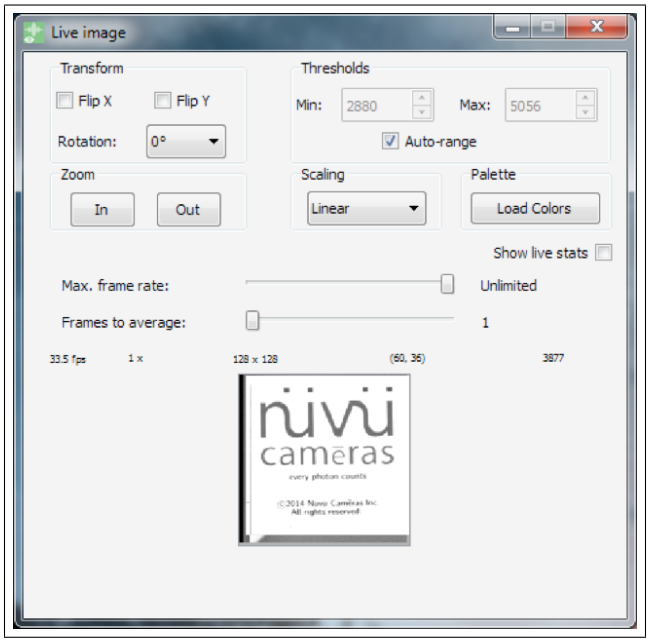

Figure 6.4: Screenshot of the *cccpView* **Live image**

general, functions that change the image layout are available on the left-hand side, functions that change the visual representation of pixel values are available on the right-hand side and functions that change the rate of image refresh are available just above the image pane itself.

Changes to the layout of displayed images are encapsulated in the **Transform** and the **Zoom** panels. The **Transform** panel provides the **Flip X** checkbox which allows you to flip the image horizontally and the **Flip Y** checkbox which allows the same vertically. In addition it provides the **Rotation** drop-down menu which allows you to select the rotation of the displayed image in increments of 90˚. The **Zoom** panel features the **In** button and **Out** button which allow you to modify the magnification of the displayed image.

The visual representation of pixel values can be modified by manipulating the **Thresholds**, value **Scaling** and color **Palette**. In the **Thresholds** panel, the **Min** and **Max** spinboxes allow you to bracket the range of pixel values that are mapped to the color palette. All values below the minimum threshold will be displayed the same color as the minimum threshold. Likewise, all values above the maximum threshold will be displayed the same color as the maximum threshold. The thresholds may also be generated automatically by selecting the **Auto-range** checkbox. This function brackets the last displayed image such that the thresholds exclude the upper and lower 1% of pixels. Between the two thresholds, the pixel values may be scaled to the color palette in various ways, to accentuate differences in small values relative to differences in large ones. To allow this, the **Scaling** panel provides a drop-down menu which may be used to select **Linear**, **Squareroot** or **Logarithmic** scaling. Finally the color palette, which determines what color represents a particular scaled pixel value, may be defined in an Adobe Color Table file (`.act` format) and loaded via the **Palette** panel's **Load Colors** button. Below this is the **Show live stats** checkbox which provides access to the **Live stats** window (see Section 6.3).

The image refresh rate can be controlled via the **Max. frame rate** and **Frames to average** sliders.

The **Max. frame rate** slider allows you to limit the rate at which *cccpView* requests images which may become necessary if there is too much latency in the connection with *cccpServer*. The **Frames to average**

slider allows you to chose how many frames are accumulated to produce each displayed image which may be necessary to discern detail when operating at very low illumination. Note that the displayed image is a running average over the number of frames requested here.

NB: saved images are individual frames received from CCCP, not the averaged image displayed in the **Live image** window.

Finally, the status line along the top of the image pane lists, in order :

- the frame rate at the server,
- the magnification of the displayed image,
- the dimensions of the displayed image (width $\times$ height),
- the image coordinates of the cursor (if it is inside the image pane),
- the value contained in the pixel beneath the cursor (if it is inside the image pane).

6.3 Live stats window

The **Live stats** window provides a set of tools for examining and analysing the pixel values of images generated by CCCP. The distribution of pixel values in individual images may be inspected via the **Histogram** tab; the optical sharpness of the image can be monitored via the **Focus** tab; and the performance of the readout sequence can be verified via the **Characterisation** tab.

6.3.1 Histogram tab

Figure 6.5 shows the layout of the **Histogram** tab.

A histogram of the pixel values constituting the entire image displayed in the **Live image** window is shown. Below the histogram, the **Pixel Values** panel summarizes the range of values in the image :

Min indicates the smallest pixel value in the image,

Max indicates the largest pixel value in the image and

Delta indicates the difference between these two, the total range of pixel values available in the image.

Next to it, the **Range** panel allows you to choose the range of the histogram using the **Min** and **Max** spinboxes. This range will be used as long as the **Specified** radio button is selected, immediately below the spinboxes. The other two radio buttons allow the histogram range to be determined automatically: the **Auto** radio button limits the histogram range so that it only contains filled bins while the **Match image** radio button forces the histogram to have the same range as that defined by the **Thresholds** panel of the **Live image** window.

Finally, the **Bin contents** panel allows you to control how the bin contents are determined and displayed. By default the height of the bar representing a bin varies linearly according to its contents: the number of pixels with values in the range covered by the bin. However, to improve the visibility of bins with a relatively small content, it is possible to choose logarithmic scaling of the displayed height of the bin contents by selecting the **Logarithmic** checkbox. Also, by default there is one bin per ADU but, if this proves too sparse (i.e. there are too many gaps in the distribution, of only a few bins wide), you may choose to combine bins in multiples of two with the **Binning** drop-down menu.

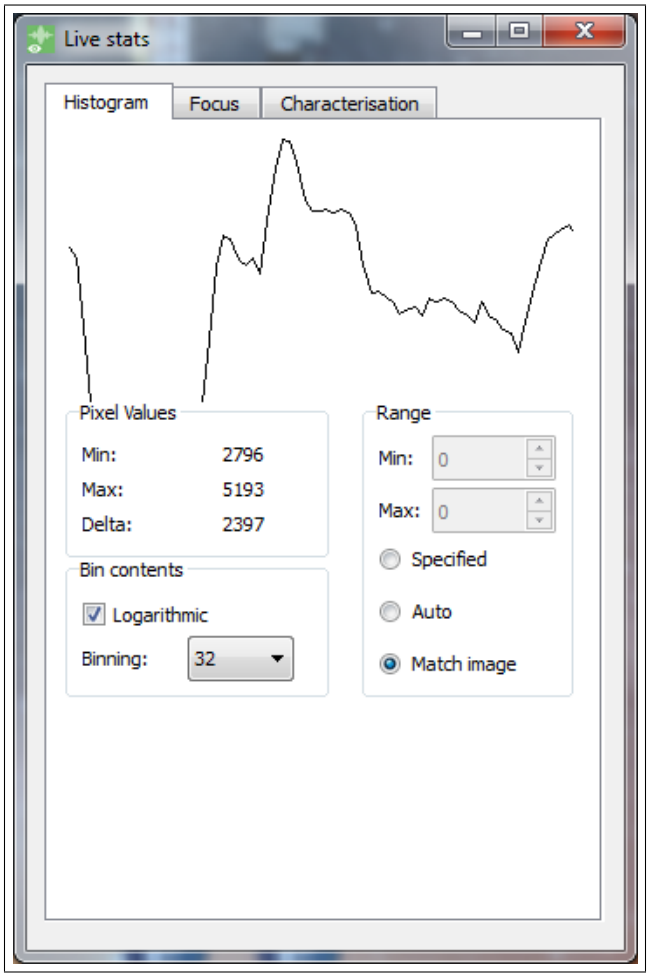

Figure 6.5: Screenshot of the *cccpView* **Live stats** window with the **Histogram** tab selected

6.3.2 Focus tab

Figure 6.6 shows the layout of the **Focus** tab.

A region of interest of the image displayed in the **Live image** window is shown. The center of this sub-image can be chosen by clicking in the image pane of the **Live image** window, while the size of the sub-image is set using the **Sub image size** slider. Below and to the right of the sub-image, selected statistics of the pixel values of which it is comprised are displayed :

Stddev indicates the standard deviation of the pixel values in the sub-image,

Mean indicates the mean of the pixel values in the sub-image,

Min indicates the smallest pixel value in the sub-image,

Max indicates the largest pixel value in the sub-image and

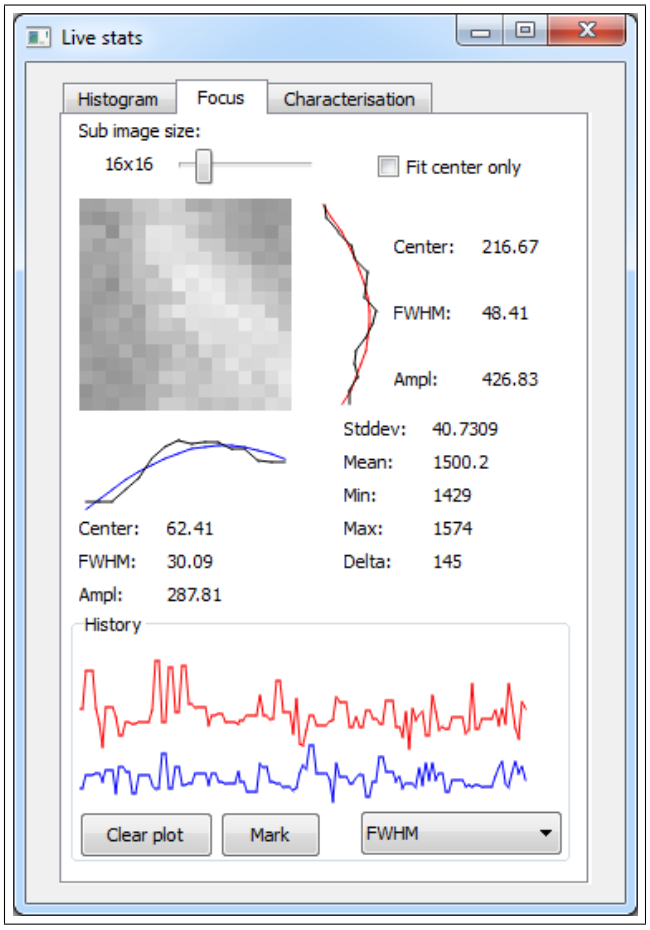

Figure 6.6: Screenshot of the *cccpView* **Live stats** window with the **Focus** tab selected

Delta indicates the difference between the latter two, the total range of pixel values available in the sub-image.

Along the vertical and horizontal axes of the sub-image, a mean projection of the pixel values onto the appropriate axis is shown. A fit of a Gaussian distribution to each of these projections is automatically performed and plotted as a coloured line over the projection. The fit quality will be best for a high contrast image In addition, the parameters of the fit functions are displayed next to the plots :

Center is the pixel coordinate, along the appropriate axis, of the peak of the fit function;

FWHM is the Full-Width Half Maximum of the fit function; and

Ampl is the amplitude of the fit function, meaning the height of the peak above the baseline.

To restrict the fitted projections to a cross-section through the center of the sub-image along each axis, the **Fit center only** checkbox may be selected.

If a point spread function is isolated in the sub-image these fits to the projections may be used as a focussing aid. To this end, the evolution of the FWHM of these fit functions is displayed in the **History** plot

at the bottom of the tab. A vertical bar can be inserted into the plot to mark a reference point in time by clicking the **Mark** button. The **History** plot may be cleared by clicking the **Clear plot** button. The **History** plot can instead be of one of several statistics of the sub-image which may be selected from a drop-down menu.

6.3.3 Characterisation tab

This tab provides access to a characterisation tool which, under the correct conditions, can automatically determine vital statistics of the performance of the CCD readout. A histogram is accumulated over many frames and is populated with pixel values drawn from a region of interest of the image displayed in the **Live image** window. For accuracy, the frames should be acquired at very low, uniform illumination such that there is an average of less than one photon per pixel per frame; this rate is determined as part of the calculation of the characterisation statistics. In particular, for determination of the gain, low illumination (a *real count rate* in the range of 0.01–0.1 per pixel per frame) should be used and for CIC characterisation no illumination (complete darkness or a closed shutter) is essential.

Before beginning, the **Setup** tab is used to define the limits of the region of interest used for generating the characterisation histogram. This is especially useful for excluding the edges of the CCD which may not have adequate illumination (if required) or noise from readout glitches, which may also appear at the edges of a grabbed image. Figure 6.7 shows the layout of the **Characterisation Setup** tab. The **X1** and **X2** textboxes each accept an integer defining the horizontal range of pixels from which values are drawn for the histogram. Likewise, The **Y1** and **Y2** textboxes each accept an integer defining the vertical range of pixels from which values are drawn for the histogram.

NB: the **CIC** tab cannot be selected until the **Setup** tab has been completed, to provide a non-zero region of interest.

The characterisation histogram is generated dynamically when the **CIC** tab is selected. Figure 6.8 shows the layout of the **Characterisation CIC** tab. The number of frames used to accumulate the characterisation histogram is shown next to the **Frames** label and the contents of the histogram can be erased, resetting the calculation of the statistics by selecting the **Reset** button.

Once the histogram has begun to accumulate, the results of the automatic interpretation of the histogram are displayed in the **CIC** tab :

Noise The exclusion threshold required to suppress noise (5 standard deviations above the bias).

Bias The average baseline signal level across the image.

Gain The estimated average analogue signal, achieved via electron multiplication, for a single charge carrier accumulated in a CCD pixel.

Counts The total number of charge carriers detected above the noise-exclusion threshold. This is the raw value, containing charge carriers due to photons, CIC from the CCD, CIC from the EM register and secondary counts due to charge transfer inefficiency. By construction, this is insensitive to pile-up: multiple photons arriving in a pixel are indistinguishable from a single charge carrier from the CCD.

Lost The estimated proportion of charge carriers not detected because they are below the noise-exclusion threshold.

Raw rate The number of counts per pixel per frame, given uniform illumination. The value in parenthesis includes the estimate of lost counts: this is an estimate of the *real count rate*.

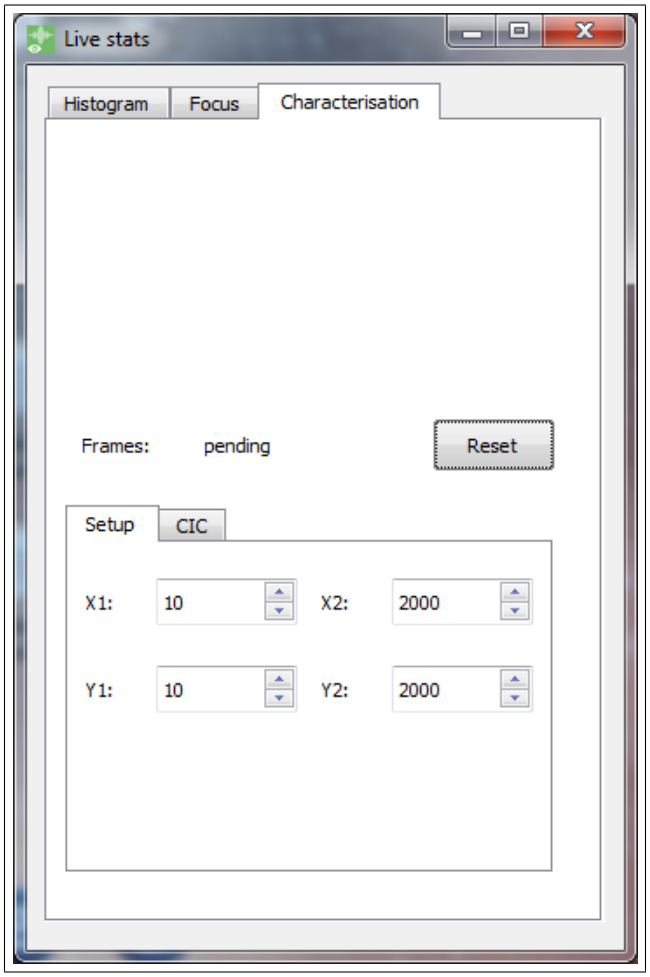

Figure 6.7: Screenshot of the *cccpView* **Live stats** window with the **Characterisation Setup** tab selected

CCD rate The estimated contribution of charge carriers from the CCD to the total count rate, including photons and CIC from the CCD but excluding both CIC generated in the EM register and secondary counts due to charge transfer inefficiency. The value is per pixel per frame. It is a measure of the true level of charge-carriers extracted from the CCD: reducing this to a minimum under totally dark conditions helps ensure that counts are genuinely due to photons accumulated by the CCD.

Bad The estimated proportion of counts that are actually secondary counts, due to charge transfer inefficiency smearing the signal from large-valued pixels.

The histogram itself displays the bin contents with logarithmic scaling and also features three colored lines :

Green This vertical line indicates the level of the noise-exclusion threshold: the **Counts** statistic is computed from entries to the right of this line.

Blue This vertical line indicates the lower limit of a fit applied to the histogram in order to determine the

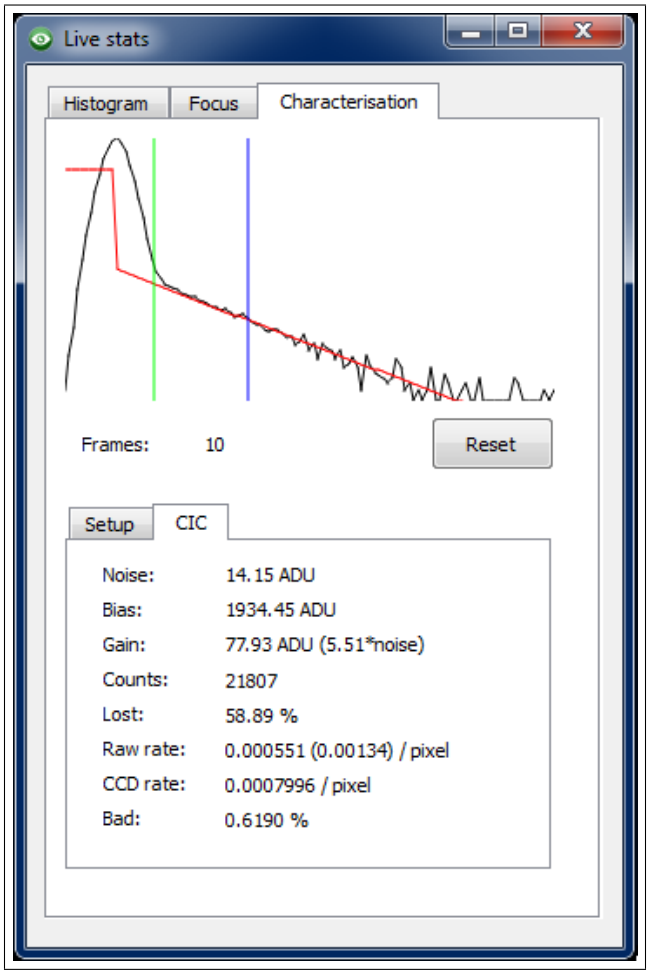

Figure 6.8: Screenshot of the *cccpView* **Live stats** window with the **Characterisation CIC** tab selected

distribution due to multiplied charge carriers in a regime relatively free of bias noise.

Red This sloped line indicates the result of a fit to the histogram applied to the right of the blue line mentioned above and extrapolated down to the bias level: the parameters of this fit are used in the computation of the **Gain**, **Lost** and **CCD rate** statistics.

Chapter 7

Hardware Details

7.1 Analogic Front-end

The Analog Front-End (AFE) is an electronic circuit used for interfacing a sensor with the digital part of the system.

The outputs of the CCD come directly onto either Preamp-EM channel for CCDout0 or Preamp-CONV channel for the CCDout1. For these two channels, the architectures are the sames (Figure 7.1) and only the analog bandwidths change. Depending on which read-out sequence is loaded on the CCD, it is also possible to select a particular bandwidth for each channel with the **svideobw** serial command. The Table 7.1 refers to the available bandwidths for each channels.

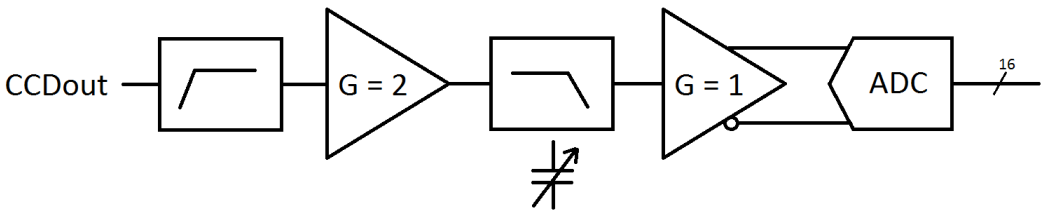

Figure 7.1: AFE overview of the CCCP.

CCCP does not buffer the pixel's data once they come out of the ADC. These data must immediately be handed to the ADC de-serializer interface for the CDS processing. The timing of the ADC and the FPGA ADC data de-serializer are thus interlocked. To synchronize correctly ADC data with the ADC clock, the user has to generate the ADC clock by using the **ads5263clk** serial command. For better performances, it is also recommended to generate this clock at a frequency of 120MHz.

If the bandwidths shown in Table 7.1 or the gain of these amplifiers is to be changed, please contact Nüvü Camēras.

Table 7.1: Possible bandwidths for the two digitalization channels

sel	EM	CONV
0	40MHz	6MHz
1	20MHz	3MHz
2	10MHz	500kHz
3	6MHz	6MHz

7.2 Resonant HV clock

In order to clock the EM register of an EMCCD, a high voltage clock must be used (25–50 V, depending on the EM gain). On the cccpv3 controller, this clock is generated with a resonant circuit, where the capacitance of the EM register resonates with an inductance on the controller board. This resonant system is driven by the digital signal HV_CLK1 (bits [4:7] of phase 14 in Table 2.2), the VHVL1 bias (bias 12 in Table 2.4) which controls the low level of the clock and the VHVH1 bias (bias 4 in the Table 2.4) which controls the high level of the clock. Once the digital signal HV_CLK1 is activated, the resulting HV clock will be sinusoidal. The resonant system takes about 40 µs to settle. Thus, one may choose to start the HV clock at the beginning of the readout of the CCD and leave it running while doing the vertical transfers, even if the HV clock is not needed during that time. The HV clock can be stopped during the integration periods. **When the HV clock is not used, the digital signal HV_CLK1 should be left high (digital value of '1'). HV_CLK1 should not be taken low for an extended period of time, as damage to the controller could occur.**
The frequency of the digital signal HV_CLK1 defines the frequency of the output HV clock, but as the controller is shipped, the resonant system on the controller is tuned to operate at either 10MHz or 20MHz[1]. The delay of the digital signal within the pattern may require tuning by the user so that the resulting HV clock is in sync with the read-out to clock the EM register. By adjusting the delay of the digital signal, the user will also adjust the delay of the resulting high voltage clock. This behavior is shown on Figure 7.2.
The HV clock has a soft-start mechanism that prevents it from overshooting upon start-up. However, if the HV clock is stopped and re-started less than 10 µs after, it may overshoot (Figure 7.3). If the HV clock is to be stopped for less than 10 µs, it should be kept active. In all circumstances, the overshoot will be for less than 8 µs. In most circumstances, the overshooting will be limited to 50 V, which is the maximum absolute rating of e2v chips. However, depending on the resonating frequency and EMCCD used, the overshoot could be beyond 50 V. Thus, repeated overshooting is not recommended as it could accelerate the ageing process of the EMCCD.

7.3 Clock drivers

The power amplifiers producing the clock outputs are model THS3091 from *Texas Instruments*. These current feedback amplifiers have a high slew rate (7300 V/µs), high bandwidth (210 MHz), and high sustained current output (250 mA). They have a thermal pad soldered to a ground plane to allow the device to dissipate heat. The schematic of the power amplifiers is shown in Figure 7.4.

In this figure, a THS3091 and an AD8002 from *Analog Devices* are shown. The AD8002 takes the differential 0–20 mA signal from the DAC and converts it into a ± 2 V single-ended signal. This signal is then

[1] The resonant frequencies are listed in the Certificate of Conformity shipped with the controller.

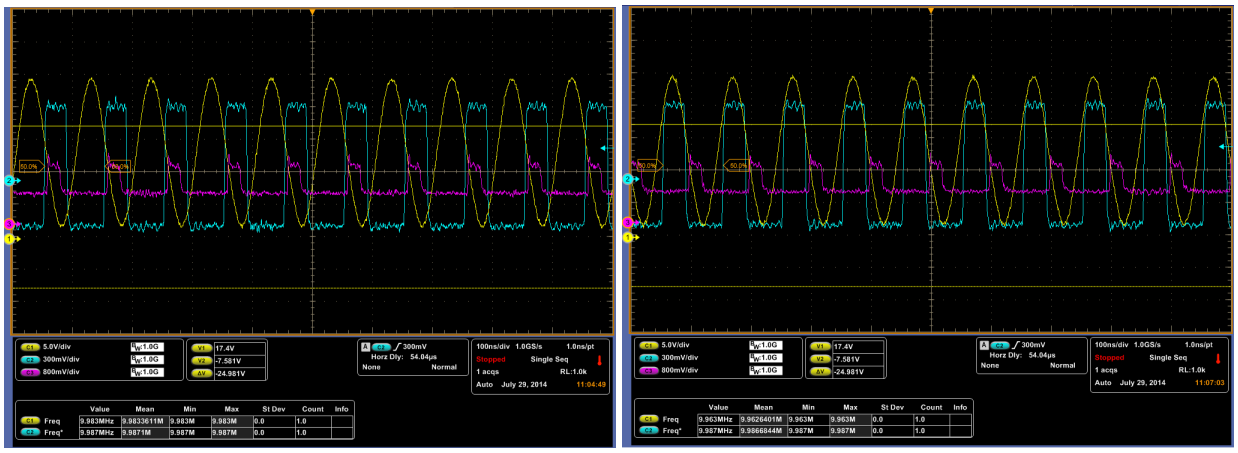

Figure 7.2: HV clock synchronisation within a pattern. The pink trace is the high resolution digital signal HV_CLK1, the yellow trace is the resulting high voltage clock from the resonant system and the blue trace is the trace with which we want to synchronise the high level of the HV clock. Adding the appropriate delay on the digital signal, it is possible to synchronise the high level of the HV clock with the high level of the blue trace, as seen on the figure on the left.

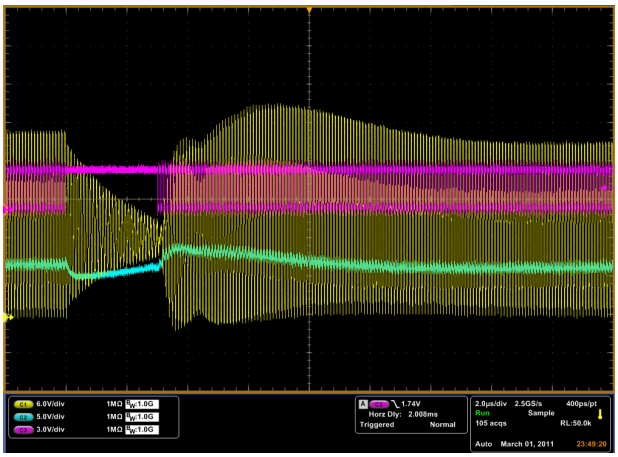

Figure 7.3: Overshooting of the HV clock that occurs if it is stopped and re-started in less than a 10 µs time interval. To prevent overshooting, avoid stopping the HV clock for less than 10 µs. The yellow trace is the HV clock and the pink trace is the digital signal of the HV_CLK1. One can see that the HV_CLK1 digital signal stops for 3 µs and the HV clock overshoots when it starts back.

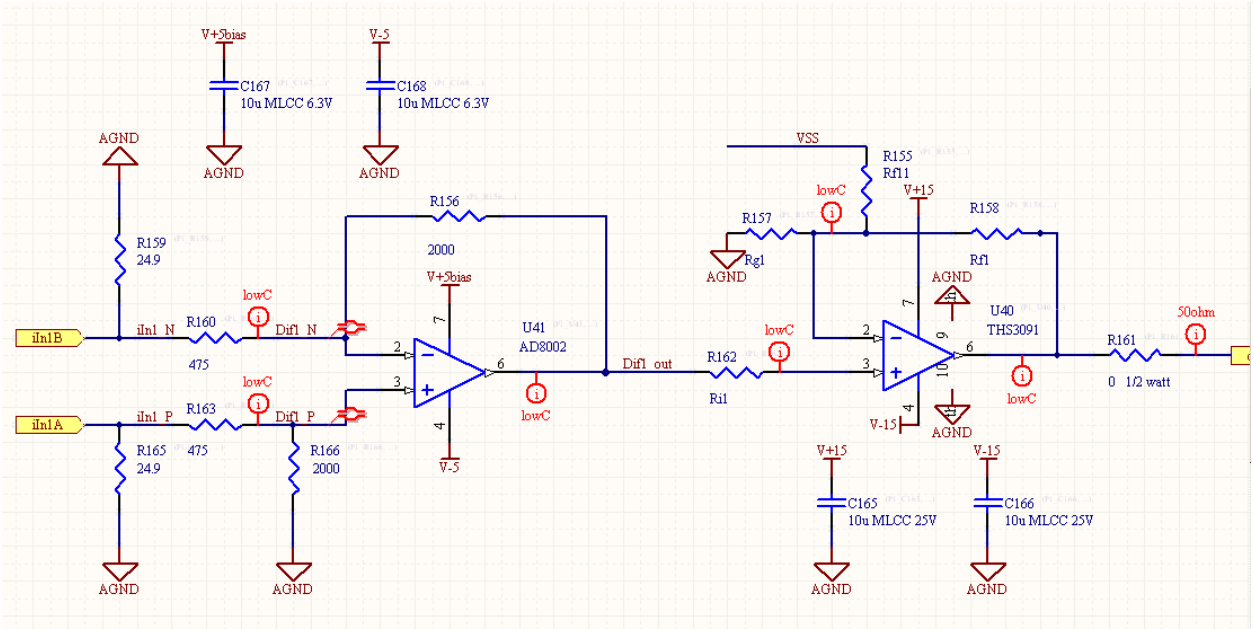

Figure 7.4: Schematic of the power amplifiers.

amplified by the THS3091 to drive the CCD pin. In virtual VSS operation, the output level of the THS3091 is negatively offset by the value of VSS. The $\mathrm{R_f}$ and $\mathrm{R_g}$ values for the THS3091 are given in Table 7.2. Note that $\mathrm{R_{f11}}$ is always equal to $\mathrm{R_{f1}}$ to yield a gain of −1 on VSS. On a current feedback amplifier, the feedback resistor ($\mathrm{R_{f1}}$) sets the bandwidth of the amplifier. The bandwidth for every phase is given in Table 2.3. If the bandwidth or the gain of these amplifiers is to be changed, please contact Nüvü Camēras.

Even though these amplifiers can dissipate a lot of power (about 2.2 W at 25 °C), care must be taken when clocking the CCD, especially when it comes to the blanking sequence. Most of the time, the blanking of the CCD involves clocking its vertical clocks while holding a dump gate in a high state. The vertical clocks are typically of high capacitance and it requires a high power to clock them. Even though the vertical clock drivers can continuously clock the vertical clocks for a moderate period of time (e.g. the time needed to do a few transfers from the image to the storage section of a CCD), it is not recommended to clock them continuously for an extended period of time. As an example, clocking the vertical clocks of a CCD97 with a 3 MHz sinusoidal clock of 12 V peak-to-peak requires the dissipation of ∼2.5 W by the power amplifier (see Figure 7.5). This exceeds the sustained power dissipation capability of the THS3091. Thus, if a blanking pattern that may run for an extended period of time would use these waveforms, **damage could occur to the power amplifiers**.

The way to overcome this effect is to add some wait-state to the clocks in a blanking pattern. Figure 7.6 shows an example of a 3.3 MHz (30 ns run time) vertical clock used for blanking. By using a 60 ns delay after the clock, the power dissipation of the THS3091 is lowered by a factor of 3. The clock cycles can keep the same frequency component as a continuously running clock (which is important to keep the clock induced charges to a low level) while ensuring the power amplifier will not overheat.

Table 7.2: Configuration of the power amplifiers resistors

Phase	R_{f1} (kΩ)	R_{g1} (kΩ)	Gain
Phase0	1.21	0.402	8.65
Phase1	1.21	0.402	8.65
Phase2	1.21	0.402	8.65
Phase3	1.21	0.402	8.65
Phase4	1.21	0.402	8.65
Phase5	1.21	0.402	8.65
Phase6	1.21	0.402	8.65
Phase7	1.21	0.402	8.65
Phase8	1.21	0.402	8.65
Phase9	1.21	0.402	8.65
Phase10	1.21	0.402	8.65
Phase11	1.21	0.402	8.65
Phase12	1.21	0.402	8.65
Phase13	1.21	0.402	8.65

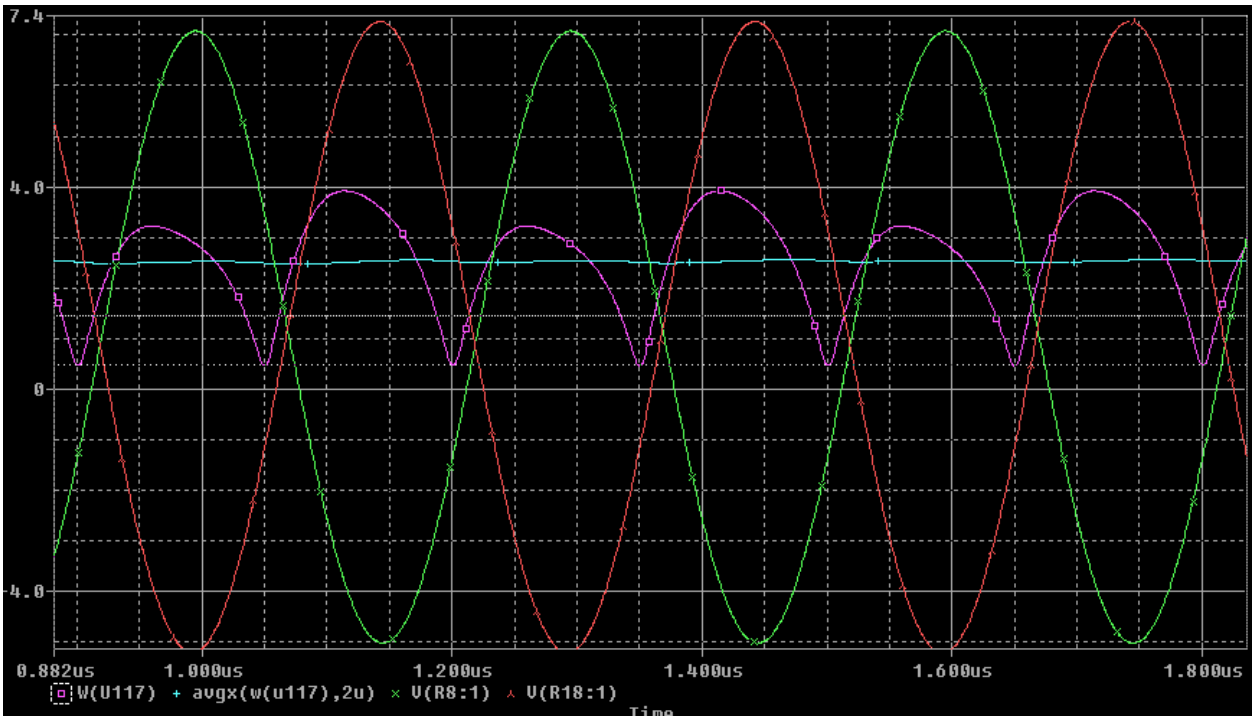

Figure 7.5: Simulation of the power amplifiers. The green and red traces show the outputs of two THS3091, these outputs are two 3 MHz sinusoids that have a 180˚ phase difference. These signals are as they would be on IΦ1 and IΦ3 of a CCD97. The purple line shows the instantaneous power dissipation of one THS3091. The cyan line shows the power dissipation of the same THS3091 averaged on a 2µs period.

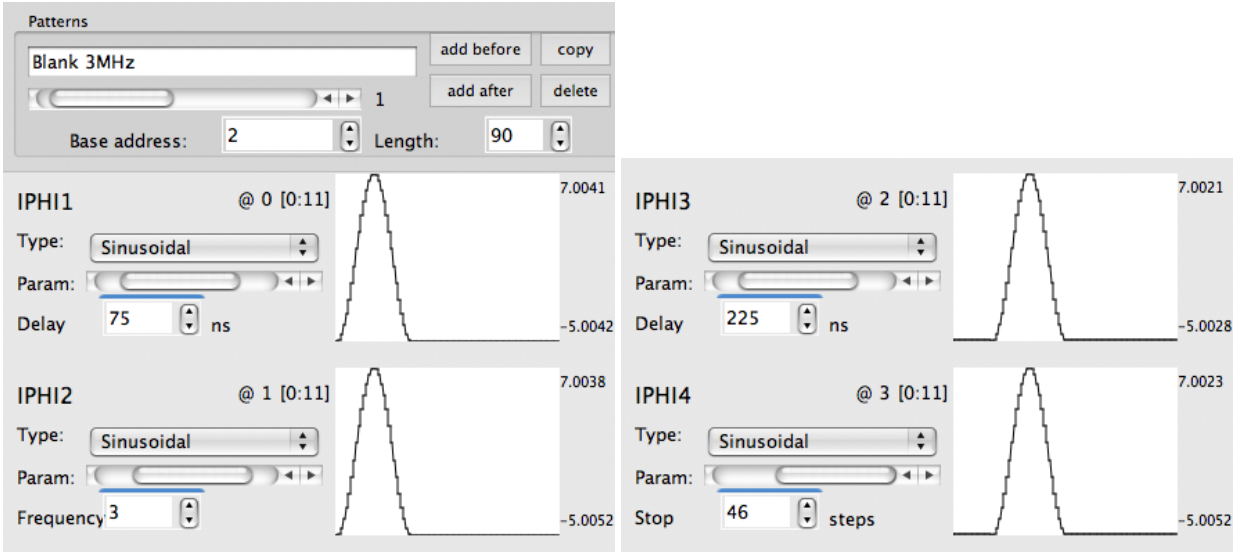

Figure 7.6: Example of a blanking pattern that uses a 3 MHz clock with a wait-state to lower the power dissipation of the THS3091.

7.4 Biases

The biases are generated with OPA4277 amplifiers connected to a 14 bits DAC that provides a voltage between 0 and 5 V. These amplifiers have a low offset, a low temperature drift, and can be powered with up to 36 V supply voltages. Each bias, as enumerated in Table 2.4, is generated with a circuit like the one shown in Figure 7.7. In all cases, the $\mathrm{R_l}$ resistor is 10 kΩ to prevent the bias from floating when powering up the controller. This insures that the bias stays at 0 V prior the initialization of the DAC.

The biases are designed to be decoupled with a 10 µF and 100 nF ceramic capacitors at the CCD pin. The 10 Ω resistor isolates the feedback circuit from the phase shift induced by the capacitive load. At high frequency, the feedback is mainly from the 220 nF capacitor. This further isolates the op-amp from the phase shift caused by the decoupling capacitor at high (≥100 kHz) frequency or transient coming from reflexion at the other end of the transmission line connecting the bias. Having placed the feedback resistor after the isolation resistor further enhances the stability of the bias when the current drawn by the CCD varies.

The $\mathrm{R_f}$ and $\mathrm{R_g}$ resistors, together with the $\mathrm{V_+}$ and $\mathrm{V_-}$ supply voltages vary from bias to bias. For every bias, the exact values of these parameters are given in Table 7.3.

7.5 Connector

The edge-mount connector located on one end of the detector board allows the construction of a custom connection board to connect to the CCD or to an hermetic connector. The connector on the controller board is a QTS-075-01-L-D-RA-WT from *Samtec*[2]. It can mate with either the surface-mount or edge-mount members of the QSS-075 connectors family[3]. The pinout of the connector located on the detector board is

[2] `http://www.samtec.com`
[3] `http://www.samtec.com/technical-specifications/Default.aspx?seriesMaster=qss`

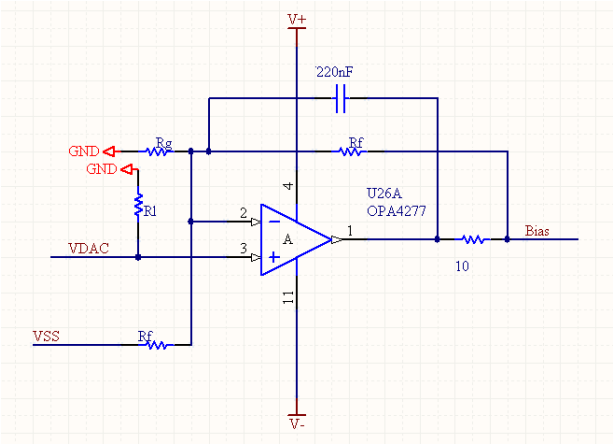

Figure 7.7: Schematic of a bias driver.

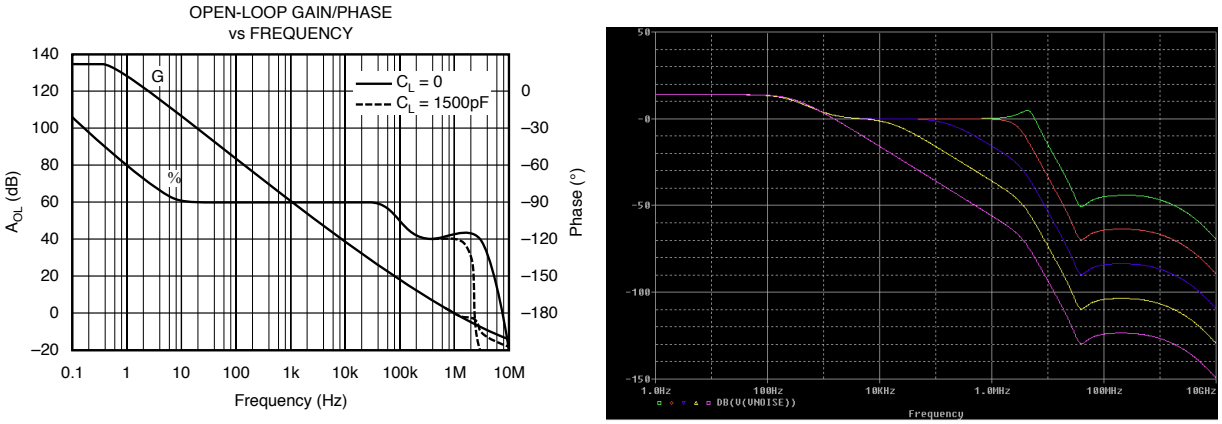

Figure 7.8: Frequency response of an OPA4277. **Left**: Open loop, taken from the OPA4277 specifications. **Right**: Closed loop, for a circuit as the one of Figure 7.7. The various plots (from green to pink) are for a decoupling capacitor of 1 nF, 10 nF, 100 nF, 1 µF, and 10 µF, respectively. It shows that decoupling capacitor should be at least of 10 nF to avoid the natural resonance at ∼2 MHz.

Table 7.3: Configuration of the biases.

Bias	R_f (kΩ)	R_g (kΩ)	V_+ (V)	V_- (V)
Bias0	3	1	+30	−5
Bias1	3	1	+30	−5
Bias2	3	1	+30	−5
Bias3	3	1	+30	−5
Bias4[1]				
Bias5	3	1	+30	−5
Bias6	4.02	4.02	+15	−15
Bias7	4.02	4.02	+15	−15
Bias8[2]	4.42	2.21	+15	−15
Bias9[2]	4.42	2.21	+15	−15
Bias10[1]				
Bias11[1]				
Bias12[1]				
Bias13	4.02	1	+30	−5
Bias14	4.02	1	+30	−5

[1] These biases are used internally to control the high and low levels of the HV clock.
[2] These biases have an inverting gain configuration that differs from the one show in Figure 7.7.

shown in Figure 7.9.

7.6 Design guidelines for the cryostat interface

EMCCDs are meant to be used at a high pixel rate. When connecting the controller to the EMCCD through the hermetic connector, great care must be taken when designing the transmission lines that will carry the signal from the clock driver to the CCD pin. The transmission delay and the self inductance of the transmission line must be considered. Figures 7.10 and 7.11 show the effect of the transmission line on both vertical and horizontal clocks.

In order to minimize the inductance of the transmission line, printed circuits boards (rigid or flexible) should be used whenever it is possible. They should have at least one plane of ground placed near the signal layers to allow a quick return path for the current, hence low inductance. If discrete wires must be used, twisted pairs with one wire connected to the ground are highly recommended for every clock.

In order to minimize the propagation delay, the controller should be hooked as close as possible to the hermetic connector. The propagation delay should be maintained below 2 ns (see the effect of the propagation delay in the top left panel of Figure 7.10). As an example, a 18 cm PCB trace and/or twisted pair yields a 1 ns of propagation delay (assuming a propagation at 0.6 c). It is highly recommended to solder the hermetic connector directly to a PCB connected to the controller with the edge-mount QTS-075 connector. Provision for damping resistors should be placed on this PCB in series with the signal path of every clock (see the effect of the damping resistors in Figures 7.10 and 7.11). The damping resistors should be rated at least 0.5 Watt and be surface-mounted to reduce their self inductance.

The capacitance and resistance of the CCD clock must also be considered. For example, the CCD97 RΦ2 has a 75 pF capacitance and a 6 Ω serial resistance. This creates a low-pass filter that will dampen the

rise and fall time of the clock.

The figures below show simulation results of a THS3091 driver hooked to a CCD pin. Square clocks are used, which are the worst case scenario as they have high frequency components. When using the controller, since the clocks can be shaped, better results should be achieved. Moreover, the arbitrary clocks of the controller allowsone to pre-distort the signal generated by the controller to lessen the overshoots and undershoots generated by the effects of the transmission line. These simulations do not show the effect of the inter-capacitance between two clocks (IΦ2 and IΦ3, for example). In order to minimize this effect, the lowest possible value should be used for the damping resistor.

Regarding the biases, the inductance of their signal path, although less critical than the one of the clocks, must still be minimized. By using ground planes into the PCBs and decoupling every bias near the CCD pin with a 10 µF and a 100 nF ceramic capacitors, this should be enough.

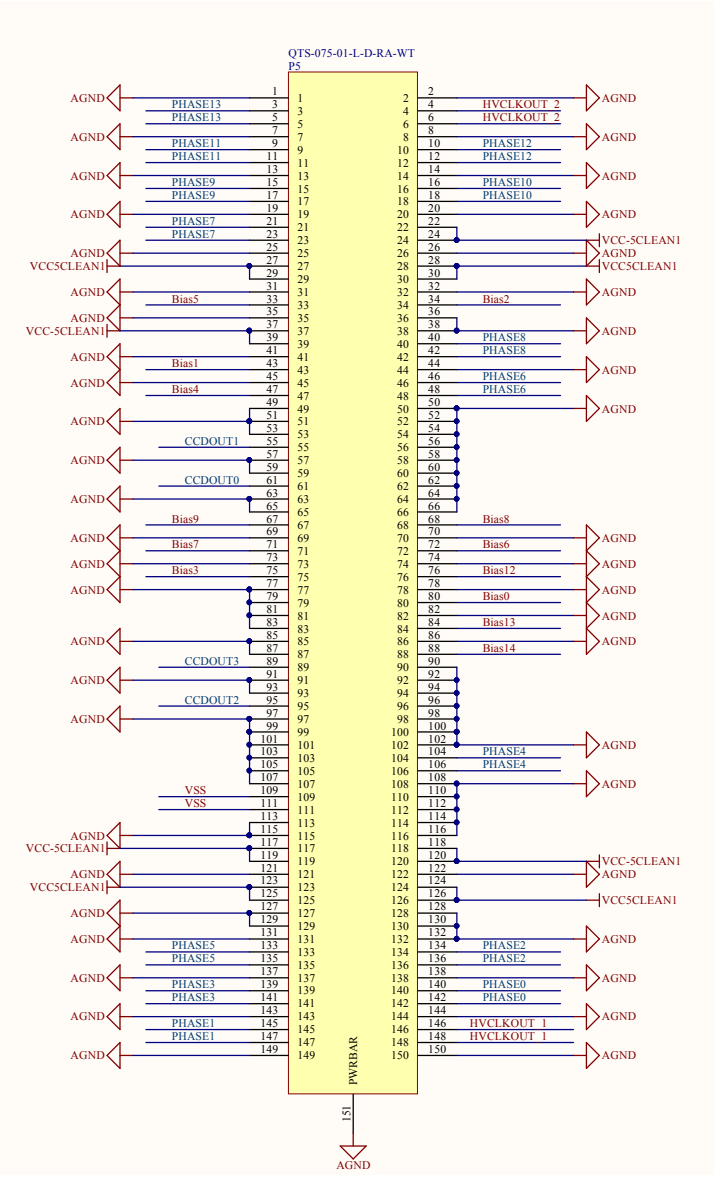

Figure 7.9: Schematic of the CCD connector.

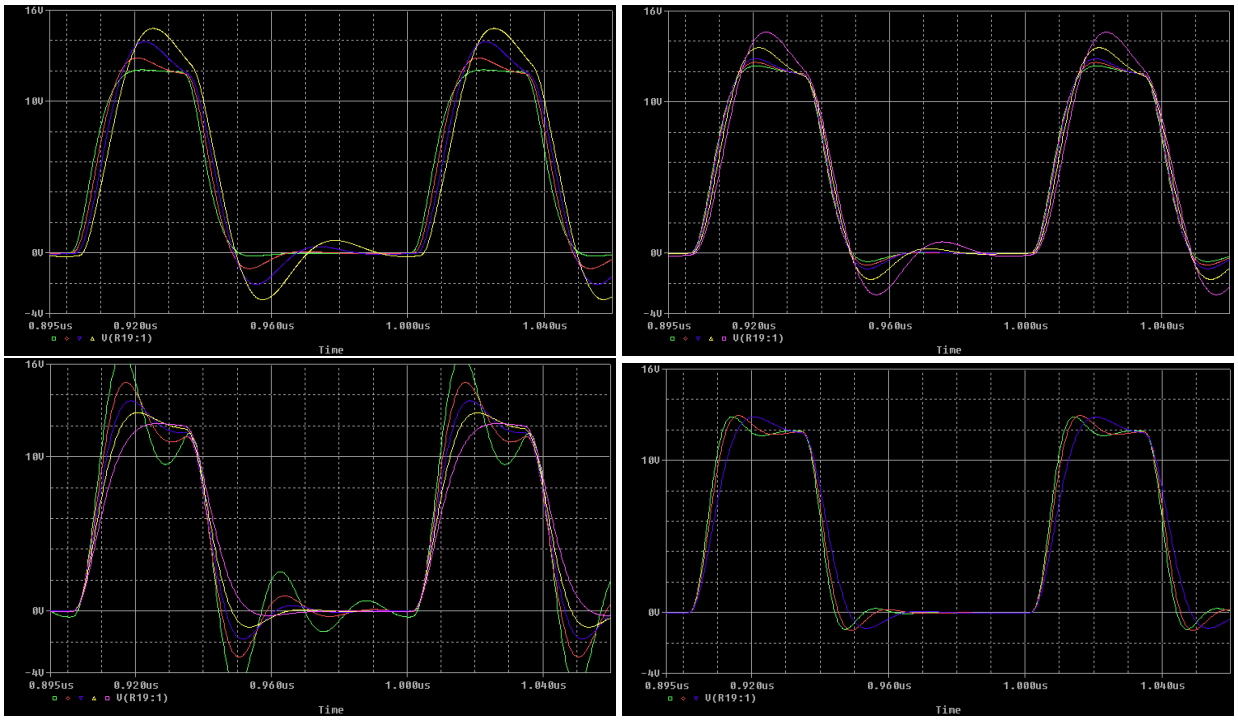

Figure 7.10: Effects of the transmission line between the clock driver and the CCD source on the horizontal clock shape. The simulations uses a square clock of 3 ns rise and fall times, 12 volts of amplitude, 10 MHz and 30 % duty cycle into a 175 pF load in series with a 6 Ω resistor, with a 1 ns propagation delay, a 20 nH self-inductance of the transmission line, and a 15 Ω serial damping resistor. **Top left**: Effect of the propagation delay. Plotted values are for 0.2, 1, 2, and 3 ns. **Top right**: Effect of the self inductance of the transmission line. Plotted values are for 0, 10, 20, 50 and 100 nH. **Bottom left**: Effect of the damping resistor. Plotted values are for 0, 5, 10, 15 and 24 Ω. **Bottom right**: Effect of the load. Plotted values are for 75, 100, and 175 pF.

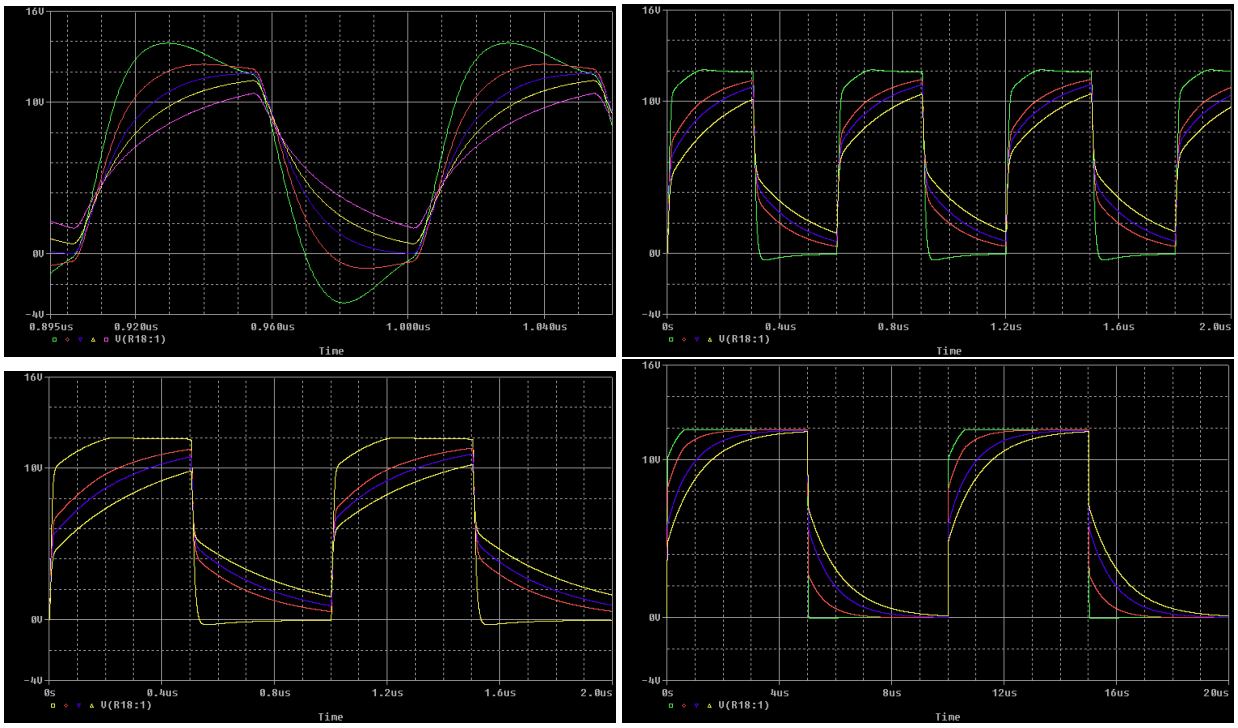

Figure 7.11: Effect of the damping resistor on vertical clocks. In all cases, a transmission line with 20 nH self inductance and 1 ns propagation delay is used for a clock of an amplitude of 12 volts and 50 % of duty cycle. The plotted values are for values of 0, 10, 15 and 24 Ω. **Top left**: A 10 MHz clock with 20 ns rise and fall times, into a 900 pF load in series with a 9 Ω resistor. This load represents the load of the vertical clocks of the CCD60 at the maximum recommended parallel shifting frequency. An extra plot for a damping resistor of 5 Ω is shown (red curve). **Top right**: A 1.6 MHz clock with a 3 ns rise and fall times, into a 5.3 nF load in series with a 17 Ω resistor. This load represents the load of the most capacitive vertical clocks of the CCD97 at the maximum recommended parallel shifting frequency. **Bottom left**: A 1 MHz clock with 3 ns rise and fall times, into a 10.2 nF load in series with a 14 Ω resistor. This load represents the load of the most capacitive vertical clocks of the CCD201 at the maximum recommended parallel shifting frequency. **Bottom left**: A 100 kHz clock with 3 ns rise and fall times into a 30 nF load in series with a 16 Ω resistor. This load represents the load of the most capacitive vertical clocks of the CCD207-40 at the maximum recommended parallel shifting frequency.

Chapter 8

Mechanics

Figure 8.1 provides a view of the housing of a single-channel CCCP version 3.

8.1 Operating and storage conditions

Operating temperature 0˚C to 35˚C ambient, relative humidity <80% (non-condensing). Storage temperature -25˚C to 55˚C.

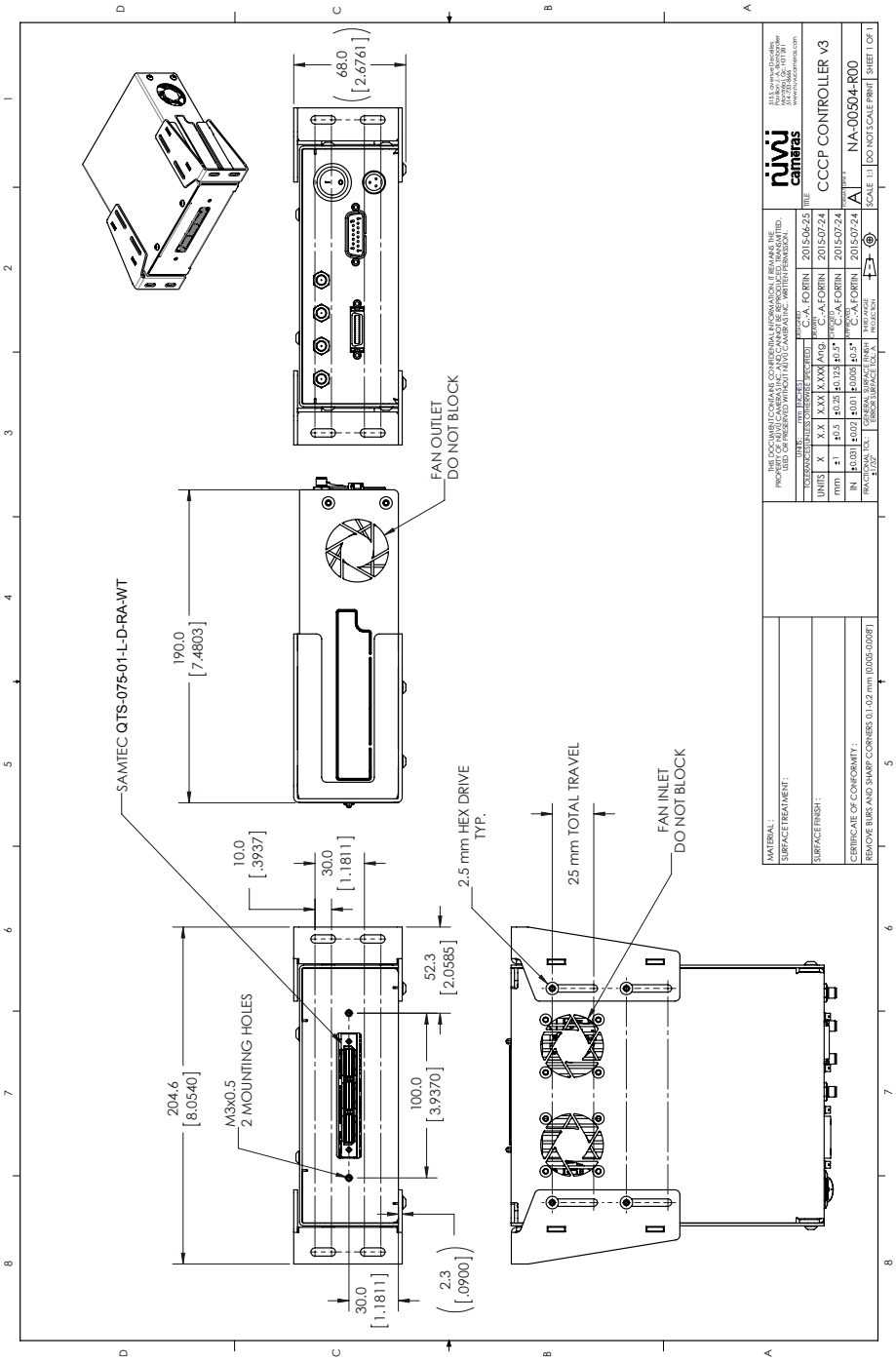

Figure 8.1: Views of the housing of a CCCPv3.

Chapter 9

Warranty

Nüvü Camēras warrants for twelve (12) months all products manufactured by Nüvü Camēras, unless noted in the Certificate of Conformity. Nüvü Camēras warrants all material free from original manufacturing defect. This warranty does not cover any products that have been subject to neglect, misuse, negligence, or accident. This warranty does not apply to any damage to products that is the result of improper installation or maintenance, or to any products that have been operated or maintained in any way contrary to the operating or maintenance instructions as specified in Nüvü Camēras' documentation.

The CCD Controller for Counting Photons (CCCP) is a precision instrument that has been optimized for use with a particular type of CCD. User modification of the controller to adapt it for use with another type of CCD may cause unpredictable results, and will void the warranty. The user is strongly advised to contact Nüvü Camēras before attempting any modifications. Nüvü Camēras will not be responsible for damages resulting from user modification, such damages including, but not being limited to, damages caused to other devices connected to the controller or for damages resulting from incorrect use of the controller. It is the responsibility of the user to follow the operating guidelines provided with the controller, and to control the operation of the CCCP in a manner that respects the limitations of other devices.

Nüvü Camēras is not responsible for any defective goods returned without a previous written arrangement with our after sales service. Nüvü Camēras will accept returned products only in their original condition.

EXCEPT AS EXPRESSLY SET FORTH IN THIS LIMITED WARRANTY AND TO THE GREATEST EXTENT ALLOWED BY LAW, NÜVÜ CAMĒRAS MAKES NO OTHER REPRESENTATIONS, WARRANTIES OR CONDITIONS, EXPRESSED OR IMPLIED, INCLUDING ANY IMPLIED REPRESENTATIONS, WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, AND NON-INTERFERENCE. ANY IMPLIED WARRANTIES THAT MAY BE IMPOSED BY LAW ARE LIMITED IN DURATION TO THE LIMITED WARRANTY PERIOD, TO THE GREATEST EXTENT ALLOWED BY LAW.