TIQI Quantech Workshop
32-Channel Arbitrary Waveform Generation for Trapped-Ion Electrode Control

Paul Venetz, Michael Marti, Benjamin van Ommen, Marco Stucki
Supervised by Martin Stadler and Vlad Negnevitsky

September 16, 2020

Abstract

Quantum information processing requires a large number of controllable quantum bits (qubits) to allow meaningful real world applications. For trapped ion quantum computers, one of the requirements for independently controlling a large number of qubits (ions) is the availability of three to four times as many trapping electrode voltages, and the capability to scale this number up easily.

The focus of this project is the testing and integration of the Fastino, a 32-channel Eurocard-sized device which is part of the Sinara Open-Source Hardware Project, into the existing control system for ion trap experiments of the Trapped Ion Quantum Information (TIQI) group at ETH Zürich. In addition to the obvious advantage of the number of output channels, other reasons to implement this device into the setup are compactness, ease of setup and maintenance, and voltage stability in comparison to the custom made AWG board currently in use, while retaining similar noise level.

The project encompassed design and manufacturing of a new Printed Circuit Board (PCB) to allow interfacing between current experimental hardware and the new Fastino, and writing Hardware Description Language (HDL) to handle the data-flow from a host-PC to the DACs. The final version allows for sending waveforms from a host-PC via USB or Ethernet to the control FPGA, where it is saved in memory and continuously output to the Fastino's DACs. This paves the way for controlling hundreds of electrodes in the ion traps of the TIQI group.

Contents

1 Introduction

In the field of quantum information processing, progress is made every day on different limiting factors and on different platforms. A primary concern is scalability in the number of qubits that can be precisely controlled. This project was started to improve this scalability in the Trapped Ion Quantum Information (TIQI) Group at ETH. Apart from the scalability, the TIQI group was looking for DACs which have a better voltage stability and thus can tightly control electrodes in order to avoid heating or losing the ions.

Trapped ion quantum computing uses lasers to manipulate the motional and internal electronic states of ions that are trapped by a rotating quadrupole and static electric fields, produced by electrodes. A road map for scaling up these ion traps to a large-scale ion-trap quantum computer is the "quantum CCD" [12], which relies heavily on a large number of trapping electrodes to move ions between memory and interaction regions, recombining and splitting the ions in these regions.

In this project the goal is to replace existing experimental hardware, which features four DACs per board, with a new, externally developed board, called "Fastino". The Fastino [10] ecosystem for trapped-ion hardware hosts 32 DACs (on a single PCB), and four of these Fastinos can be controlled by one Xilinx MicroZed, which normally controls four DACs, amounting to a potential 8-fold increase in DAC number. The Fastino is part of the ARTIQ (Advanced Real-Time Infrastructure for Quantum physics) control system, however since most of the TIQI hardware is developed in-house some work needs to be done to integrate the Fastino.

1.1 State of the Art

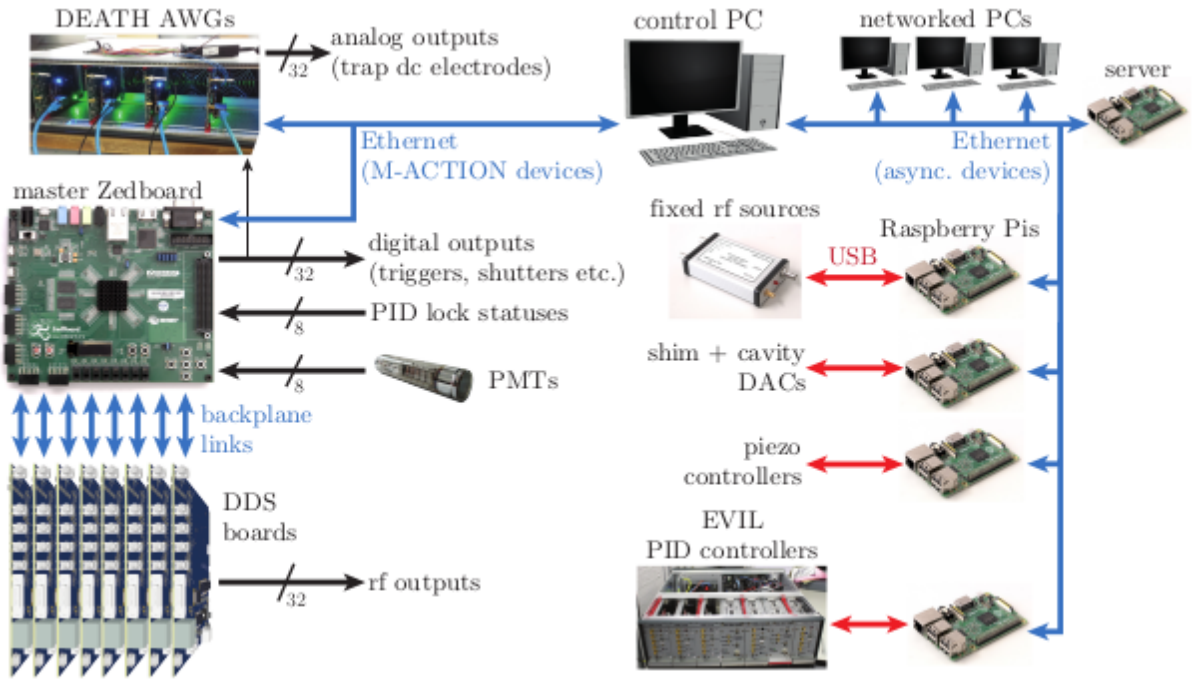

Figure 1: Experimental control in the mixed-species setup. [7]

DEATH (Direct Ethernet Adjustable Transport Hardware) is the current board used to create the arbitrary waveforms in the TIQI lab. It fits in a broader control system depicted in figure 1. Control over the DEATHs is executed from a control PC and a Zedboard interfacing with the FPGA via Ethernet. The DEATH board features 4 DAC channels per board. It is a custom design by the TIQI group and is described in detail in Ludwig de Clerq's thesis [3]. At the moment at most ten DEATHs are used in a single trapping setup to dynamically control electrodes.

The DEATHs are connected to the control PC and to the master Zedboard via Ethernet, and they can also be triggered by the master Zedboard via a separate TTL pulse. The DEATHs receive their

waveforms over the Ethernet connection and a digital line from the Zedboard triggers them to output the waveforms to the respective DACs. [7]

As one can see, the DEATHs fit into a larger experimental control system, and any future replacement would have to be able to fit into this system. Broadly speaking, a replacement should be remote controlled, easily scaled-up and maintained, while staying flexible enough to allow quick reconfiguration. It goes without saying that devices should also be stable and reliable[1].

In figure 2, a top-down view of a DEATH board (green) with a MicroZed (red) on it is shown. The DEATH board provides the MicroZed with 4 DAC channels. To develop our MicroZed-to-Fastino PCB we adapted the current DEATH design, removing the DACs and adding connectors for the Fastinos.

The SoC (System on Chip) on the MicroZed formerly used with the DEATH is a Xilinx Zynq 7010, which contains two ARM core A9 processors and an FPGA. This design has become very popular, as the FPGA firmware can communicate with the processors which can be programmed in C/C++. Embedded in the FPGA fabric is 240kB of Block Random Access Memory (BRAM) which would allow for storage of over 16 waveforms with each including 1000 steps for the four independent analog outputs. [3]

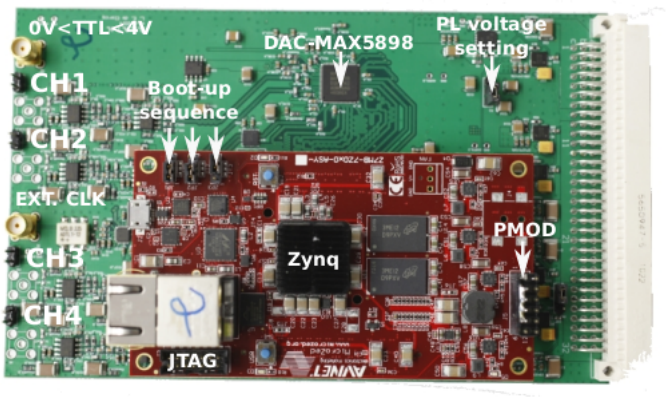

Figure 2: Top-down view of the DEATH with the MicroZed attached. The second DAC is located under the MicroZed board. [3]

At the time the original DEATHs were designed, the important criteria used for the DAC choice were high update rate, low noise, and good voltage stability [3]. For this reason the MAX5898 [6] by Maxim was chosen.

The MAX5898 is a dual channel, 16 bit DAC and updates the output voltage at up to 500MS/s. The data input is a single parallel LVDS bus with a width of 16-bits and a clock speed of 125 MHz (DDR). This means only 125 MS/s are sent to each channel and 4-times interpolation has to be applied to arrive at the aforementioned sampling rate.

1.2 Reason for Upgrade

As already implied at the beginning of the introduction, at a certain point increasing the number of controllable qubits in a trap becomes inevitable if one wishes to implement more sophisticated quantum protocols. In fact, to implement anything useful one needs at least 50+ high fidelity qubits. To control more qubits, we need more electrodes to trap more ions, and thus we need more DACs.

State of the art is four channels per DEATH board, with four DEATHs per rack adding up to 16 channels per rack. This suffices to control traps with tens of electrodes, but this quickly becomes infeasible when thinking about hundreds of electrodes per trap. Therefore, a replacement board should feature significantly more than the current four DACs per board.

Some conditions that were imposed on the DEATH DACs can also be relaxed a little, now that more experimental experience has been gained. First of all, the cutoff frequency of the DACs in the current setup is lowered to around 100 kHz to keep noise to acceptable levels (down from a planned 500 kHz). Secondly, the maximum sample rate was rarely used in real-world waveforms [2]. This would thus allow for DACs with a much lower sample rate to be used.

[1] V. Negnevitsky, private correspondence
[2] V. Negnevitsky, private correspondence

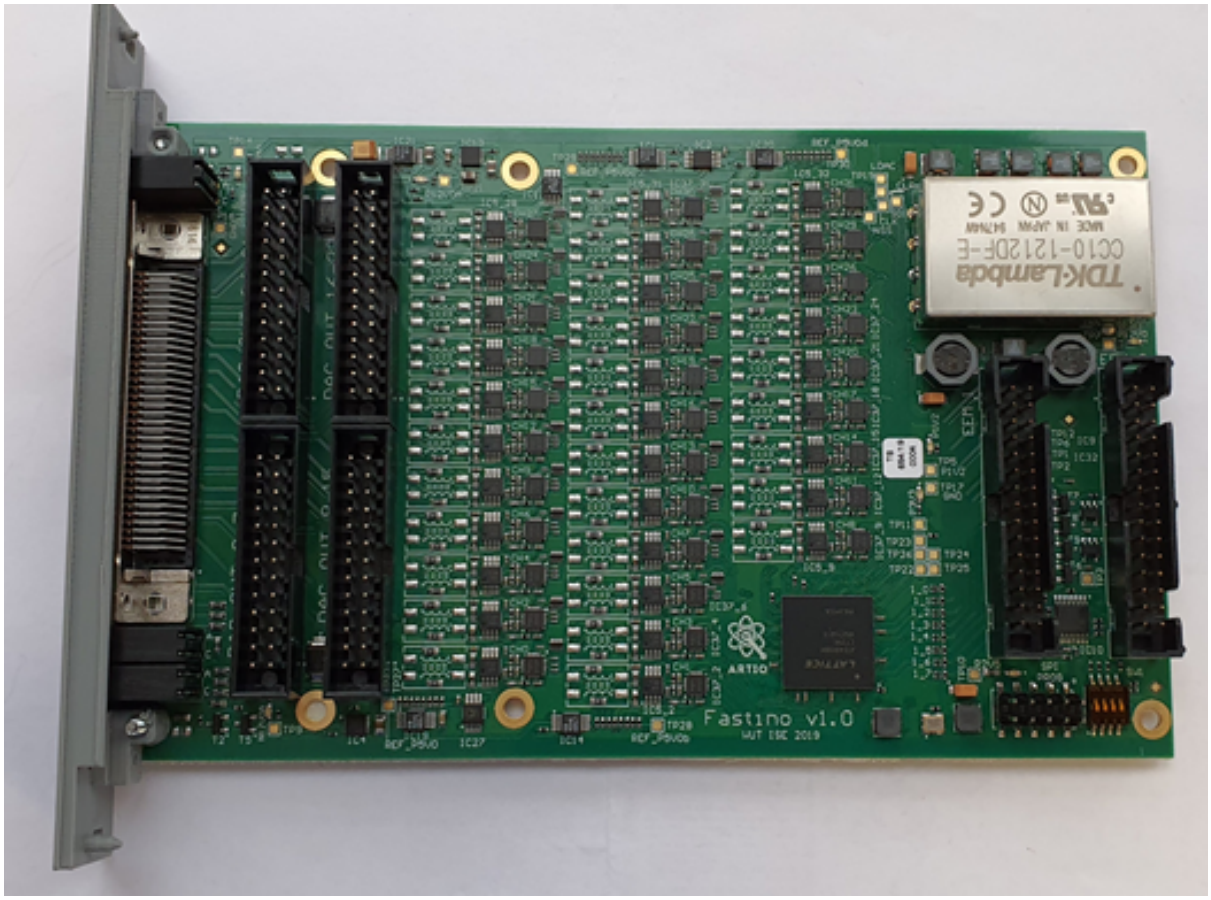

Figure 3: A top-down view of the Fastino. On the left hand one has the DAC outputs as 4 headers and as a SCSI connector. On the right hand side there are 2 headers to connect to control hardware. One header features 8 LVDS pairs, a 12V and 3.3V power rail, and an I2C bus.

The aforementioned Fastino meets the criteria quite well. Using it will provide 32 DACs per rack slot with the only overhead being the MicroZed on the break-out board we have designed, which would take up one rack slot itself too.

> The DEATHs have served us well, however there are a few limitations that we want to overcome. This is where you can help us out!
>
> (Vlad Negnevitsky)

1.3 New Hardware - Fastino

The Fastino (figure 3 is a eurocard-sized 32 channel AWG (Arbitrary Waveform Generator). As mentioned before it is part of the sinara ecosystem for trapped-ion hardware. It features 2 input headers, 32 DACs with 32 output stages, on-board power supplies and DAC voltage references, it has its own FPGA to forward data to the DACS and there is an I2C bus for temperature monitoring and FPGA flashing. There are also 8 user LEDs on the front, and 4 status LEDs.

Input The Fastino features two input headers. Each input header has power rails (3.3V and 12V), 8 LVDS pairs for in or output, and an I2C bus. The power rails provide all the power the board needs. In our design 1 LVDS pair is used as a clock for the FPGA, 6 pairs are used as data input (carrying the DAC data), and one pair is used as a return line to report errors that might occur on the Fastino.

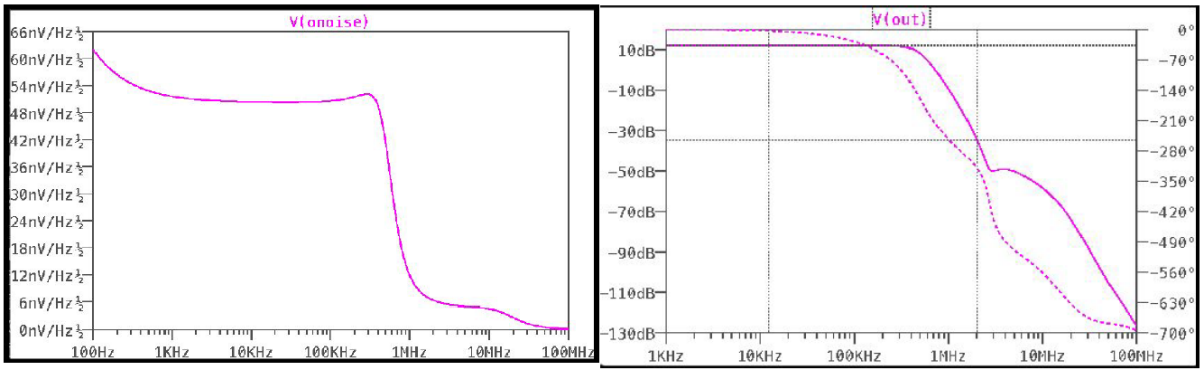

Figure 4: Noise spectral density and transfer of the output stages on the Fastino as provided by the manufacturer [10]

DACs and output stages The Fastino features 32 AD5542ABCPZ DACs. In the next section these DACs are compared to the DACs that were present on the old DEATH boards. The outputs of these DACs are connected to two opamps (OPA2197IDGKR). The first opamp provides 4-times amplification of the DAC output, and the second works as low-pass filter and buffer. The output range is -10 to +10 Volt. In figure 4 the transfer and output noise as a function of frequency are shown, as provided by the designers. Since we did not have access to a spectrum analyzer during this project we were not able to verify this.

The voltage outputs are at the pins on the side of the board as seen in figure 3 and also on a SCSI connector on the front of the board.

Power Supply The Fastino features its own power supply, which converts the incoming 12 Volts to the voltages required for the FPGA (2.5, 3.3 and 1.8 Volts) and to the voltages required for the DACs (5 Volts for reference and ±12 Volts for the output opamps).

FPGA The Fastino features an iCE40HX8K CT256 FPGA designed by Lattice Semiconductor. The FPGA's primary functions is to convert the data that arrives on the input headers into SPI transactions with each of the 32 DACs.

1.4 Comparing Existing to New Hardware

The main reason for upgrading the system is to increase the number of DACs. In this perspective the Fastino is superior to the old hardware as just one board already includes 32 individual DAC outputs, i.e. 8 times more DACs per board. The Fastinos do have some extra overhead however, as for every four Fastinos we need one control-MicroZed. This also means having more DACs controlled by a single Ethernet connection, more DACs controlled by a single trigger, et cetera.

The primary difference between the board is of course the actual DACs used on the boards. While the DEATHs used MAX5898's, the Fastino uses Analog Devices AD5542ABCPZ chips. The specifications are compared in table 1. Although the AD5542s are slower, their accuracy is very comparable to the MAXs, and the gain temperature stability of the AD5542s is claimed to be much higher in fact.

Because the Fastino hosts so many DACs it hosts its own FPGA to handle the data flow from the external connections.

1.5 Project Goals

To be able to embed the new hardware in the current control system an adaptation of the DEATH PCB was required, as well as firmware for the FPGA on the Fastino and firmware for the MicroZed. The goals laid out for this report are:

- Design a PCB to connect the MicroZed to (up to) four Fastinos at 2.4 Gbit/s, at 400 Mbit/s per line.
- Flash the Fastino FPGA with an FTDI USB-to-SPI converter chip.

DAC	AD5542ABCPZ (Fastino)	MAX5898 (DEATH)
# bits	16	16
MS/s	∼3	500
Communication Protocol (DACwords)	SPI (50 MHz CMOS)	16 Parallel LVDS inputs
t_{settle} (ns)	∼1000	11
Offset Drift (ppm/ °C)	±0.2	±0.03
Gain-Error Drift (ppm/ °C)	±0.1	±110
INL	±1.0 LSB	±3 LSB
DNL	±1.0 LSB	±1.0 LSB
Output Noise Spectral Density*	11.8 nV/$\sqrt{\text{Hz}} \approx -178.56$ dBFS/Hz	-156 dBFS/Hz
... including amplification	60 nV/$\sqrt{\text{Hz}}$ at 100 Hz	63 nV /$\sqrt{\text{Hz}}$ [3] (page 59)

Table 1: Comparison of a few selected specifications of the Fastino and DEATH DACs. dBFS conversion calculated using 10 V as full scale. *For the Fastino the real Noise Spectral Density is actually higher, as the output stage also consists of two op-amps. It is at least larger than 60 nV/$\sqrt{\text{Hz}}$. (figure 4)

- Write firmware for the Fastino that allows testing the DACs. We opted for a UART controlled bitstream.
- Write firmware that allows the Fastino to take in DAC data (2.4 Gbit/s, 400 Mbit/s per line) during run-time.
- Write firmware that allows the MicroZed to send DAC data to a Fastino.
- Have the MicroZed control a Fastino and use 32 DACs at the same time.

This 2.4 Gbit/s, at 400 Mbit/s per line, figure is very important, as it constrains our design substantially. We will discuss how this figure came about in section 2.4.

2 Execution

Already at the start of the project, we split our group of four students up into two groups. Paul and Michael formed the PCB group, while Benjamin and Marco formed the Fastino group.

The task of the PCB group was to design the breakout board for the MicroZed by adapting the DEATH board while the Fastino group wrote testing firmware for the Fastino. After this first phase, Michael and Paul continued by writing the firmware for the MicroZed, while Benjamin wrote the firmware of the Fastino and Marco wrote and tested a program for using an FTDI USB-to-SPI converter chip to program the Fastino.

2.1 PCB design

The main task was to redesign the former DEATH board by removing all DAC and amplification stage related components, and adding ribbon cable connectors and power supplies to connect up to four Fastino boards.

Power Supply The new DEATH PCB (figure 5) powers both the MicroZed and up to four Fastino boards. It is supplied from the Eurocard connector with two different voltages, one at 5V and the second at 12V. The Fastinos are mainly powered through the 12V line and board management power is provided by a 3.3V rail created on-board of the newly designed PCB. On the other side, the MicroZed receives 5V as primary supply voltage. The I/O bank voltage of the Zynq 7020 is set using a jumper on the PCB. Either 2.5V or 3.3V can be connected. There are in total four different power nets which are all converted from the 5V backplane power supplies. Their nominal value and function shall be shortly summarized below.

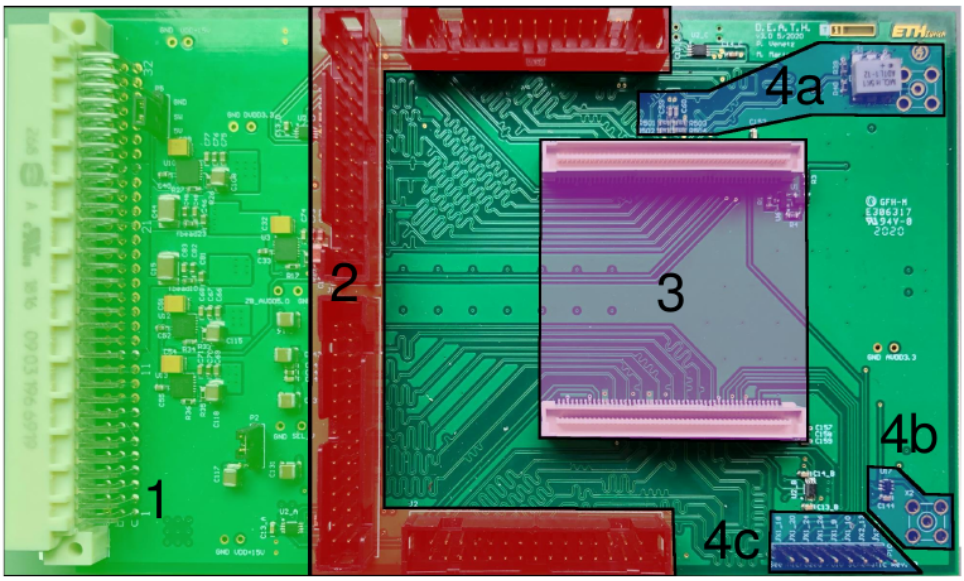

Figure 5: Overview of functional blocks. 1: Power supply. 2: 30-pin ribbon cable connectors for link with the Fastino cards. 3: MicroZed expansion header. 4a: External clock input. 4b: TTL trigger. 4c: Fastbranch header.

power rail (schematic name)	nominal value [V]	dependencies
ZB_AVDD2.5	2.5	I/O banks of the Zynq 7020 FPGA. This will be the default voltage rail as is required by the by the FPGA to drive LVDS pairs.
ZB_AVVD3.3	3.3	Second option for powering the I/O banks of the Zynq 7020. This supply is unused for operation with the Fastino.
AVDD3.3	3.3	Management circuitry on the Fastino board. Should be powered up before the main 12V supply.
DVDD3.3	3.3	There are different instances of level shifters incorporated in the new DEATH PCB. The two main categories are level shifters used for I2C interfaces for additional data exchange with the Fastino and one shifter for translating an external TTL trigger input to the selected I/O bank voltage.

The MicroZed 5V and Fastino 12V power rails are directly powered by the backplane connector and are filtered with a ferrite bead and decoupling capacitors. All mentioned additional power rails are generated on-board using low-dropout regulators (LDOs).

Level Shifter In addition to the 8 LVDS pairs, there is an optional I2C bus which can be used for programming the iCE40 FPGA and for receiving temperature readings from the Fastino. The bank voltage has to be set to 2.5V for the differential lines as the Zynq 7000 Series FPGAs cannot drive LVDS pairs when the I/O bank voltage is higher than 2.5V. The I2C bus to the Fastino however requires 3.3V. For this reason, bidirectional auto-sensing level shifters where inserted into the existing design to account for the logic-level difference on the I2C bus.

External Clock The DEATH board has the capacity to provide an external clock signal to the MicroZed which can be used as the primary clocking source for the FPGA.

Length Matching To ensure error free signal transmission between the MicroZed and the Fastino at high data rate, all differential pairs connecting the MicroZed to the Fastino are length matched taking into account the residual trace length on both MicroZed and Fastino.

Impedance Matching The geometry of the layer stack-up and PCB traces has been chosen to achieve a wave impedance of 100 Ω for differential lines and 50 Ω for single-ended lines. This minimizes reflections at interfaces, e.g. ribbon cable to pin header to PCB traces, which degrade signal integrity.

Manufacture and Assembly The board was manufactured by Gold Phoenix PCB [3] in China and the top layer was assembled at pragmaSol GmbH [4] in Rubigen BE. We soldered bottom layer SMD and through-hole components ourselves. Gold Phoenix PCB was chosen as manufacturer as former revisions of the DEATH PCB were manufactured and assembled by Gold Phoenix PCB as well. Depending on manufacturer, parts for assembly have to be shipped to the manufacturing site. To save time by avoiding shipping and assembly time [5], we decided to ship components and PCBs to Switzerland for assembly. Assembly of SMD (surface-mounted device) components usually requires access to a reflow oven for soldering (especially QFN and BGA packages). During the decision-making process, it was unclear if we would have access to a reflow oven at ETH.

2.2 Fastino Testing/Programming

The Fastino features the iCE40 HX8K CT256 FPGA by Lattice Semiconductor. The iCE40 devices are favoured for their small size and low power consumption, but are in turn limited in capability, compared to the Xilinx 7th series FPGAs on the MicroZeds. For the Fastino that is not a problem however, as the FPGA only needs to forward the incoming LVDS data to 32 SPI buses, and do some CRC calculations.

As described in the goals, we have created two different hardware designs: one design purely for testing the Fastino itself, and one design that would allow the Fastino to be controlled by a MicroZed.

For the testing bitstream we wanted an easy way to control the Fastino from a computer and therefore we opted for a UART (Universal asynchronous receiver-transmitter) controllable Fastino firmware. UART was convenient as we could use the programming pins on the Fastino for the UART communication, and the FTDI USB-serial chip [1] we had planned to use for programming the FPGA supported UART communication out of the box.

There were two communication protocols needed to get output on the Fastino: UART, to communicate from the computer to the Fastino while running it for testing purposes (control LEDs, write waveforms to BRAM, play these waveforms on combinations of DACs), and SPI (Serial Peripheral Interace), which serves to communicate the waveforms from the FPGA to the DACs.

2.2.1 Communication from PC with the Fastino FPGA

To simplify the testing of the Fastino, a communication channel for transmitting certain commands or data is necessary. UART is used as a communication protocol after the FPGA on the Fastino is programmed from the computer.

UART UART is a widely used asynchronous communication protocol. It requires only two lines: one for sending and one for receiving data. Single bits are sent serially over these channels at a predefined frequency (baud rate) starting usually with the least significant bit. The transmission channel is held high by default. To indicate that a transmission has started the channel is pulled low for one cycle (the length of which is given by the baud rate). This start bit is followed by some data bits, then follows an optional parity bit and lastly a high stop bit ends the message, after which a start bit can start another data message or the line can be left high in the idle state.

Implementation In our implementation we chose a message length of 8 bits and omitted the parity bit. The chosen baud rate was 115200. For a UART input and output line we used two of the pins which are also used for programming. The signals are sent and received by the FTDI chip, which is connected to a PC via USB and can be interfaced using a python library called `pyserial`. Using a Python script data can be sent and read over the terminal.

In figure 6 you can see a block diagram of the HDL. We connect a computer through the FTDI chip to the serial in- and outputs, and the SPI outputs go to the DACs. The HDL code can be found on the `fastino-testing` Gitlab repository.

[3] `https://www.goldphoenixpcb.com/`, Gold Phoenix PCB Co., Ltd.

[4] `http://www.pragmasol.ch/`, Thunstrasse 25A, 3113 Rubigen

[5] The reader is reminded that the project has taken place during federal lockdown induced by the global COVID-19 pandemic.

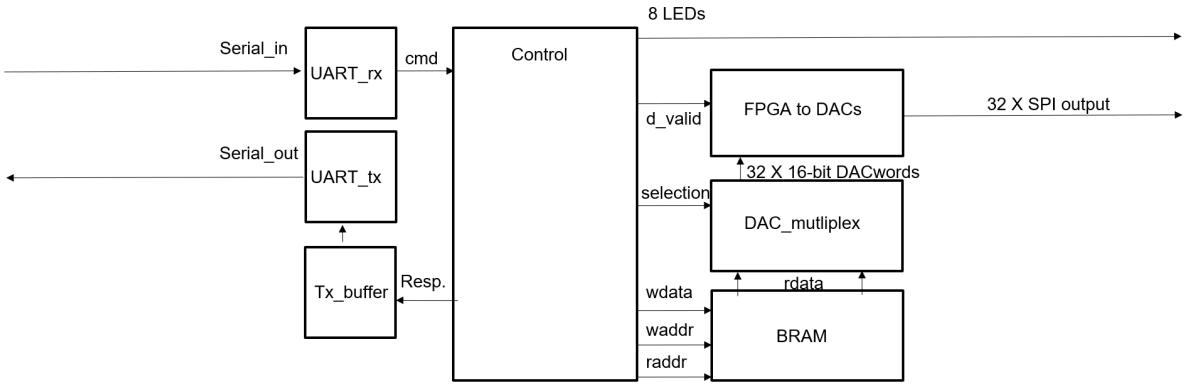

Figure 6: Block diagram of the UART bitstream HDL. The serial connections on the left hand side connect to a computer via the FTDI USB-to-serial converter. The SPI outputs on the right hand side connect to the DACs.

These are the implemented commands, the letter in front is how to call them in the console:

- L – followed by a number between 0 and 255 allows to set the eight LEDs on the front of the Fastino to light up in a binary representation of the set number.
- D – followed by two numbers between 0 and 31 allows to choose which of the 32 DACs outputs the waveform stored in BRAM1 and BRAM2. The DAC with the first number will be linked to BRAM1 and the second to BRAM2.
- W – followed by a path to a file initiates writing to the BRAMs on the FPGA. Requires a 4096-byte file. The first 2048 bytes will be written to BRAM1 the other half to BRAM2.
- P – Plays the currently stored sequence on the BRAMs to the selected DACs.
- R – Toggles looping mode. When looping is enabled, the sequence on the BRAMs will be played continuously on the DACs until looping mode is toggled again.
- M – Toggles output mode. When toggled on, the contents of BRAM1 will be played on all DACs. Contents on BRAM2 are then not used.

2.2.2 Data transfer from the Fastino FPGA to the DACs

To send the DACs voltage values an SPI protocol is used.

SPI SPI is a synchronous communication protocol, which means that it transmits a clock. At either the rising or falling clock edge, depending on the SPI mode, the data is transmitted from the master to the slave. The protocol is full duplex capable, which means that both the master and the slave device can communicate simultaneously.[5]

In figure 7 you can see the typical connections between the master and the slave devices. The signals transmitted are:

- $\overline{\text{CS}}$ is called slave or chip select. The chip select is sent from the master to address the respective slave device, since SPI can involve multiple slaves connected to the same SPI bus, where multiple CS lines would be used. CS is usually an active low signal, that means it is high by default (disconnected from the slave) and low during communication with the corresponding slave.[5]
- SCLK is the SPI clock signal, which determines the speed of the transmission, since a single bit is transferred on each positive or negative edge of the clock.
- MOSI stands for master out slave in. It is the channel to transmit data from master to slave.
- MISO stands for master in slave out. It is the line for data transmitted from slave to master.

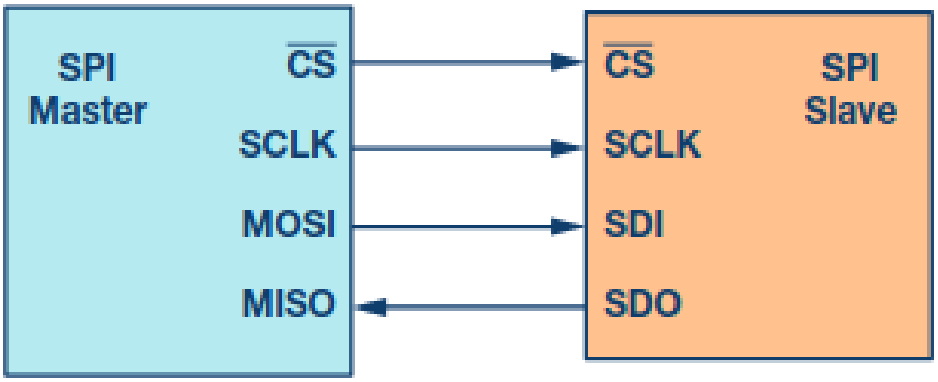

Figure 7: SPI configuration with one master and one slave device. [5]

An SPI transmission starts with the master pulling the CS low to activate the slave and then transmitting the SCLK to the slave for the whole transmission while simultaneously sending and receiving data over the MOSI and MISO channels. [5]

Implementation A schematic overview from the DAC datasheet shows the relevant input signals (figure 8):

- $\overline{\text{CS}}$ (Chip Select)
- $\overline{\text{LDAC}}$ (Latch)
- SCLK (SPI Clock)
- DIN (Data in)
- $\overline{\text{CLR}}$ (Clear)

The Fastino features 32 separate SPI connections to all the DACs, so for each DAC the FPGA has 5 pins connecting them (CLR has just one channel with a fan-out to each DAC). The communication to the DACs is unidirectional, so no response is sent back. The data sent on DIN is clocked in on a rising edge of SCLK, if CS is low. When 16 bits are clocked in and the CS signal goes high, they are shifted to the latch register. Then, when LDAC is pulled down, the latch register is shifted into the actual DAC register. The active low CLR signal will reset the DAC registers to a reset value [4]. CLR and LDAC are asynchronous, so they can be sent at any time during (or after) the SPI transmission.

In our implementation, we wrote hardware description language (HDL) both for a synchronous LDAC operation (LDAC asserted when all DACs are ready for a new word) and an asynchronous operation (assert LDAC for one specific DAC whenever it is ready). We later found out that asynchronous operation can also be achieved by pulling LDAC low at all times. CLR is not used. The SPI clock runs at the highest possible frequency of 50 MHz, limited by the DACs [4].

2.2.3 iCEcube2

iCEcube2 is an "electronic design automation" software tool developed by Lattice Semiconductor. In this project we used it to synthesize and implement our HDL code for the iCE40. It uses Synplify Pro for HDL synthesis and uses its own routines for placing and routing the design. iCEcube2 produces the bitstream which then is sent to the Fastino as discussed in section 2.3 and is used to program the FPGA.

2.2.4 Project iCEStorm

We also briefly attempted to use the open-source toolchain from project iCEStorm[6]. The toolchain consists of yosys[7] for the HDL synthesis and then using nextpnr[8] for placing and routing.

[6] http://www.clifford.at/icestorm/
[7] http://www.clifford.at/yosys/
[8] https://github.com/YosysHQ/nextpnr

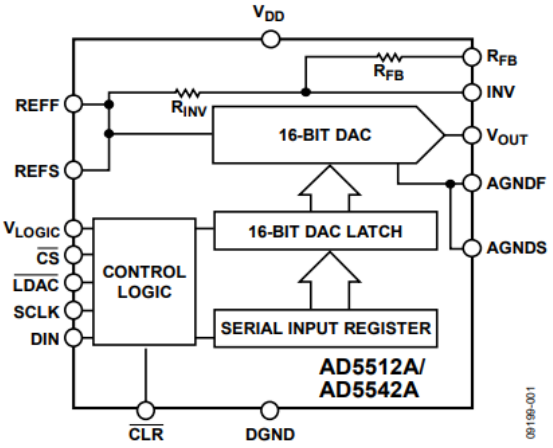

Figure 1. 16-Lead TSSOP and 16-Lead LFCSP

Figure 8: Schematic overview of the DACs[4]

We tried to use Project iCEStorm to see if it generated bitstreams with better timing performances, however after giving it a try and getting much worse results than with iCEcube2 we decided it would be more productive for us to optimize the hdl for iCEcube2 than to try and get better results from iCEStorm.

2.2.5 Icarus Verilog, GTKWave and Aldec Active-HDL

The website of Icarus Verilog concisely describes the software:

> Icarus Verilog is a Verilog simulation and synthesis tool. It operates as a compiler, compiling source code written in Verilog (IEEE-1364) into some target format.
>
> (`http://iverilog.icarus.com/`)

It is an open source program, that we used to simulate our HDL code and export debug data to GTKWave[9] (also open source).

In figure 9 you can see a visualization of the top module of the Fastino's firmware. Going through the waveforms allows accessible low-level debugging of the HDL. Even if all possible test-scenarios are considered, errors can be introduced in synthesis of the design or caused by physical limitations of the target FPGA. There can even be issues with the hardware itself. That's why it is mandatory to also test the programmed Fastino.

iCEcube2 also provides its own HDL simulator, an Aldec Active-HDL license. Although the supplied version is quite limited, it allows us to simulate the FPGA netlist after placing and routing the design, which proved to be crucial on multiple occasions as iCEcube2 is not entirely trustworthy. Sometimes, state machines were encoded as one-hot, however their status register would be reset to all zeroes, an undefined state. We are not completely sure what can be done to circumvent this. In some cases adding a default case to state machines seemed to fix this. One of the limitations of iCEcube2 is that the synthesis tool can only reset registers to 0, even if in a `if reset` case registers are set to 1. It's also possible that the broken state machines can be fixed by making sure that the starting state for a state machine is also the state bound to 0 in the HDL.

2.3 Programming the Fastino

After writing and and verifying correct behavior in simulations of the HDL code, programming the Fastino was the next challenge.

We chose to try to program the Fastino on our own, without buying the official programmer, as the signals of the original Lattice Semiconductor programmer are very well documented (see figure 10) and could quite easily be emulated using an FTDI chip (FT232H by Adafruit). The documentation defines a programming frequency between 1 and 25 MHz [9]. We chose the lower limit, as a logic analyzer of our

[9] `http://gtkwave.sourceforge.net/`

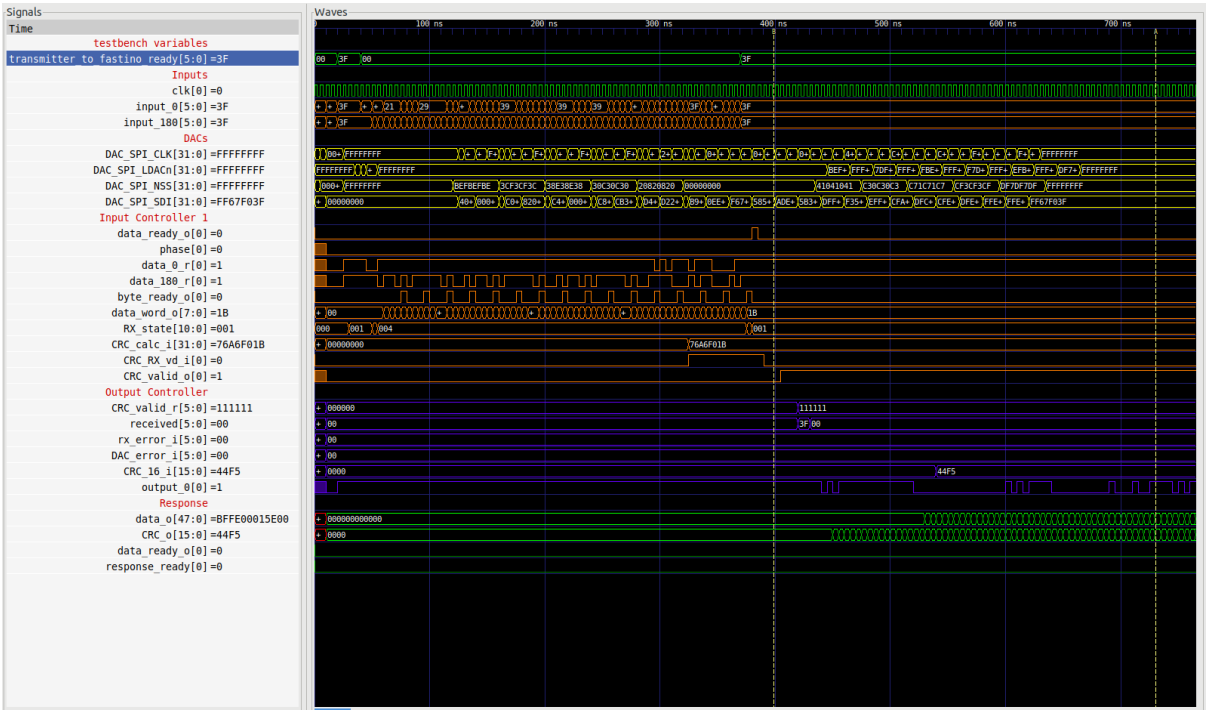

Figure 9: Waveforms of the Fastino's top module simulated in a testbench using Icarus Verilog and visualized with GTKWave

programming sequence using a FTDI device revealed idle sequences in the programming transmission. We thought this would be an issue and the reason why the programming did not initially, so we tried different methods for programming. Two of them are explained in some detail in the coming paragraphs. After some further testing, we found out, that our first assumption was not the actual issue. The iCE40 can handle the idle sequences without a problem, the real issue was, that the programming cables were not connected properly.

Another way of programming the Fastino would be by using the NOR-Flash chip present on the board, instead of trying to program the FPGA directly as we tried first. On power-up, the iCE40 automatically tries to read a bitstream from the flash chip to program itself. This is by far the most convenient method, as the FPGA then keeps its configuration after a power-cycle.

pyftdi `pyftdi` is a pure python implemented driver for FTDI USB - serial devices. Using the pyftdi library it is rather easy to open a GPIO (general purpose input/output) connection over USB to an 8-bit wide port. We used the GPIO driver to transmit a file, generated through a python script, which would bit-bang the SPI transmission at a given frequency required to program the FPGA.

This did not work at first try, as we confused two pins. After realizing the actual issue, this custom method worked just as well.

iceprog Since the pyftdi programming method did not work in the beginning, one of our supervisors, Martin Stadler, made us aware of "Project IceStorm" which could also be used for programming.

"Project IceStorm"[10] is an open source Verilog-to-Bitstream flow for the iCE40 FPGAs. It is notable for being one of the few open source FPGA toolchains. iceprog is a tool from "Project IceStorm" and is used to program iCE40 evaluation/development boards. As it so happens, these boards use an FTDI chip to convert USB to serial communication, and thus we can use the same software to program our own iCE40 if we connect the chip properly to the FPGA programming bus. After installing, it works flawlessly from the terminal if the FTDI device is connected correctly to the FPGA. iceprog does not require to generate a new binary file for the programming (to properly bit-bang the programming sequence). We use iceprog to program the iCE40 over our own pyftdi solution out of convenience.

[10] `http://www.clifford.at/icestorm/`

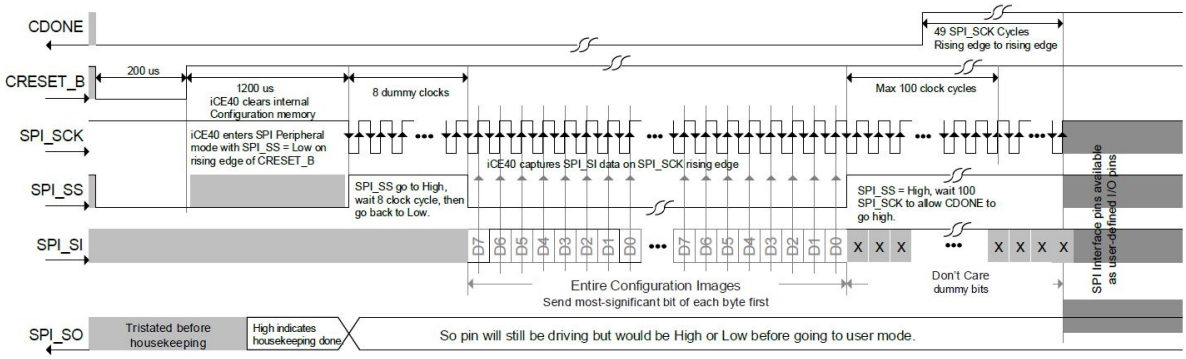

Figure 10: In this figure you can see the detailed signal schedule needed to program the iCE40 FPGA. (source of figure: [9]) . The lower four signal are once more an SPI communication to transmit the bitmap, where the hardware design is encoded. The CDONE in the top line indicates if the programming was successful, when it goes high at the end of the sequence. CRESET_B needs be tied low at the beginning of the sequence to clear the FPGA's programming memory.

2.3.1 Experimental Set-Up for Programming

The Fastino needs to be powered by a 12V and 3.3V source at specific pins. Then the FT232H needs to be connected to the SPI pins on the Fastino (figure 11 and figure 12). The MISO/FLASH_SDI cable is not connected because this port got damaged by during testing, most likely because of bad grounding.

2.4 MicroZed and Fastino

The Fastino has two input headers, allowing for 2×8 LVDS inputs. However, to allow one MicroZed to control 4 Fastinos, we need each Fastino to be controlled entirely by just one header, so by 8 lines. One line is needed for the clock from MicroZed to Fastino, one line is needed for error and status reporting from Fastino to MicroZed, which leaves 6 lines for transmitting data from MicroZed to Fastino.

The DACs on the Fastino (AD5542A) have a resolution of 16 bits, and can take data in at 50 MHz over SPI. Including a small protocol overhead of one bit, this results in an update rate of roughly 3 MHz (50 MHz / 17 bit = 2.94 MS/s) for each DAC, meaning we need 32×16 bits every 335 ns. To ensure data integrity we also include a 32 bit CRC check with each message from the MicroZed.

We opted for a data rate of 400 Mbit/s over each input line, which gives us 134 bits over each line for each 335 ns DAC update-cycle. We divide this up in the following way:

- The first five data lines control 6 DACs each (96 bits) and send a 32 bit CRC.
- The sixth data line controls 2 DACs (32 bits) and has 64 bits spare, followed by a 32 bit CRC. The spare bits could be used for configuration purposes like pausing the continuous DAC update for example.
- Each line uses 4 startbits ("0101") and 2 stopbits ("11"), which allow synchronization of the receiver and transmitter.

To achieve this data rate, we have to use Double Data Rate (DDR) clocking on the input lines, as our FPGA's are not capable of internally running at 400 MHz. A natural choice is then to have both devices run on a 200 MHz clock. We opted for 200 Mbit/s SDR communication on the return line, as there is not a lot of data that has to be sent back from the Fastino to the MicroZed.

This section closely follows the gitlab Readme[11] of the relevant project. As mentioned before the ultimate goal of the project is to control the Fastino through a MicroZed. For this we need to define a communication scheme that allows reliable data transmission to the Fastino, and also allows detection of any errors.

The idle state of each data line is high. When sending data, even bits are clocked into the Fastino on the rising edge, and odd bits on the falling edge.

The data messages are structured as follows:

[11] `https://gitlab.phys.ethz.ch/tiqi-projects/quantech/fastino`

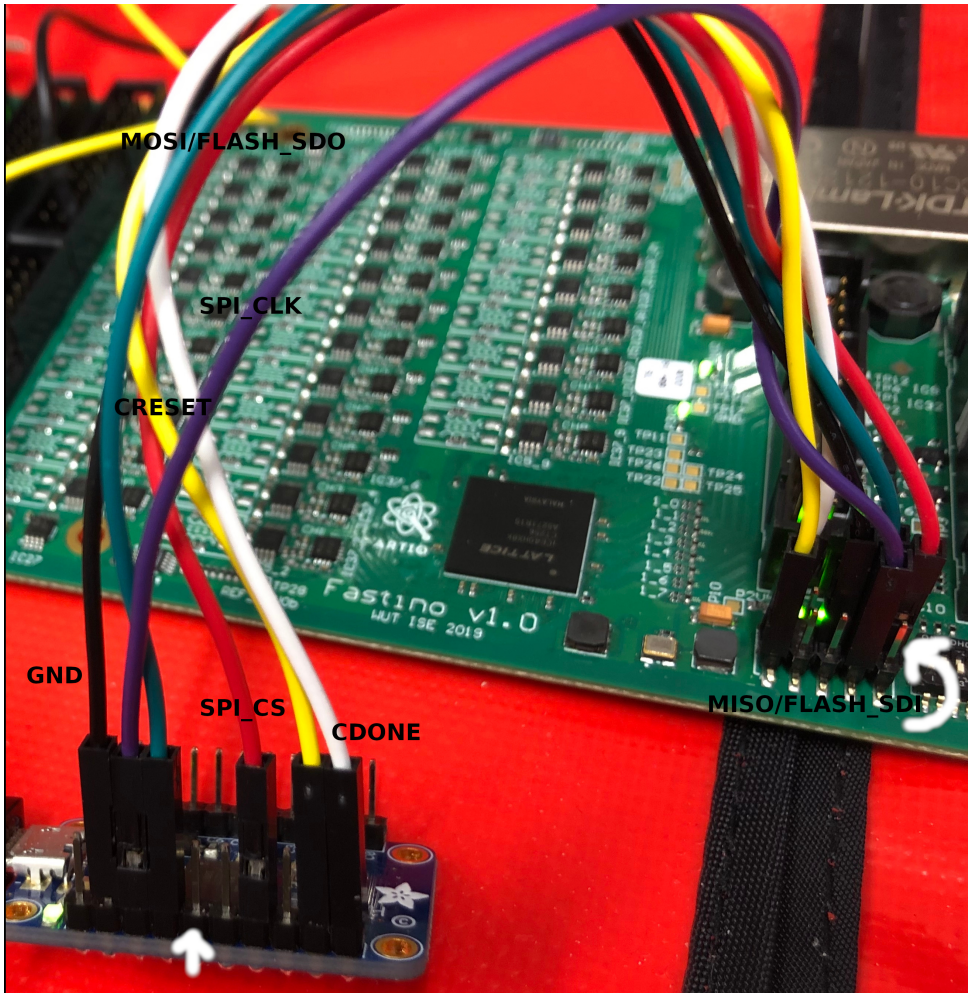

Figure 11: Programming the Fastino with the adafruit FT232H board [1]. Starting from left the cables are connected to the following pins of the Fastino (see figure 12: black cable to GND, purple cable (D0) to SPI_CLK, green cable (D1) to MOSI/FLASH_SDO, red cable (D4) to SPI_CS, yellow cable (D6) to CRESET, white cable (D7) to CDONE. One cable is missing in the picture, it would connect the pins indicated by the white arrows and is the MOSI/FLASH_SDO connection.

- START: “0101”
- DAC0: [16 bit DACword, MSB first]
- DAC1: [16 bit DACword, MSB first]
- DAC2: [16 bit DACword, MSB first]
- DAC3: [16 bit DACword, MSB first]
- DAC4: [16 bit DACword, MSB first]
- DAC5: [16 bit DACword, MSB first]
- CRC32: [32 bits of CRC of the 6 DACwords, MSB first]
- END: “11”

The 6th data line can use DAC 2 through to 5 for other purposes.

The Fastino will send a response if either each input pair received a message, or if one input pair has received 2 messages, while another pair has received no messages. This makes sure that even if a pair is damaged a return message will be sent.

The return message is structured as follows:

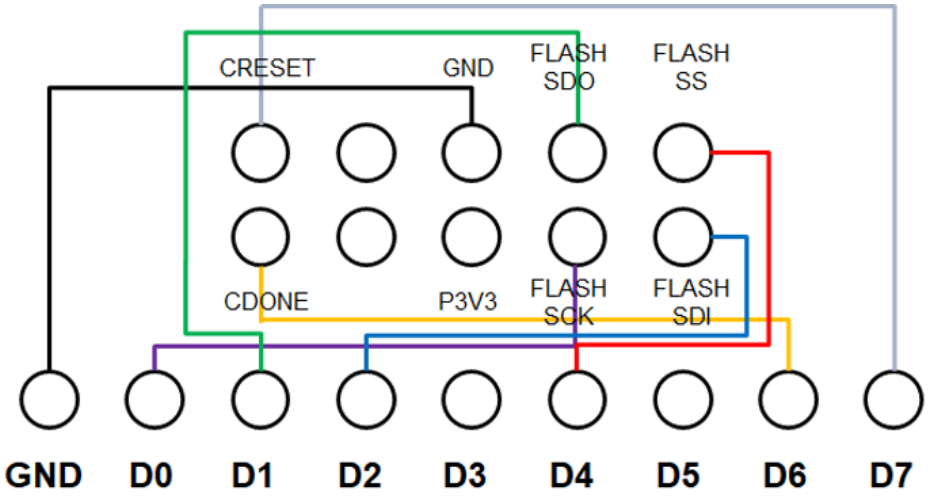

Figure 12: Schematical connections between the FTDI device and the Fastinos SPI input.

- START: “0101”
- CRC VALID: [6 bits, high for a passed CRC check on an input pair]
- MSG RCVD: [6 bits, high for a message received on an input pair]
- RX ERROR: [6 bits, high for an error in the “0101” START sequence]
- DAC ERROR: [6 bits, high indicates an SPI transaction was still in progress when new data should have been sent. NB should not be possible to occur]
- Spare: 21 bits
- CRC16: [16 bit CRC over “101” + the rest of the message.]

Errors reported on the return line can be used to detect if the two boards have lost synchronisation. The MicroZed can then reset the SPI to re-synchronize the communication.

2.5 CRC

2.5.1 What is CRC?

Cyclic Redundancy Checks (CRCs) allows a computer to detect errors in data sent over a channel. The CRC is calculated by doing the equivalent of polynomial division in base-2 on the message data. The remainder of this division is then appended to the actual data. The receiver can divide the incoming data by the same polynomial, and can then compare its own result with the received result. If there is a discrepancy one or more bit flips must have occurred.

The polynomial must be carefully chosen to provide the highest degree of error detection. There are common industry-standard polynomials.

2.5.2 Implementation of CRC

As CRCs are used widely in many applications, much thought has been put into their efficient computation. As the iCE40 is quite strained in our application (both in available flip-flops and in meeting timing constraints) we thought it would be a good idea to use the BRAMs on the iCE40 in aiding the CRC calculation, as these BRAMs are otherwise unused in our design. Also, because data is clocked in 2 bits per cycle, we need a way to calculate multiple CRC bits at the same time.

CRC can be quickly calculated one byte at a time using a look-up table [8]. The look-up table contains all the possible remainders of dividing a byte of data by the CRC polynomial (the remainder is as long as the polynomial itself). For 16 bits of CRC, two tables are needed that each store 256 bytes, and for 32 bits of CRC four such tables are required. This reduces the processing of a data byte to two sequential 8-bit-wide XORs, and a RAM data lookup.

The 32 bit CRC check on the data messages to the Fastino use the polynomial 0x04C11DB7, has an initial register of 0xFFFFFFFF and inverts the final result. For the 16 bit CRC we use CRC-16-IBM, or CRC BUYPASS, with polynomial 0x8005 and initial register 0x0000.

2.6 Fastino HDL structure (For MicroZed Communication)

Apart from the testing bitstream described in section 2.2, we created a bitstream13 to receive data from the MicroZed according to the scheme described in 2.4, and to forward the received data to the DACs.

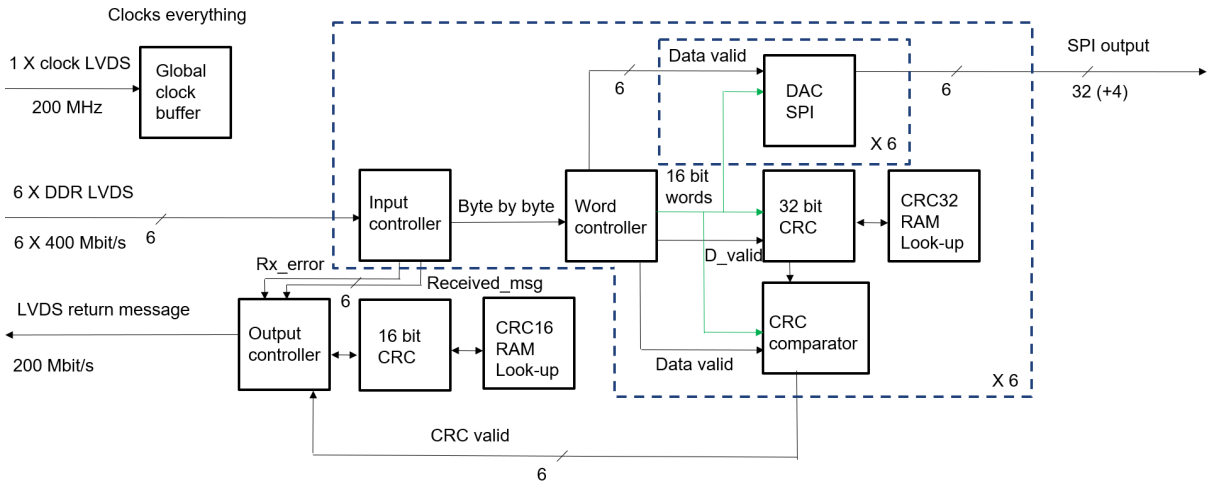

Figure 13: Schematic diagram of the receiving HDL on the Fastino. The left hand side lines connect to the MicroZed through the new DEATH PCB and the right hand side output connects to the DACs. The dashed blocks indicate repeated components. All registers are clocked by the LVDS clock originating from the MicroZed. The DAC error is missing in the schematic. Note the CRC look-up RAMs connected to the CRC modules to accelerate CRC calculation.

All registers are clocked by the LVDS clock coming from the MicroZed. Each LVDS input pair is sent byte-by-byte from the input controller to the word controller, which controls a 16-bit bus that connects to 6 DAC SPI modules, a CRC calculator and a CRC comparator. The word controller also has `data valid` signals to each module connected to the bus. This single bus helps with meeting timing constraints.

2.7 MicroZed HDL structure

At the current design stage, the MicroZed manages the storage of waveform data and implements the master side of the modified SPI. Modifications to a pure SPI implementation are the usage of DDR compared to SDR and that the chip select signal is omitted as it uses point-to-point communication.

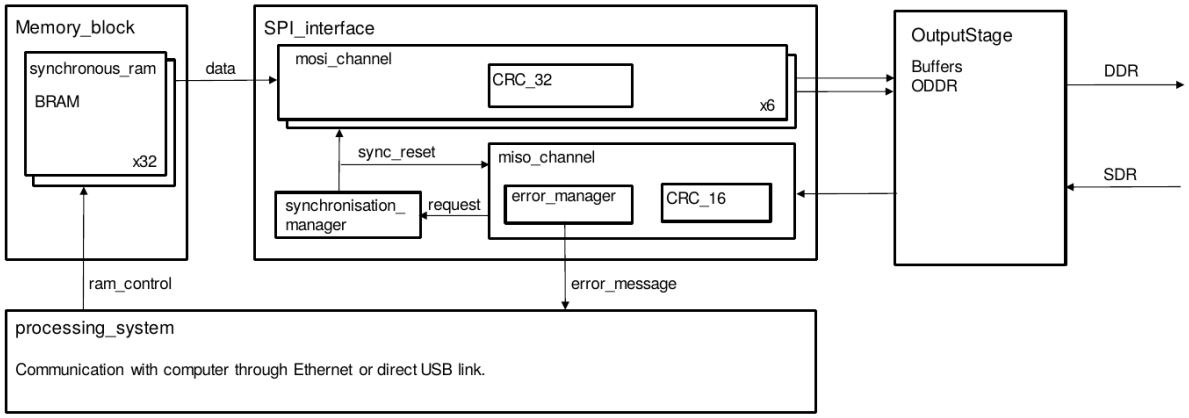

Figure 14: Diagrammatic overview of the MicroZed HDL structure.

2.7.1 SPI

The MicroZed firmware is mainly composed of a quasi `SPI_interface` module which sends DAC data to the Fastino in 6 separate MOSI channels following the structure described in section 2.4. The `OutputStage` module then converts signals to DDR output and includes I/O buffers which connect to the pads of the Zynq 7020.

One MISO channel per Fastino allows for error detection. The communication structure is also described in section 2.4. If any error is detected, an error message is produced and stored in a 8 bits wide register (`error_manager` module) until another error is produced or a certain `KEEP_TIME` (measured in clock cycles) has passed. To allow for debugging, the error message is passed to the processing system of the Zynq 7020. This enables users to output messages in a terminal, which is handy for debugging. The `KEEP_TIME` was implemented to increase the chance of the processing system to detect single isolated errors, since it checks the register at a much slower rate than the FPGA potentially outputs distinct error messages. This polling approach for reading errors could later be replaced by an error interrupt approach.

If too many errors are detected, Fastino and MicroZed have probably lost synchronization. This is indicated by a status bit which is used for sending re-synchronization requests. The `synchronisation_manager` module reads this bit and takes action accordingly. In the current firmware, this module acts rather aggressively and will reset the SPI interface as soon as this status bit is asserted high. Re-synchronization is performed by waiting till the return line from the Fastino has been idle for long enough to be sure that currently no response is being transmitted by the Fastino, indicating that it is ready for a new message.

2.7.2 Waveform Storage

Each DAC has its own RAM associated with it. The RAM is implemented as BRAM in the fabric of the Zynq 7020 FPGA. The current firmware will cycle through the whole BRAM, making the output waveforms periodic. While easily scalable, each RAM currently only has an address width of 9 bits providing room for 512 DAC samples. It is possible to write to each DAC RAM individually using the ARM CPU such that arbitrary manipulations to the waveforms can be performed. Communication between the CPU and the programmable logic is established using the AXI GPIO v2.0 IP provided by Xilinx. A separate module `ram_controller` manages the request by broadcasting the address and data to all DAC RAMs but only enabling one specific RAM for a write transaction. This is far from ideal as it is possible to use RAM controllers which are directly connected to the Processing System (PS) of the Zynq 7020 using an AXI bus.

2.7.3 Additional Control Signals

There are three additional signals which are either external inputs or used to enable/disable functionality on the PCB.

ttl_trigger External TTL input used by the control system to start an experiment. Could be used to start waveform transmission to the Fastino.

fastino_en This signal can be used to enable/disable the LDO which produces the AVDD_3.3 supply rail. This can be used to power-cycle all Fastino boards at the same time. However, the current layout does not allow for power-cycling a specific Fastino.

shifter_en Enables/disables the level shifter used for the I2C bus implementation. When disabled, the I/O lines of the shifter are set to a high-impedance state.

3 Results

3.1 Initial Fastino Tests

After some initial difficulties with programming the iCE40 on the Fastino, we managed to get a first bitstream which makes the status LEDs blink. The management power connector was not connected to the management power net on the PCB, which required us to add a wire on the bottom side of the board to power the board from the input header. Once this worked we also managed to write this bitstream to the flash memory on the Fastino, and the iCE40 was able to program itself after a power-cycle by reading the bitstream from the flash memory. Due to an electrical accident the FPGA_SDO pin was not able to communicate anymore, so sadly this specific iCE40 cannot program itself anymore.

We then programmed the Fastino with the bitstream that exposes a UART interface for dynamic modification of the waveforms sent to the DACs, and we could successfully communicate with the iCE40 through the FTDI chip. After fixing some bugs we were then able to write waveforms to the iCE40 BRAM and output these on DACs of our choice. In the end we only managed to run this bitstream at 150 MHz, instead of our desired 200 MHz, due to timing constraints in the HDL design.

This bitstream was used to do some preliminary tests of the DACs, where we discovered that DAC16's output voltage ranged only from -10 to +5 volt. Going through the schematics we have determined that this is probably caused by a faulty feedback resistor in the amplifier stage of channel 16. The actual DAC chip seems to be fine.

3.2 Initial MicroZed Tests

Before connecting the MicroZed and Fastino to the breakout PCB, initial electrical tests have been performed to avoid potential damage to either board and to assess connectivity between the Zynq 7020 FPGA and the pin headers used for data transmission.

On-Board Power Supplies The on-board power supplies were tested for their output voltage to make sure they are well in operating range. The measurements were made using a simple digital multimeter.

power rail	nominal value [V]	measured value [V]	dependencies
ZB_AVDD2.5	2.5	2.48 ± 0.01	I/O banks MicroZed
ZB_AVVD3.3	3.3	3.28 ± 0.01	I/O banks MicroZed
AVDD3.3	3.3	3.28 ± 0.01	management power on the Fastino
DVDD3.3	3.3	3.28 ± 0.01	TTL level shifter, I2C level shifters

Connectivity A separate hardware design was used to test the connectivity between the FPGA pads and the pin headers. For greater flexibility each I/O line can be set either as an input or as an output using the on-chip processing system. The current PCB layout allows for reading and writing any pin used for data transmission.

Level Shifter Each of the two lines can be used either as input or as output and auto-sensing translation direction was found to be working. This was verified by connecting both lines and repeatedly writing to one pin followed by reading from the other. After every read/write step, the roles were exchanged.

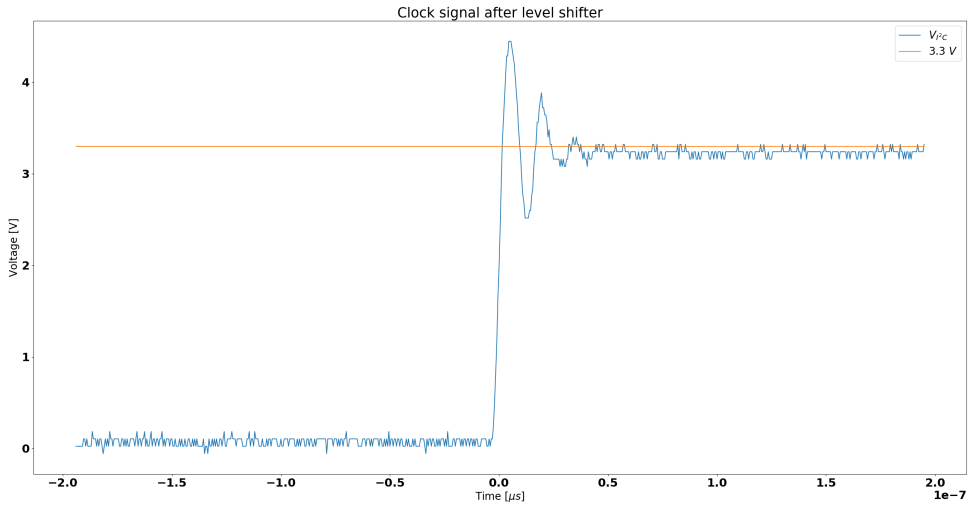

Figure 15: Overshoot during a rising edge on the new DEATH PCB I2C bus, no additional termination to the probe.

3.2.1 Design Issues

Level Shifter: Transients When checking the I2C lines for the ability to output data, strong over- and undershoots (depending on the type of edge) have been observed. A representative transient is shown in (figure 15). An easy way to alleviate this problem would be to add series resistance to the data lines in the next iteration of the breakout PCB. To avoid accidental damage, one should investigate if the severity of the overshoot (undershoot) would persist even with the presence of a pull-up resistor on the Fastino side of the data lines.

Differential Pairs: Polarity There are two kind of Fastino connectors: Polarity matching ones (J1, J4) and non-matching ones(J2, J3). For non-matching differential lines, the P signal from the Zynq 7020 is connected to the N signal of the iCE40 and vice versa. This has been done for easier routing of the differential traces on the DEATH breakout PCB. Data sent using J2 or J3 would therefore need to be bit-wise inverted before transmission.

Due to errors in the schematic there are two lines with non-swapped polarities. The following table summarizes which signals are physically connected (row-wise):

Connector	Znyq 7020 Pin	Signal MicroZed	Signal Fastino
3	F19	JX2_LVDS_15_P	EEM0_4_P
3	F20	JX2_LVDS_15_N	EEM0_4_N
2	Y18	JX1_LVDS_16_P	EEM0_1_P
2	Y19	JX1_LVDS_16_N	EEM0_1_N

To avoid further confusion, the signal name on the MicroZed side was chosen as in the MicroZed HW Users Guide [2].

AVDD_3.3: Missing Pull-Down In the current design, there is no pull-down resistor placed for the power rail AVDD_3.3. When disabled using the control signal, the output remains floating. This in turn does not power-cycle the Fastino. Luckily, it is quite easy to solder a through-hole resistor on the solder side of the PCB which connects the LDO output directly to ground.

Silkscreen Errors Silkscreen around the fastbranch header is faulty. The wrong MicroZed Zynq chip name and revision is written down, as well as the incorrect names for the fastbranch signals.

3.3 Communication between MicroZed and Fastino

Once both devices had been tested separately, we were ready to connect them together using a ribbon cable, with the Fastino controlled and powered by the new DEATH PCB(figure 16). Initial tests at 100 MHz worked as expected (figure 17). At 200 MHz we were unable to use the oscilloscope to observe the waveforms, so we had to rely on the error reporting on the Fastino to check whether communication was established correctly.

While doing this, we found out that even though iCEcube2 reported a supported frequency of 200 MHz, and no CRC errors or other errors were reported by the Fastino, we saw a few DACs that had a corrupted output (figure 18). In the end we don't know exactly what caused this, however after optimizing timing a little more in the SPI module that communicates with the DACs we managed to compile a bitstream that had the intended output on each DAC. It's possible that we did not properly set timing constraints for the outputs to the DACs.

With our optimized module we were able to control the Fastino with the MicroZed at 200 MHz using any of the four ribbon cable connectors. We were able to induce errors by resetting the Fastino during a message to check the error detection. Also, we discovered that we had made a faulty ribbon cable. While moving the ribbon cable slightly, we could generate a stream of errors. With a new ribbon cable the errors were gone.

To conclude, our project was successful: We managed to have the MicroZed communicate with the Fastino at 2.4 Gbit/s, updating the DACs at around 3 MHz, and controlling 32 DACs at a time. These same results were observed on each connector. An assortment of simultaneously output waveforms is shown in figure 16 and figure 19.

3.4 DAC measurements

3.4.1 Slew Rate and Settling Time

The settling time, the slew rate and the time constant (time interval in which $1 - 1/e = 63,2\%$ of the maximum is reached) were measured for a step-function starting and the minimal voltage of the DACs (0x0000, $\sim$ -10V) to the maximal voltage (0xFFFF, $\sim$ 10V) and also for a smaller step-function (0x9388 to 0x9B58, the difference is 2000 in decimal numbers, which results in a step of about 0.6V) using an oscilloscope (normalized plots in figure 20 and figure 21). For the maximal step-function the slew rate is mostly around 19-20 V/µs.
The settling time for the maximal step-function was found to be around 1 µs (mean: 1.03 µs $\pm$ 0.03 µs), which is in agreement with the Fastinos Github Wiki [10]. For the smaller step-function the settling time is similar as to the maximal step-function, but slightly lower (mean: 0.96 µs $\pm$ 0.03 µs). The slew rate on the other hand is completely different: instead of values between 19-20 V/µs we have a slew rate around 0.56 V/µs. The reason is visible in the data, the rise time for both step-functions is very similar (around 1 µs). This helps explaining the lower in slew rate observed for DAC output 16 in both plots, as the voltage is re-scaled to 3/4 of what the other DACs achieve, hence if the rise time is similar and the voltage difference are smaller, the resulting slew rate is smaller.
The average time constant of the maximal step- function is 0.65 µs $\pm$ 0.03 µs and for the smaller step 0.55 µs $\pm$ 0.01 µs, where the uncertainty is given by the standard deviation over all DACs. DAC 16 shows similar behaviour here.

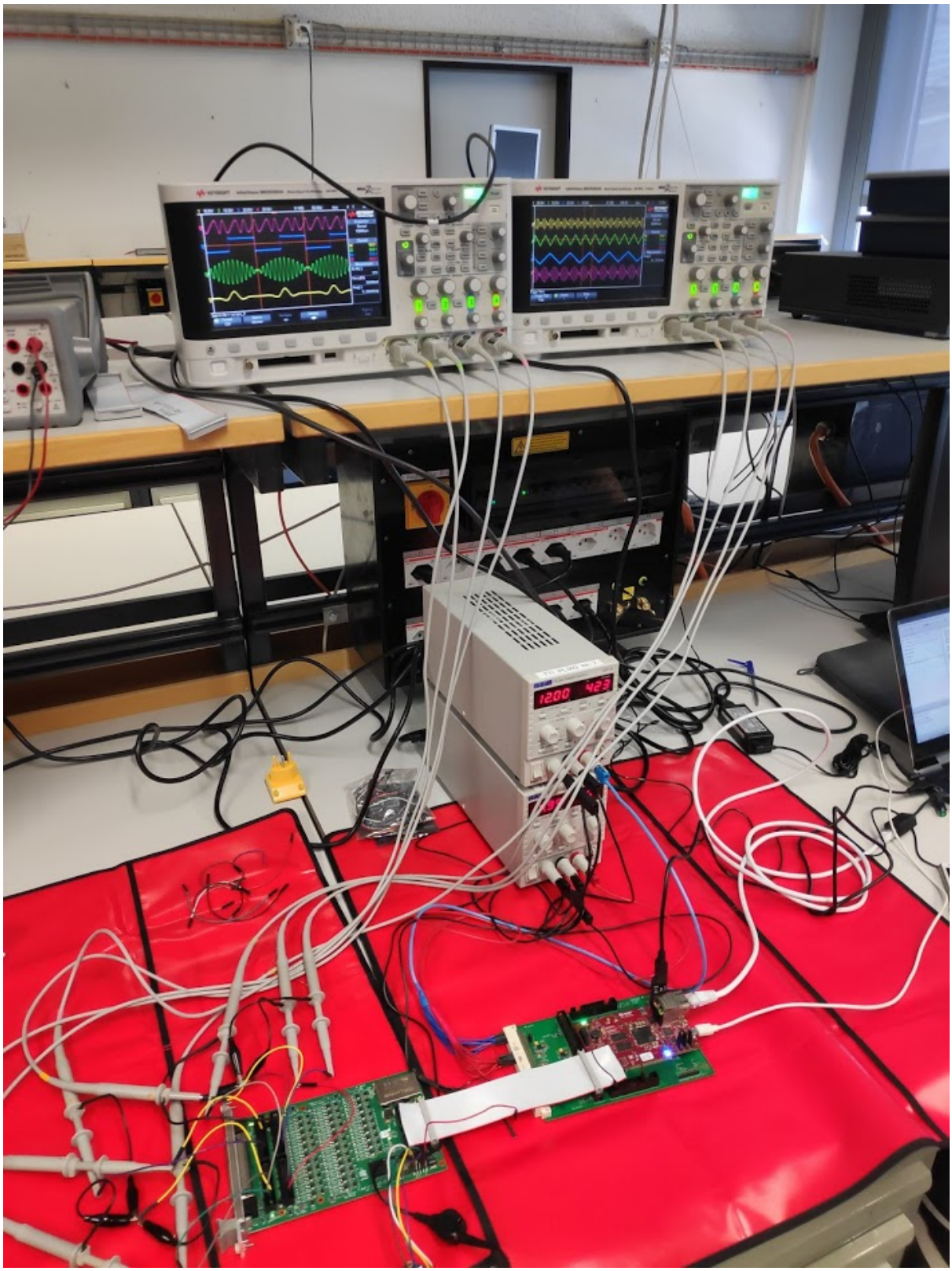

Figure 16: Set-up of the Fastino connected to the MicroZed through a ribbon cable. Note the waveform recorded from the DACs on the oscilloscopes, and the MicroZed connected with USB and Ethernet to a laptop

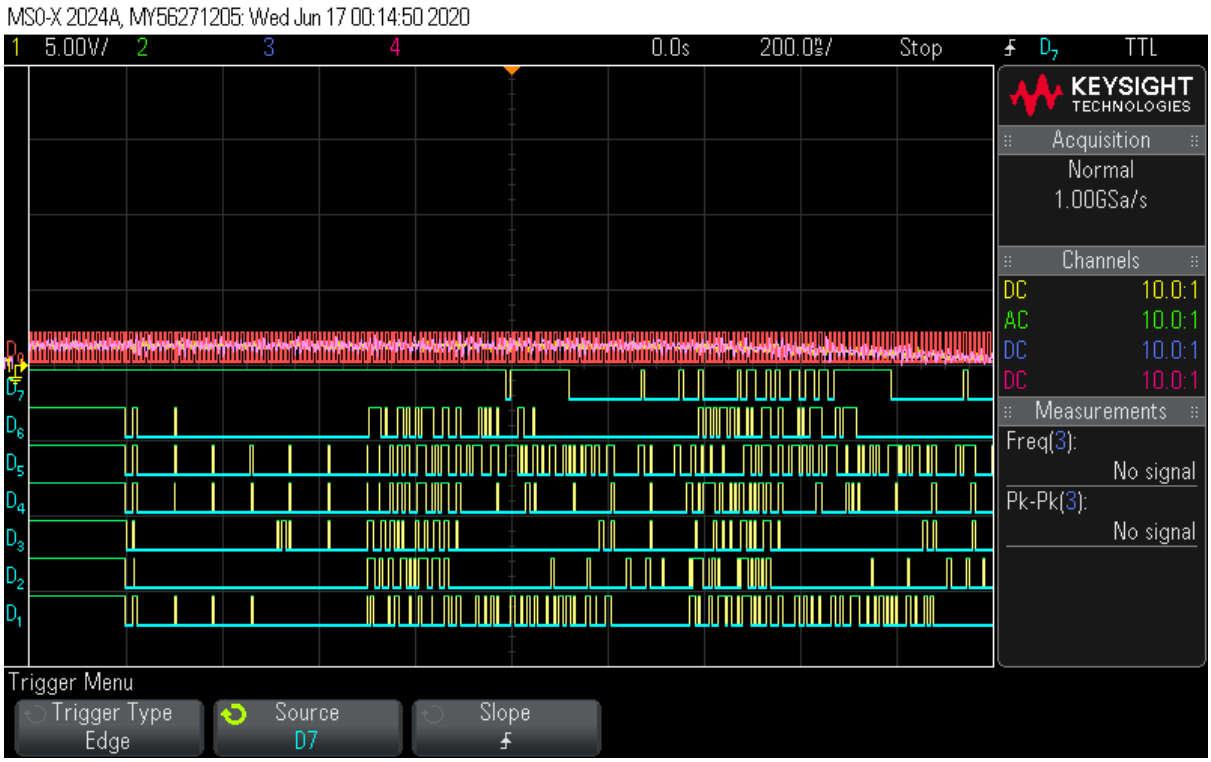

Figure 17: Digital communication between Fastino and MicroZed at 100 MHz. Note that all lines start simultaneously. Due to aliasing the proper start sequence (0101) cannot be distinguished.

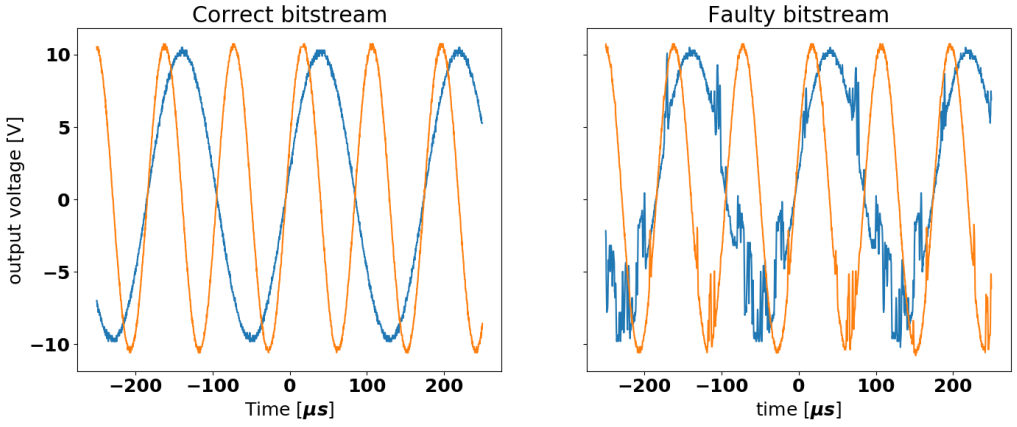

Figure 18: Left: Clean DAC output. Right: Corrupted DAC output. The aliasing in the clean picture is caused by using a small BRAM to encode a fast waveform, and is not a limitation of the DACs. Note that the corruption of the waveform is deterministic, i.e. it is the same every period. This points to the cause being a systematic error in the FPGA, as opposed to random noise on the transmission line.

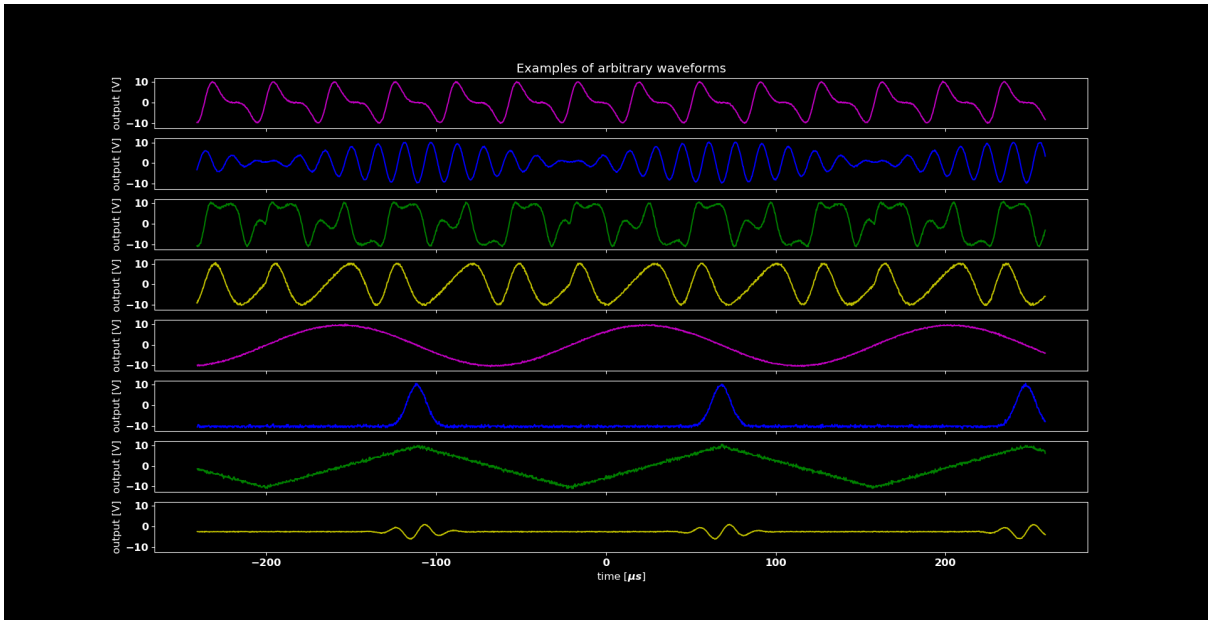

Figure 19: Eight waveforms captured with the set-up in figure 16, one oscilloscope is triggered by the other, effectively creating an eight channel oscilloscope. This was meant as a definitive "proof of concept", conclusive evidence that we have an AWG that has at least eight channels. Of course, we have 32 channels, but measuring 32 channels simultaneously was impossible with our equipment.

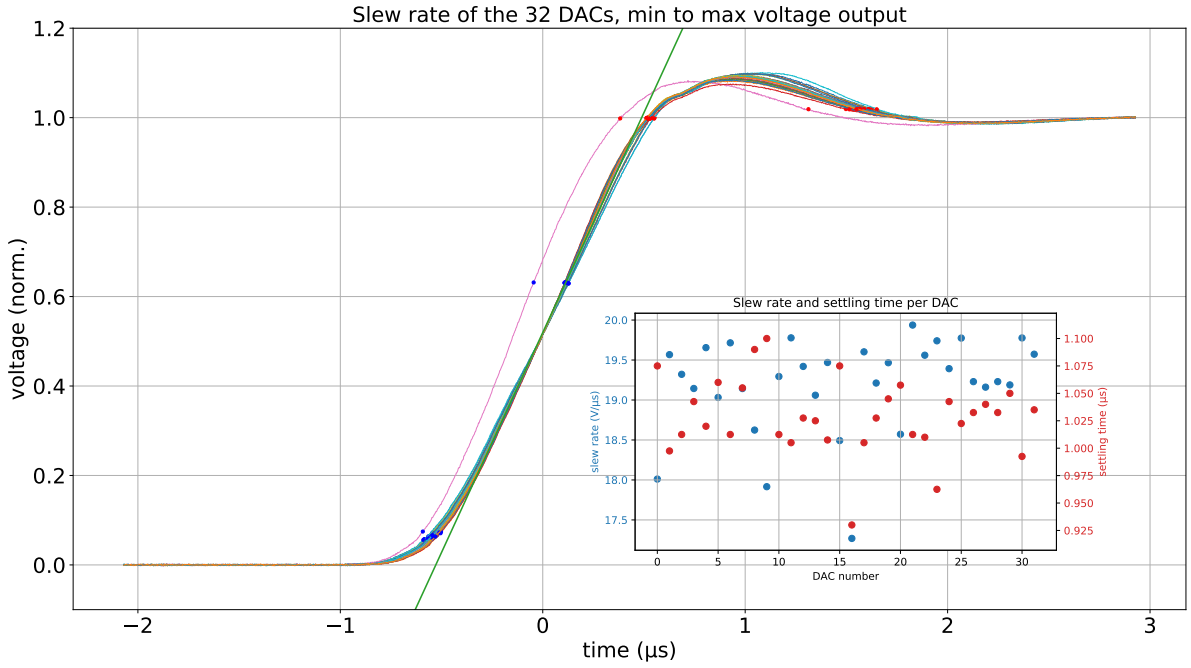

Figure 20: Normalized measurements of a step-function from minimal voltage (∼ -10V) to maximal voltage (∼ 10V). The red dots show in which interval the settling is measured (tolerance of 2% difference from the final value). The dark blue dots indicate the interval in which the time constant was measured. The first blue dot is at the timestamp of the interception of a linear fit (fit between 25% and 75% of max. val., see green line as example) and the ground line. The second blue dot is simply 63,2% of the final value. It is clearly visible that all DACs behave very similarly except DAC 16, which has, as mentioned before, a reduced range (can only achieve values between -10V to +5V). In the inset the slew rate (blue dots) is plotted with the axis on the left, while the settling time (red dots) is plotted with the axis on the right.

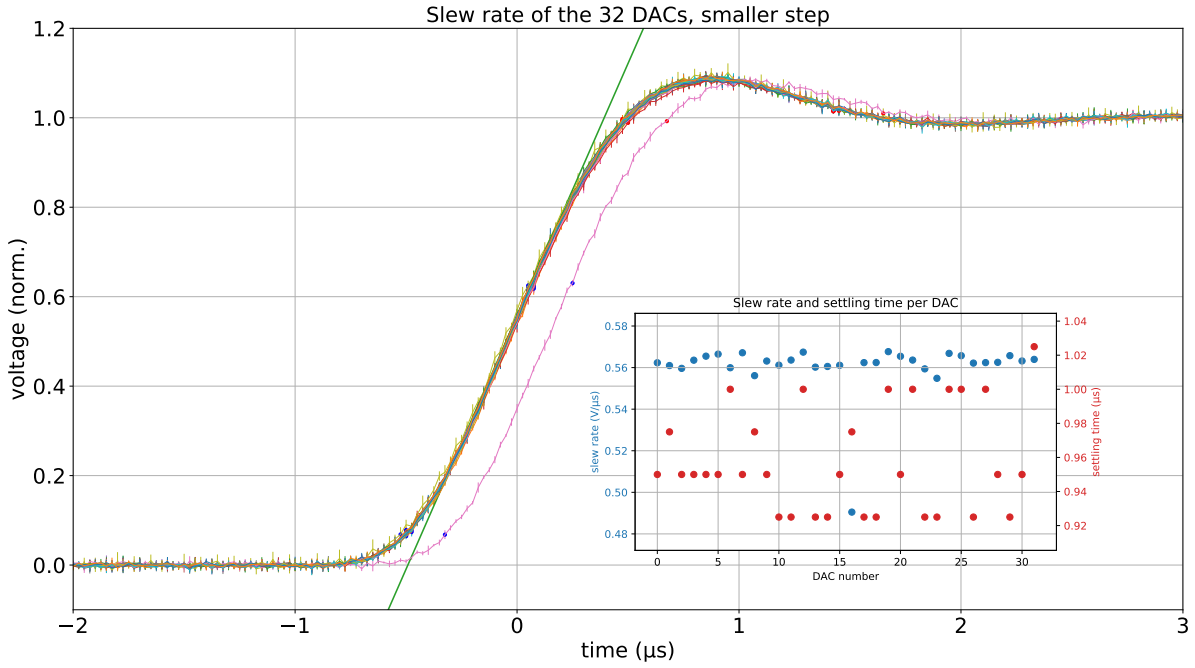

Figure 21: This plot is of the same form as figure 20, with the difference that it shows measurements of a smaller step-function. From 0x9388 to 0x9B58. The voltage ranges from 1.54 V to 2.14 V. The data was too noisy to easily calculate the settling time; for this reason we have averaged the data points in groups of five. The errorbars are the standard devations of each group of five. The data behaves very similarly, just the slew rate of DAC 16 is much different.

3.4.2 Linearity of the DACs

The voltages on the DACs are set with a sixteen bit number over a range from around -10V to 10V. In this measurement, five evenly distributed values of each DAC was measured to check how linear the dependence of the DACs are to this sixteen bit number. The chosen values are:

Number in hex	0x0000	0x3FFF	0x7FFF	0xBFFF	0xFFFF
Ideal voltage (rounded)	-10V	-5V	0V	5V	10V

Figure 22 visualizes that most of the DACs are evenly distributed around the average value (red line). The fact that all curves are not flat indicates a clear bias in the voltage outputs, which should be adjusted for if exact voltage values need to be achieved. The DACs tend to be too negative below zero and too positive above zero. This behaviour can be explained by taking into account that the DACs evenly distribute the possible voltage values not as assumed between -10V and 10V but over a slightly bigger range (minimal possible output voltage to maximal voltage). In any case, a linear distribution of voltages to input numbers seems to be a very good approximation, the maximum deviation from linearity is 0.2 mV, which corresponds to 2/3 LSB (figure 23).

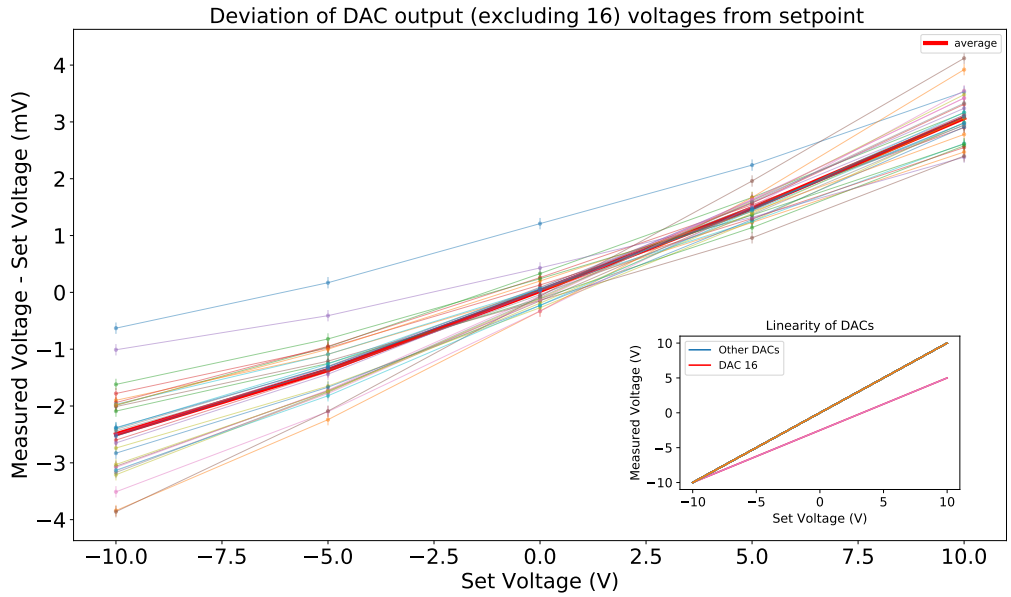

Figure 22: Voltage outputs of the DACs subtracted by their ideal value (see table page 25). The red line in the middle is the average over all DACs except DAC 16. In the inset is the measurement of the voltages plotted without subtraction, it highlights the good linearity/similarity of the DACs and the discrepancy of DAC 16.

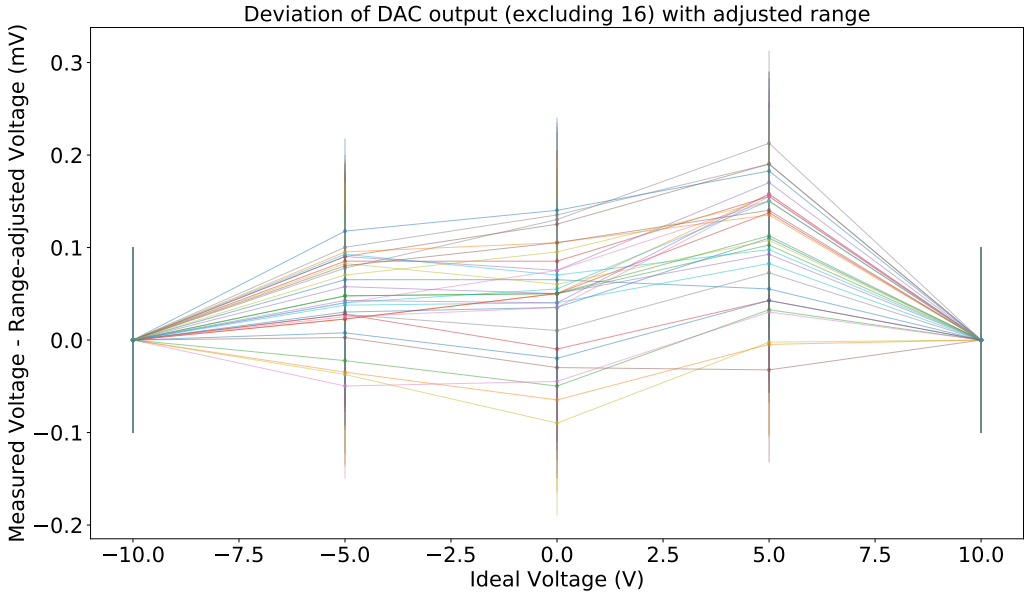

Figure 23: Voltage outputs of the DACs subtracted by their rescaled ideal value. The subtracted voltage for each DAC was calculated by linearly distributing all 0xFFFF + 1 values over the whole range of said DAC to visualize the INL (integral nonlinearity). Looking at it this way exhibits, that all of them deviate from their expected linear behaviour only by about 0.1 to 0.2 mV, which is less than one significant bit (2/3 of a LSB). The errorbars are 0.1 mV, which is given by the datasheet of the digital multi-meter we used [11].

The raw data of this measurement and fitting parameters are in the Appendix section 5.2.

3.4.3 Output Stability

We measured the output of one DAC over the course of an afternoon. The value was set to 10V and drifted by less than 300 µV (which is less than one LSB), indicating high stability. We observed changes over the course of more than four hours. Data is in the appendix in section 5.1 on page 29.

4 Conclusion and Outlook

Conclusion A major challenge for the development of quantum computers is their scalability. Through the work in this report we have attempted to improve the scalability of the number of electrodes in an ion trap, which will in turn increase the maximum number of individually controllable ions in an ion trap. This is key to having a quantum computer with many qubits. The major accomplishments of this report are verification of the Fastino functionality and its testing, and the design, manufacturing and assembly of a PCB for powering both MicroZed and Fastino as well as for allowing communication between the two. We hope that our work will be of much use to the TIQI group.

Outlook There are still some points that need to be taken care of before the new hardware can be used in the existing TIQI architecture, which provide a basis for a new project. The remaining tasks can be split into two categories. First, the evaluation of the Fastino needs to be completed. A detailed characterization of the noise, as well as measurements of temperature dependencies, of the DAC outputs need to be performed. After confirming the suitability of the new hardware, it has to be integrated into the existing control system. The sequencer called */texttt*Scythe* needs to be adapted for usage with potentially up to 128 channels. The communication between the Processing System and Programmable Logic should be refined for better performance. Upstream software/firmware also need adaptation to accommodate for the increased channel number. Only after these steps it will be possible to fully integrate the new set-up and perform actual experiments.

References

[1] Adafruit. *Adafruit FT232H Breakout*. https://www.adafruit.com/product/2264#technical-details. Accessed: 2020-07-12. 2020.

[2] Avnet. *MicroZed^{TM} Zynq®Evaluation Kit and System on Module Hardware User Guide*. http://zedboard.org/sites/default/files/documentations/5276-MicroZed-HW-UG-v1-7-V1.pdf. Version 1.7. Mar. 2017.

[3] L. E. de Clercq. "Transport Quantum Logic Gates for Trapped Ions". PhD thesis. ETH Zurich, 2015.

[4] Analog Devices. *AD5512A/AD5542A*. https://www.analog.com/media/en/technical-documentation/data-sheets/AD5512A_5542A.pdf. Rev. C. 2017.

[5] P. Dhaker. "Introduction to SPI Interface". In: *Analog Dialogue* 52.6 (Sept. 2018), pp. 49–54.

[6] Maxim. *16-Bit, 500Msps, Interpolating and ModulatingDual DAC with Interleaved LVDS Inputs*. https://datasheets.maximintegrated.com/en/ds/MAX5898.pdf. Rev. 2. 2010.

[7] V. Negnevitsky. "Feedback-stabilised quantum states in a mixed-species ion system". PhD thesis. ETH Zurich, 2018.

[8] D. V. Sarwate. "Computation of Cyclic Redundancy Checks via Table Look-Up". In: *Commun. ACM* 31.8 (Aug. 1988), pp. 1008–1013. ISSN: 0001-0782. DOI: 10.1145/63030.63037. URL: https://doi.org/10.1145/63030.63037.

[9] Lattice Semiconductor. *iCE40 Programming and Configuration, Technical Note*. http://www.latticesemi.com/view_document?document_id=46502. FPGA-TN-02001-3.2. Mar. 2020.

[10] Sinara. *Fastino*. https://github.com/sinara-hw/Fastino/wiki. Accessed: 2020-07-10. 2019.

[11] Keysight Technologies. *Digital Multimeters34460A, 34461A, 34465A (6 digit), 34470A (7 digit)*. https://www.keysight.com/us/en/assets/7018-03846/data-sheets/5991-1983.pdf. 5991-1983EN. Apr. 2020.

[12] D. J. Wineland et al. "Architecture for a large-scale ion-trap quantum computer". In: *Nature* 417 (2002). URL: https://doi.org/10.1038/nature00784.

5 Appendix

The data for the slew rate and settling time as well as .csv files of the data here is on the git repository:

5.1 Stability Measurement Data

The data discussed in section 3.4.3. The last two values were taken, while the Fastino was constantly updating the voltage value of the corresponding DAC to 10V. The uncertainty of the digital multi-meter is 0.1 mV in the chosen range [11].

Time of measurement	Voltage value [V]
11:42	10.00387
11:18	10.00378
12:16	10.00370
13:14	10.00363
13:38	10.00361
13:50	10.00360
14:32	10.00362
15:12	10.00363
15:34	10.00362

5.2 Linearity Measurement Data

The data of the measurement discussed in section 3.4.2. The hex numbers indicated the values the DACs were set to. The units of the measurements are in Volts, with uncertainty of ± 0.01 mV.

DAC number	0x0000	0x3FFF	0x7FFF	0xBFFF	0xFFFF
0	-10.00063	-4.99983	0.00121	5.00224	10.00353
1	-10.0019	-5.00109	3E-05	5.00123	10.00247
2	-10.00199	-5.00096	0.00033	5.00167	10.00316
3	-10.00194	-5.00095	0.00026	5.00154	10.0029
4	-10.0025	-5.00131	8E-05	5.00158	10.00312
5	-10.00306	-5.00173	-0.00015	5.00146	10.00312
6	-10.00308	-5.00172	-6E-05	5.00164	10.00342
7	-10.00241	-5.0013	9E-05	5.00151	10.00294
8	-10.00274	-5.00165	-0.00027	5.00125	10.00299
9	-10.00197	-5.00109	6E-05	5.00125	10.00262
10	-10.00283	-5.00167	-0.00023	5.00128	10.00291
11	-10.00384	-5.00224	-0.00033	5.00167	10.00392
12	-10.00209	-5.00125	-0.00011	5.00114	10.00258
13	-10.00178	-5.00098	0.00013	5.0013	10.00255
14	-10.00101	-5.00041	0.00043	5.00132	10.00238
15	-10.00201	-5.00121	-0.00014	5.00096	10.0024
16	-10.00385	-6.25268	-2.50133	1.25004	5.00151
17	-10.00351	-5.0021	-0.00033	5.00151	10.00355
18	-10.00244	-5.00137	-4E-05	5.00137	10.00295
19	-10.00321	-5.00176	-0.00011	5.00161	10.00348
20	-10.00239	-5.00126	6E-05	5.00143	10.00298
21	-10.00253	-5.00134	6E-05	5.00148	10.00298
22	-10.00198	-5.001	0.0002	5.00142	10.00278
23	-10.00162	-5.00082	0.00024	5.00136	10.00261
24	-10.0026	-5.0014	4E-05	5.00157	10.00331
25	-10.00265	-5.00144	3E-05	5.00162	10.00324
26	-10.00386	-5.00209	-7E-05	5.00196	10.00412
27	-10.00314	-5.00178	-0.00013	5.00163	10.00342
28	-10.00317	-5.00175	-9E-05	5.00159	10.00333
29	-10.00303	-5.00174	-0.00019	5.00139	10.00307
30	-10.00313	-5.00182	-0.00023	5.00144	10.00317
31	-10.00238	-5.00131	3E-05	5.00146	10.00309

Fitting parameters for linear fits off all the DACs.

DAC number	slope	offset in Volts	stv. of fit	max. dev. in LSB
0	1.00021	0.001304	1e-05	0.22
1	1.00022	0.000148	1e-05	0.53
2	1.00026	0.000442	1e-05	0.34
3	1.00024	0.000362	8e-06	0.52
4	1.00028	0.000194	8e-06	0.57
5	1.00031	-7.2e-05	7e-06	0.63
6	1.00033	4e-05	9e-06	0.5
7	1.00027	0.000166	7e-06	0.71
8	1.00029	-8.4e-05	1.4e-05	0.3
9	1.00023	0.000174	1e-05	0.28
10	1.00029	-0.000108	1e-05	0.37
11	1.00039	-0.000164	1.4e-05	0.22
12	1.00023	5.4e-05	1.3e-05	0.17
13	1.00022	0.000244	1e-05	0.46
14	1.00017	0.000542	1e-05	0.31
15	1.00022	-0.0	1.3e-05	0.11
16	0.75027	-2.501262	6e-06	0.33
17	1.00035	-0.000176	1.3e-05	0.17
18	1.00027	9.4e-05	1.1e-05	0.24
19	1.00034	2e-06	9e-06	0.36
20	1.00027	0.000164	9e-06	0.33
21	1.00028	0.00013	7e-06	0.61
22	1.00024	0.000284	8e-06	0.45
23	1.00021	0.000354	1e-05	0.38
24	1.0003	0.000184	1.1e-05	0.14
25	1.0003	0.00016	1e-05	0.53
26	1.0004	1.2e-05	8e-06	0.47
27	1.00033	-0.0	1e-05	0.52
28	1.00033	-1.8e-05	7e-06	0.63
29	1.00031	-0.0001	8e-06	0.5
30	1.00032	-0.000114	9e-06	0.5
31	1.00027	0.000178	1.2e-05	0.14