FPGA Based Readout System for testing multi-Needle Langmuir Probe ASIC

Sondre Fortun Slettemoen

Thesis submitted for the degree of
Master in Electrical Engineering, Informatics and Technology
60 credits

Department of Physics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2021

FPGA Based Readout System for testing multi-Needle Langmuir Probe ASIC

Sondre Fortun Slettemoen

FPGA Based Readout System for testing multi-Needle Langmuir Probe ASIC

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

The multi-Needle Langmuir Probe (m-NLP) instrument is an electron density sensor made for sounding rockets and satellites. It is capable of a much higher sampling rate and spacial resolution compared to a traditional single probe system, making it possible to detect small-scale structures in the ionosphere, which is important for space weather analysis. An application-specific integrated circuit (ASIC) is being developed at UiO to replace the off-the-shelf components currently used. A field-programmable gate array (FPGA) based system has been used to test this ASIC before, but in a constrained manner as it lacked the ability to save data and was not very intuitive to use. In this thesis a read-out system for testing and reading-out of this ASIC using a PYNQ-Z2 development board was developed, improving on the downsides of the previous system. This read-out system was successfully tested, with performed measurements for the ASIC ADC and the front-end.

Nomenclature

ADC Analog-to-digital Converter

ASIC Application-Specific Integrated Circuit

DAC Digital-to-analog Converter

EIDEL Eidsvoll-Electronics

EM Electro-Magnetic

ENOB Effective Number of Bits

EUV Extreme Ultra-Violet

FE Front-end

FPGA Field-Programmable Gate Array

GNSS Global Navigation Satellite System

GPS Global Positioning System

HDL Hardware Description Language

I-V Current-Voltage

IEEE Institute of Electrical and Electronics Engineers

IP Intellectual Property

IRI International Reference Ionosphere

LSB Least Significant Bit

m-NIC Multi-Needle Integrated Circuit

m-NLP Multi-Needle Langmuir Probe

MCU Microcontroller Unit

MSB Most Significant Bit

OtS Off-the-Shelf

PL Programmable Logic

PS Processing System

SoC System-on-Chip

TEC Total Electron Content

TIA Transimpedance Amplifier

UiO University of Oslo

VHDL Very High Speed Integrated Circuit Hardware Description Language

Acknowledgements

This work was carried out in the period from January 2020 to June 2021. I would like to thank my main supervisor Ketil Røed for introducing me to space electronics and the m-NLP instrument. Thanks to Candice Quinn for all your help with my writing, as well as the help in the lab and designing the PCB. Thanks to Joar Martin Østby for answering all my questions regarding the design of the m-NIC, and a big thanks to Girish Aramanekoppa Subbarao for all your help regarding the operation of the m-NIC. I want to thank Olav Stanly Kyrvestad for all the help related to the lab and ordering new parts whenever it was needed, and Philipp Häfliger for good advice during our bi-weekly meeting. Thanks to my great uncle Olav for providing useful tips and feedback on my writing.

Thanks to everyone at 333 and SEF for making the days more fun and tolerable. Lastly I want to thank my parents, friends and girlfriend for support and patience, especially through these last weeks.

Contents

Chapter 1

Introduction

The University of Oslo's (UiO) multi-Needle Langmuir Probe (m-NLP) instrument was first developed by T.A Bekkeng in 2009 [1]. Since then, multiple revisions have been adapted for different missions and have been present on both sounding rockets and satellites. Previous versions have relied on off-the-shelf (OtS) components installed on a custom PCB. Utilizing OtS components has its advantages, such as low cost and short development time, but does come with drawbacks such as a large area requirement and high power consumption. Choosing the right components and optimizing PCB layout are two ways of reducing the effects of these downsides, but due to the larger form factor it will in most cases use more power than an ASIC counterpart. An ASIC design called the multi-Needle Integrated Circuit (m-NIC) is currently in development at UiO, with the goal of replacing all of the OtS components used in the original system. One problem with ASIC development is complexity as a small error on a revision can lead to failure of the entire chip, something which would then require a new revision, compared to a PCB where doing modifications is possible to some extent. m-NIC is currently on revision two (m-NIC2), adding additional features and changing some existing modules from the first revision (m-NIC1). For the m-NIC to become flight ready its current problems has to be resolved. Both integrated circuits (IC) have been tested to some extent before but mostly as a proof of concept. Extensive testing on each of the IC's internal module needs to be performed to properly reveal all problems, as well as testing on the system as a whole. A field-programmable gate array (FPGA) based readout system has been suggested in order to thoroughly test and prototype the ASIC system. In future revisions it is proposed to integrate a FPGA readout design into the ASIC, something this readout system would be an early prototype of.

1.1 Challenge with the current readout system

An FPGA based readout system is already in place and has been used in previous rounds of testing. Currently, the state of it is more akin to a collection of code meant for proof-of-concept measurements rather than a fully functioning readout system. One major challenge is its functionality to produce output data back for interpretation, as it currently shows output data on a HEX-display and a row of leds, and has no method of saving data to be properly analysed.

1.2 Goal

The goal of this thesis is to develop a testing focused readout system for both the m-NIC1 and m-NIC2. Something which would enable the possibility of performing measurements for characterizing the internal ADC and DAC, as well as testing the front end in both a table-top configuration and with a Langmuir probe in a plasma chamber.

1.3 Thesis outline

Given below are an overview of this thesis' structure and contents:

- **Chapter 2: Background theory** gives a brief background of the Ionosphere, plasma, Langmuir probes and UiO's m-NLP instrument. A description of the relevant parts of both the m-NIC1 and m-NIC2 will be provided. As well as an introduction to FPGA based readout systems.
- **Chapter 3: Measurement System Design** describes the m-NIC PCB and readout system developed in this thesis
- **Chapter 4: Measurement Test Setup** describes the test-setup for characterization measurements and bench-top testing of the front-end.
- **Chapter 5: Results** provides results from characterization measurements as well as bench-top testing. Results regarding the performance of the readout system will also be given.
- **Chapter 6: Discussion** discusses the measurement results, as well as the resulting readout system.
- **Chapter 7: Conclusion** concludes and summarizes the work done in this thesis.
- **Appendix** contains very high speed integrated circuit hardware description language (VHDL) code for the programmable logic (PL) design, state machine diagrams, block diagram from Vivado, constraints for I/O assignments, python code for readout and for post-processing and m-NIC PCB schematics.

Chapter 2

Background theory

Each of this chapter's four sections contain a different topic necessary to understand this thesis. A brief introduction to the ionosphere and plasma is presented in the first section. Langmuir probe theory is explained in the second section, and the third section is an overview of the present state of the m-NIC chips. Finally, an introduction to FPGA based readout systems is given.

2.1 Ionosphere

From approximately 60 km up to around 1000 km above the Earths surface lies the ionosphere [2]. The ionosphere ionized, but the ionization grade varies by orders of magnitude depending on height, position and time of day. When solar winds interact with the Earth's magnetic field, particles from the Sun are directed along the field lines down towards the polar regions. Polar regions are the areas around the geomagnetic poles and get the highest amount of mass-particles from the Sun, which can cause disturbances in the ionosphere. Disturbances like this causes accuracy problems for Global Navigation Satellite Systems (GNSS) like Global Positioning System (GPS), as described in paragraph 2.1.2.1.

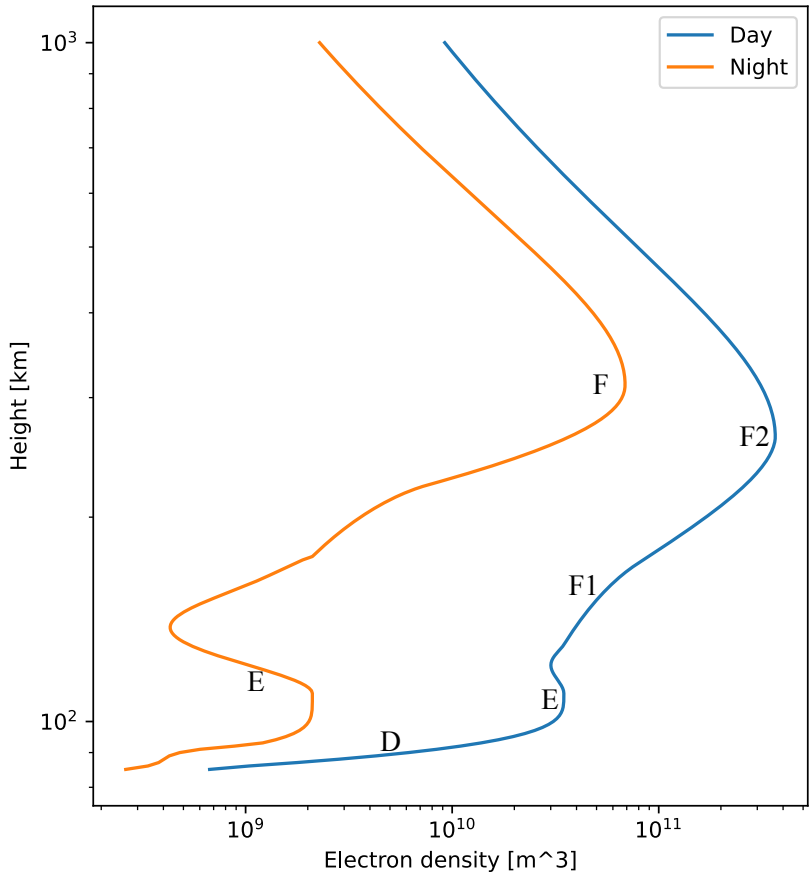


Figure 2.1: Plot of ionospheric electron density depending on day or night-time. Data from International reference Ionosphere (IRI) model [3]

2.1.1 Ionospheric regions

The ionosphere consists of five layers that can change and vary throughout the day due to solar activity. There are no firm boundaries for these regions due to the fact that the ionosphere changes its characteristics. An example of the electron density in the ionosphere depending on day or night is shown in Figure 2.1.

D Region The D region is the lowest and smallest region. Starting around 60 - 70 km above the surface, and extends to approximately 90 km. Since the density of the atmosphere is much greater there than further up, collisions between particles are more common which changes the dynamics

compared to higher altitudes. But it is still a high enough altitude that high energy electro-magnetic (EM) waves and particles can also ionize the atmosphere [4].

E Region Stretching from 90-100 km to about 120 - 150 km lies the E region. Here, collisions are much less frequent [4], and ionization rate is much higher compared to the D region. Soft X-rays and extreme ultra-violet (EUV) are the predominant drivers of ionization in this region, with the former being the highest. A higher ionization rate will then natural lead to a higher electron density.

F Region Starting around 150 km, the F region will vary greatly in height. Collisions are rare, and the ionization is mainly driven by high energy EM-waves from the sun. Sometimes the F region is referred to F1 and F2 due to changes in its characteristics depending on the time of day [4]. F1 exist only during daylight, the F region then changes into F2 at night.

Plasmasphere Above the F region lies the Plasmasphere, it is also sometimes called the inner magnetosphere. Here, the movement of the plasma is dominated by the Earth's magnetic field, and is therefore relatively stable and irregularity free compared to the lower regions [5].

Magnetosphere is the area around the Earth which is controlled by its magnetic field. It is directly affected by the solar wind, which compresses on the day side and creates a long tail of magnetic field lines on the night side. Similar to how a boat compresses the water in front and leaves a long tail behind.

2.1.2 Plasma

Plasma is a naturally occurring substance and is one of the four fundamental states of matter, it is the matter that makes up the majority of the visible universe. It consists of both neutral and charged particles that together create an ionized gas. At Earth, plasma occurs during lightning strikes and flows around the planet in the Ionosphere. On a smaller day-to-day scale plasma is also created in the fraction of a second during an electro-static discharge (ESD), for example when getting a shock from touching a metallic door handle. In modern-day technology, plasma is used in plasma TV's. To describe a plasma, one usually refers to a few parameters: electron density, electron temperature, plasma potential and magnetization.

2.1.2.1 Small-scale structures

Small-scale structures can be considered as a form of turbulence in the Plasma. Plasma blobs and bubbles are irregularities in the plasma, which can range from hundreds of kilometers to a few meters [6]. These structures are one of the challenges with space weather as it can disturb radio communication. Another issue created by small-scale structures is scintillations, which is caused by variation in refractive index of the plasma that occurs due to difference in n_e. Scintillations disrupt and change the path an EM wave has to travel, increasing the distance it has to travel as seen in Figure 2.2.

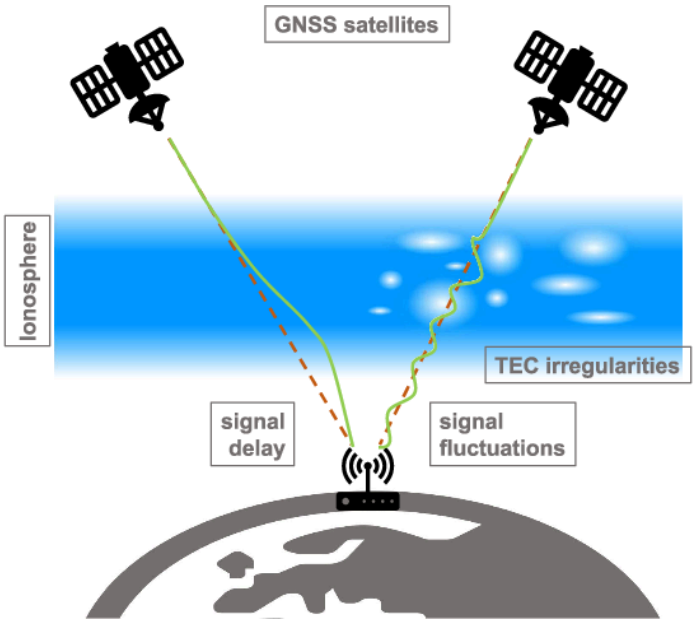


Figure 2.2: Figure showing the effects of the ionosphere on GNSS signals [7], where total electron count (TEC) irregularities is plasma bubbles which causes scintillations.

Scintillation effects on GNSS Scintillations decrease the accuracy of GNSS services such as GPS because it alters the EM waves as they travel through the ionosphere[7]. Figure 2.2 shows the difference in possible paths the waves can take through a less turbulent plasma versus one that has irregularities. The present method used to minimize the affects of a non-turbulent ionosphere is the total electron content (TEC), which is an approximation the the amount of electrons between a receiver and a transmitter [8].

2.2 Langmuir probes

Since Irving Langmuir invented the Langmuir probe in the 1920s, it has been widely used to measure different plasma parameters. When a voltage bias is placed on the probe in a plasma it will either attract or repel electrons depending on a positive or negative voltage bias. An I-V curve, as seen in 2.3, is obtained by performing a linear voltage sweep. A linear voltage sweep is a constant change in voltage that has a starting point and a stopping point. For example, 0 V to 10 V. From this sweep the electron density ,n_e, and the electron temperature ,T_e, can be determined.

2.2.1 Current-Voltage Characteristics

I-V characteristics of a Langmuir probe are divided into three different regions, ”ion saturation”, ”retardation region” and ”electron saturation”. These regions can be seen in Figure 2.3. In the ion saturated region, V_b is more negative than V_p, the negative voltage will then repel electrons and

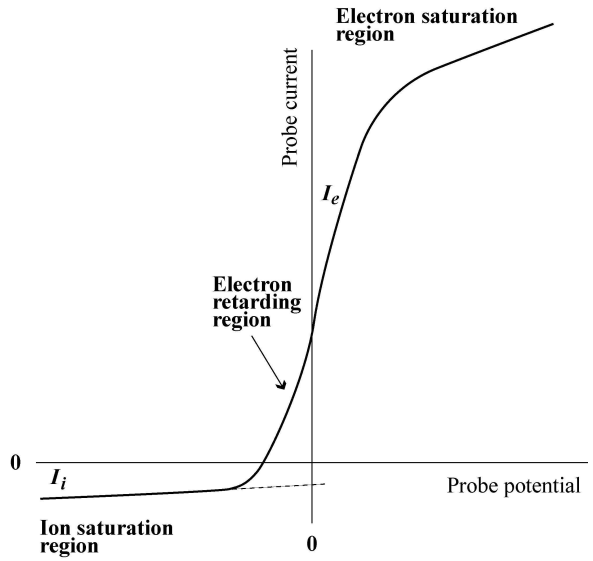


Figure 2.3: Langmuir Probe I-V characteristics illustrating the three main regions. Figure from [9]

attract ions so that the ion current dominates. As in an electronic circuit, the electrons will always follow the least resistive path to a more positive voltage. Langmuir probes will attract electrons and repel ions when V_b is more positive than V_p. In this region the electron current will dominate and increase approximately linear. In the electron retardation region a gradual shift from ion dominating current to electron dominating current will happen as the voltage becomes more positive.

2.2.2 Parameter Calculation

Electron Density The current collected from a probe with the voltage potential V is given in Eq. 2.1 as presented in [10].

$$I_c^2 = \frac{k_B T_e}{2\pi m_e}(n_e q 2\pi r l)^2 \frac{4}{\pi}(1 + \frac{qV}{k_B T_e}) = \frac{2k_B T_e}{m_e}(n_e q 2 r l)^2 + \frac{2q}{m_e}(n_e q 2 r l)^2 V \tag{2.1}$$

Where n_e and T_e are the electron density and electron temperature respectively, q is the electron charge and m_e is the electron mass. k_B is the Boltzmann's constant. r Is the radius of the probe and l is the length, V is the probe potential. The two unknown parameters T_e and n_e are separated. Since T_e is not dependent upon the bias voltage, taking the difference in current between two different biases will remove this part of the equation.

$$I_{c2}^2 - I_{c1}^2 = \frac{2k_B T_e}{m_e}(n_e q 2 r l)^2 - \frac{2k_B T_e}{m_e}(n_e q 2 r l)^2 + \frac{2e}{m_e}(n_e q 2 r l)^2 V_2 - \frac{2e}{m_e}(n_e q 2 r l)^2 V_1$$

$$\Delta(I_c)^2 = \frac{2q}{m_e}(n_e q 2 r l)^2 \Delta V$$

$$n_e^2 = \frac{m_e}{2q(q2rl)^2}\frac{\Delta(I_c^2)}{\Delta V}$$

$$n_e = \sqrt{K\frac{\Delta(I_c^2)}{\Delta V}} \tag{2.2}$$

Where $K = \frac{m_e}{2q(q2rl)^2}$, which is decided upon by the probes geometry. This derivation originates from [10].

Electron Temperature T_e is determined from the electron retardation region [1].

$$T_e = \frac{e}{k_B A_{ret}} \tag{2.3}$$

where A_{ret} represents the slope of the retardation region.

2.2.3 The m-NLP instrument

For plasma measurements in the ionosphere, the major drawback of a single-probe sweep setup is spacial resolution. Given a circular low-earth orbit (LEO) velocity of 7.5 km/s, a sweep of 1 s will give a spacial resolution of 7.5 km. To combat this the m-NLP instrument was developed [10] [1]. Four Langmuir probes are utilized here with individual voltage bias', V_b, where V_b is measured with respect to the spacecraft potential. A curve fit is performed from these four points to create an approximate I-V curve, this eliminates the need for an AC sweep. Samples can now be gathered much faster due to each probe having a static bias voltage, increasing the sampling rate and therefore the spacial resolution.

UiO's m-NLP instrument has been present on 9 sounding rocket launches, as well as a few satellites such as NorSat-1 and ExAlta-1 [11]. Both of which were apart of the QB50 nano satellite mission which is a CubeSats mission for lower thermosphere and re-entry research [12]. Together with Eidsvoll Electronics (EIDEL), the m-NLP has been made into a commercial product, and is per today the only commercial instrument capable of delivering sub meter resolution [13]. A fixed voltage bias increases the sampling rate from mHz up to the kHz range, which can reduce the spacial resolution down to the meter scale [14]. A meter scale resolution introduces the ability to see and analyze small-scale structures in the plasma, such as the mentioned plasma bubbles. Measuring these plasma bubbles will help develop a better understanding of the dynamics of the ionosphere.

2.3 multi-Needle Integrated Circuit

The Nanoelectronics group (NANO) at the Department of Informatics (IFI) has together with the 4DSPACE initiative at the Department of Physics developed two iterations of an ASIC. In the future it is desired that this chip will be used as a replacement for today's OtS components based instrument. Both m-NIC iterations consist of an analog front end, a system for serial communication and the same 16-bit capacitor-resistor hybrid (CR-hybrid) successive approximation register (SAR) ADC [15]. A 7-bit DAC and a programmable front-end controlled through a serial register was added for the

second revision. When referring to m-NIC1 or m-NIC2 in this thesis, it is referred to either revision one or two respectively. When only m-NIC is written without a number, it is meant as a reference to the series of chips and not one revision in particular. m-NICs common ADC will be described in Section 2.3.1, then both iterations of the chip will be described in the following Sections. Table 2.1 provides an overview of the functionality of both chips.

Component	m-NIC1	m-NIC2
ADC	Functional	Functional
ADC Noise	Not quantified	Not quantified
Number of channels	1	2
DAC:	N/A	Semi-functional
ENOB:	12	12
Front end:	Functional	Programmable, non-functioning
Front end noise:	Undocumented	Undocumented
Interface:	4-bit parallel bus	Serial shift register, also a non-functioning SPI module.

Table 2.1: Status overview of both m-NLP IC's.

2.3.1 m-NIC ADC

The ADC is unchanged from the first to second revision. From previous rounds of testing, the ADC has been determined to be functional but noisy [16]. The ADC has previously been calculated to have an effective number of bits (ENOB) of 12 bits in the linear region, this region is defined from 50 mV to 2.7 V. After 2.7 V, the relation between the input voltage and the converted value is no longer linear [16].

2.3.1.1 CR-hybrid SAR ADC

A SAR ADC works by using a comparator to compare the output of a DAC with the input of the ADC. It will begin by comparing the input with the output of the DAC, which will be equal to $V_{Ref}/2$. V_{Ref} is the reference voltage which determines the maximum value. In other words, it is checking if the analog input is higher or lower than the most significant bit (MSB) of the DAC. If it's higher, the MSB of the ADC result becomes a logic high, if not it becomes a logic low. The comparator output is then fed back into the SAR logic, see Figure 2.4. The SAR then moves to the next most significant bit. The same operation is done successively on all bits to find an approximation to the analog input. After finishing with all bits in the SAR, an end-of-conversion (EOC) signal will go high. If doing continuous conversions, the next clock cycle will then give the value of the MSB.

2.3.1.2 Operation

There are two different methods for reading out this ADC. By using the serial register interface described more detailed in Section 2.3.3.1, or by directly connecting to the comparator and EOC output. The ADC uses three external voltage references for the DAC. VL, VM and VH. VH is the

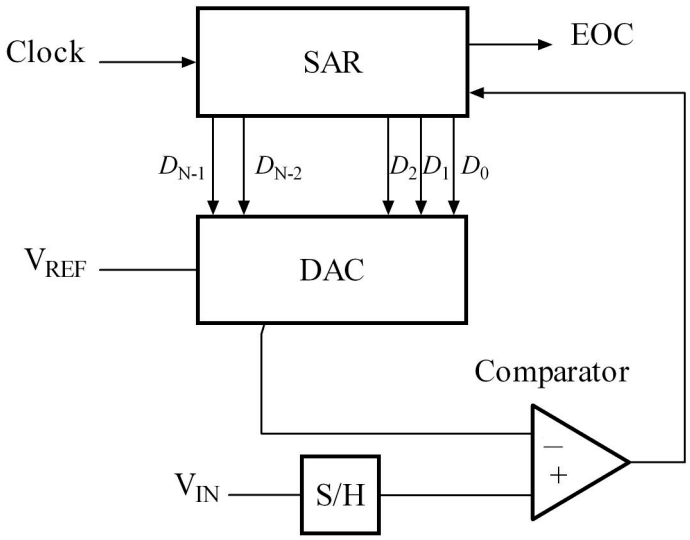


Figure 2.4: Block diagram of a SAR ADC. S/H is a sample and hold block, which makes sure the input of the comparator is kept the same for an entire clock cycle.

high voltage, earlier mentioned as V_{ref}. VM is the middle value which should be as close to $VH/2$ as possible. VL is the low voltage, and is designed to be connected to ground. The comparator is dependent on a $5\mu A$ bias current to function properly.

2.3.1.3 Propagation delay

During verification of the readout system, an undocumented property of the ADC appeared. After a conversion and EOC has been high for one ADC_CLK period, EOC should go low and MSB will be asserted high if the analog input is higher than V_M. This is not quite what happens, as there is a propagation delay for both EOC and COMP. EOC is asserted around 50 ns after ADC_CLK, and COMP after 100-300 ns. The delay of EOC is constant, while the timing of COMP varies depending on the input voltage, as shown in Figure 2.5. A voltage that is slightly greater than V_M will yeild a longer delay, while a voltage closer to V_H will result in a shorter delay. However, this is expected behaviour as the input voltage increases, the difference between this voltage and the threshold voltage to flip the most significant bit increases. A larger difference will give a faster result as the comparator output will stabilize faster.

2.3.1.4 Performance

As previously mentioned, the ADC has been calculated to have an ENOB of 12-bits in its linear range [16]. During the initial round of testing in 2018, a measurement of the output with a constant 1.65V input was performed. This measurement showed a difference of 35 least-significant-bits (LSBs) between the maximum and minimum value. This indicates a resolution of less than 12-

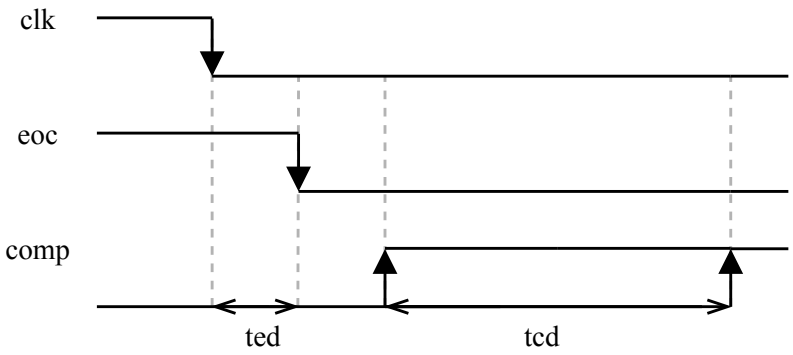


Figure 2.5: Visuilization of the propegation delay, where t_{ed} is EOCs falling delay and t_{cd} is the variation in COMPs MSB rising delay.

bits. However, this is not conclusive evidence of the ADCs ENOB, as measurements for ground noise and noise from the voltage source is not present. To date, a max sampling rate has not been formally tested. A very clear dip in the resolution and an increase in noise has been measured to be more visible once the sampling rate reaches 40-50 kS/s.

2.3.2 m-NIC1

As seen in table 2.1, m-NIC1 consists of a SAR-ADC, an arbiter and an analog front-end. Only the ADC and front-end are relevant for this thesis and are therefore the only ones described in further detail.

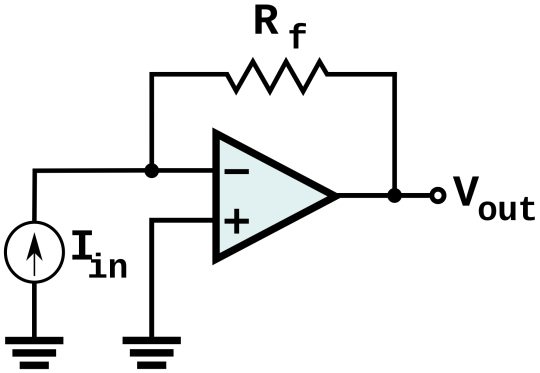


Figure 2.6: A simplified trans-impedance amplifier (TIA) [17] with a current source and feedback resistor connected to the inverting input.

Module	m-NIC1 -> m-NIC2 change description
ADC	Mostly unchanged, but now with the option of being multiplexed between two channels.
DAC	7-bit DAC was added
Channels	m-NIC2 contains two Langmuir probe channels instead of one
Front-end	Programmable gain was added, as well as a redesigned op-amp. However, a design error in the current bias circuitry rendered this module useless.
Communication	A serial register was added to configure front-end, DAC and read from ADC. An SPI module was added but is not functional.

Table 2.2: Overview of most relevant changes from m-NIC1 to m-NIC2.

2.3.2.1 Front end

A front end is required in order to convert the Langmuir probe current to a voltage so that it can be measured, this conversion is done with a trans-impedance amplifier (TIA). To enduce a current from the plasma, a bias is applied to the probe via a follower, which is a biasing method using an op-amp. Current then flows into the TIA. Due to the TIA output not being between 0 - 3.3 V, but rather between Vbias - 10 V, a level-shifter and inverter is needed to create an output voltage (OutLS) between 3.3 - 0 V.

A TIA works in principle by receiving an unknown current, which is delivered to an op-amp with a known feedback resistor, and reading the output. In this case the unknown current is the plasma current from the Langmuir probe. Figure 2.6 illustrates a simplified TIA.

2.3.3 m-NIC2

m-NIC2 Is the second and latest revision of the m-NICs. An overview of the changes made is shown in table 2.2

2.3.3.1 Serial interface

The serial interface is a custom interface designed to communicate with the m-NIC2's modules. It consists of writing to 54-bit long serial register using the SI (serial in) port, then asserting a logic high on the SWRITE port. In table 2.3a an overview of the pinout of the serial interface is described. Table 2.3b provides more detailed information about the serial register for controlling the front-end.

Number of bits #	Description	Direction
14	Channel 1	In
14	Channel 2	
8	Independent test DAC	
16	ADC Result	Out
1	ADC Select channel 1 bit	
1	ADC End-of-conversion bit	

(a) Overview of the m-NIC2 shift-register for controlling the front-end, DAC and reading from the ADC.

Number of bits #	Description
CFBc	Adds 0.5 pF to the 0.5 pF TIA feedback, will lower feedback bandwidth and noise.
G1Mc G1Sc	Sets gain of TIA.
G2Hc G2Mc	Sets gain of second stage invert and level shift amplifier.
FHc	High sets 10 kHz corner frequency for sixth order lowpass filter, low sets it to 1 kHz.
FLPc	High signal enables the low pass filter.
Vscr[6:0]	Digital input of the DAC

(b) More detailed view of the register for controlling the Channel 1/2 register in 2.3a.

Table 2.3: Overview and detailed overview of the m-NIC2 serial-register.

2.3.3.2 DAC

The m-NIC2 DAC is a 0-10 V, 7-bit DAC controlled by a serial register. It has three outputs, two of them are used for setting the screen voltage on the Langmuir probes while the third is a testing DAC (TDAC) and is meant for testing purposes. Table 2.4 presents the current issues with the DAC.

Issue	Description
Range	For the screen voltage outputs the range is not 0-10 V, but 1.5 V - 10 V. This is because the output goes through a buffer to which does not give any less voltage than 1.5 V.
MSB Switching spike	Another distortion appears at higher frequencies. A spike is visible at around 5V, when the most significant bit changes. This effect is caused by two things. One reason is shifting further in the R2R network that is used in the DAC. The second reason is due to switching between the PMOS amplifier and the NMOS amplifier. When the PMOS is switched off and the NMOS is on. This was a distortion which was also noticed during simulation.
Probe output bias	There is a design flaw with the biasing circuitry for the DAC. The main current reference does not provide enough current for the output buffers. Both VSCR1 and VSCR2[1] are then affected, and will not be able to drive even small loads and will also struggle at higher frequencies. This effect was noticed during testing with the readout system developed in this thesis.

Table 2.4: Overview of the issues related to the m-NIC2 DAC.

2.3.4 Existing readout method

During previous testing of both chips, an older Cyclone 2 FPGA board from Intel was used to test the individual modules. In adition to the FPGA, it contained a row of LEDs, switches, headers and four 7-segment display. Using this method, the system could display data on the 7-segmented displays or an array of leds. Averaging filters and the option to display max an min values, it was able to do some debugging and analisys. Saving data was then performed by manually writing down values on a spreadsheet, this was the systems largest limitation. Doing formal characterization would be impossible with the current system, as it is only possible to do measurements on DC signals or very slow mHz waveforms.

2.4 FPGA based readout systems

FPGAs are a popular tool used in readout systems. There are a plethora of reasons for this, for example flexibility in interfacing non-standard communication protocols as well as speed, as FPGAs can process large amounts of data in a parallelized fashion.

2.4.1 FPGA Introduction

An FPGA is primarily built up of Configurable Logic Blocks (CLB), which are essentially blocks programmed to give a certain output given a specific input. Most common building blocks of a CLB are of Look-up-tables (LUT), flip-flops (FF) and often a multiplexter (MUX). Figure 2.7 illustrates the build-up of a CLB. These are the building blocks of an FPGA and are connected togheter with interconnects to create a large mesh of programmable logic. Other blocks such as Block Random Access Memory (BRAM) and Digital Signal Processing (DSP) cells are also a part of an FPGA, but serve specific purposes related to memory or calculations.

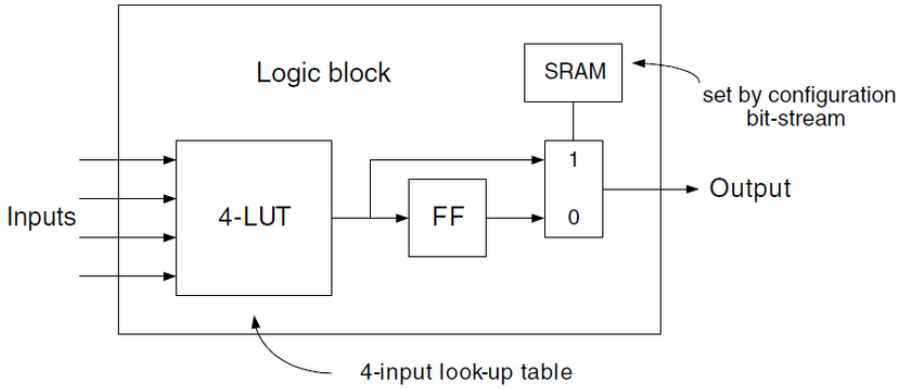


Figure 2.7: Example CLB with a 4-bit LUT, SRAM block and FF. Taken from [18]

2.4.2 Advanced eXtencible Interface

AXI is first and foremost a high bandwidth, parallel, multi-master and multi-slave interface. It is mostly used for on-chip communication. As other other high bandwidth interface utilizes, AXI uses one channel for each ”type” of communication. This means that there is a seperate bus for data, addresses, ready signals etc. Compared to a 3-wire interface like SPI where the different words (address, data) will be transferred after each-other sequentially. AXI has a separate channel for both read/write addresses and data. There are three types of AXI4 interfaces: AXI4, AXI4-Lite and AXI4-Stream, where the latter two are the focus of this thesis. In this thesis the focus will lie on AXI4-Lite and AXI4-Stream. From here on, AXI4-Stream will be referred to as AXIS and is designed for high-speed data streaming. AXI4-Lite is for simple, low-throughput memory-mapped communication. Both AXI4-Lite and AXIS are compatible with different clock frequencies on the master and slave side.

2.4.2.1 Master/Slave

The AXI interface is based on the Master/Slave model of communication. This means that there will be one device/interface which will be the master, and one or multiple slaves that will follow the

masters instructions. A master will initiate and control the communication, the slave then follows the given instructions.

2.4.2.2 AXI4-Lite

The AXI4-Lite interface consist of five different channels, with read and write channels for both the address and data channel. A write response channel is also present, as the slave will acknowledged a write operation performed by the master. AXI4-Lite is able to do both read and write operations simultaneously, which mean that communication can flow between the master and slave at the same time.

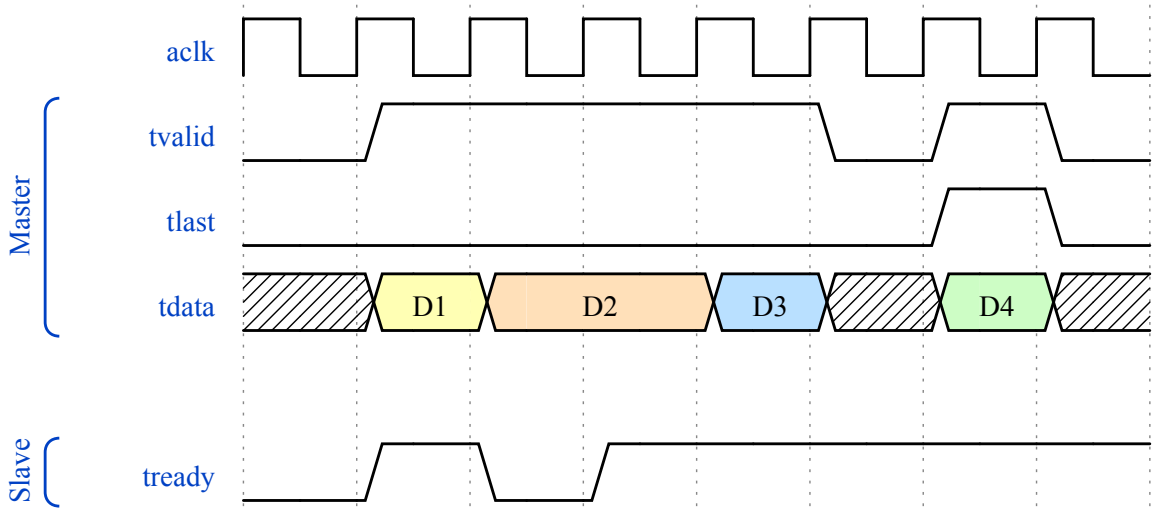


Figure 2.8: Example waveform for a shared clock AXIS transfer, functionality of both tvalid and tready are shown.

2.4.2.3 AXI4-Stream

AXIS is different from AXI4-Lite due to it being a one way data transfer protocol, as it can only transfer data from the master to the slave and not the other direction. The advantage of AXIS comes from the fact that the amount of data to be streamed is unlimited. AXIS utilizes a READY and VALID bus. The slave pulls READY high whenever it is ready to read data, and the master pulls VALID high when it has data ready. A transaction is done when both READY and VALID are high. An optional LAST signal is asserted high for one clock cycle when the master is finished streaming. An example illustrating this behaviour is shown in Figure 2.8.

2.4.3 PL - PS Hybrids

A modern development in SoC FPGA technology is the combination of programmable logic (PL) and a processing system (PS). Combined it has the benefits of an FPGA which can run computa-

tions in parallel and easily interface a non-standard communication protocol, And the single thread performance of a PS, along with the PS running an OS which often has functionality to interface an external PC/CPU that can handle and save data for later analysis.

2.4.3.1 PYNQ-Z2

PYNQ-Z2 utilizes a Zynq7020 which is an aforementioned SoC FPGA, but which is specialized in low development time and usability. On the PS, and embedded Linux version is running a Jupyter notebook. This means that data can be transferred from the PL directly to a python environment, using only a single python command with a custom library.

Chapter 3

Measurement System Design

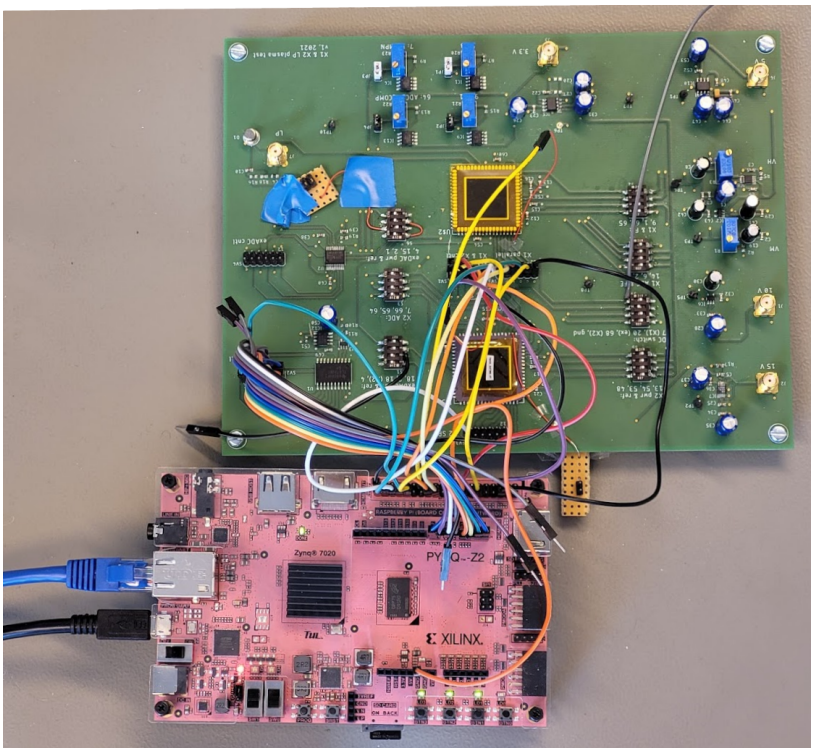


Figure 3.1: Picture of the PYNQ board (bottom) and m-NIC PCB (top) connected together.

In this chapter the measurement system designed will be described, Figure 3.2 gives an overview of the whole system from a broad perspective where a single ADC/DAC combination is used. Section 3.1 describes a PCB containing both revisions of the m-NIC, as well as an OtS ADC and DAC. Section 3.2 describes a readout and control system developed for this PCB.

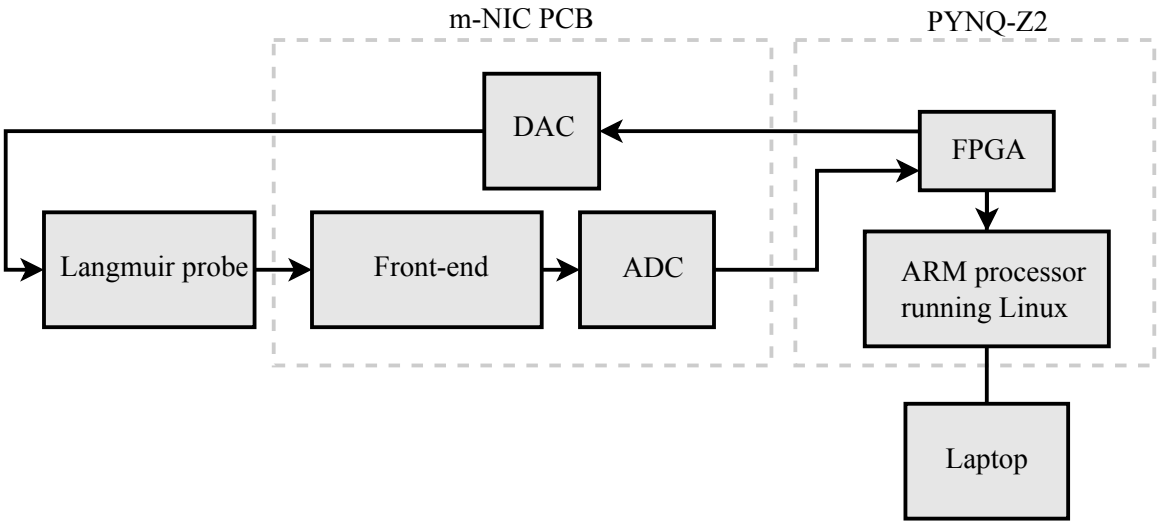


Figure 3.2: Block diagram showing the connection between the m-NIC PCB and PYNQ-Z2 based readout system developed in this thesis.

Component name	**Brief description**
m-NIC1	First revision m-NIC, see 2.3.2
m-NIC2	Second revision m-NIC, see 2.3.3
MAX1133	16-bit, 200 kS/s ADC
TLC7226	8-bit, 0 - 10 V DAC

Table 3.1: Overview of relevant PCB components

3.1 m-NIC PCB

The m-NIC PCB is designed by PhD student Candice Quinn in which both m-NIC chips and reference converters are mounted on. Reference converters are included to be able to perform the same tests as with the m-NIC converters, but with a component which is already characterized and documented by the manufacturer [19] [20]. Both reference components were chosen based on having a higher ENOB, and will therefore be able to produce higher resolution result that may reveal more information compared to the internal converters. Figure 3.3 shows a simplified block diagram of the schematic where only connections between each component is shown and external connections are not included. Table 3.1 gives a short overview of the relevant components on the PCB.

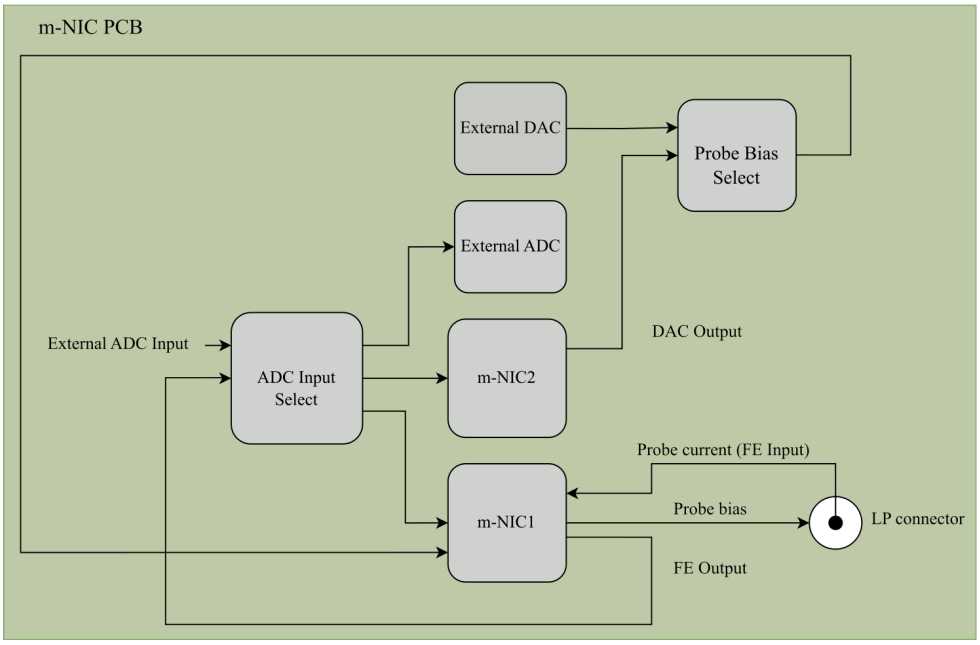


Figure 3.3: Block diagram meant for illustrating the connections between the different PCB components. Not visible in this Figure: Power management, I/O connections and other general circuit components not necessary to understand its function.

3.1.1 Post-assembly PCB Modifications

Prior to the PCB's design, the chips had never been tested together and under the same conditions before. A complex PCB which is being tested for the first time have a high chance of containing some sort of error. After component population some modifications had to be performed in order for it the PCB to function as desired.

Change	Description
Comparator and EOC trace cut	Both m-NICs comparator output, as well as the digital-out (DOUT) port of the MAX1133 ADC were connected to the same header. Since only one ADC were to be active at any given time, this was not seen as an issue but a way to reduce the amount of cables. However, during testing the comparator output was not able to be driven to 3.3 V, and was instead only able to reach around 1V. To solve this, all three signals had to be separated and brought out on headers. A simple trace cut was then performed on m-NIC1 and m-NIC2 comparator trace, while DOUT was left on the original header. m-NIC2s EOC signal was also separated from m-NIC1s EOC in the same fashion as the comparator outputs.
MAX1133 Power Rails change	After production, it was noticed that the digital power rail of the MAX1133 was designed for $5\,\mathrm{V} \pm 0.25\,\mathrm{V}$, a voltage that is too high and potentially harmful to the electronics on the PYNQ board. This did not become an issue as the ADC operated the same way with a DVDD of 3.5 V. A trace cut to open the connection between AVDD and DVDD was made, as well as soldering on a header to provide 3.5 V to DVDD.
Change gain non-inverting op-amp	The output of the TLC7226 was connected to a TLV217 op-amp in a non-inverting configuration with a gain of 3. Therefore, all outputs greater than 3.3 V from the DAC became 10 V from the output and any input signal greater than 3.3 V became saturated. To solve this, the feedback resistor was replaced by a $0\,\Omega$ resistor, bringing the gain down to 1. Full range off the DAC was still not achived as the maximum output of the op-amp was VDD-0.5, resulting in about 95% of the DAC output range being used.
Trace cuts for reference IC	The REF5010AIDR IC is used to provide a stable 10 V reference voltage for TLC7226. Three of its pins which were specified to be unconnected was connected to ground [21], and did therefore not function properly. By cutting these traces, or snipping its legs the connection was eliminated and the reference circuitry performed as intended.

Table 3.2: Table describing the modifications made to the PCB after component placement.

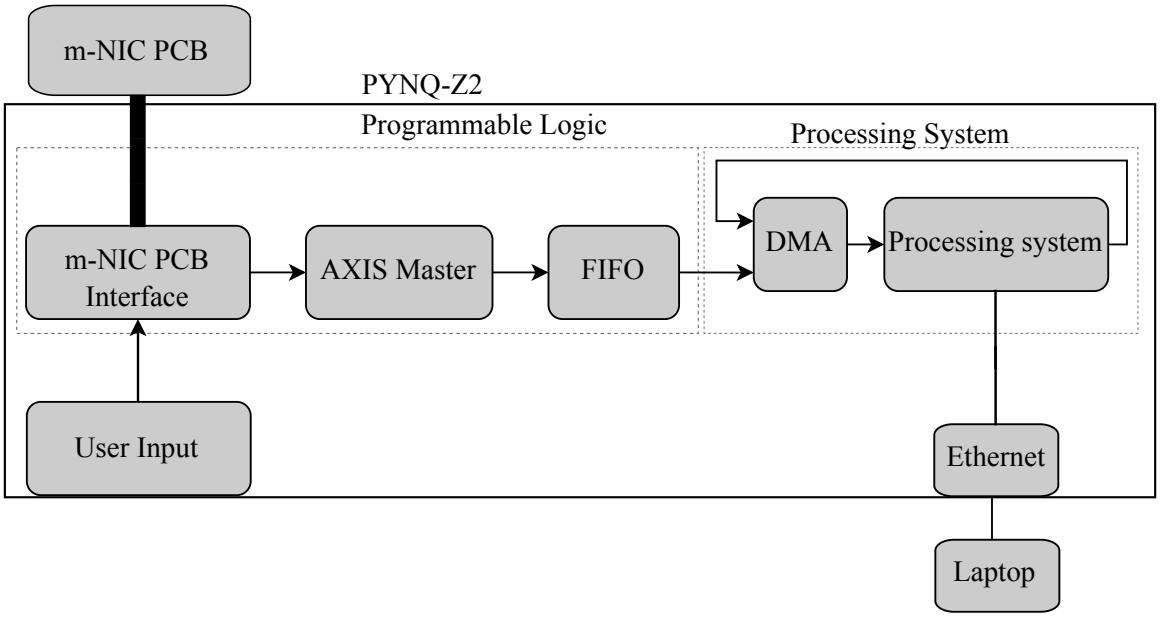


Figure 3.4: Simplified block diagram of the readout system.

3.2 Embedded readout system design

In this section the PYNQ based readout system will be presented.

3.2.1 Overview

The readout system itself contains two main modules referred to as the ”m-NIC PCB interface” and ”Data transfer module”. m-NIC PCB interface connects to the m-NIC PCB to control, configures and readout from its components. Then the data that is going to be sent to the data transfer module is selected. AXIS Master, a 32-bit word module that takes this data and sends it to a FIFO using an AXIS interface. From there the data is gathered by the DMA which then again delivers data to the PS. Now the data is accessible to the user through the python environment which is further explained in Section 3.2.5.1. The Jupyter notebook runs on the Linux operative system and is accessible through Ethernet.

3.2.1.1 User Input

After the SoC is programmed the user can control certain aspects of it operation with button and switches. An active high reset signal is mapped to a button on the PYNQ board. Problems can appear when the button signal is not a square wave, which the output of buttons rarely are when being pressed by a person. To counter this a debouncer is used, and works by detecting activity from the button, then sending a generated square pulse as a replacement for the pure button output. There

are also two switches on the board, SW0 and SW1 that are used to determine which mode is used, as seen in table 3.3.

Mode[0:1]	m-NIC2 DAC	External DAC	m-NIC1/2 ADC	External ADC
”00”	Enabled	Disabled	Enabled	Disabled
”01”	Enabled	Disabled	Enabled	Disabled
”10”	Disabled	Enabled	Disabled	Enabled
”11”	Disabled	Enabled	Disabled	Enabled

Table 3.3: Table explaining what modules are being used for different mode inputs.

Since the physical buttons and switches are not enough to control all the modes of the FPGA design, there is also used an IP module added called ”Virtual Input/Output”. To be able to use this module one must use the Vivado Synthesis Tool [1] to program the FPGA. Table overviews the operation of this module.

Input	Description
DAC_DATA[7:0]	Selects input data for the DAC when DC mode is selected.
SWEEP_FREQUENCY[20:0]	Selects the divider for the SWEEP clock provided for sine and sawtooth generators.
WAVE_TYPE[1:0]	Selects a sinewave(”00”), sawtooth(”01”), DC mode (”10”) or random waveform (”11”).

Table 3.4: Description of input possibilities with the VIO module.

3.2.1.2 Clocking

The internal main clock (mclk) is a 100 MHz clock from the IO Phase Locked Loop (PLL) clock source. All interconnects and IPs, as well as the m-NIC PCB interface uses this clock. During programming of the FPGA through Jupyter, only the IO PLL will be considered. What this means is that if the design utilizes the ARM PLL source with a generated clock of 80 MHz, the PYNQ will select the closest frequency to that but using the IO PLL clock source. Due to these sources being divided from different oscillators with different base frequencies the resulting frequency might be 79 MHz, without any warning given to the developer. A problem which was noticed after analyzing the ADC data where it seemed to be out of sync from the same data recorded by as oscilloscope.

To create clocks used in other parts of the system, the main 100 MHz clock is divided into slower frequencies. These clocks are generated and sent out to either the m-NIC or one of the other external components, there are problems with this method however. Generating clocks and using them in the FPGA before they are sent out to external pins sends the clocks into the interconnect mesh which

[1] Vivado Synthesis Tool is Xilinx’s tool for creating designs, synthesis, implementation and programming of a physical FPGA.

#	Description
R.1	Inexpensive. Due to speed requirements being low compared to today's technology it should not cost more than a low-cost FPGA development kit, this would mean to less than 3000 NOK.
R.2	Speed, it should be able to handle a transfer-rate of 1.92 Mb/s.
R.3	Ease of use, with the ability to operate with little to no FPGA knowledge.
R.4	Reusability and further development

Table 3.5: Summary of the most important requirements related to the FPGA based readout system.

is inside the FPGA. Causing unnecessary timing delays for the clock compared to on a dedicated clocking tree.

3.2.2 Requirements

As mentioned in [16] the desired sampling rate of the internal ADC is 20 kS/s and requires a clock frequency of 340 kHz [2], this means a data transfer rate of

$$20\,\mathrm{kbit/s} * 16\,\mathrm{bit} = 320\,\mathrm{kbit/s}$$

must be achieved. This will cover the transfer rate for one ADC at 20 kS/s. In the future the goal is four channels at 20 kS/s, as well as the 8-bit DACs. Including this the requirement is now

$$4 * 20\,\mathrm{kS/s} * 16\,\mathrm{bit} + 4 * 20\,\mathrm{kS/s} * 8\,\mathrm{bit} = 1.92\,\mathrm{Mbit/s}$$

With the current technology this is well within what is expected to be achieved. Expected resource utilization was not a great concern, therefore speed and resource specifications were not qualities which were deemed highest priority.

What is important, however, is finding a development board which would require minimal effort to get started and would also be easy to use while performing measurements. There is also a need for 30-40 input/output (I/O) pins. In the end the TUL PYNQ-Z2 board was selected as the platform to be used, as the board focused on fast development time and flexibility due to having a processing system running Linux. It allows the user (read: not designer) to need no experience with the any FPGA toolchain, but instead program and control the SoC FPGA via python. Requirements for this system are summarised in Table 3.5.

3.2.3 m-NIC PCB interface

In this section the module which controls the components on the PCB is described, it is callen "m-NIC PCB Interface"[3] contains three other modules: External ADC Control, m-NIC ADC Control

[2] It is actually 20.008 kS/s and 340.136 kHz, due to clock division due to 100 M/340 k not being an integer

[3] If looking through the code in the appendix, this module will be called pcb_interface_v3 and has changed name in the thesis to make it more readable.

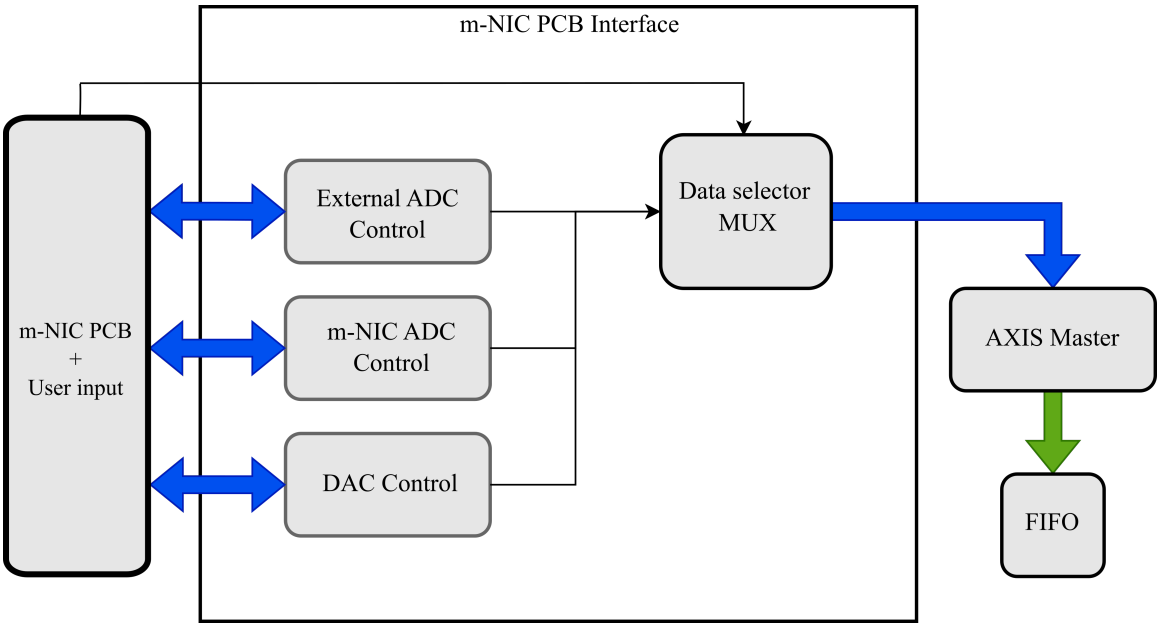


Figure 3.5: Block diagram of the m-NIC PCB interface, which is the hardware abstraction layer (HAL) between the m-NIC PCB and processing system.

and DAC Control. These three modules are together responsible for controlling, configuring and reading-out the physical ADCs and DACs. A MUX controlled by SW1 and SW0 is used to select which data is stored, this is described in table 3.3.

3.2.3.1 External ADC control

This module is responsible for configuring and reading out from the MAX1133 ADC. Figure 3.6 illustrates the timing diagram of the ADC. Internal clocking mode is selected, which means that the internal clock of the ADC is used for the conversion and the external clock provided by the PYNQ board is used for communication only.

Clock generation SCLK is the external serial clock for MAX1133 and should be kept low during the conversion period to improve noise performance [19]. To solve this an enable signal and an and gate was introduced to the output. However, glitches in the combinational logic could then lead to missed clock cycles and short pulses. A common workaround for this issue is most often a change in the structure of the module which is being controlled, which is not possible in this situation.

3.2.3.2 m-NIC1/2 Comparator method

One method of reading out from the adc implemented in both mNIC1 and 2 is to directly read out from the comparator port. This is done by looking at the COMP output and the EOC signal, and

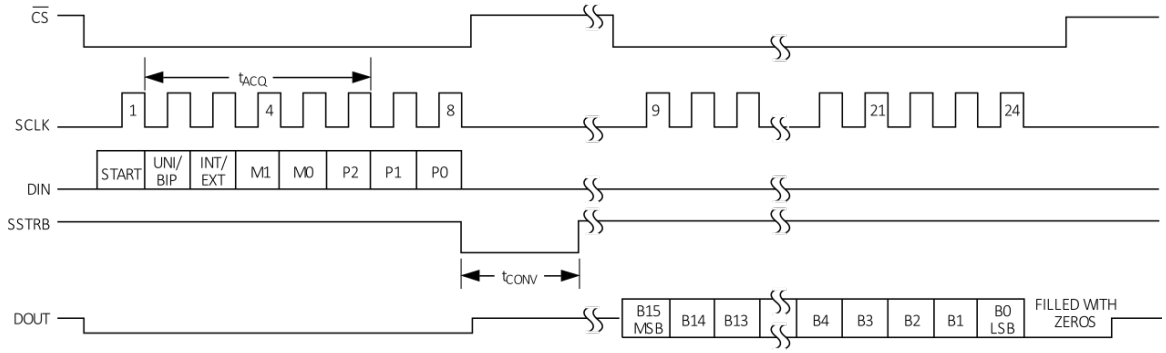


Figure 3.6: Inputs and outputs of the MAX1133 ADC during a conversion. From the MAX1133 Datasheet [19].

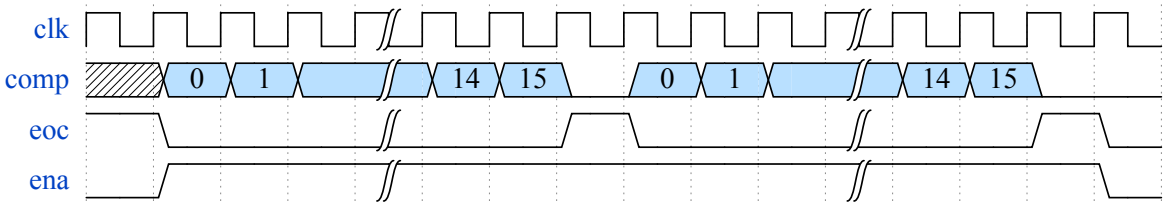


Figure 3.7: Detailed timing diagram of two convertions for the m-NIC ADC, looking at the eoc, comp and ADC enable ports. Where index 0 represents the MSB.

is illustrated with an example in Figure 3.7. A sample of the produced data from this method is shown in Figure 3.8. An unexpected difference between this method and the serial register data was observed, as reading directly from the comparator port led to a signal containing much larger levels of noise, as well as abnormally large spikes. The reason for this is most likely that the comparator output will take time to settle if the input is close to the threshold between high and low. Reading this value too early might then result in an interpreted value much higher or lower than the previous and proceeding values.

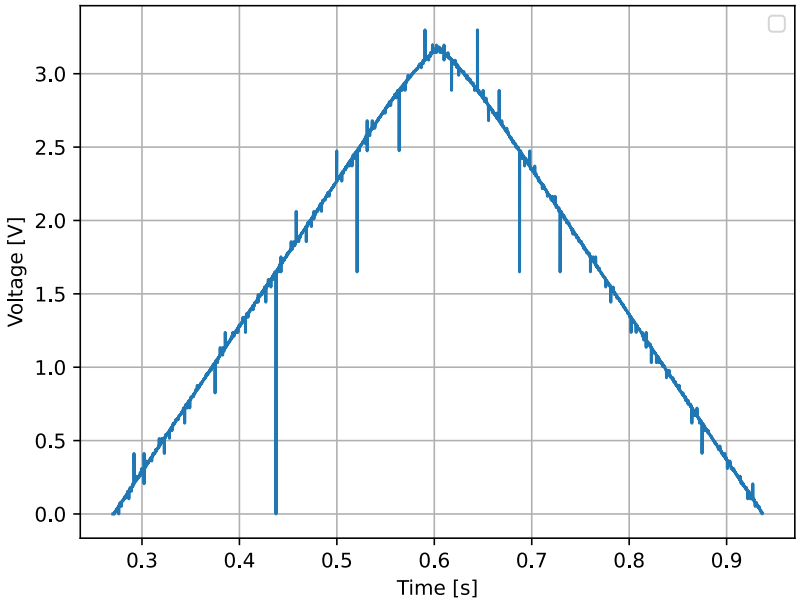


Figure 3.8: Readout results from the m-NIC1 ADC by using the comparator and EOC port, showing large spikes due to misinterpreted data. Input waveform is a 1.5 Hz triangle wave.

3.2.3.3 m-NIC2 Shift-register control

Controlling the shift register is done with 4 ports and a clock input for SCLK. These 4 ports are SWRITE, SREADB, SO and SI, where SI and SO are data input and output to the register respectively. To perform a write operation, data starts to shift out on the SI port. After 53 clock cycles SWRITE should then be pulled high to perform a write operation. SCLK must be low during this write pulse. A faster write operation can be performed by pulling SWRITE high earlier. To read, a similar operations is performed. SREADB, which should be high when not active, will go low for one SCLK period. 18 bits will then be fed out of the SO port, where the first 16 are the last converted ADC value. Due to the previous readout system had already created a module to read from the shift-register, this module was slightly modified and re-used for this system.

3.2.3.4 DAC control

DAC control function is to control both the internal and external DAC. It is the only module containing sub modules, as it creates different waveforms which is used by both DACs.

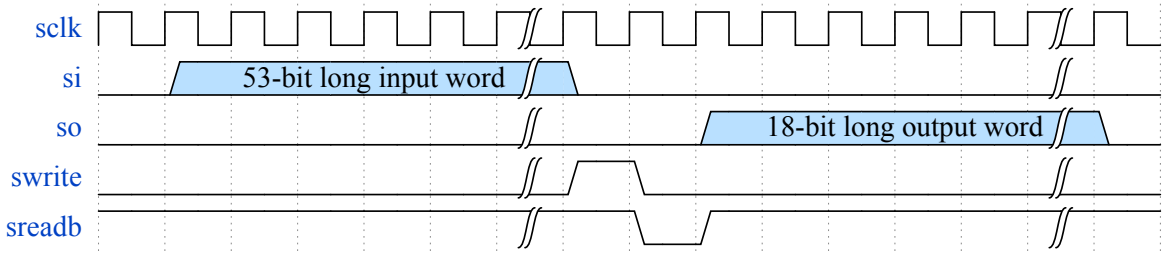


Figure 3.9: Timing diagram for the m-NIC2 serial register with one write operation and one read operation.

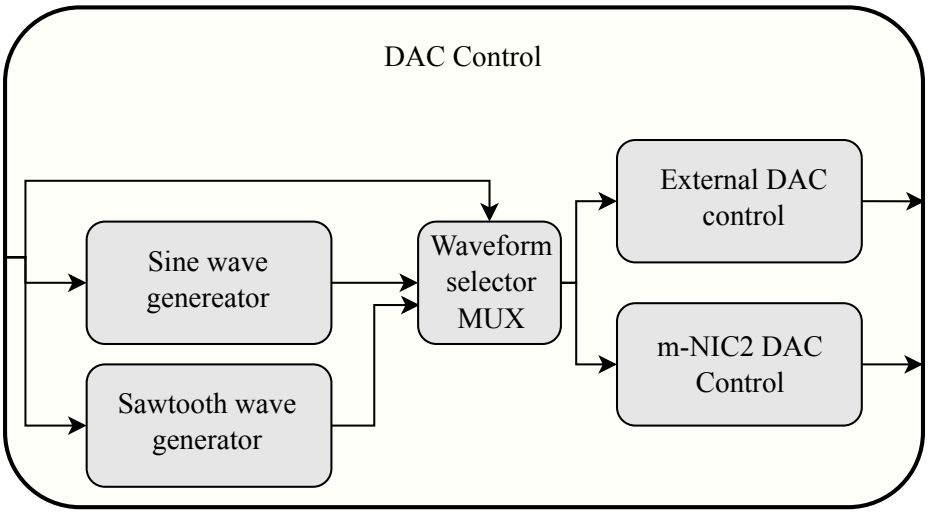


Figure 3.10: Block diagram of the DAC control module, showing the internal structure.

Random DAC Waveform To create a pseudo-random signal for the DAC, a 16-bit 4-tap[4] Linear-Feedback shift Register (LFSR) is used. The last 8/7-bits, depending on the DAC, are used for the DAC input. The reason for choosing a 16-bit length is due to the fact that an 8-bit LFSR would create a 128 bit long sequence, while with an 16-bit LFSR the sequence is 65536 bits long. With a longer sqeuence it will take longer before it repeats itself.

3.2.4 Data transfer

Transferring data consists of four main components: AXIS Master, FIFO, DMA and the PS wrapper. A port called TDATA_ASYNC is a 32-bit bus that feeds into the AXI stream master, and the 32-bit word is shown in table 3.6. The AXIS master is generated from the Vivado tool, and has only a slight modification to it. An internal signal called stream_data is set to be the aforementioned TDATA_ASYNC input, resulting in the module functioning to something akin a translator from a

[4]An LFSR feeds some elements in a register back to the input of the register. All feedbacks go through an XOR port with another ”tap”, which is what a feedback is called. A 16-bit 4-tap then means a 16-bit register with 4 feedbacks.

Bits	Description
0-15	Selected ADC data
16	Synchronize wave from the waveform generator
17	Synchronize wave output from the DAC
18	Oscilloscope trigger value
19-22	Always low
23-30	DAC value
31	Always high, as this will keep the length of the same when doing string manipulation in python

Table 3.6: Overview of 32-bit word transferred from the PL design to the DMA.

32-bits bus to AXIS. These AXIS signals are then connected to an AXIS FIFO IP. Both the FIFO AXIS slave and output of the AXIS Master are driven by the same clock, this is described in the next paragraph. AXI FIFO is configured with a depth of 32768, which is then sent to the DMA at a 100 MHz rate. A clock which is used for both the output of the AXIS master and FIFO slave is generated to be the same as the ADC sampling rate.

3.2.5 Software

Python scripts were run in Jupyter to retrieve and plot the data which were received from the FPGA. Jupyter is a python environment run on the PS and is accessible on a PC connected to the PYNQ board. Other code was a simple python program used for plotting and simple analysis.

3.2.5.1 Jupyter

Programming the FPGA is done in Jupyter by using a python Pynq library, that also contains functionality which allows for controlling the DMA. Data capture is initiated by calling a transfer function from said library and waiting for a buffer to be returned, where one buffer is 32-bit wide and $2^{15} = 32768$ long. As previously mentioned, the 32-bit word from the PL contains different data. By reading this word as a string, it is possible to use string manipulation in python to save each individual type of data to variables. After this, the data is then stored in columns in a comma separated values (.csv) file.

Chapter 4

Measurement setup

In this chapter, setups for characterization measurements of the internal converters and testing of the front-end is presented. Two front-end setups are described, with one using a current source and the other using a biased Langmuir probe in a plasma chamber as the input.

4.1 Characterization setup

The m-NIC ADC and DAC are not formally characterised. This needs to be done in order to continue further testing of the chip. Both the ADC and DAC characterization setups utilizes a digital oscilloscope for analog measurements. This is to analyse the DAC output and ADC input. More specifically it is a Keysight DSOX1202G, which is a 200 MHz 2GS/s scope. More in depth description of the setups are found below in Section 4.1.2 and 4.1.3. Since the m-NIC2 DAC has problems with the output buffer, as mentioned in Section 2.3.3.2, it's the TLC7226 output which will be analyzed.

4.1.1 Requirements

For proper statistical analysis of both the ADC and DAC enough samples must be captured. Based on the Institute of Electrical and Electronics Engineers' (IEEE) standards, a minimum of 2^{22} samples will be needed per waveform for ADC analysis and 2^{18} samples for the DAC[22] [23]. There are different requirements for types of waveforms for the ADC and DAC, all waveform types and frequencies can be seen in Table 4.1 for the ADC and Table 4.2 for the DAC.

4.1.1.1 Frequencies

There are three different categories of waveform frequencies which are to be tested: Fine, Medium and Coarse [22] [23]. Medium and coarse are selected to both cause and not cause errors with aliasing, while fine frequencies are selected to specifically get hits on all the converters codes. To do this Eq 4.1 is used. "Where J is an integer that is relatively prime to M. f_{UPDATE} is the update

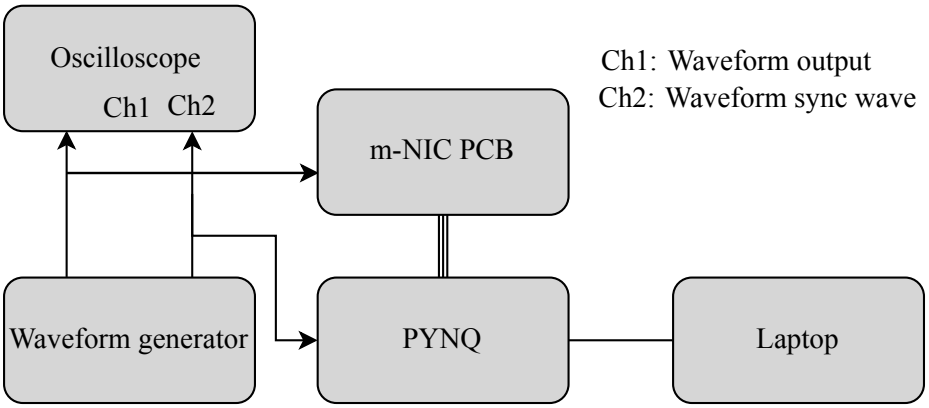


Figure 4.1: Block diagram of ADC characterization setup

rate and M is the record length." [22].

$$f_i = \frac{J}{M} f_{UPDATE} \tag{4.1}$$

4.1.2 ADC

ADC characterization is performed by sending a known signal to the ADC and analyzing the output. For the signal source, a Keysight 33500B waveform generator is used. A splitter connects the waveform generator output to channel 1 on the oscilloscope. The other side of the splitter connects to the ADC_EXT_IN pin on the m-NIC2 ADC. This pin can be selected as the ADC input by pulling ADC_EXT_SEL to a logic high. Keysight 33500B also has a sync output, a square wave with a 50% duty cycle where one period equals one period of whatever waveform is selected for channel 1. Channel 2 on the oscilloscope and pin AR11 the Pynq board receives this sync wave. By doing this the data from the oscilloscope and PYNQ board can be synced up, despite not sharing absolute time or sampling rate. Figure 4.1 illustrates the setup connections.

4.1.2.1 Trigger

To ensure that the data capture starts at the same time for both the Pynq board and oscilloscope, there is introduced a shared trigger by creating a signal which is 0 when the system is being reset. Calling the data capture function described in Section 3.2.5.1 starts the capture, as long as the system is not actively being reset. Pressing the reset button before calling this function, to then release it will start the capture and pull the trigger high. A high trigger will also start the data capture on the oscilloscope.

Categories	Requirements	Description
Scale	Full scale (0 - 3.3V) Attenuated	0 to 3.3V 0 to 2.97V
Waveforms	Sawtooth Sine wave	
Frequencies	Fine Medium Coarse	1.273 Hz, 17.867 Hz, 369.41 Hz, 1 kHz 3.01 kHz, 5.01 kHz, 6 kHz 7 kHz, 8 kHz, 9 kHz

Table 4.1: Overview of the waveforms and frequencies to be used during ADC characterization, based on a sampling rate of 20.008 kHz.

4.1.3 DAC

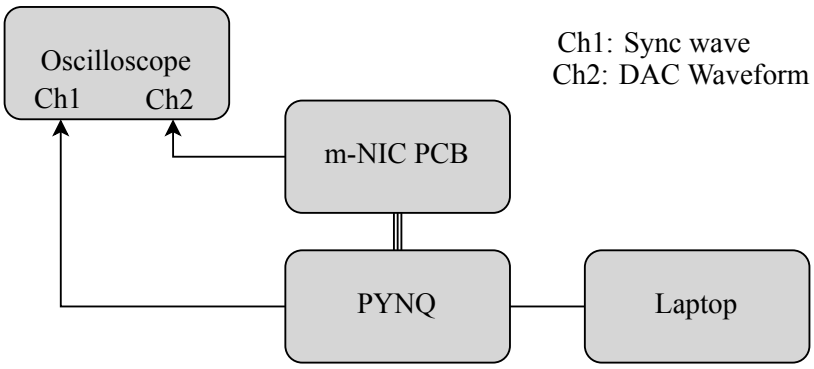


Figure 4.2: Schematic of DAC characterisation setup

Characterizing the DAC is similar to the ADC characterization setup, but does not require a waveform generator as the waveform is generated on the PYNQ board and sent to the DAC. Channel 1 on the scope is connected to the DAC output, and channel 2 is connected to a pin on the PYNQ board, which produces a square wave of the same nature as the sync wave generated from the waveform generator in Section 4.1.2. Due to issues with the m-NIC2 DAC, the TLC7226 reference DAC will be used for all DAC related measurements instead.

Categories	Requirements	Description
Scale	Full scale	0 to 3.3V
Waveforms	Sawtooth Sine wave	
Frequencies	Fine Medium Coarse	2.86 Hz, 77.6 Hz, 1181.98 Hz, 1457.4 Hz 3161 Hz, 4709 Hz, 6022 Hz, 6595 Hz 17658 Hz, 21467 Hz, 23152 Hz, 24468 Hz

Table 4.2: Waveforms and frequencies to be used during DAC characterisation, based on a sampling frequency of 50 kS/s.

4.2 Front-end table test

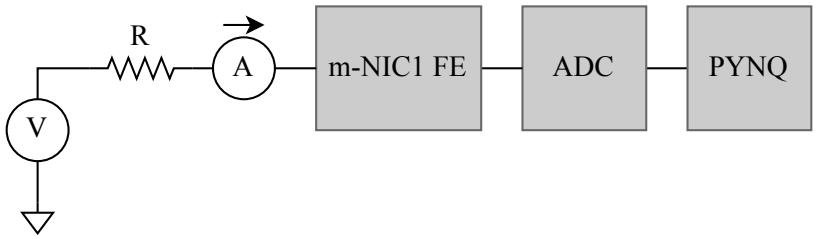


Figure 4.3: Measurements setup for testing m-NIC1 front-end, the arrow indicates the direction of a negative current.

In the previous section, a setup for capturing ADC and DAC measurements was described for doing characterization of both modules. Now, the ADC will be used together with the m-NIC1 front-end to perform a functional test of the chip. As a way to test that the setup is done correctly and that the PCB connections perform as intended, a mock-LP run is performed. A mock-run in this case will be to simulate a positive biased Langmuir probe in a plasma, drawing a negative current. If this tests works, the system should then be ready for measurement with a Langmuir probe in a plasma chamber. To simulate a probe, a voltage source and resistors in the $M\Omega$ range is used and connected to the InS port of m-NIC1.

4.2.1 Probe-current vs output voltage

From simulations done during development of the m-NIC front-end, the measurable current range is shown to be from 1 nA - 2500 nA [15]. After the simulation however, the front-end has not been characterized. To actually quantify the performance of the system as a whole, the relation between the collected current ,I_c, going into the TIA and the produced output voltage on OutLS must be found. This is done by sending in a known current and measuring the output, by doing this for multiple input currents an I-V curve can be found.

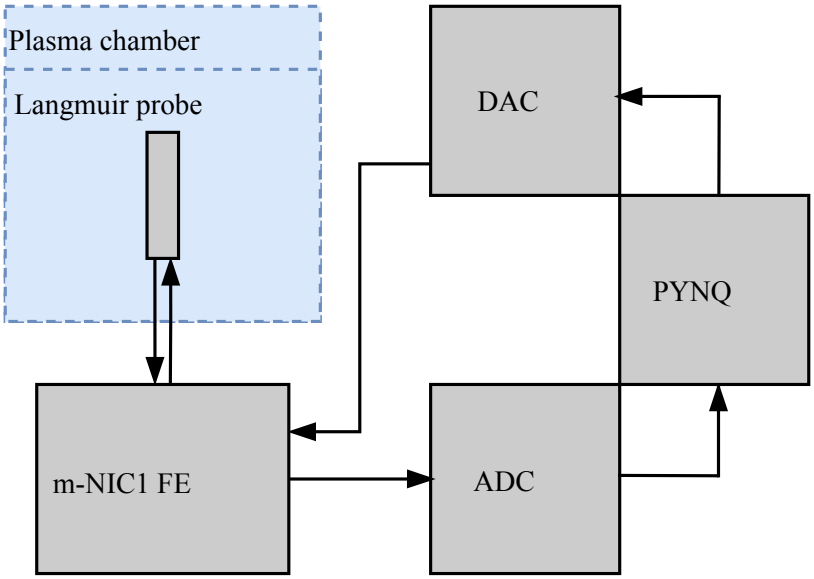


Figure 4.4: Measurements setup for plasma chamber testing

4.3 Proposed plasma chamber test

Using the m-NIC with a Langmuir probe in a plasma environment has never been done before and is not only a test of the readout system, but the functionality of the ICs themselves. The setup closely resembles the one described in Section 4.2, but the current source is now switched with a biased probe in a plasma chamber, as seen in Figure 4.4. Instead of a power supply, the external DAC TLC7226 is used to provide a voltage bias to the front-end which in turn biases the probe, this creates a negative current draw to the front-end input.

Chapter 5

Results

This chapter will present measurements performed with the readout system developed in this thesis, as well as some resource utilization of the FPGA design.

5.1 Measurements

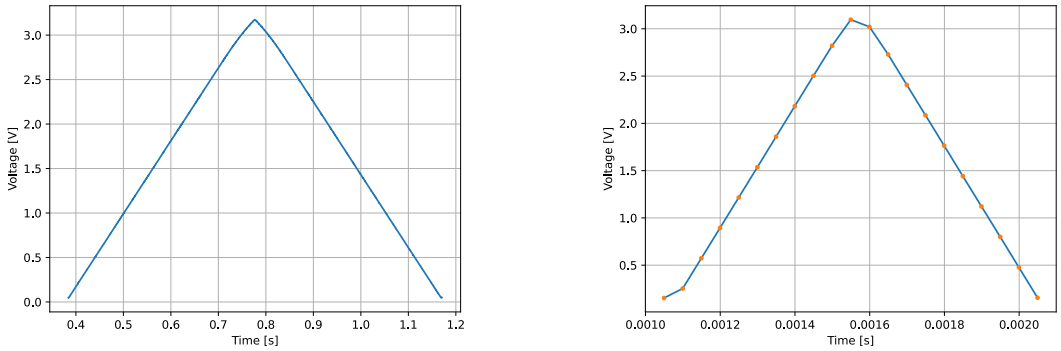
Measurements from a generated input wave to the ADC will be presented, as well as measurements from a m-NIC1 and m-NIC2 table test.

Measurement type	Status
ADC Characterization	Performed for the m-NIC2 ADC, but an error 5.1.1.1 occurred for all measurements which was not noticed until a later date. Included in the measurements section are characterization measurements for two frequencies to demonstrate the functionality of the readout system.
DAC characterization	Not performed.
Front-end Table measurements	Performed for both m-NIC1 and m-NIC2 ADC.
Plasma chamber measurements	Not performed.

Table 5.1: Current status of which measurements have been performed.

Frequency	Standard deviation [LSB]	Maximum deviation [LSB]
1.273 Hz	10.06	36
4 mHz	9.92	60

Table 5.2: Calculation of parameters for the deviation from a regress line at different frequencies, as shown in Figure 5.2 and 5.3.



(a) 1.273 Hz

(b) 1 kHz

Figure 5.1: Triangle wave input at 1.273 Hz and 1 kHz, measured with the m-NIC2 ADC using the shift-register readout.

5.1.1 ADC

Shown in Figure 5.1 are zoomed in data of a 210 second capture of two input signals. From the 1.273 Hz measurement a linear

Figure 5.2 displays the deviation from a regress line in the linear region for the waveform displayed in Figure 5.1 a). For this wave, the amount of samples analysed in the linear region is approximately 6500 samples which is too low to get one sample per LSB step.

Another measurement was performed, this time a sawtooth wave with a frequency of 4 mHz. With this measurements there are about $4 * 10^6$ samples, this equals approximate 77 samples per LSB step, giving more detailed view. Results from a regress line similar to what was perfomed on the 1.273 Hz waveform is found in Figure 5.3.

Calculations for both maximum deviation and standard deviation for Figure 5.2 and 5.3 is found in Table 5.2.

For both Figure 5.2 and 5.3 a clear spike can be seen at 1.65 V, this is around VM for the ADC and is where the MSB will be asserted high. This transition is shown in more detail in Figure 5.4

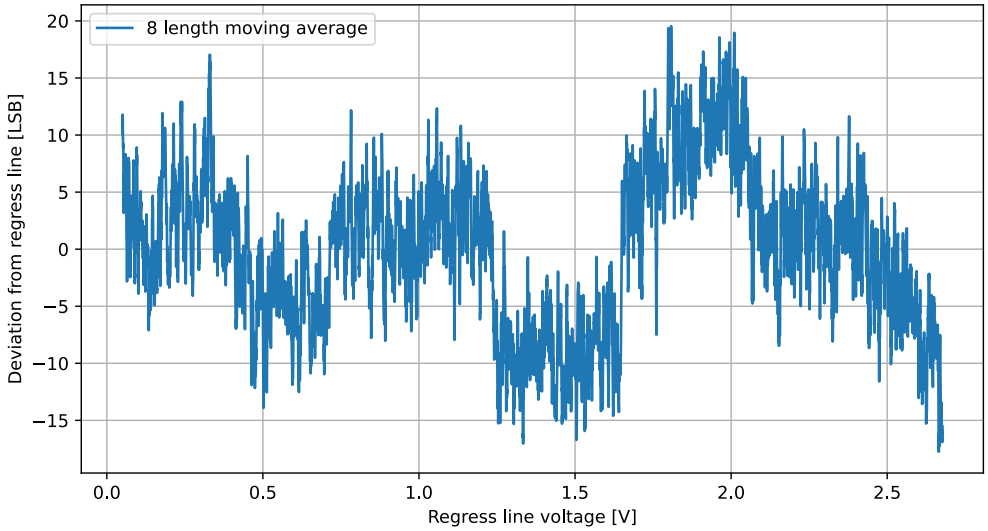


Figure 5.2: Deviation in LSBs from a linear fit in the linear region (50 mV to 2.7 V). Taken from the 1.273 Hz measurement as seen in figure 5.1 a).

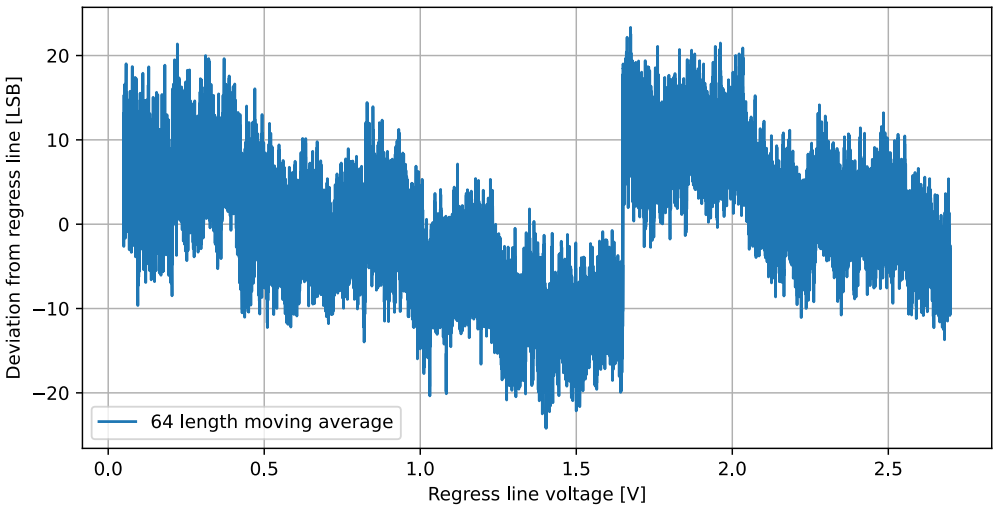


Figure 5.3: Deviation in LSBs from a linear fit in the linear region from a 4 mHz sawtooth wave.

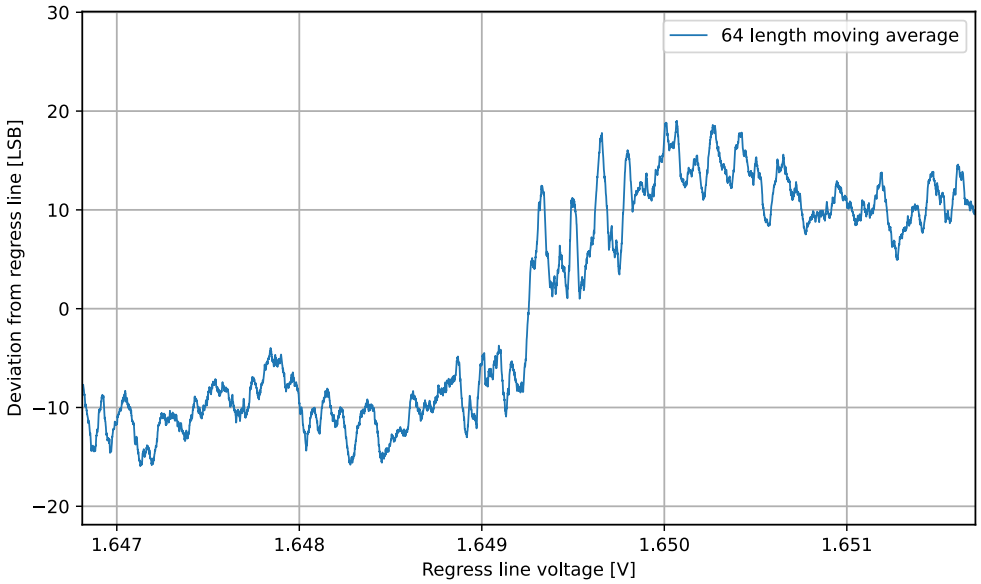
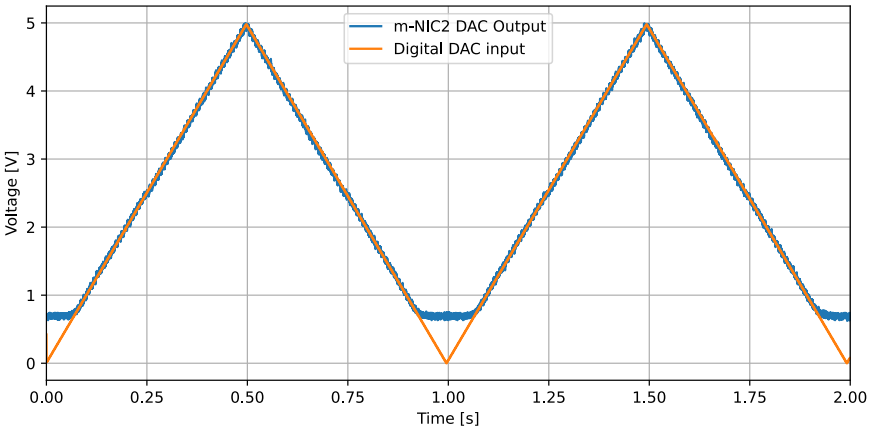


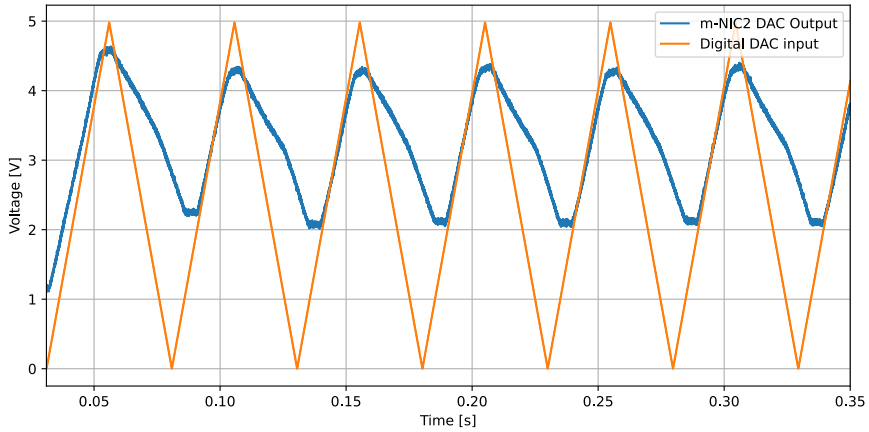
Figure 5.4: A zoomed in view of the transition of MSB low to MSB high from the deviation plot in Figure 5.3.

5.1.1.1 ADC issues caused by DAC sweeping

During the data capture of the ADC characterization measurements the external DAC was performing sweeps. It was connected to the output of the m-NIC2 DAC output, which caused the ADC to not perform as expected. Capturing of the input wave was done on a oscilloscope to be synced with the measured ADC data as a way of double checking that everything was correct, this showed that nothing was wrong with the input.



(a) 1 Hz



(b) 20 Hz

Figure 5.5: Triangle wave output of the m-NIC2 DAC, demonstrating the issues which occur at higher frequencies.

5.1.2 DAC

As described in Table 2.4 there are multiple issues regarding the m-NIC2 DAC, two of which are visible in Figure 5.5. In Figure 5.5a the issue with the output buffer regarding lower voltages is seen to be approximate 0.8 V. Figure 5.5b shows an effect which has not been documented prior to these measurements. The output voltage of the DAC struggles to follow the input, and settles at an offset close to 3.1 V with a peak to peak voltage of approximate 2.2 V.

5.1.3 Front-end table test

Figure 5.6 shows the relation between input current on the TIA input port InS and its voltage output $OutLS$. An approximate linear relation can be seen as the current becomes more negative[1]. For the m-NIC2 -1500 nA measurement an unknown error occurred, which the reason for the output voltage being zero, in the linear fit performed in Figure 5.6 this measurements is left out on purpose.

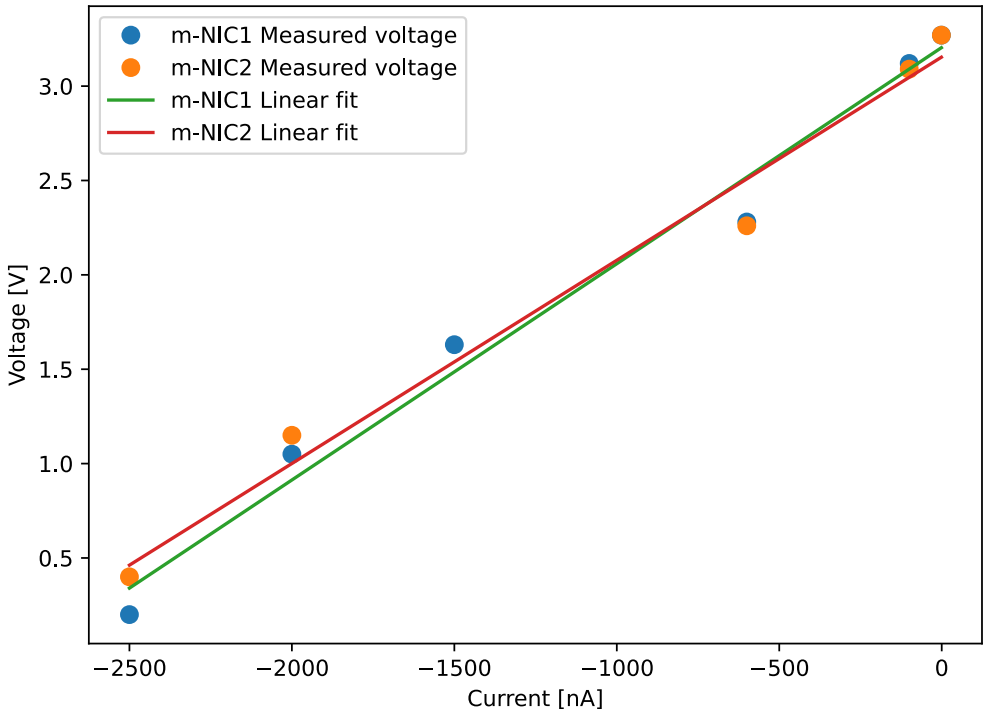


Figure 5.6: Comparison between m-NIC1 ADC and m-NIC2 ADC of the measured output voltage of the m-NIC1 front-end.

5.2 Readout system performance

In this section, the FPGA design resource utilization and timing performance will be presented.

5.2.1 Resource utilization

Table 5.3 presents the FPGA resources used for both the full system, as well as only the m-NIC PCB interface module. It is clearly seen that the m-NIC PCB interface is far less resource demand-

[1] Current negative current refers to an electron flow into the chip.

Resource	Total utilization	m-NIC PCB Interface utilization	Amount available
LUT	2994	756	53 200
FF	4273	735	106 400
BRAM	35.5	0	140
I/O	43	43	125

Table 5.3: Table containing the resource utilization of the whole system, as well as only the m-NIC PCB interface.

ing compared to the data transfer module, and the whole system itself is not very resource heavy compared to what is available.

Chapter 6

Discussion

In this final chapter the measurements performed will be discussed and the readout systems performance will be evaluated.

6.1 Measurements

Measurements for both the m-NIC1 and m-NIC2 ADC were performed while testing the m-NIC1 front-end, as well as a measurement of the m-NIC1 ADC with a sawtooth wave as an input. Due to time constraints related to the delivery of this thesis, proper characterization did not have time to take place and plasma chamber testing were scrubbed.

6.1.1 ADC

The measurements for the n-NIC2 ADC gave similar results to what has been shown in earlier rounds of testing, something which proves the functionality of the developed readout system. With the added functionality of saving data automatically, more formal characterisation can be performed.

Comparing the results from this thesis to what was seen in 2018 for the m-NIC1 ADC and there are some differences. In the 2018 round of testing the standard deviation from a linear regress line in the linear region was 8 LSB, and the maximum deviation was 23 [16]. These numbers are lower than the ones found in this thesis, which were 9.92 and 60 respectively for the largest dataset, see 5.2. However, the 2018 measurements were performed with a sampling frequency of 1 S/s compared to 20.008 kS/s which were used in this thesis. A different conversion value when comparing multiple sampling frequencies was also noticed in 2018 [16]. Comparing these two measurements is therefore not straight forward and it is uncertain if the lower standard deviation found in 2018 is due to a less noisy setup, if the m-NIC1 ADC has less noise compared to m-NIC2, if the lower sampling rate helped to reduce noise or if the old measurements simply did not contain enough samples. Proper characterization measurements for the m-NIC2 ADC following IEEE standard 1241-2010[22] will be carried out following the submission of this thesis.

Requirement #	Status
R.1	Achieved
R.2	Achieved
R.3	Partly achieved
R.4	Partly achieved

Table 6.1: Table summarising the requirements described in Section 3.2.2.

6.1.2 DAC

Measurements performed on the DAC illustrated an effect which has previously not been documented as described in Section 5.1.2. DAC characterization of the m-NIC2 DAC was planned, but due to its performance, further measurements for characterization were deemed unnecessary. Increasing the frequency futher from what was shown in Section 5.1.2 only decreased the peak to peak voltage of the output wave, and it shared more and more similarities with a noisy DC signal.

6.1.3 Front-end measurements

Observing the front-end results presented in Section 5.1.3 and a clear trend of a decreasing output voltage is observed for an more negative current (electrons flowing into the front-end). These results are, however, different from what has been documented earlier. Depending on the bias voltage set on the front-end the current range will be different, as in the output voltage will stop decreasing in a linear fashion at different current levels, with a 5 V bias voltage the range was previously determined to be 0 to 1.5 μA [16]. This is not what the results in this thesis show. When performing measurements manually using an oscilloscope the results were reported to be much more similar to what was seen in [16]. The reason for the differing results is not known and the measurements with the m-NIC ADC should be attempted to reproduced.

6.2 Readout system

In this section the performance of the readout system will be discussed and weather it achieved its intended purpose.

6.2.1 Requirements

This section will discuss the requirements and to what degree each of them were fulfilled.

R.1 A PYNQ-Z2 board, together with accessories such as cables, protective case and an SD-card costs 2 013,00 NOK at farnell.no the 27 of May 2021. This is an inexpensive board and satisfies the requirement of keeping the price below 3000 NOK.

R.2 The speed requirement of 2 Mbps was easily achieved by transferring data with the DMA. As power consumption was not a consideration during the development of this design, there was not much reason to change the 100 MHz clock. Seeing that the speed requirement was achieved by such a good margin, the clock frequency could be reduced to save power if this became desirable.

R.3 Performing measurements with the ADC requires the usage of physical switches and running commands in Jupyter, while to control the DAC Vivado is also needed. While it has more functionality compared to the old readout system, it is not easier to use due to needing user input from three different sources to operate with full functionality. Moving the physical button and virtual input/output to the AXI GPIO would create a more intuitive user interface, but would only be a small step in the right direction. To take full advantage of the systems potential, a AXI4-Lite control register should be added, as described in Section 7.1.

R.4 Reusability for this system can be split into two parts, the data-transfer module and the m-NIC PCB interface module. As of now, the data-transfer module can operate at high speeds and easily change its data source. It is not perfect however, as it currently relies on using a clock which is synchronized with the incoming data to transfer at a correct rate. A better option would be to fully utilize the AXIS bus signals, and add functionality for a TVALID [1] signal. The m-NIC PCB interface module is reusable to an extent, as its components can be used in different designs to control its physical counterparts, but the top design would require more changes. Transferring this design over to later revisions of the m-NIC might not be applicable either, as the communication interface might, and should, be changed to a more standard protocol.

6.2.2 Limitations

Due to using Vivado specific IP's the data transfer module is locked to the PYNQ platform. Moving to a different Xilinx PS-PL hybrid platform will require some changes but is still possible given correct specifications, if the decision to use a different manufacturer is made this module would then require a re-design. Being locked to a PS-PL hybrid will also make the transition to integrate the readout design into an ASIC with the front-end functionality more challenging.

Most of the resources used by the design comes from the data transfer module. This module is capable of far higher speeds that what is necessary for this test setup, but does not impact anything in a negative way here. However, for prototyping an FPGA design which could be further developed into the ASIC this would be far from an ideal solution. A UART, SPI or I^2C protocol could be implemented instead to reduce the resource utilization of the data transfer module, and therefore have a smaller area impact on an ASIC.

[1] TVALID is one of the masters signals in the AXIS bus. It will go high when it has valid data to transfer, and data will only transfer when it is high as mentioned in 2.4.2

Chapter 7

Conclusion

This thesis main goal was to enable the ability to further understand the m-NIC project by developing a readout system towards both revisions and some reference converter components. A goal which was achieved, and some measurements were successfully done using the developed system, improving on the previous readout system which had been used for similar tests. The two readout modules developed in this thesis, being the m-NIC PCB interface and the data transfer module are both valuable additions to the project. By modifying the m-NIC PCB interface module functionality for a future chip can be integrated, meaning much less work and far less time needs to be dedicated to characterizing future revisions. There are still some issues however, most notable being the user interface to the whole system being a combination of physical inputs, Jupyter[1] and virtual inputs through Vivado[2]. Over the following weeks, characterization measurements for the internal converters and plasma chamber measurements should be performed. This will help the gain better insight in the current state of the m-NIC project.

7.1 Future work

Additional measurements In the near future both characterisation measurements and plasma chamber measurements will be performed. This will help quantify performance and reveal useful information for the next revisions of the m-NIC. More extensive measurements on the front-end is also necessary to find a proper I-V characteristic for different voltage biases.

AXI4-Lite Control register The readout system now is more difficult to control and operate than what it needs to be. To improve usability a memory mapped control register should be added to take over the functionality. This would eliminate the need for the physical switches and the virtual input/output module in Vivado. Writing data to the control register would then be performed in Jupyter and would make the user interface exceedingly less complex and more automatic, something which

[1] Python environment initiating the data capture code.
[2] Xilinx software to program the Zynq7020-SoC.

would also allow for longer characterization captures as scripts could run without the supervision of a person.

Improved data transfer After the data is transferred from the PL to the PS it needs to be saved. Currently, if performing a 210 second data capture at 20 kS/s it will take an additional 10 minutes after the capture is complete to save the data. This data is saved in a csv format with the values being floats. One solution for saving data faster could be to save it as the raw 32-bit word, and do the decoding in post-process. Data could also be saved in smaller files, this could be performed during data capture as it takes approximately 1.6 s to fill the FIFO and 350 μs to empty it.

m-NLP RISC-V Another master project [24] looked at and developed a custom RISC-V core for MNLP. It proved real-time calculations of the electron density within the timing constraints of a 20 kS/s sampling rate was very doable. This reduces the data cost and would allow to capture data at maximum sampling rate for a longer duration.

Appendix A

VHDL Code

A.1 pcb_interface_v3.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.std_logic_1164.ALL;
3  USE IEEE.NUMERIC_STD.ALL;
4  use work.sine_package.all;
5  --use work.sweep_buffer_pkg.all;
6
7  ENTITY pcb_interface_v3 IS
8
9      PORT (-- Board input
10         mrst            : in  std_logic;
11         mclk            : in  std_logic;
12         mode            : in std_logic_vector(1 downto 0);
13         adc_switch      : in std_logic;
14         dbg_led         : out std_logic;
15         adc_selected    : out std_logic;
16         trigger         : out std_logic;
17         -- Generated clocks
18         int_adc_clk     : out std_logic;--std_logic_vector(1 downto 0); -- Should run at
           ↪  17*samplefreq, samplefreq: 1-10kHz max
19         max1132_clk     : out std_logic; -- Can do multiple MHz
20         sclk            : out std_logic; -- Uncertain, 1MHz maybe?
21         axis_clk_in         : in std_logic;
22         axis_clk_out        : out std_logic;
23         -- ADC control
24         -- Internal ADC
25         adc_en       : out std_logic;--std_logic_vector(1 downto 0);
26         adc_eoc      : in  std_logic;--std_logic_vector(1 downto 0);
27         adc_comp            : in std_logic;--std_logic_vector(1 downto 0); -- Read out
           ↪  directly from ADC comparator output
28
29         qp4d        : in std_logic_vector(3 downto 0);
30         -- External ADC
31         max1132_dout        : in std_logic; -- Dout
```

```
32          max1132_sstrb       : in std_logic; -- SSTRB
33          max1132_din         : out std_logic; -- digital in, write to adc
34          max1132_rst         : out std_logic; -- reset adc
35          max1132_shdn        : out std_logic; -- drive shdn low to put the adc in shutdown
            ↪ mode
36          max1132_cs          : out std_logic;
37
38          -- DAC Control
39          --dac_sweep_ena : in std_logic; -- Or always on? Pullup-header to keep system in
            ↪ idle
40          wave_type   : in std_logic_vector(1 downto 0); -- Selects sine or sawtooth for DAC
            ↪ input
41          sweep_mclk     : in std_logic; -- Clock from PS to DAC, can be changed via jupyter.
            ↪
42
43          dac_data    : in std_logic_vector(31 downto 0);
44          -- Serial interface (internal DAC)
45          -- Need more shif-register control? What happens during x2 dac + adc testing?
46          adc_selch1  : out std_logic;
47          adc_ext_sel : out std_logic;
48          scr_enbias  : out std_logic;
49          mresb       : out std_logic_vector(1 downto 0);
50          si          : out std_logic;
51          sreadb      : out std_logic;
52          swrite      : out std_logic;
53          so          : in std_logic;
54
55          -- External DAC
56          dac_out     : out std_logic_vector(7 downto 0);
57          wr          : out std_logic;
58          A           : out std_logic_vector(1 downto 0);
59
60          -- AXI
61          adc_result      : out std_logic_vector(31 downto 0);
62          axis_clk_counter    : out std_logic_vector(31 downto 0);
63          led             : out std_logic_vector(1 downto 0);
64          comp_delayed    : out std_logic;
65          eoc_delayed     : out std_logic;
66          tvalid          : out std_logic;
67        -- sweep_buffer   : out buffer_array
68
69        -- DEBUG
70        reg_bank_sel     : in std_logic_vector(2 downto 0);
71        sweep_sync       : out std_logic;
72        input_sync       : in std_logic;
73        dac_frequency_vio    : in std_logic_vector(20 downto 0);
74        dac_latch        : in std_logic;
75        dbg_adc_data_ila         : out std_logic_vector(15 downto 0)
76    );
77
78  END ENTITY pcb_interface_v3;
79
```

```
80  ARCHITECTURE arch OF pcb_interface_v3 IS
81
82      component debounce is
83          port (
84              mclk    : in std_logic;
85              mrst    : in std_logic;
86              button_inp  : in std_logic;
87              button_stable   : out std_logic
88
89          );
90      end component;
91      component dac_control is
92          port(
93              clk     : in std_logic;
94              sweep_mclk : in std_logic;
95              reset   : in std_logic;
96              enable  : in std_logic;
97              sclk    : out std_logic;
98              si      : out std_logic;
99              sreadb  : out std_logic;
100             swrite  : out std_logic;
101             sweep_ena : in std_logic;
102             dac_data    : in std_logic_vector(7 downto 0);
103
104             conf_reg_data   : in std_logic_vector(6 downto 0);
105             reg_bank_sel    : in std_logic_vector(2 downto 0);
106             conf_reg_latch  : in std_logic;
107             dac_out : out sine_vector_type;
108             wr      : out std_logic;
109             A       : out std_logic_vector(1 downto 0);
110             adc_eoc : in std_logic;
111             sweep_sync  : out std_logic;
112             dac_frequency : in std_logic_vector(20 downto 0);
113             dac_latch   : in std_logic;
114             wave_type   : in std_logic_vector(1 downto 0);
115             wr_clk_out  : out std_logic
116         );
117     end component dac_control;
118
119     component int_adc_control is
120         port (
121             mclk_adc    : in std_logic;
122             reset   : in std_logic;
123             enable_control  : in std_logic;
124             -- ADC select
125             adc_selch1   : out std_logic;
126             adc_ext_sel  : out std_logic;
127             scr_enbias   : out std_logic;
128             -- ADC control
129             mresb        : out std_logic;
130             adc_clk      : out std_logic;
131             adc_en       : out std_logic;
```

```
132             adc_eoc      : in  std_logic;
133             adc_comp     : in std_logic; -- Read out directly from ADC comparator output
                ↪
134             adc_result   : out std_logic_vector(15 downto 0);
135             comp_delayed     : out std_logic;
136             eoc_delayed      : out std_logic;
137             tvalid           : out std_logic
138         );
139     end component int_adc_control;
140 
141     component ext_adc_control is
142         port (
143             mclk_ext_adc     : in std_logic;
144             enable   : in std_logic; -- outside decider of exADC is being used
145             reset    : in std_logic; -- reset signal for exADC
146 
147             dout     : in std_logic; -- Dout
148             sstrb    : in std_logic; -- SSTRB
149             din      : out std_logic; -- digital in, write to adc
150             adc_rst : out std_logic; -- reset adc
151             exadc_clk    : out std_logic; -- clock for adc
152             shdn     : out std_logic; -- drive shdn low to put the adc in shutdown mode
153             cs       : out std_logic;
154 
155             exADC_result     : out std_logic_vector(15 downto 0) -- result to pcb_interface
156         );
157     end component ext_adc_control;
158 
159     component tb_adc_arbiter_fifo_3 is
160         PORT (-- Reset and clk input
161         reset          : IN  STD_LOGIC;
162         clk            : IN  STD_LOGIC;
163         -- Reset out
164         reset_out      : OUT STD_LOGIC;
165          -- ADC
166           clk_adc        : OUT STD_LOGIC;
167           adc_en         : OUT STD_LOGIC;
168           adc_eoc        : IN  STD_LOGIC;
169           qpc            : OUT STD_LOGIC_VECTOR ( 1 DOWNTO 0);
170           qp4d           : IN  STD_LOGIC_VECTOR ( 3 DOWNTO 0);
171         -- FIFO
172           write_clk      : OUT STD_LOGIC;
173           clk_arb        : OUT STD_LOGIC;
174           disp_3         : OUT STD_LOGIC_VECTOR ( 6 DOWNTO 0);
175           disp_2         : OUT STD_LOGIC_VECTOR ( 6 DOWNTO 0);
176           disp_1         : OUT STD_LOGIC_VECTOR ( 6 DOWNTO 0);
177           disp_0         : OUT STD_LOGIC_VECTOR ( 6 DOWNTO 0);
178          -- Debug port
179          dbg_sel_swt    : IN  STD_LOGIC_VECTOR ( 1 DOWNTO 0);
180           dbg_port_sel   : OUT STD_LOGIC_VECTOR ( 3 DOWNTO 0);
181           dbg_port_out   : IN  STD_LOGIC_VECTOR ( 3 DOWNTO 0);
```

```
182             dbg_sts       : OUT STD_LOGIC_VECTOR (15 DOWNTO 0)        -- dbg_port_sts  : OUT
                ↪  STD_LOGIC_VECTOR (3 DOWNTO 0)
183             );
184     end component;
185
186     component tb_adc_sawtooth is
187     PORT (-- Board input
188         reset_in     : IN  STD_LOGIC;  -- R22 (KEY[0])
189         clk_in       : IN  STD_LOGIC;  -- A12 (24 MHz)
190         -- ADC select
191         adc_selch1   : OUT STD_LOGIC;  -- G18 (GPIO_1[25])
192         adc_ext_sel  : OUT STD_LOGIC;  -- G20 (GPIO_1[24])
193         scr_enbias   : OUT STD_LOGIC;  -- E18 (GPIO_1[23])
194         -- ADC control
195         mresb        : OUT STD_LOGIC;  -- E19 (GPIO_1[22])
196         adc_clk      : OUT STD_LOGIC;  -- F20 (GPIO_1[21])
197         adc_en       : OUT STD_LOGIC;  -- E20 (GPIO_1[20])
198         adc_eoc      : IN  STD_LOGIC;  -- D20 (GPIO_1[19])
199         -- ADC output interface
200         sclk         : OUT STD_LOGIC;  -- D19 (GPIO_1[18])
201         so           : IN  STD_LOGIC;  -- C20 (GPIO_1[17])
202         si           : OUT STD_LOGIC;  -- C19 (GPIO_1[16])
203         sreadb       : OUT STD_LOGIC;  -- C18 (GPIO_1[15])
204         swrite       : OUT STD_LOGIC;  -- C17 (GPIO_1[14])
205         -- 7-Segment LED display
206         adc_data_3   : OUT STD_LOGIC_VECTOR ( 6 DOWNTO 0);
207         adc_data_2   : OUT STD_LOGIC_VECTOR ( 6 DOWNTO 0);
208         adc_data_1   : OUT STD_LOGIC_VECTOR ( 6 DOWNTO 0);
209         adc_data_0   : OUT STD_LOGIC_VECTOR ( 6 DOWNTO 0);
210         -- Average ADC output
211         dsp_sel_swt  : IN  STD_LOGIC;  -- L22 (SW[0])
212         -- Test points
213         test_point_0 : OUT STD_LOGIC;  -- H12 (GPIO_1[0])
214         test_point_1 : OUT STD_LOGIC;  -- H13 (GPIO_1[1])
215         test_point_2 : OUT STD_LOGIC;   -- H14 (GPIO_1[2])
216         -- sawtooth generation
217         sawtooth_signal     : OUT std_logic;
218         adc_result      : out std_logic_vector(15 downto 0)
219 );
220     end component;
221
222     CONSTANT ADC_CLK_PERIOD   : integer  := 5000; -- 600;
223     CONSTANT SER_CLK_PERIOD   : integer  := 2400;
224     CONSTANT AXI_MASTER_CLK_HALFPERIOD    : integer := 2499; --2517; --2499; --250;
225
226     signal cur_ser_read       : std_logic;
227     signal nxt_ser_read       : std_logic;
228
229     signal cur_adc_en         : std_logic;
230     signal nxt_adc_en         : std_logic;
231
232     signal cur_adc_clk        : std_logic;
```

```
233     signal nxt_adc_clk        : std_logic;
234     signal cur_adc_clk_count  : integer := 0;
235     signal nxt_adc_clk_count  : integer;
236
237     signal cur_adc_clk_n      : std_logic;
238     signal nxt_adc_clk_n      : std_logic;
239     signal cur_adc_clk_n_count: integer := ADC_CLK_PERIOD;
240     signal nxt_adc_clk_n_count: integer;
241
242     --CONSTANT ADC_CLK_PERIOD     : integer         := 25000000;
243
244     -- Serial interface
245     signal int_sclk  : std_logic;
246
247     -- External DAC signals
248     signal int_chipsel : std_logic;
249     signal int_enable_sine : std_logic := '1';
250     signal conf_reg_data    : std_logic_vector(6 downto 0);
251
252     --signal enable : std_logic;
253     --signal reg_bank_sel : std_logic_vector(2 downto 0);
254     signal conf_reg_latch : std_logic;
255
256
257     signal adc_rst      : std_logic;
258
259     -- Int ADC signals (adc_comp readout)
260     signal int_adc_reset    : std_logic;
261     --signal int_adc_ena       : std_logic;
262     signal int_adc_en       : std_logic; -- bedre navn?
263
264
265     -- External ADC signals
266
267     signal exADC_enable     : std_logic;
268     signal exADC_reset      : std_logic;
269     signal exADC_dout       : std_logic;
270     signal exADC_sstrb      : std_logic;
271     signal exADC_tconv      : std_logic;
272     signal exADC_din        : std_logic;
273     signal exADC_clk        : std_logic;
274     signal exADC_shdn       : std_logic;
275     signal exADC_result     : std_logic_vector(15 downto 0);
276     signal exADC_cs         : std_logic;
277
278     -- Test mode management
279     signal int_mode     : std_logic_vector(1 downto 0);
280     signal ext_adc_ena  : std_logic;
281     signal int_adc_ena  : std_logic; --_vector(1 downto 0);
282     signal int_dac_ena  : std_logic;
283     signal ext_dac_ena  : std_logic;
284
```

```
285     signal dummy_enable : std_logic;
286
287     signal dac_sweep_ena  : std_logic;
288
289     signal int_int_adc_clk : std_logic;
290     signal int_max1132_clk : std_logic;
291
292     signal int_adc_result   : std_logic_vector(15 downto 0);
293     signal max1132_result   : std_logic_vector(15 downto 0);
294
295     signal adc_result_msb   : std_logic;
296     --signal comp_delayed : std_logic;
297     --signal eoc_delayed  : std_logic;
298
299     --signal nxt_sweep_buffer : buffer_array;
300
301     signal int_adc_clk_slow : std_logic;
302     signal dac_out_int      : std_logic_vector(7 downto 0);
303
304     -- Serial clock
305
306     SIGNAL cur_ser_clk        : STD_LOGIC;
307     SIGNAL nxt_ser_clk        : STD_LOGIC;
308     SIGNAL cur_ser_clk_count  : INTEGER := 0;
309     SIGNAL nxt_ser_clk_count  : INTEGER;
310
311     signal adc_result_s1 : std_logic_vector(15 downto 0);
312
313     signal mrst_stable  : std_logic;
314     signal adc_switch_stable    : std_logic;
315
316     signal mNIC1ADC_ena : std_logic;
317     signal mNIC2ADC_ena : std_logic;
318
319     signal int_adc_result_x1    : std_logic_vector(15 downto 0);
320     signal int_adc_result_x2    : std_logic_vector(15 downto  0);
321
322     --signal axis_clk_int         : std_logic;
323     signal sweep_sync_s1        : std_logic;
324
325     signal axis_clk_ena     : std_logic := '0';
326
327     signal HIGH : std_logic := '1';
328     signal LOW  : std_logic := '0';
329
330     signal temp_so : std_logic;
331
332     signal axis_clk_out_s1 : std_logic;
333     signal axis_clk_out_s2 : std_logic;
334
335     --signal tvalid   : std_logic;
336     signal adc_eoc_prev : std_logic;
```

```
337     signal wr_clk_dac    : std_logic;
338
339 begin
340     -- Testing signals/values
341     --int_adc_clk(1) <= int_adc_clk_slow(1);
342     int_adc_clk <= int_adc_clk_slow;
343
344     dac_sweep_ena <= '1';
345     adc_result_msb <= adc_result_s1(15);
346     dbg_led <= adc_result_msb;
347
348     adc_selected <= mNIC2ADC_ena;
349     dbg_adc_data_ila <= int_adc_result_x1;
350
351
352
353
354     -- Clockstuff
355     int_sclk            <= cur_ser_clk;
356     nxt_ser_clk         <=  cur_ser_clk            WHEN cur_ser_clk_count < SER_CLK_PERIOD
        ↪  ELSE (cur_ser_clk XOR '1');
357     nxt_ser_clk_count   <= (cur_ser_clk_count + 1) WHEN cur_ser_clk_count < SER_CLK_PERIOD
        ↪  ELSE 0;
358     --int_sclk <= mclk;
359     --int_adc_clk <= mclk;
360     int_max1132_clk <= mclk;
361     --axis_clk <= axis_clk_int;
362
363     --sclk <= int_sclk;
364     --max1132_clk <= int_max1132_clk;
365     int_int_adc_clk <= mclk;
366
367     dac_out <= dac_out_int;
368     --reg_bank_sel <= "011"; -- Setter output på VSRC1_10
369     conf_reg_latch <= '0';
370     sweep_sync <= sweep_sync_s1;
371
372     adc_en <= int_adc_en;
373     adc_result(15 downto 0) <= adc_result_s1;
374     adc_result(16) <= input_sync;
375     adc_result(17) <= sweep_sync_s1;
376     adc_result(31) <= '1';
377     adc_result(30 downto 23) <= dac_out_int;
378
379     axis_clk_counter(31) <= '1';
380
381     --temp_so <= adc_comp;
382     --adc_result(31) <= '1';
383     --adc_result(29 downto 22) <= dac_out_int;
384
385     --CLK_GEN_0: clock_generator port map(
386         --mclk    => mclk,
```

```
387         --rst     => mrst,
388         --ext_adc_clk   => ext_adc_clk,
389         --int_adc_clk  => int_adc_clk,
390         --dac_clk    => dac_clk);
391
392
393
394     ADC_SWITCH_DEBOUNCER: debounce port map(
395         mclk    => mclk,
396         mrst    => mrst,
397         button_inp => adc_switch,
398         button_stable => adc_switch_stable
399
400     );
401
402     DAC_0: dac_control port map(
403         clk     => mclk,
404         sweep_mclk => sweep_mclk,
405         reset   => mrst,
406         enable  => ext_dac_ena,
407         sclk    => open,
408         si      => open,
409         sreadb  => open,
410         swrite  => open,
411         sweep_ena => dac_sweep_ena,
412         dac_data    => dac_data(7 downto 0),
413         conf_reg_data   => conf_reg_data,
414         reg_bank_sel    => reg_bank_sel,
415         conf_reg_latch  => conf_reg_latch,
416         dac_out => dac_out_int,
417         wr      => wr,
418         A       => A,
419         adc_eoc => adc_eoc,
420         sweep_sync => sweep_sync_s1,
421         dac_frequency => dac_frequency_vio,
422         dac_latch => dac_latch,
423         wave_type   => wave_type,
424         wr_clk_out  => wr_clk_dac
425     );
426
427     ext_adc_0: ext_adc_control port map(
428         mclk_ext_adc    => int_max1132_clk, -- mulig lage en clock genereator?
429         enable  => ext_adc_ena,
430         reset   => mrst,
431         dout    => max1132_dout,
432         sstrb   => max1132_sstrb,
433         din     => max1132_din,
434         adc_rst => max1132_rst,
435         exadc_clk   => max1132_clk,
436         shdn        => max1132_shdn,
437         cs          => max1132_cs,
438         exADC_result    => max1132_result
```

```
439     );
440
441     mNIC2ADC: int_adc_control port map(
442         mclk_adc    => int_int_adc_clk,
443         reset   => mrst,
444         enable_control  => mNIC2ADC_ena,
445         adc_selch1      => open,
446         adc_ext_sel     => open,
447         scr_enbias      => open, -- trenger for adc control? er ikke dette dac relatert
448         mresb           => open,
449         adc_clk         => open,
450         adc_en          => open,
451         adc_eoc         => adc_eoc,
452         adc_comp        => adc_comp,
453         adc_result      => open,
454         comp_delayed    => comp_delayed,
455         eoc_delayed     => eoc_delayed,
456         tvalid          => open
457     );
458
459     mNIC1ADC: int_adc_control port map(
460         mclk_adc    => int_int_adc_clk,
461         reset   => mrst,
462         enable_control  => mNIC1ADC_ena,
463         adc_selch1      => open,
464         adc_ext_sel     => open,
465         scr_enbias      => open, -- trenger for adc control? er ikke dette dac relatert
466         mresb           => mresb(0),
467         adc_clk         => open,
468         adc_en          => open,
469         adc_eoc         => adc_eoc,
470         adc_comp        => adc_comp,
471         adc_result      => int_adc_result_x1,
472         comp_delayed    => open,
473         eoc_delayed     => open,
474         tvalid          => open
475     );
476
477     mNIC1ADC_qp4d: tb_adc_arbiter_fifo_3 port map(
478         reset   => mrst,
479         clk     => mclk,
480         reset_out => open, --mresb(0)
481         clk_adc => open,
482         adc_en  => open,
483         adc_eoc => adc_eoc,
484         qpc     => open,
485         qp4d    => qp4d,
486         write_clk   => open,
487         clk_arb     => open,
488         disp_3      => open,
489         disp_2      => open,
490         disp_1      => open,
```

```
491         disp_0      => open,
492         dbg_sel_swt => "00",
493         dbg_port_sel    => open,
494         dbg_port_out    => "0000",
495         dbg_sts         => open
496         );
497 
498     mNIC2ADC_sr: tb_adc_sawtooth port map(
499         reset_in    => mrst,
500         clk_in      => mclk,
501         adc_selch1  => open,
502         adc_ext_sel => open,
503         mresb       => mresb(1),
504         adc_clk     => int_adc_clk_slow,
505         adc_en      => int_adc_en,
506         adc_eoc     => adc_eoc,
507         sclk        => sclk,
508         so          => adc_comp,
509         si          => si,
510         sreadb      => sreadb,
511         swrite      => swrite,
512         adc_data_3  => open,
513         adc_data_2  => open,
514         adc_data_1  => open,
515         adc_data_0  => open,
516         dsp_sel_swt => HIGH,
517         test_point_0    => open,
518         test_point_1    => open,
519         test_point_2    => open,
520         sawtooth_signal => open,
521         adc_result      => int_adc_result_x2
522     );
523 
524     --P_BUFFER: process(adc_eoc,int_adc_clk_slow,mclk, mrst) is
525     --    variable count : integer := 256;
526     --    begin
527     --        if(mrst = '1') then
528     --            nxt_sweep_buffer <= (others => (others => '0'));
529     --        elsif rising_edge(adc_eoc) then
530     --            if(count = 0) then
531     --                sweep_buffer <= nxt_sweep_buffer;
532     --                nxt_sweep_buffer <= (others => (others => '0'));
533     --            else
534     --                count := count - 1;
535     --                nxt_sweep_buffer(count) <= int_adc_result & "0000000" & int_dac_ena &
        ↪  dac_out_int;
536     --            end if;
537     --        end if;
538     --end process;
539 
540     -- Managing enable signals for different modules
541     int_mode <= mode;
```

```
542
543     axis_clk_out <= axis_clk_out_s1;--  when mode = ""; -- int_adc_clk_slow; --axis_clk_ena
        ↪  and axis_clk_in;
544
545     PROCESS(mclk, mrst)
546     BEGIN
547         IF (mrst = '1') THEN
548               cur_ser_clk         <= '0';
549               cur_ser_clk_count   <= 0;
550
551
552        ELSIF rising_edge(mclk) THEN
553               cur_ser_clk         <= nxt_ser_clk;
554                cur_ser_clk_count   <= nxt_ser_clk_count;
555
556        END IF;
557
558    END PROCESS;
559    -- E . . . . . . . . . . . . . . . . . E . . . . . . . . . . . . . . . . . E
560
561    P_AXIS_CLK: process(mclk, mrst) is --adc_eoc
562         variable counter : integer := 0;
563         begin
564             if(mrst = '1') then
565                 axis_clk_out_s1 <= '0';
566                 counter := 0;
567             --elsif (adc_eoc = '1') then
568             --    axis_clk_out_s1 <= '0';
569             --    counter := 0;
570             elsif rising_edge(mclk) then
571                 if(counter = AXI_MASTER_CLK_HALFPERIOD) then
572                     axis_clk_out_s1 <= not axis_clk_out_s1;
573                     counter := 0;
574                 end if;
575                 counter := counter + 1;
576             end if;
577     end process;
578
579     axis_clk_out_s2 <= wr_clk_dac;
580
581     P_TVALID: process(axis_clk_out_s1, mrst) is -- tvalid, adc_eoc_prev
582         begin
583             if(mrst = '1') then
584                 tvalid <= '0';
585             elsif rising_edge(axis_clk_out_s1) then
586                 adc_eoc_prev <= adc_eoc;
587                 tvalid <= '1';
588
589                 if(adc_eoc_prev < adc_eoc) then -- rising edge
590                     tvalid <= '1';
591                 end if;
592             end if;
```

```
593     end process;
594
595     P_TRIGGER: process(mclk, mrst) is
596         begin
597             if(mrst = '1') then
598                 trigger <= '0';
599                 adc_result(18) <= '0';
600             elsif rising_edge(mclk) then
601                 trigger <= '1';
602                 adc_result(18) <= '1';
603             end if;
604     end process;
605
606     P_AXIS_CLK_COUNTER: process(axis_clk_in,mrst) is
607         variable count : integer := 0;
608         begin
609             if(mrst = '1') then
610                 count := 0;
611             elsif rising_edge(axis_clk_in) then
612                 if(count < 1073741823) then -- 32-bit max value -2
613                     count := count + 1;
614                 else
615                     count := 0;
616                 end if;
617                 axis_clk_counter(29 downto 0) <= std_logic_vector(to_unsigned(count,30));
618             end if;
619     end process;
620
621
622     P_ADC_SWITCH: process(mrst, adc_switch_stable) is
623         begin
624             if(mrst = '1') then
625                 mNIC2ADC_ena <= '1';
626                 mNIC1ADC_ena <= '0';
627             elsif(adc_switch_stable'EVENT and adc_switch_stable = '1') then
628                 mNIC1ADC_ena <= not mNIC1ADC_ena;
629                 mNIC2ADC_ena <= not mNIC2ADC_ena;
630             end if;
631     end process;
632
633     P_MODE_SELECT: process(mclk, int_mode, int_adc_result_x2) is
634         begin
635             case int_mode is
636                 when "00" =>
637                     int_dac_ena <= '1';
638                     int_adc_ena <= '1';
639                     ext_dac_ena <= '0';
640                     ext_adc_ena <= '0';
641
642                     led <= "00";
643                 when "01" =>
644                     int_dac_ena <= '1';
```

```
645                     int_adc_ena <= '0';
646                     ext_dac_ena <= '0';
647                     ext_adc_ena <= '1';
648                     adc_result_s1(15 downto 0) <= max1132_result;
649                     led <= "01";
650                 when "10" =>
651                     int_dac_ena <= '0';
652                     int_adc_ena <= '1';
653                     ext_dac_ena <= '1';
654                     ext_adc_ena <= '0';
655                     led <= "10";
656                     --if(mNIC2ADC_ena = '1') then
657                     --adc_result_s1(15 downto 0) <= int_adc_result_x2;
658                     --elsif(mNIC1ADC_ena = '1') then
659                     adc_result_s1(15 downto 0) <= int_adc_result_x1;
660                     --end if;
661                 when "11" =>
662                     int_dac_ena <= '0';
663                     int_adc_ena <= '0';
664                     ext_dac_ena <= '1';
665                     ext_adc_ena <= '1';
666                     adc_result_s1(15 downto 0) <= max1132_result;
667                     led <= "11";
668                 when others =>
669                 end case;
670         end process;
671     --ext_adc_ena       <= int_mode(0);
672     --int_adc_ena     <= not int_mode(0);
673     --int_dac_ena     <= not int_mode(1);
674     --ext_dac_ena     <=  int_mode(1);
675     -- shift register control
676     adc_ext_sel <= '1'; -- ext_dac_ena;
677     scr_enbias <= '1'; -- Always high?
678     adc_selch1 <= '0'; -- Always low i think
679 
680     --adc_result <= int_adc_result when int_adc_ena = '1' else max1132_result;
681 
682 
683 
684 end architecture;
```

A.2 dac_control.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  USE IEEE.NUMERIC_STD.ALL;
4  use work.sine_package.all;
5
6  entity dac_control is
7      port (
8          clk     : in std_logic;
9          sweep_mclk : in std_logic;
10         reset   : in std_logic;
11         enable  : in std_logic;
12         sclk    : out std_logic;
13         sweep_ena : in std_logic;
14         si      : out std_logic;
15         sreadb  : out std_logic;
16         swrite  : out std_logic;
17         dac_data    : in std_logic_vector(7 downto 0);
18
19         conf_reg_data   : in std_logic_vector(6 downto 0);
20         reg_bank_sel    : in std_logic_vector(2 downto 0);
21         conf_reg_latch  : in std_logic;
22         dac_out : out std_logic_vector(7 downto 0); -- 8 bit vector
23         wr      : out std_logic;
24         A       : out std_logic_vector(1 downto 0);
25         adc_eoc : in std_logic;
26         sweep_sync : out std_logic;
27         dac_frequency   : in std_logic_vector(20 downto 0);
28         dac_latch   : in std_logic;
29         wave_type   : in std_logic_vector(1 downto 0);
30         wr_clk_out  : out std_logic
31         );
32 end entity dac_control;
33
34 architecture dac_control_arch of dac_control is
35
36    -- component sine_wave is
37     --    port(clock, reset, enable : in std_logic;
38      --       wave_out : out sine_vector_type);
39     --end component sine_wave;
40
41     component ext_dac_control is
42         port(
43             clk             : in std_logic;
44             reset           : in std_logic;
45             enable          : in std_logic;
46             sweep_gen_data  : in std_logic_vector(7 downto 0);
47             ext_dac_out     : out std_logic_vector(7 downto 0);
48             wr              : out std_logic;
49             A               : out std_logic_vector(1 downto 0)
50         );
```

```
51      end component ext_dac_control;
52
53      component sine_wave is
54          port(
55              clk : in std_logic;
56              reset : in std_logic;
57              enable : in std_logic;
58              wave_out : out std_logic_vector(7 downto 0)
59          );
60      end component;
61
62      component int_dac_control is
63          port(
64              reset_in       : IN  STD_LOGIC;
65              clk_in         : IN  STD_LOGIC;
66              ena             : in std_logic;
67              -- Shift register control
68              mresb          : OUT STD_LOGIC;
69              scr_enbias     : OUT STD_LOGIC;
70              adc_ext_sel    : OUT STD_LOGIC;
71              adc_selch1     : OUT STD_LOGIC;
72              -- Shift register
                ↪  I/O
73              sclk           : out STD_LOGIC;
74              --                                      so             : IN  STD_LOGIC;
75              si             : OUT STD_LOGIC;
76              sreadb         : OUT STD_LOGIC;
77              swrite         : OUT STD_LOGIC;
78              -- Configuration register (Serial register)
79              conf_reg_data  : IN  STD_LOGIC_VECTOR( 6 DOWNTO 0); -- dc sweep
80              reg_bank_sel   : IN  STD_LOGIC_VECTOR( 2 DOWNTO 0);
81              conf_reg_latch : IN  STD_LOGIC
82          );
83      end component;
84
85      component sawtooth_wave is
86          generic(
87              MAX_VALUE : integer := 243;
88              MIN_VALUE : integer := 0
89          );
90          port(
91              clk, reset : in std_logic;
92              sweep_sync_out  : out std_logic;
93              wave_out : out std_logic_vector(7 downto 0));
94      end component;
95
96
97      constant SWEEP_CLK_PERIOD : integer := 977; --195(15kHz); --3333; --97656 (1Hz);
        ↪  --48828; -- 1 -> 50kHz. 25000 for 1Hz, 1 (må være ~54 ganger større enn serial
        ↪  clock til intern DAC)
98      constant WRITE_HALF_PERIOD : integer := 1000; -- 50kHz write/sample frequency
99      -- 1Hz : 97656
```

```
100     -- 2Hz :
101     constant sine_clk_count    : integer := 50000000;
102     signal sine_reset : std_logic;
103     signal int_dac_out : std_logic_vector(7 downto 0);
104     signal int_reset : std_logic;
105     signal int_enable : std_logic;
106     signal ext_enable : std_logic;
107     signal sine_counter : integer := 0;
108     signal sine_clk     : std_logic;
109     signal sweep_gen_data   : std_logic_vector(7 downto 0);
110     signal sweep_gen_data_wr_freq   : std_logic_vector(7 downto 0); -- Sweep data updated
        ↪  every WRITE_FREQUENCY
111
112     signal wr_clk   : std_logic;
113     signal mresb : std_logic;
114     signal scr_enbias : std_logic;
115     signal adc_ext_sel : std_logic;
116     signal adc_selch1   : std_logic;
117
118     signal sweep_clk    : std_logic;
119
120     signal ext_dac_out  : std_logic_vector(7 downto 0);
121     -- DAC frequency calculation:
122     -- CLK = 100MHz -- sweep_clk = CLK/(2*dac_frequency_integer)
123     -- sweep period = sweep_clk_period * 512
124     signal dac_frequency_integer    : integer := 97656; -- 1Hz default
125
126     signal sine_ena : std_logic;
127     signal sine_wave_data : std_logic_vector(7 downto 0);
128     signal sawtooth_ena : std_logic;
129     signal sawtooth_wave_data : std_logic_vector(7 downto 0);
130     signal dc_ena   : std_logic;
131
132
133
134
135     begin
136
137         wr_clk_out <= wr_clk;
138         int_enable <= not enable;
139         ext_enable     <= enable;
140         int_reset <= reset;
141         dac_out <= sweep_gen_data_wr_freq;
142         --sine_reset <= not reset; -- sine_wave uses active high reset
143         dac_frequency_integer <= to_integer(unsigned(dac_frequency));
144
145
146         -- Only one enable signal will  be high at a given moment.
147         sawtooth_ena <= wave_type(0) and (not dc_ena);
148         sine_ena    <= not wave_type(0) and (not dc_ena);
149         dc_ena      <= wave_type(1);
150
```

```
151
152         dac1: ext_dac_control port map(
153             clk     => clk,
154             reset   => reset,
155             enable  => ext_enable,
156             sweep_gen_data  => sweep_gen_data_wr_freq,
157             ext_dac_out => int_dac_out,
158             wr      => wr,
159             A       => A);
160
161         dac2: int_dac_control port map(
162             reset_in    => reset,
163             clk_in      => clk,
164             ena         => int_enable,
165             mresb       => mresb,
166             scr_enbias  => scr_enbias,
167             adc_ext_sel => adc_ext_sel,
168             adc_selch1  => adc_selch1,
                ↪
169             sclk        => sclk,
170             si          => si,
171             sreadb      => sreadb,
172             swrite      => swrite,
173             conf_reg_data  => sweep_gen_data(6 downto 0), --dac_data(6 downto
                ↪  0),--sweep_gen_data(6 downto 0),
174             reg_bank_sel   => reg_bank_sel,
175             conf_reg_latch => dac_latch);
176
177         sine_comp: sine_wave port map(
178             clk => sweep_clk,
179             reset => reset,
180             enable => sine_ena,
181             wave_out => sine_wave_data);
182
183         sawtooth_comp: sawtooth_wave port map(
184             clk         => sweep_clk,
185             reset       => reset,
186             sweep_sync_out  => sweep_sync,
187             wave_out    => sawtooth_wave_data
188         );
189
190
191         --P_BIAS_CONTROLER: process(clk, reset) is
192         --    variable dac_out_int : integer range 0 to 255 :=0;
193         --    begin
194         --        if(reset = '1') then
195         --            dac_out_int <= "00000000";
196         --        elsif(dcbias_ena = '1') then
197
198         --            sweep_gen_data <= std_logic_vector(to_signed(dac_out_int,8));
199         --       end if;
200         --end process;
```

```
201
202
203         -- SWEEP_CLK_PERIOD --> dac_frequency_integer
204         P_SWEEP_CLK: process(sweep_mclk,reset) is
205             variable count : integer := 0;
206             begin
207                 if(reset = '1') then
208                     sweep_clk <= '0';
209                     count := 0;
210                 elsif rising_edge(sweep_mclk) then
211                     if(count = dac_frequency_integer) then
212                         sweep_clk <= not sweep_clk;
213                         count := 0;
214                     elsif(count > dac_frequency_integer) then
215                         count := 0;
216                     else
217                         count := count + 1;
218                     end if;
219                 end if;
220         end process;
221
222         P_WR_CLK: process(clk,reset) is
223             variable count : integer := 0;
224             begin
225                 if(reset = '1') then
226                     wr_clk <= '0';
227                 elsif rising_edge(clk) then
228                     if(count = WRITE_HALF_PERIOD) then
229                         wr_clk <= not wr_clk;
230                         count := 0;
231                     else
232                         count := count + 1;
233                     end if;
234                 end if;
235         end process;
236
237         P_SWEEP_WR_FREQ: process(clk, reset) is
238             begin
239                 if(reset = '1') then
240                     sweep_gen_data_wr_freq <= (others => '0');
241                 elsif rising_edge(wr_clk) then
242                     sweep_gen_data_wr_freq <= sweep_gen_data;
243                 end if;
244         end process;
245
246         P_SWEEP_DATA_GEN: process(clk, reset) is
247             begin
248                 if(reset = '1') then
249                     sweep_gen_data <= (others => '0');
250                 elsif rising_edge(sweep_clk) then
251                     if(dc_ena = '1') then
252                         sweep_gen_data <= dac_data;
```

```
253                     elsif(sawtooth_ena = '1') then
254                         sweep_gen_data <= sawtooth_wave_data;
255                     elsif(sine_ena = '1') then
256                         sweep_gen_data <= sine_wave_data;
257                     else
258                         sweep_gen_data <= (others => '0');
259                     end if;
260                 end if;
261         end process;
262 end architecture dac_control_arch;
263
264
265
266
267         -- sine_clk_process: process(clk, reset) is
268         --     begin
269                   --variable counter : integer;
270          --         if(rising_edge(clk)) then
271           --            if(sine_counter < sine_clk_count) then
272             --              sine_counter <= sine_counter + 1;
273               --        else
274                 --          sine_counter <= 0;
275                   --        sine_clk <= not sine_clk;
276                     --  end if;
277                  --elsif reset = '0' then
278                     --  sine_clk <= '0';
279                  -- end if;
280          --end process sine_clk_process;
```

A.3 ext_dac_control.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
use work.sine_package.all;

entity ext_dac_control is

    port (
        clk     : in std_logic;
        reset   : in std_logic;
        enable  : in std_logic;
        sweep_gen_data  : in std_logic_vector(7 downto 0);
        ext_dac_out : out std_logic_vector(7 downto 0);
        wr      : out std_logic;
        A       : out std_logic_vector(1 downto 0)
    );
end entity ext_dac_control;

architecture dac_control_arch of ext_dac_control is

    -- Assuming 1MHz clk frequency
    constant sine_clk_count : integer := 1000; -- 1 -> 50kHz. 25000 for 1Hz, 1
    constant WR_FREQ   : integer := 1000000;   -- 1 us = 1

    signal sine_reset : std_logic;
    signal int_dac_out : std_logic_vector(7 downto 0);
    signal int_reset : std_logic;
    signal int_enable : std_logic;
    signal sine_counter : integer := 0;
    signal sine_clk     : std_logic;
    signal sine_out     : sine_vector_type;
    signal nxt_wr       : std_logic;



    begin
        int_enable <= enable;
        int_reset <= reset;
        sine_reset <= not reset; -- sine_wave uses active high reset
        --int_dac_out <= sweep_gen_data;
        ext_dac_out <= int_dac_out;

        A <= "00"; -- Or whatever channel is used

        -- Action                   |  Min time requirement
        ----------------------------|------
        -- Setup time for data      : 45ns
        -- Hold time, data          : 10ns
        -- Pulse duration, WR low   : 50ns
        -----------------------------------
```

```
51          -- settling time to 1/2 LSB is 5-7us depending on power supply setup
52          --
53
54          wr <= nxt_wr;
55
56          P_WRITE_DATA: process(clk, reset) is
57              variable count : integer := 0;
58              begin
59                  if(reset = '1') then
60                      nxt_wr <= '1';
61                  elsif rising_edge(clk) then
62                      if(count = WR_FREQ) then
63                          --nxt_wr <= '0'; -- Keeps wr low for 2 clock periods
64                          nxt_wr <= not nxt_wr; -- writes every 10 us (100kHz)
65                          count := 0;
66                          int_dac_out <= sweep_gen_data;
67                      else
68                          count := count + 1;
69                      end if;
70                  end if;
71          end process;
72
73
74          sine_clk_process: process(clk, reset) is
75              begin
76                  --variable counter : integer;
77                  if (reset = '0') then
78                      sine_clk <= '0';
79                  elsif (rising_edge(clk)) then
80                      sine_clk <= '0';
81                      if(sine_counter < sine_clk_count) then
82                          sine_counter <= sine_counter + 1;
83                      else
84                          sine_counter <= 0;
85                          sine_clk <= not sine_clk;
86                      end if;
87                  end if;
88          end process sine_clk_process;
89
90  end architecture dac_control_arch;
91
92  --if(int_reset = '0') then
93  --    int_dac_out <= (others => '0');
94  --elsif(rising_edge(clk)) then
95  --    dac_out <= int_dac_out;
96  --end if;
97  --end process;
98
99  --if(int_enable = '1') then
100 --int_dac_out <= wave_out;
101 --end if;
```

A.4 int_dac_control.vhd

```
1  library ieee;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  USE IEEE.NUMERIC_STD.ALL;
4  use work.sine_package.all;
5
6  entity int_dac_control is
7      port (
8          reset_in       : in std_logic;
9          clk_in         : in std_logic;
10         ena             : in std_logic;
11         -- Shift register control
12         mresb          : out std_logic;
13         scr_enbias     : out std_logic;
14         adc_ext_sel    : out std_logic;
15         adc_selch1     : out std_logic;
16         -- Shift register
           ↪  I/O
17         sclk           : out std_logic;
18         --                                  so             : int std_logic;
19         si             : out std_logic;
20         sreadb         : out std_logic;
21         swrite         : out std_logic;
22         -- Configuration register (Serial register)
23         conf_reg_data  : in std_logic_vector(6 DOWNTO 0); -- sweep data
24         reg_bank_sel   : in std_logic_vector(2 DOWNTO 0);
25         conf_reg_latch : in std_logic   -- T21 (KEY[3])
26     );
27
28 end entity int_dac_control;
29
30 architecture int_dac_control_arch of int_dac_control is
31 -- 7-segment display interface
32
33     CONSTANT DB_COUNT         : unsigned := "000001";
34     CONSTANT SER_CLK_PERIOD   : INTEGER  := 40; -- 2400 = 41.67 kHz
35     CONSTANT LATCH_PER          : INTEGER   := 100000000;
36
37     TYPE SER_IN_STATE_TYPE IS (ser_in_init, ser_in_data_write, ser_in_swrite);
38
39     SIGNAL cur_ser_in_state   : SER_IN_STATE_TYPE;
40     SIGNAL nxt_ser_in_state   : SER_IN_STATE_TYPE;
41
42     SIGNAL tb_reset           : std_logic;
43     SIGNAL tb_sclk            : std_logic;
44     SIGNAL tb_si                   : std_logic;
45     signal int_sclk             : std_logic;
46     signal int_sclk_n           : std_logic;
47
48     SIGNAL cur_ser_data_count : INTEGER := 0;
49     SIGNAL nxt_ser_data_count : INTEGER;
```

```
50
51      SIGNAL cur_swrite         : std_logic;
52      SIGNAL nxt_swrite         : std_logic;
53      --
54      SIGNAL cur_si             : std_logic;
55      SIGNAL nxt_si             : std_logic;
56
57      SIGNAL cur_ser_clk        : std_logic;
58      SIGNAL nxt_ser_clk        : std_logic;
59      SIGNAL cur_ser_clk_count  : INTEGER := 0;
60      SIGNAL nxt_ser_clk_count  : INTEGER;
61
62      SIGNAL cur_ser_clk_n      : std_logic;
63      SIGNAL nxt_ser_clk_n      : std_logic;
64      SIGNAL cur_ser_clk_n_count: INTEGER := 0;
65      SIGNAL nxt_ser_clk_n_count: INTEGER;
66
67      SIGNAL reset_db           : std_logic;
68      SIGNAL cur_db_count       : unsigned( 5 DOWNTO 0);
69      SIGNAL nxt_db_count       : unsigned( 5 DOWNTO 0);
70
71      SIGNAL conf_reg_input     : std_logic_vector(53 DOWNTO 0);
72
73      SIGNAL tdac               : std_logic_vector( 6 DOWNTO 0);
74      SIGNAL scr_1_dac          : std_logic_vector( 6 DOWNTO 0);
75      SIGNAL scr_1_conf         : std_logic_vector( 6 DOWNTO 0);
76      SIGNAL scr_2_dac          : std_logic_vector( 6 DOWNTO 0);
77      SIGNAL scr_2_conf         : std_logic_vector( 6 DOWNTO 0);
78
79      SIGNAL cur_tdac           : std_logic_vector( 0 TO 6);
80      SIGNAL nxt_tdac           : std_logic_vector( 0 TO 6);
81
82      SIGNAL cur_scr_1_dac      : std_logic_vector( 0 TO 6);
83      SIGNAL nxt_scr_1_dac      : std_logic_vector( 0 TO 6);
84
85      SIGNAL cur_scr_1_conf     : std_logic_vector( 0 TO 6);
86      SIGNAL nxt_scr_1_conf     : std_logic_vector( 0 TO 6);
87
88      SIGNAL cur_scr_2_dac      : std_logic_vector( 0 TO 6);
89      SIGNAL nxt_scr_2_dac      : std_logic_vector( 0 TO 6);
90
91      SIGNAL cur_scr_2_conf     : std_logic_vector( 0 TO 6);
92      SIGNAL nxt_scr_2_conf     : std_logic_vector( 0 TO 6);
93
94      SIGNAL nibble_0           : unsigned ( 3 DOWNTO  0);
95      SIGNAL nibble_1           : unsigned ( 3 DOWNTO  0);
96      SIGNAL nibble_2           : unsigned ( 3 DOWNTO  0);
97      SIGNAL nibble_3           : unsigned ( 3 DOWNTO  0);
98
99      signal int_enable   : std_logic;
100     signal conf_reg_latch_test  : std_logic;
101
```

```
102     -- Function to reverse array bits stream
103     FUNCTION reverse_array (in_array: std_logic_vector) RETURN std_logic_vector IS VARIABLE
        ↪  out_array: std_logic_vector(6 DOWNTO 0);
104     BEGIN
105
106         FOR i in in_array'RANGE LOOP
107
108             out_array(i) := in_array(i);
109
110         END LOOP;
111
112         RETURN out_array;
113
114     END FUNCTION;
115
116 BEGIN
117
118     -- Shift register I/O
119     mresb         <= not reset_in;
120
121     --sclk          <= tb_sclk AND (NOT(cur_swrite));
122     --tb_sclk <= sclk;
123     si            <= tb_si;
124
125     tb_si         <= cur_si;
126     --scr_enbias           <= '1';
127     --adc_ext_sel   <= '0';  -- External source for ADC
128     --adc_selch1    <= '1';  -- N/A when adc_ext_sel = 1  -- '0'
129     int_enable <= ena;
130
131     nibble_3      <= "0" & unsigned(cur_scr_1_conf(0 TO 2));
132     nibble_2      <= unsigned(cur_scr_1_conf(3 TO 6));
133     nibble_1      <= "0" & unsigned(cur_scr_1_dac(0 TO 2));
134     nibble_0      <= unsigned(cur_scr_1_dac(3 TO 6));
135
136     -- Configuration bit stream
137     tdac       <= reverse_array(cur_tdac);
138     scr_1_dac  <= reverse_array(cur_scr_1_dac);
139     scr_1_conf <= reverse_array(cur_scr_1_conf);
140     scr_2_dac  <= reverse_array(cur_scr_2_dac);
141     scr_2_conf <= reverse_array(cur_scr_2_conf);
142
143     --conf_reg_input <= scr_1_conf & scr_1_dac & scr_2_conf & scr_2_dac & tdac & tdac(0) &
        ↪  "1000000000000000001";
144     conf_reg_input <= "0000000" & conf_reg_data & "00000000000000000000" &
        ↪  "0000000000000000";
145
146     --Serial clock
147     sclk <= int_sclk and not(cur_swrite);
148
149     -- Switch (reset) debounce
150     --tb_reset      <= not reset_in;--reset_db; -- NOT(reset_db);
```

```
151     reset_db      <= '1'             WHEN cur_db_count = "000000" ELSE '0';
152     nxt_db_count  <= (OTHERS => '0') WHEN cur_db_count = "000000" ELSE cur_db_count + 1;
153
154     -- Shift enable
155     sreadb <= '1';
156     swrite <= cur_swrite;
157
158     P_SCLK_GEN: process(clk_in, reset_in) is
159         variable count : integer := 0;
160         begin
161             if(reset_in = '1') then
162                 count := 0;
163                 int_sclk <= '0';
164                 int_sclk_n <= '1';
165             elsif rising_edge(clk_in) then
166                 count := count + 1;
167                 if(count = SER_CLK_PERIOD) then
168                     int_sclk <= not int_sclk;
169                     int_sclk_n <= not int_sclk_n;
170                     count := 0;
171                 end if;
172             end if;
173     end process;
174
175     PROCESS(clk_in, reset_in)
176     BEGIN
177         IF (reset_in = '1') THEN
178             cur_db_count        <= DB_COUNT;
179             --cur_ser_clk         <= '0';
180             cur_ser_clk_count   <= 0;
181             cur_ser_clk_n       <= '1';
182             cur_ser_clk_n_count <= 0;
183
184         ELSIF (clk_in'EVENT AND clk_in = '1') THEN
185             cur_db_count        <= nxt_db_count;
186             --cur_ser_clk         <= nxt_ser_clk;
187             cur_ser_clk_count   <= nxt_ser_clk_count;
188             cur_ser_clk_n       <= nxt_ser_clk_n;
189             cur_ser_clk_n_count <= nxt_ser_clk_n_count;
190
191         END IF;
192
193     END PROCESS;
194
195     ser_in_process: PROCESS(cur_ser_in_state, nxt_ser_in_state, cur_ser_data_count, cur_si,
        ↪  cur_swrite, conf_reg_input)
196     BEGIN
197
198         nxt_ser_in_state   <= cur_ser_in_state;
199         nxt_ser_data_count <= cur_ser_data_count;
200         nxt_si             <= cur_si;
201         nxt_swrite         <= cur_swrite;
```

```
202
203         CASE cur_ser_in_state IS
204
205             WHEN ser_in_init =>
206
207                     nxt_swrite       <= '0';
208                     nxt_ser_in_state <= ser_in_data_write;
209
210             WHEN ser_in_data_write =>
211
212                     nxt_si <= conf_reg_input(cur_ser_data_count);
213
214                     IF (cur_ser_data_count = 53) THEN --53
215
216                         nxt_ser_data_count <= 0; -- resets the serial data count
217                         nxt_swrite         <= '1';  -- Assert the write pulse
218                         nxt_ser_in_state
                            ↪  <=        ser_in_swrite;
219                     ELSE
220                         nxt_ser_data_count <= cur_ser_data_count + 1;
221                     END IF;
222
223             WHEN ser_in_swrite =>
224
225                     nxt_swrite       <= '0';  -- Deassert the write
                        ↪  pulse
226                     nxt_ser_in_state <= ser_in_init;
227
228             WHEN OTHERS =>
229
230                     nxt_ser_in_state <= ser_in_init;
231
232         END CASE;
233
234     END PROCESS;
235
236     ser_pos_clk_process: PROCESS(reset_in, int_sclk)
237     BEGIN
238
239         IF (reset_in = '1') THEN
240
241             cur_ser_in_state   <= ser_in_init;
242             cur_ser_data_count <= 0;
243             cur_swrite         <= '0';
244     --           cur_si             <= '0';
245
246         ELSIF rising_edge(int_sclk) THEN
247
248             cur_ser_in_state   <= nxt_ser_in_state;
249             cur_ser_data_count <= nxt_ser_data_count;
250             cur_swrite         <= nxt_swrite;
251     --            cur_si             <= nxt_si;
```

```
252
253         END IF;
254
255     END PROCESS;
256
257     ser_neg_clk_process: PROCESS(reset_in, int_sclk_n)
258     BEGIN
259
260         IF (reset_in = '1') THEN
261
262                 cur_si  <= '0';
263
264         ELSIF rising_edge(int_sclk_n) THEN
265
266             cur_si  <= nxt_si;
267
268         END IF;
269
270     END PROCESS;
271
272
273     PROCESS(conf_reg_latch, reg_bank_sel, conf_reg_data, cur_tdac, cur_scr_2_dac,
        ↪  cur_scr_2_conf, cur_scr_1_dac, cur_scr_1_conf)
274     BEGIN
275
276         nxt_tdac       <= cur_tdac;
277         nxt_scr_2_dac  <= cur_scr_2_dac;
278         nxt_scr_2_conf <= cur_scr_2_conf;
279         nxt_scr_1_dac  <= cur_scr_1_dac;
280         nxt_scr_1_conf <= cur_scr_1_conf;
281
282         IF (conf_reg_latch = '0') THEN
283
284             CASE reg_bank_sel IS
285
286                 WHEN "000" => nxt_tdac       <= conf_reg_data;
287
288                 WHEN "001" => nxt_scr_2_dac  <= conf_reg_data;
289
290                 WHEN "010" => nxt_scr_2_conf <= conf_reg_data;
291
292                 WHEN "011" => nxt_scr_1_dac  <= conf_reg_data;
293
294                 WHEN "100" => nxt_scr_1_conf <= conf_reg_data;
295
296                 WHEN OTHERS =>
297
298             END CASE;
299
300         END IF;
301
302     END PROCESS;
```

```
303
304     PROCESS(reset_in, int_sclk)
305     BEGIN
306
307         IF (reset_in = '1') THEN
308
309             cur_tdac       <= (OTHERS => '0');
310             cur_scr_2_dac  <= (OTHERS => '0');
311             cur_scr_2_conf <= (OTHERS => '0');
312             cur_scr_1_dac  <= (OTHERS => '0');
313             cur_scr_1_conf <= (OTHERS => '0');
314
315         ELSIF rising_edge(int_sclk) THEN
316
317             cur_tdac       <= nxt_tdac;
318             cur_scr_2_dac  <= nxt_scr_2_dac;
319             cur_scr_2_conf <= nxt_scr_2_conf;
320             cur_scr_1_dac  <= nxt_scr_1_dac;
321             cur_scr_1_conf <= nxt_scr_1_conf;
322
323         END IF;
324
325 END PROCESS;
326 end architecture int_dac_control_arch;
```

A.5 tb_adc_sawtooth.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  USE IEEE.NUMERIC_STD.ALL;
4  --USE WORK.spi_defs_pkg.ALL;
5
6  ENTITY tb_adc_sawtooth IS
7
8          PORT (-- Board input
9                          reset_in     : IN  STD_LOGIC;  -- R22 (KEY[0])
10               clk_in       : IN  STD_LOGIC;  -- A12 (24 MHz)
11                                  -- ADC select
12                                  adc_selch1   : OUT STD_LOGIC;  -- G18 (GPIO_1[25])
13                                  adc_ext_sel  : OUT STD_LOGIC;  -- G20 (GPIO_1[24])
14                                  scr_enbias   : OUT STD_LOGIC;  -- E18 (GPIO_1[23])
15                                  -- ADC control
16                                  mresb        : OUT STD_LOGIC;  -- E19 (GPIO_1[22])
17                                  adc_clk      : OUT STD_LOGIC;  -- F20 (GPIO_1[21])
18                                  adc_en       : OUT STD_LOGIC;  -- E20 (GPIO_1[20])
19                                  adc_eoc      : IN  STD_LOGIC;  -- D20 (GPIO_1[19])
20                                  -- ADC output interface
21                                  sclk         : OUT STD_LOGIC;  -- D19 (GPIO_1[18])
22                                  so           : IN  STD_LOGIC;  -- C20 (GPIO_1[17])
23                                  si           : OUT STD_LOGIC;  -- C19 (GPIO_1[16])
24                                  sreadb       : OUT STD_LOGIC;  -- C18 (GPIO_1[15])
25                                  swrite       : OUT STD_LOGIC;  -- C17 (GPIO_1[14])
26                              -- 7-Segment LED display
27                         adc_data_3   : OUT STD_LOGIC_VECTOR ( 6 DOWNTO 0);
28                         adc_data_2   : OUT STD_LOGIC_VECTOR ( 6 DOWNTO 0);
29                         adc_data_1   : OUT STD_LOGIC_VECTOR ( 6 DOWNTO 0);
30                         adc_data_0   : OUT STD_LOGIC_VECTOR ( 6 DOWNTO 0);
31                                  -- Average ADC output
32                     dsp_sel_swt  : IN  STD_LOGIC;  -- L22 (SW[0])
33                                  -- Test points
34                                 test_point_0 : OUT STD_LOGIC;  -- H12 (GPIO_1[0])
35                                 test_point_1 : OUT STD_LOGIC;  -- H13 (GPIO_1[1])
36                   test_point_2 : OUT STD_LOGIC;   -- H14 (GPIO_1[2])
37                -- sawtooth generation
38                                 sawtooth_signal     : OUT std_logic;
39                                 adc_result : out std_logic_vector(15 downto 0)
40             );
41
42 END ENTITY tb_adc_sawtooth;
43
44 ARCHITECTURE tb_adc_sawtooth_arch OF tb_adc_sawtooth IS
45
46     -- 7-segment display interface
47     TYPE DISP_CODE_ARRAY IS ARRAY ( 0 TO 15) OF STD_LOGIC_VECTOR( 7 DOWNTO 0);
48     CONSTANT DISP_CODE : DISP_CODE_ARRAY := (X"40", X"79", X"24", X"30", X"19", X"12",
       ↪  X"02", X"78", X"00", X"10", X"08", X"03", X"46", X"21", X"06", X"0E");
49
```

```
50      CONSTANT DB_COUNT         : UNSIGNED := "000001";
51      CONSTANT ADC_CLK_PERIOD   : INTEGER  :=  147;  -- 10 -- 147
52      CONSTANT SER_CLK_PERIOD   : INTEGER  :=  20; --147;  --  2 --10
53
54      TYPE ADC_CONV_STATE_TYPE IS (adc_conv_init, adc_conv_start, adc_eoc_wait,
        ↪  adc_read_value, adc_conv_stop);
55          TYPE SER_IO_STATE_TYPE   IS (ser_io_init, ser_io_start, ser_io_data_ready,
            ↪  ser_io_data_latch, ser_io_data_read, ser_io_stop);
56          TYPE ADC_AVG_CALC_STATE_TYPE IS (wait_calc_start, calc_adc_avg);
57
58      SIGNAL cur_adc_conv_state : ADC_CONV_STATE_TYPE;
59      SIGNAL nxt_adc_conv_state : ADC_CONV_STATE_TYPE;
60
61          SIGNAL cur_ser_io_state   : SER_IO_STATE_TYPE;
62          SIGNAL nxt_ser_io_state   : SER_IO_STATE_TYPE;
63
64      SIGNAL tb_reset           : STD_LOGIC;
65      SIGNAL tb_adc_clk         : STD_LOGIC;
66          SIGNAL tb_adc_en          : STD_LOGIC;
67          SIGNAL tb_adc_eoc         : STD_LOGIC;
68          SIGNAL tb_sclk            : STD_LOGIC;
69          SIGNAL tb_so              : STD_LOGIC;
70          SIGNAL tb_si                  : STD_LOGIC;
71          SIGNAL tb_sreadb          : STD_LOGIC;
72          SIGNAL tb_swrite               : STD_LOGIC;
73
74          SIGNAL cur_adc_val_rdy    : STD_LOGIC;
75          SIGNAL nxt_adc_val_rdy    : STD_LOGIC;
76
77          SIGNAL cur_ser_read       : STD_LOGIC;
78          SIGNAL nxt_ser_read       : STD_LOGIC;
79
80          SIGNAL cur_adc_en         : STD_LOGIC;
81          SIGNAL nxt_adc_en         : STD_LOGIC;
82
83          SIGNAL cur_sreadb         : STD_LOGIC;
84          SIGNAL nxt_sreadb         : STD_LOGIC;
85
86          SIGNAL cur_swrite         : STD_LOGIC;
87          SIGNAL nxt_swrite         : STD_LOGIC;
88
89          SIGNAL cur_si             : STD_LOGIC;
90          SIGNAL nxt_si             : STD_LOGIC;
91
92          SIGNAL cur_so             : STD_LOGIC;
93          SIGNAL nxt_so             : STD_LOGIC;
94
95          SIGNAL cur_adc_bit_count  : INTEGER := 0;
96          SIGNAL nxt_adc_bit_count  : INTEGER;
97
98      SIGNAL cur_adc_data       : STD_LOGIC_VECTOR (17 DOWNTO  0);
99      SIGNAL nxt_adc_data       : STD_LOGIC_VECTOR (17 DOWNTO  0);
```

```
100
101     SIGNAL nibble_0           : UNSIGNED ( 3 DOWNTO  0);
102     SIGNAL nibble_1           : UNSIGNED ( 3 DOWNTO  0);
103     SIGNAL nibble_2           : UNSIGNED ( 3 DOWNTO  0);
104     SIGNAL nibble_3           : UNSIGNED ( 3 DOWNTO  0);
105
106 --        SIGNAL inv_adc_data_1     : STD_LOGIC_VECTOR ( 0     TO 15);
107         SIGNAL inv_adc_data_2     : UNSIGNED (15 DOWNTO  0);
108
109         SIGNAL cur_inv_adc_data   : STD_LOGIC_VECTOR (15 DOWNTO  0);
110         SIGNAL nxt_inv_adc_data   : STD_LOGIC_VECTOR (15 DOWNTO  0);
111
112     SIGNAL cur_adc_clk        : STD_LOGIC;
113     SIGNAL nxt_adc_clk        : STD_LOGIC;
114     SIGNAL cur_adc_clk_count  : INTEGER := 0;
115     SIGNAL nxt_adc_clk_count  : INTEGER;
116
117     SIGNAL cur_adc_clk_n      : STD_LOGIC;
118     SIGNAL nxt_adc_clk_n      : STD_LOGIC;
119     SIGNAL cur_adc_clk_n_count: INTEGER := ADC_CLK_PERIOD;
120     SIGNAL nxt_adc_clk_n_count: INTEGER;
121
122     SIGNAL cur_ser_clk        : STD_LOGIC;
123     SIGNAL nxt_ser_clk        : STD_LOGIC;
124     SIGNAL cur_ser_clk_count  : INTEGER := 0;
125     SIGNAL nxt_ser_clk_count  : INTEGER;
126
127     SIGNAL cur_ser_clk_n      : STD_LOGIC;
128          SIGNAL nxt_ser_clk_n      : STD_LOGIC;
129     SIGNAL cur_ser_clk_n_count: INTEGER := SER_CLK_PERIOD;
130     SIGNAL nxt_ser_clk_n_count: INTEGER;
131
132     SIGNAL reset_db           : STD_LOGIC;
133     SIGNAL cur_db_count       : UNSIGNED( 5 DOWNTO 0);
134     SIGNAL nxt_db_count       : UNSIGNED( 5 DOWNTO 0);
135
136     SIGNAL cur_avg_calc_state : ADC_AVG_CALC_STATE_TYPE;
137     SIGNAL nxt_avg_calc_state : ADC_AVG_CALC_STATE_TYPE;
138
139     SIGNAL cur_adc_avg        : UNSIGNED(16 DOWNTO 0);
140     SIGNAL nxt_adc_avg        : UNSIGNED(16 DOWNTO 0);
141
142     SIGNAL cur_min_adc_val    : UNSIGNED(16 DOWNTO 0);
143     SIGNAL nxt_min_adc_val    : UNSIGNED(16 DOWNTO 0);
144
145     SIGNAL cur_max_adc_val    : UNSIGNED(16 DOWNTO 0);
146         SIGNAL nxt_max_adc_val    : UNSIGNED(16 DOWNTO 0);
147
148         signal cur_eoc_trigger        : std_logic;
149         signal nxt_eoc_trigger        : std_logic;
150
151         signal nxt_rdy                : std_logic;
```

```
152
153 BEGIN
154
155     -- ADC select
156     adc_ext_sel   <= '1';  -- External source for ADC
157          adc_selch1    <= '1';  -- N/A when adc_ext_sel = 1  -- '0'
158          scr_enbias    <= '0';  -- Screen voltage from off chip
159
160          -- ADC control
161     mresb         <= tb_reset;
162     adc_clk       <= tb_adc_clk;
163          adc_en        <= tb_adc_en;
164          tb_adc_eoc    <= adc_eoc;
165
166          -- ADC output interface
167          sclk          <= tb_sclk;
168          tb_so         <= so;
169          si            <= tb_si;
170          sreadb        <= tb_sreadb;
171          swrite        <= tb_swrite;
172
173          tb_adc_en     <= cur_adc_en;
174          tb_sreadb     <= cur_sreadb;
175          tb_swrite     <= cur_swrite;
176          tb_si         <= cur_si;
177
178          -- 7-segment Display output
179          adc_data_3    <= DISP_CODE(TO_INTEGER(nibble_3))(6 DOWNTO 0);
180          adc_data_2    <= DISP_CODE(TO_INTEGER(nibble_2))(6 DOWNTO 0);
181          adc_data_1    <= DISP_CODE(TO_INTEGER(nibble_1))(6 DOWNTO 0);
182     adc_data_0    <= DISP_CODE(TO_INTEGER(nibble_0))(6 DOWNTO 0);
183
184          nibble_3      <= inv_adc_data_2(15 DOWNTO 12) WHEN dsp_sel_swt = '0' ELSE
          ↪  cur_adc_avg(15 DOWNTO 12);
185          nibble_2      <= inv_adc_data_2(11 DOWNTO  8) WHEN dsp_sel_swt = '0' ELSE
          ↪  cur_adc_avg(11 DOWNTO  8);
186          nibble_1      <= inv_adc_data_2( 7 DOWNTO  4) WHEN dsp_sel_swt = '0' ELSE
          ↪  cur_adc_avg( 7 DOWNTO  4);
187          nibble_0      <= inv_adc_data_2( 3 DOWNTO  0) WHEN dsp_sel_swt = '0' ELSE
          ↪  cur_adc_avg( 3 DOWNTO  0);
188
189           inv_adc_data_2  <= UNSIGNED(cur_inv_adc_data);
190 --  inv_adc_data_2  <= inv_adc_data_1;
191 --         inv_adc_data_1  <= cur_adc_data(17 DOWNTO 2);
192
193     -- 1-segment Display output
194          --sr_out <= cur_adc_data;
195
196          -- Test points
197     test_point_0        <= cur_adc_val_rdy;
198     test_point_1        <= cur_ser_read;
199
```

```
200 --    test_point_0        <= '0';
201 --    test_point_1        <= '0';
202     test_point_2        <= '0';
203 
204           -- ADC clock
205           --tb_adc_clk        <=  cur_adc_clk;
206           nxt_adc_clk       <=  cur_adc_clk           WHEN cur_adc_clk_count <
              ↪  ADC_CLK_PERIOD ELSE (cur_adc_clk XOR '1');
207           nxt_adc_clk_count <= (cur_adc_clk_count + 1) WHEN cur_adc_clk_count <
              ↪  ADC_CLK_PERIOD ELSE 0;
208 
209           nxt_adc_clk_n       <=  cur_adc_clk_n           WHEN cur_adc_clk_n_count <
              ↪  ADC_CLK_PERIOD ELSE (cur_adc_clk_n XOR '1');
210           nxt_adc_clk_n_count <= (cur_adc_clk_n_count + 1) WHEN cur_adc_clk_n_count <
              ↪  ADC_CLK_PERIOD ELSE 0;
211 
212           -- Serial clock
213           tb_sclk           <=  cur_ser_clk;
214           nxt_ser_clk       <=  cur_ser_clk           WHEN cur_ser_clk_count <
              ↪  SER_CLK_PERIOD ELSE (cur_ser_clk XOR '1');
215           nxt_ser_clk_count <= (cur_ser_clk_count + 1) WHEN cur_ser_clk_count <
              ↪  SER_CLK_PERIOD ELSE 0;
216 
217           nxt_ser_clk_n       <=  cur_ser_clk_n           WHEN cur_ser_clk_n_count <
              ↪  SER_CLK_PERIOD ELSE (cur_ser_clk_n XOR '1');
218           nxt_ser_clk_n_count <= (cur_ser_clk_n_count + 1) WHEN cur_ser_clk_n_count <
              ↪  SER_CLK_PERIOD ELSE 0;
219 
220     -- Switch debounce for Reset
221     tb_reset      <= reset_db; -- NOT(reset_db);
222           reset_db      <= '1'            WHEN cur_db_count = "000000" ELSE '0';
223           nxt_db_count  <= (OTHERS => '0') WHEN cur_db_count = "000000" ELSE cur_db_count +
              ↪  1;
224 
225           PROCESS(clk_in, reset_in)
226           BEGIN
227 
228               IF (reset_in = '1') THEN
229 
230                       cur_db_count        <= DB_COUNT;
231                           cur_adc_clk         <= '0';
232                           cur_adc_clk_n       <= '0';
233                           cur_ser_clk         <= '0';
234                           cur_ser_clk_n       <= '0';
235                           cur_adc_clk_count   <= 0;
236                           cur_adc_clk_n_count <=
                              ↪  ADC_CLK_PERIOD;
237                           cur_ser_clk_count   <= 0;
238                           cur_ser_clk_n_count <= SER_CLK_PERIOD;
239 
240           ELSIF (clk_in'EVENT AND clk_in = '1') THEN
241 
```

```
242                     cur_db_count        <= nxt_db_count;
243                          cur_adc_clk         <= nxt_adc_clk;
244                          cur_adc_clk_n       <=
                             ↪  nxt_adc_clk_n;
245                          cur_ser_clk         <= nxt_ser_clk;
246                               cur_ser_clk_n       <= nxt_ser_clk_n;
247                          cur_adc_clk_count   <= nxt_adc_clk_count;
248                          cur_adc_clk_n_count <= nxt_adc_clk_n_count;
249                               cur_ser_clk_count   <= nxt_ser_clk_count;
250                               cur_ser_clk_n_count <= nxt_ser_clk_n_count;
251
252         END IF;
253
254         END PROCESS;
255
256         P_TB_ADC_CLK: process(clk_in, reset_in) is
257         variable counter : integer := 0;
258         begin
259                 if(reset_in = '1') then
260                         tb_adc_clk <= '0';
261                         counter := 0;
262                 elsif rising_edge(clk_in) then
263                         if(counter = ADC_CLK_PERIOD) then
264                                 tb_adc_clk <= not tb_adc_clk;
265                                 counter := 0;
266                         end if;
267                         counter := counter + 1;
268                 end if;
269         end process;
270
271
272          adc_conv_process: PROCESS(cur_adc_conv_state, cur_adc_val_rdy, cur_adc_en,
             ↪  cur_ser_read, tb_adc_eoc)
273          BEGIN
274
275              nxt_adc_conv_state <= cur_adc_conv_state;
276                   nxt_adc_val_rdy    <= cur_adc_val_rdy;
277                   nxt_adc_en         <= cur_adc_en;
278                   nxt_eoc_trigger         <= cur_eoc_trigger;
279
280                   CASE cur_adc_conv_state IS
281
282                       WHEN adc_conv_init =>
283
284                                       --eoc_trigger        <= '0';
285                                  nxt_adc_conv_state <= adc_conv_start;
286
287                                WHEN adc_conv_start =>
288
289                                       nxt_adc_en         <= '1';
290                                       nxt_eoc_trigger <= '0';
291                                  nxt_adc_conv_state <= adc_eoc_wait;
```

```
292
293                                 WHEN adc_eoc_wait
                                    ↪  =>
294
295                                     --IF (tb_adc_eoc = '1') THEN
296
297                                             nxt_eoc_trigger <= '1';
298                                               nxt_adc_val_rdy    <= '1';
299                                               nxt_rdy <= '0';
300                                              nxt_adc_conv_state <= adc_conv_stop; --
                                                 ↪  adc_read_value;
301                                              --adc_result <= cur_adc_data(16 downto 1);
302                                              nxt_adc_en <= '0'; -- '1';
303                                              --eoc_trigger <= '1';
304
305                                         --END IF;
306
307                                 WHEN adc_read_value =>
308
309 --                                    nxt_adc_val_rdy    <= '1';
310                                       nxt_adc_conv_state <= adc_conv_stop;
311
312                                 WHEN adc_conv_stop =>
313
314                             --IF (cur_ser_read = '0') THEN
315
316                                       nxt_adc_val_rdy    <= '0';
317                                       nxt_adc_conv_state <=
                                          ↪  adc_conv_init;
318
319                                       -- END IF;
320
321                     WHEN OTHERS =>
322
323                                   nxt_adc_conv_state <= adc_conv_init;
324
325         END CASE;
326
327          END PROCESS;
328
329     adc_pos_clk_process: PROCESS(tb_reset, tb_adc_clk)
330     BEGIN
331
332         IF (tb_reset = '0') THEN
333
334             cur_adc_conv_state <= adc_conv_init;
335             cur_adc_en         <= '0';
336                          cur_adc_val_rdy    <= '0';
337                       cur_eoc_trigger         <= '0';
338
339                   ELSIF (tb_adc_clk'EVENT AND tb_adc_clk = '1') THEN
340
```

```
341                 cur_adc_conv_state <= nxt_adc_conv_state;
342                      cur_adc_val_rdy    <= nxt_adc_val_rdy;
343                               cur_adc_en         <= nxt_adc_en;
344                               cur_eoc_trigger <= nxt_eoc_trigger;
345
346
347                  END IF;
348
349      END PROCESS;
350
351           ser_io_process: PROCESS(cur_ser_io_state, cur_adc_bit_count, cur_adc_data,
             ↪  cur_ser_read, cur_sreadb, cur_swrite, cur_si, cur_adc_val_rdy, cur_so,
             ↪  cur_inv_adc_data)
352           BEGIN
353
354               nxt_ser_io_state  <= cur_ser_io_state;
355                    nxt_adc_bit_count <= cur_adc_bit_count;
356                    nxt_adc_data      <= cur_adc_data;
357                    nxt_ser_read      <= cur_ser_read;
358                    nxt_sreadb        <= cur_sreadb;
359                    nxt_swrite        <= cur_swrite;
360                    nxt_si            <= cur_si;
361          nxt_inv_adc_data  <= cur_inv_adc_data;
362
363                    CASE cur_ser_io_state IS
364
365              WHEN ser_io_init =>
366
367                                      nxt_sreadb       <= '1';
368                                           nxt_swrite       <= '0';
369                                           nxt_ser_read     <= '1';
370                                           --adc_result                <= (others => '0');
371                                           nxt_ser_io_state <= ser_io_start;
372
373              WHEN ser_io_start =>
374
375                                      --if(cur_adc_val_rdy = '1')
                                         ↪  then
376                                          nxt_sreadb       <=
                                             ↪  '0';
377                                               nxt_ser_io_state <= ser_io_data_ready;
378
379                                           --END IF;
380
381                                 WHEN ser_io_data_ready =>
382
383                                      nxt_sreadb       <= '1';  -- '0';
384                                           nxt_ser_io_state <= ser_io_data_read; --
                                             ↪  ser_io_data_latch;
385
386  --            WHEN ser_io_data_latch =>
387  --
```

```
388 --                              nxt_sreadb       <= '1';
389 --                                      nxt_ser_io_state <= ser_io_data_read;
390
391             WHEN ser_io_data_read =>
392
393                                     nxt_adc_data(0)           <= cur_so;
394                                         nxt_adc_data(17 DOWNTO 1) <= cur_adc_data(16
                                            ↪  DOWNTO 0);
395                                         nxt_si                    <= '0';
396
397                                         IF (cur_adc_bit_count = 17) THEN  -- 17
398
399                                              nxt_ser_read      <=
                                                 ↪  '0';
400                                                 nxt_adc_bit_count <= 0;
401                                                 nxt_ser_io_state  <= ser_io_stop;
402
403                                         ELSE
404
405                                             nxt_adc_bit_count <= cur_adc_bit_count + 1;
406
407                                         END IF;
408
409             WHEN ser_io_stop =>
410
411                                     --IF (cur_adc_val_rdy = '0') THEN
412
413                                               nxt_ser_read      <=
                                                  ↪  '1';
414                                               nxt_inv_adc_data  <= cur_adc_data(16 DOWNTO
                                                  ↪  1); -- cur_adc_data(16 DOWNTO 1); --
415                                               adc_result        <= cur_adc_data(16 downto
                                                  ↪  1);
416                                               nxt_ser_io_state  <= ser_io_init;
417
418                                         --END IF;
419
420                     WHEN OTHERS =>
421
422                                     nxt_ser_io_state <= ser_io_init;
423
424         END CASE;
425
426          END PROCESS;
427
428     ser_pos_clk_process: PROCESS(tb_reset, tb_sclk)
429     BEGIN
430
431         IF (tb_reset = '0') THEN
432
433             cur_inv_adc_data        <= (OTHERS => '0');
434             cur_ser_io_state  <= ser_io_init;
```

```
435                         cur_adc_bit_count <= 0;
436                        cur_adc_data      <= (OTHERS => '1');        -- '0'
437                             cur_ser_read      <= '0';
438                                  cur_swrite        <= '0';
439                                  cur_si            <= '0';
440 --            cur_so            <= '0';
441 --            cur_test_point_0  <= '0';
442 --            cur_test_point_1  <= '0';
443 --            cur_test_point_2  <= '0';
444 
445                     ELSIF (tb_sclk'EVENT AND tb_sclk = '1') THEN
446 
447             cur_inv_adc_data        <= nxt_inv_adc_data;
448                  cur_ser_io_state  <= nxt_ser_io_state;
449                       cur_adc_bit_count <= nxt_adc_bit_count;
450                      cur_adc_data      <= nxt_adc_data;
451             cur_ser_read      <= nxt_ser_read;
452                               cur_swrite        <= nxt_swrite;
453                          cur_si            <= nxt_si;
454 --            cur_so            <= tb_so;
455 --            cur_test_point_0  <= nxt_test_point_0;
456 --            cur_test_point_1  <= nxt_test_point_1;
457 --            cur_test_point_2  <= nxt_test_point_2;
458 
459                     END IF;
460 
461     END PROCESS;
462 
463     ser_neg_clk_process: PROCESS(tb_reset, cur_ser_clk_n)
464     BEGIN
465 
466         IF (tb_reset = '0') THEN
467 
468             cur_so            <= '0';
469 --            cur_ser_read      <= '0';
470                               cur_sreadb        <= '1';
471 
472                      ELSIF (cur_ser_clk_n'EVENT AND cur_ser_clk_n = '1') THEN
473 
474             cur_so            <= tb_so;
475 --            cur_ser_read      <= nxt_ser_read;
476                               cur_sreadb        <= nxt_sreadb;
477 
478                     END IF;
479 
480     END PROCESS;
481 
482 
483 
484 
485 END;
486 
```

```
487           -- Data selection for display output
488   --        nibble_0        <= cur_adc_data( 3 DOWNTO  0) WHEN disp_sel_swt = "00" ELSE
489   --                           cur_adc_data(19 DOWNTO 16) WHEN disp_sel_swt = "01" ELSE
490   --                                                      cur_adc_data(35 DOWNTO 32) WHEN
      ↪  disp_sel_swt = "10" ELSE
491   --                                                      cur_adc_data(51 DOWNTO 48);
492   --        nibble_1        <= cur_adc_data( 7 DOWNTO  4) WHEN disp_sel_swt = "00" ELSE
493   --                           cur_adc_data(23 DOWNTO 20) WHEN disp_sel_swt = "01" ELSE
494   --                                                      cur_adc_data(39 DOWNTO 36) WHEN
      ↪  disp_sel_swt = "10" ELSE
495   --                                                      ("00" & cur_adc_data(53 DOWNTO
      ↪  52));
496   --        nibble_2        <= cur_adc_data(11 DOWNTO  8) WHEN disp_sel_swt = "00" ELSE
497   --                           cur_adc_data(27 DOWNTO 24) WHEN disp_sel_swt = "01" ELSE
498   --                                                      cur_adc_data(43 DOWNTO 40) WHEN
      ↪  disp_sel_swt = "10" ELSE
499   --                                                      "0000";
500   --        nibble_3        <= cur_adc_data(15 DOWNTO 12) WHEN disp_sel_swt = "00" ELSE
501   --                           cur_adc_data(31 DOWNTO 28) WHEN disp_sel_swt = "01" ELSE
502   --                                                      cur_adc_data(47 DOWNTO 44) WHEN
      ↪  disp_sel_swt = "10" ELSE
503   --                                                      "0000";
```

A.6 ext_adc_control.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  USE IEEE.NUMERIC_STD.ALL;
4
5  entity ext_adc_control is
6      port (
7          mclk_ext_adc    : in std_logic;
8          enable  : in std_logic; -- outside decider of exADC is being used
9          reset   : in std_logic; -- reset signal for exADC
10
11         dout    : in std_logic; -- DOUT
12         sstrb   : in std_logic; -- SSTRB is low during adc conversion, for tconv time
13
14         din     : out std_logic; -- digital in, write to adc
15         adc_rst : out std_logic; -- reset adc
16         exadc_clk   : out std_logic; -- clock for adc
17         shdn    : out std_logic; -- drive shdn low to put the adc in shutdown mode
18         cs      : out std_logic;
19
20         exADC_result    : out std_logic_vector(15 downto 0) -- result to pcb_interface
21     );
22 end entity ext_adc_control;
23
24 architecture arch of ext_adc_control is
25
26
27     constant CALIBRATE_WORD     : std_logic_vector(7 downto 0) := "11101000";  -- 1: Start,
       ↪  1: unipolar, 1: internal clock, 01: Start calibration, 000: Don't care?
28     constant START_WORD         : std_logic_vector(7 downto 0) := "11100000";  -- 1: Start,
       ↪  1: unipolar, 1: internal clock, 00: Short acqusition mode(24 ext clk
       ↪  periods/conversion), 000: Don't care?
29     constant ADC_CLK_PERIOD     : integer := 25; -- 25 = 2MHz
30     constant ENABLE_CLOCK       : integer := 1000;
31
32     signal ext_clk_ena      : std_logic := '0';
33     signal ext_adc_clk      : std_logic;
34
35     --signal cs             : std_logic;
36
37     TYPE EXT_ADC_CONV_STATE is (ADC_IDLE, ADC_WRITE_CALIBRATE,ADC_WRITE_START,
       ↪  ADC_CLK_START, ADC_CS_LOW, ADC_START, ADC_WAIT, ADC_READ, ADC_DONE);
38
39     signal state : EXT_ADC_CONV_STATE;
40     signal nxt_state : EXT_ADC_CONV_STATE;
41
42     signal result   : std_logic_vector(15 downto 0);
43
44     signal dout_d   : std_logic;
45
46     signal int_enable : std_logic;
```

```
47
48
49      begin
50          -- ADC Clock generation
51
52          --exadc_clk <= ext_adc_clk when(sstrb = '1') else '0'; -- external serial clock
            ↪  goes low during conversion, this gives better noise performance
53
54          exADC_result <= result;
55
56         -- P_TEST: process(sstrb,ext_adc_clk) is
57         --     begin
58         --         if(sstrb = '1') then
59         --             exadc_clk <= ext_adc_clk;
60         --         elsif(sstrb = '0') then
61         --             --exadc_clk <= '0';
62         --         end if;
63         -- end process;
64
65         --exadc_clk <= ext_adc_clk and ext_clk_ena;
66         shdn <= '1';
67
68          P_INT_ENABLE_CLK: process(ext_adc_clk, reset) is
69              variable count : integer := 0;
70              begin
71                  if(reset = '1') then
72                      count := 0;
73                      --ext_adc_clk <= '0';
74                      --ext_clk_ena <= '0';
75                      --exadc_clk   <= '0';
76                      int_enable <= '1';
77                  elsif rising_edge(ext_adc_clk) then
78                      if(count = ENABLE_CLOCK) then
79                          int_enable <= '1';
80                          count := 0;
81                      else
82                          count := count + 1;
83                          int_enable <= '0';
84                      end if;
85                  end if;
86          end process;
87
88          P_EXT_ADC_GEN: process(mclk_ext_adc, reset) is
89              variable count : integer := 0;
90              begin
91                  if(reset = '1') then
92                      count := 0;
93                      ext_adc_clk <= '0';
94                      --ext_clk_ena <= '0';
95                      exadc_clk   <= '0';
96                  elsif rising_edge(mclk_ext_adc) then
97                      count := count + 1;
```

```
98                  if(count = ADC_CLK_PERIOD) then
99                      ext_adc_clk <= not ext_adc_clk;
100                     count := 0;
101                     if(ext_clk_ena = '1') then
102                         exadc_clk   <= not ext_adc_clk;
103                     elsif(ext_clk_ena = '0') then
104                         exadc_clk   <= '0';
105                     end if;
106                 end if;
107             end if;
108         end process;
109
110         P_DOUT_SAMPLING: process(ext_adc_clk, reset) is
111             begin
112                 if(reset = '1') then
113                     dout_d <= '1';
114                 elsif falling_edge(ext_adc_clk) then
115                     dout_d <= dout;
116                 end if;
117         end process;
118
119
120         P_RESET: process(ext_adc_clk, reset) is
121             begin
122                 if(reset = '1') then
123                     --state <= ADC_IDLE;
124                     adc_rst <= '0';
125                 elsif  rising_edge(ext_adc_clk) then
126                     --state <= nxt_state;
127                     adc_rst <= '1';
128                 end if;
129         end process;
130
131         P_ADC_CONV_FALLING: process(reset, ext_adc_clk) is
132             variable word_bit_cnt : integer :=0;
133             variable bit_cnt : integer := 0;
134             begin
135                 --cs <= '1';
136                 --ext_clk_ena <= '0';
137                 if(reset = '1') then
138                     cs <= '1';
139                     din <= '0';
140                     word_bit_cnt := 0;
141                     bit_cnt := 0;
142                 elsif falling_edge(ext_adc_clk) then
143                     cs  <= '1';
144                     ext_clk_ena <= '0';
145                     --din <= '0';
146                     case state is
147                         when ADC_IDLE =>
148                             --cs <= '1';
149                             --ext_clk_ena <= '0';
```

```
150                         when ADC_WRITE_CALIBRATE =>
151                             cs <= '0';
152                             ext_clk_ena <= '1';
153                             --din <= CALIBRATE_WORD(7);
154                             if(word_bit_cnt < 7) then
155                                 din <= CALIBRATE_WORD(7 - word_bit_cnt);
156                                 word_bit_cnt := word_bit_cnt + 1;
157                             elsif(word_bit_cnt = 7) then
158                                 din <= CALIBRATE_WORD(0);
159                                 word_bit_cnt := 0;
160                             end if;
161                         when ADC_WRITE_START =>
162                             cs <= '0';
163                             ext_clk_ena <= '1';
164                             --din <= CALIBRATE_WORD(7);
165                             if(word_bit_cnt < 7) then
166                                 din <= START_WORD(7 - word_bit_cnt);
167                                 word_bit_cnt := word_bit_cnt + 1;
168                             elsif(word_bit_cnt = 7) then
169                                 din <= START_WORD(0);
170                                 word_bit_cnt := 0;
171                             end if;
172                         when ADC_WAIT =>
173                             --cs <= '1';
174                             --ext_clk_ena <= '0';
175                             din <= '0';
176 
177                         when ADC_READ =>
178                             cs <= '0';
179                             ext_clk_ena <= '1';
180 
181                         when others =>
182                     end case;
183                 end if;
184             end process;
185 
186             P_ADC_CONV_FSM: process(state,nxt_state,ext_adc_clk, sstrb,reset) is
187                 variable word_bit_cnt : integer :=0;
188                 variable bit_cnt : integer := 15;
189             begin
190 
191                 if(reset = '1') then
192                     state <= ADC_IDLE;
193                     --din <= '0';
194                     --cs  <= '1';
195                     word_bit_cnt := 0;
196                     bit_cnt     := 0;
197                 elsif rising_edge(ext_adc_clk) then
198 
199                     case state is
200                         when ADC_IDLE =>
201                             if(int_enable = '1') then
```

```
202                             state <= ADC_WRITE_START;
203                             --cs <= '0';
204                             --din <= CALIBRATE_WORD(7);
205                             --int_enable <= '0';
206                         end if;
207                     when ADC_WRITE_CALIBRATE =>
208                         if(word_bit_cnt < 7) then
209                             word_bit_cnt := word_bit_cnt + 1;
210                         elsif(word_bit_cnt = 7) then
211                             word_bit_cnt := 0;
212                             state <= ADC_WAIT;
213                         end if;
214                     when ADC_WRITE_START =>
215                         if(word_bit_cnt < 7) then
216                             word_bit_cnt := word_bit_cnt + 1;
217                         elsif(word_bit_cnt = 7) then
218                             word_bit_cnt := 0;
219                             state <= ADC_WAIT;
220                         end if;
221                     when ADC_WAIT =>
222                         if(sstrb = '1') then
223                             state <= ADC_READ;
224                         end if;
225                     when ADC_READ =>
226                         if(bit_cnt = 15) then
227                             result(0) <= dout;
228                             --ext_clk_ena <= '0';
229                             --cs <= '0';
230                             bit_cnt := 0;
231                             state <= ADC_DONE;
232                         else
233                             result(15 - bit_cnt) <= dout;
234                             bit_cnt := bit_cnt + 1;
235                         end if;
236 
237                     when ADC_DONE =>
238                         --int_enable <= '0';
239                         state <= ADC_IDLE;
240 
241                     when others => state <= ADC_IDLE;
242                 end case;
243             end if;
244 
245         end process;
246 
247 
248 
249 end architecture arch;
```

A.7 int_adc_control.vhd

```
1  LIBRARY IEEE;
2  USE IEEE.std_logic_1164.ALL;
3  USE IEEE.NUMERIC_STD.ALL;
4
5  entity int_adc_control is
6      port (
7          mclk_adc    : in std_logic;
8          reset   : in std_logic;
9          enable_control  : in std_logic;
10         -- ADC select
11         adc_selch1   : out std_logic;
12         adc_ext_sel  : out std_logic;
13         scr_enbias   : out std_logic;
14         -- ADC control
15         mresb        : out std_logic;
16         adc_clk      : out std_logic;
17         adc_en       : out std_logic;
18         adc_eoc      : in  std_logic;
19         adc_comp    : in std_logic; -- Read out directly from ADC comparator output
20         adc_result  : out std_logic_vector(15 downto 0);
21         comp_delayed    : out std_logic;
22         eoc_delayed     : out std_logic;
23         tvalid          : out std_logic
24     );
25 end entity int_adc_control;
26
27 architecture arch of int_adc_control is
28
29     type adc_conv_state is(adc_conv_idle, adc_conv_read, adc_conv_stop);
30
31     CONSTANT ADC_CLK_PERIOD   : INTEGER  :=  147;-- 147 ca 20k samples/s --56 => 850kHz =>
       ↪  50ks/s. 5000;  --1M/100k = 1k
32
33     -- ADC control signals
34     signal int_adc_en   : std_logic;
35     signal nxt_adc_result : std_logic_vector(16 downto 0);
36     signal cur_adc_result : std_logic_vector(15 downto 0);
37     -- ADC clock signal s
38     signal  cur_adc_clk        : std_logic;
39     signal  nxt_adc_clk        : std_logic;
40     signal  cur_adc_clk_count  : INTEGER := 0;
41     signal  nxt_adc_clk_count  : INTEGER;
42
43     signal  cur_adc_clk_n      : std_logic;
44     signal  nxt_adc_clk_n      : std_logic;
45     signal  cur_adc_clk_n_count: INTEGER := ADC_CLK_PERIOD;
46     signal  nxt_adc_clk_n_count: INTEGER;
47
48     -- fsm
49     signal cur_adc_conv_state   : adc_conv_state;
```

```
50      signal nxt_adc_conv_state   : adc_conv_state;
51      signal cur_adc_en           : std_logic;
52      signal adc_eoc_prev         : std_logic;
53      signal adc_comp_int         : std_logic;
54
55      --type result_buffer is array (255 downto 0) of std_logic_vector(15 downto 0);
56
57      signal buf_rdy              : std_logic;
58      signal adc_result_rdy       : std_logic;
59      signal adc_comp_delayed     : std_logic;
60      signal adc_eoc_delayed      : std_logic;
61      signal adc_eoc_int          : std_logic;
62      signal tvalid_switch                : std_logic := '0';
63      signal tvalid_switch_d              : std_logic := '0';
64
65      begin
66
67          -- ADC select
68          --adc_ext_sel   <= '0';  -- External source for ADC
69          --adc_selch1    <= '1';  -- N/A when adc_ext_sel = 1
70          --scr_enbias    <= '0';  -- Screen voltage from off chip
71
72          -- ADC control
73          mresb <= not reset; -- mresb active low, asic is wierd
74          --adc_en <= int_adc_en;
75          adc_result <= cur_adc_result;
76          --adc_result(0) <= '0';
77          adc_comp_int <= adc_comp_delayed;
78          adc_eoc_int <= adc_eoc_delayed;
79
80          comp_delayed <= adc_comp_delayed;
81          eoc_delayed <= adc_eoc_delayed;
82
83          -- ADC clock generation
84          adc_clk <= cur_adc_clk;
85          nxt_adc_clk       <=  cur_adc_clk           WHEN cur_adc_clk_count <
            ↪  ADC_CLK_PERIOD ELSE (cur_adc_clk XOR '1');
86          nxt_adc_clk_count <= (cur_adc_clk_count + 1) WHEN cur_adc_clk_count <
            ↪  ADC_CLK_PERIOD ELSE 0;
87
88          nxt_adc_clk_n       <=  cur_adc_clk_n           WHEN cur_adc_clk_n_count <
            ↪  ADC_CLK_PERIOD ELSE (cur_adc_clk_n XOR '1');
89          nxt_adc_clk_n_count <= (cur_adc_clk_n_count + 1) WHEN cur_adc_clk_n_count <
            ↪  ADC_CLK_PERIOD ELSE 0;
90
91          -- signal  assignments for testing
92          int_adc_en <= enable_control; --'1';
93
94          P_COMP_DELAY: process(cur_adc_clk,reset)
95          begin
96              if(reset = '1') then
97                  adc_comp_delayed <= '0';
```

```
98              elsif falling_edge(cur_adc_clk) then
99                  adc_comp_delayed <= adc_comp;
100                 adc_eoc_delayed <= adc_eoc;
101             end if;
102         end process;
103
104         P_TVALID: process(mclk_adc,reset)
105         begin
106             if(reset = '1') then
107                 tvalid <= '0';
108             elsif rising_edge(mclk_adc) then
109                 tvalid_switch_d <= tvalid_switch;
110                 if(tvalid_switch /= tvalid_switch_d) then
111                     tvalid <= '1';
112                 else
113                     tvalid <= '0';
114                 end if;
115             end if;
116         end process;
117
118
119         P_MRESET: process(mclk_adc, reset)
120         begin
121             if(reset = '1') then
122                 cur_adc_clk         <= '0';
123                 cur_adc_clk_n       <=
                    ↪  '0';
124                 cur_adc_clk_count   <= 0;
125                 cur_adc_clk_n_count <= ADC_CLK_PERIOD;
126
127             elsif rising_edge(mclk_adc) then
128                 cur_adc_clk         <= nxt_adc_clk;
129                 cur_adc_clk_n       <= nxt_adc_clk_n;
130                 cur_adc_clk_count   <= nxt_adc_clk_count;
131                 cur_adc_clk_n_count <= nxt_adc_clk_n_count;
132             end if;
133         end process;
134
135
136         p_adc_conv_process: process(cur_adc_clk, reset) -- adc control xfab2
            ↪  ,cur_adc_conv_state, nxt_adc_conv_state, cur_adc_en
137             variable count : integer := 255;
138         begin
139             if(reset = '1') then
140                 cur_adc_conv_state <= adc_conv_idle;
141                 cur_adc_result <= (others => '0');
142                 --
143             elsif rising_edge(cur_adc_clk) then
144                 --nxt_adc_conv_state <= cur_adc_conv_state;
145                 adc_eoc_prev <= adc_eoc_int;
146                 adc_result_rdy <= '0';
147                 buf_rdy     <= '0';
```

```
148                 adc_en <= '0';
149                 --nxt_adc_en          <= cur_adc_en;
150
151                 case cur_adc_conv_state IS
152                     when adc_conv_idle =>
153                         if(int_adc_en = '1') then
154                             cur_adc_conv_state <= adc_conv_read;
155                         end if;
156
157                     when adc_conv_read =>
158                         adc_en <= '1';
159                         if(adc_eoc_int = '1') then
160                             --bit_count := 16;
161                             if(adc_eoc_prev = '0') then
162                                 adc_result_rdy <= '1';
163                                 cur_adc_result <= nxt_adc_result(15 downto 0);
164                                 tvalid_switch <= not tvalid_switch;
165                                 count := count - 1;
166                                 --result_buffer(count)(15 downto 0) <= nxt_adc_result(15
                                    ↪  downto 0);
167                                 if(count = 0) then
168                                     --result_buffer <= cur_result_buffer;
169                                     buf_rdy <= '1';
170                                 end if;
171                             end if;
172                             --if(enable_control = '1') then
173                                 cur_adc_conv_state <= adc_conv_read;
174                             --else
175                                 --cur_adc_conv_state <= adc_conv_stop;
176                             --end if;
177                         elsif(adc_eoc_int = '0') then
178                             --nxt_adc_result(0) <= '1';
179                             nxt_adc_result(0) <= adc_comp_int;
180                             nxt_adc_result(16 downto 1) <= nxt_adc_result(15 downto 0);
181                             --bit_count := bit_count - 1;
182                         end if;
183
184                     when adc_conv_stop =>
185                         nxt_adc_result <= (others => '0');
186                         if int_adc_en = '1' then
187                             --cur_adc_conv_state  <=  adc_conv_read;
188                             cur_adc_conv_state <= adc_conv_idle;
189                         else
190                             cur_adc_conv_state <= adc_conv_idle;
191                         end if;
192                     when others => cur_adc_conv_state <= adc_conv_idle;
193                 end case;
194             end if;
195
196         end process;
197
198
```

```
199         --  p_adc_pos_clk_process: process(reset, cur_adc_clk)
200         --  begin
201         --      if (reset = '1') then
202         --          cur_adc_conv_state <= adc_conv_idle;
203         --          cur_adc_en <= '0';
204         --
205         --      elsif rising_edge(cur_adc_clk) then
206         --          cur_adc_en <= '1';
207         --          cur_adc_conv_state <= nxt_adc_conv_state;
208                   --cur_adc_en <= nxt_adc_en; -- why? adc_en må synces?
209         --      end if;
210         --  end process;
211
212
213
214 end architecture;
215
216 --adc_neg_clk_process: process(reset, cur_adc_clk_n)
217 --begin
218 --    if (reset= '0') then
219         --cur_adc_val_rdy    <= '0';
220 --    ELSif (adc_clk_n'event and adc_clk_n = '1') then
221         --cur_adc_val_rdy    <= nxt_adc_val_rdy;
222 --    end if;
223 --end process;
```

A.8 sawtooth_wave.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity sawtooth_wave is
    generic(
        SAWTOOTH_MAX_VALUE : integer := 243;
        SAWTOOTH_MIN_VALUE : integer := 0
    );

  port( clk, reset: in std_logic;
        sweep_sync_out : out std_logic;
        wave_out: out std_logic_vector(7 downto 0));
end;

architecture arch of sawtooth_wave is
    signal sweep_sync : std_logic;
    signal sawtooth_wave_data : std_logic_vector(7 downto 0);

begin

    wave_out <= sawtooth_wave_data;
    sweep_sync_out <= sweep_sync;

    P_SAWTOOTH_WAVE: process(clk, reset) is --linear sweep
    variable dir : integer range -1 to 1 := 1;
    variable dac_out_int : integer range 0 to 255 := 0; -- unsigned(7 downto 0) :=
    ↪  "00000000";
    begin

        if(reset = '1') then
            dir := 1;
            dac_out_int := 0;
        elsif rising_edge(clk) then
            if(dac_out_int = SAWTOOTH_MAX_VALUE) then
                dir := -1;
                sweep_sync <= '0';
            elsif(dac_out_int = SAWTOOTH_MIN_VALUE) then
                dir := 1;
                sweep_sync <= '1';
            end if;
            dac_out_int := dac_out_int + dir;
            sawtooth_wave_data <= std_logic_vector(to_signed(dac_out_int,8));
        end if;
    end process;

end;
```

A.9 sine_wave.vhd

```
-- Synthesisable design for a sine wave generator
-- Copyright Doulos Ltd
-- SD, 07 Aug 2003

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.sine_package.all;

entity sine_wave is
  port( clk, reset, enable: in std_logic;
        wave_out: out sine_vector_type);
end;

architecture arch1 of sine_wave is
  type state_type is ( counting_up, change_down, counting_down, change_up );
  signal state, next_state: state_type;
  signal table_index: table_index_type;
  signal positive_cycle: boolean;
begin

  process( clk, reset )
  begin
    if reset = '1' then
      state <= counting_up;
    elsif rising_edge( clk ) then
      if enable = '1' then
        state <= next_state;
      end if;
    end if;
  end process;

  process( state, table_index )
  begin
    next_state <= state;
    case state is
      when counting_up =>
        if table_index = max_table_index then
          next_state <= change_down;
        end if;
      when change_down =>
        next_state <= counting_down;
      when counting_down =>
        if table_index = 0 then
          next_state <= change_up;
        end if;
      when others => -- change_up
        next_state <= counting_up;
    end case;
  end process;
```

```
51
52    process( clk, reset )
53    begin
54      if reset = '1' then
55        table_index <= 0;
56        positive_cycle <= true;
57      elsif rising_edge( clk ) then
58        if enable = '1' then
59          case next_state is
60            when counting_up =>
61              table_index <= table_index + 1;
62            when counting_down =>
63              table_index <= table_index - 1;
64            when change_up =>
65              positive_cycle <= not positive_cycle;
66            when others =>
67              -- nothing to do
68          end case;
69        end if;
70      end if;
71    end process;
72
73    process( table_index, positive_cycle )
74      variable table_value: table_value_type;
75    begin
76      table_value := get_table_value( table_index );
77      if positive_cycle then
78        wave_out <= std_logic_vector(to_signed(table_value,sine_vector_type'length));
79      else
80        wave_out <= std_logic_vector(to_signed(-table_value,sine_vector_type'length));
81      end if;
82    end process;
83
84  end;
```

A.10 sine_package.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  package sine_package is
5
6    constant max_table_value: integer := 127;
7    subtype table_value_type is integer range 0 to max_table_value;
8
9    constant max_table_index: integer := 127;
10   subtype table_index_type is integer range 0 to max_table_index;
11
12   subtype sine_vector_type is std_logic_vector( 7 downto 0 );
13
14   function get_table_value (table_index: table_index_type) return table_value_type;
15
16 end;
17
18 package body sine_package is
19
20   function get_table_value (table_index: table_index_type) return table_value_type is
21     variable table_value: table_value_type;
22   begin
23     case table_index is
24       when 0 =>
25         table_value := 1;
26       when 1 =>
27         table_value := 2;
28       when 2 =>
29         table_value := 4;
30       when 3 =>
31         table_value := 5;
32       when 4 =>
33         table_value := 7;
34       when 5 =>
35         table_value := 9;
36       when 6 =>
37         table_value := 10;
38       when 7 =>
39         table_value := 12;
40       when 8 =>
41         table_value := 13;
42       when 9 =>
43         table_value := 15;
44       when 10 =>
45         table_value := 16;
46       when 11 =>
47         table_value := 18;
48       when 12 =>
49         table_value := 19;
50       when 13 =>
```

```
51            table_value := 21;
52          when 14 =>
53            table_value := 22;
54          when 15 =>
55            table_value := 24;
56          when 16 =>
57            table_value := 26;
58          when 17 =>
59            table_value := 27;
60          when 18 =>
61            table_value := 29;
62          when 19 =>
63            table_value := 30;
64          when 20 =>
65            table_value := 32;
66          when 21 =>
67            table_value := 33;
68          when 22 =>
69            table_value := 35;
70          when 23 =>
71            table_value := 36;
72          when 24 =>
73            table_value := 38;
74          when 25 =>
75            table_value := 39;
76          when 26 =>
77            table_value := 41;
78          when 27 =>
79            table_value := 42;
80          when 28 =>
81            table_value := 44;
82          when 29 =>
83            table_value := 45;
84          when 30 =>
85            table_value := 46;
86          when 31 =>
87            table_value := 48;
88          when 32 =>
89            table_value := 49;
90          when 33 =>
91            table_value := 51;
92          when 34 =>
93            table_value := 52;
94          when 35 =>
95            table_value := 54;
96          when 36 =>
97            table_value := 55;
98          when 37 =>
99            table_value := 56;
100         when 38 =>
101           table_value := 58;
102         when 39 =>
```

```
103           table_value := 59;
104         when 40 =>
105           table_value := 61;
106         when 41 =>
107           table_value := 62;
108         when 42 =>
109           table_value := 63;
110         when 43 =>
111           table_value := 65;
112         when 44 =>
113           table_value := 66;
114         when 45 =>
115           table_value := 67;
116         when 46 =>
117           table_value := 69;
118         when 47 =>
119           table_value := 70;
120         when 48 =>
121           table_value := 71;
122         when 49 =>
123           table_value := 72;
124         when 50 =>
125           table_value := 74;
126         when 51 =>
127           table_value := 75;
128         when 52 =>
129           table_value := 76;
130         when 53 =>
131           table_value := 78;
132         when 54 =>
133           table_value := 79;
134         when 55 =>
135           table_value := 80;
136         when 56 =>
137           table_value := 81;
138         when 57 =>
139           table_value := 82;
140         when 58 =>
141           table_value := 84;
142         when 59 =>
143           table_value := 85;
144         when 60 =>
145           table_value := 86;
146         when 61 =>
147           table_value := 87;
148         when 62 =>
149           table_value := 88;
150         when 63 =>
151           table_value := 89;
152         when 64 =>
153           table_value := 90;
154         when 65 =>
```

```
155           table_value := 91;
156         when 66 =>
157           table_value := 93;
158         when 67 =>
159           table_value := 94;
160         when 68 =>
161           table_value := 95;
162         when 69 =>
163           table_value := 96;
164         when 70 =>
165           table_value := 97;
166         when 71 =>
167           table_value := 98;
168         when 72 =>
169           table_value := 99;
170         when 73 =>
171           table_value := 100;
172         when 74 =>
173           table_value := 101;
174         when 75 =>
175           table_value := 102;
176         when 76 =>
177           table_value := 102;
178         when 77 =>
179           table_value := 103;
180         when 78 =>
181           table_value := 104;
182         when 79 =>
183           table_value := 105;
184         when 80 =>
185           table_value := 106;
186         when 81 =>
187           table_value := 107;
188         when 82 =>
189           table_value := 108;
190         when 83 =>
191           table_value := 109;
192         when 84 =>
193           table_value := 109;
194         when 85 =>
195           table_value := 110;
196         when 86 =>
197           table_value := 111;
198         when 87 =>
199           table_value := 112;
200         when 88 =>
201           table_value := 112;
202         when 89 =>
203           table_value := 113;
204         when 90 =>
205           table_value := 114;
206         when 91 =>
```

```
207           table_value := 114;
208         when 92 =>
209           table_value := 115;
210         when 93 =>
211           table_value := 116;
212         when 94 =>
213           table_value := 116;
214         when 95 =>
215           table_value := 117;
216         when 96 =>
217           table_value := 118;
218         when 97 =>
219           table_value := 118;
220         when 98 =>
221           table_value := 119;
222         when 99 =>
223           table_value := 119;
224         when 100 =>
225           table_value := 120;
226         when 101 =>
227           table_value := 120;
228         when 102 =>
229           table_value := 121;
230         when 103 =>
231           table_value := 121;
232         when 104 =>
233           table_value := 122;
234         when 105 =>
235           table_value := 122;
236         when 106 =>
237           table_value := 123;
238         when 107 =>
239           table_value := 123;
240         when 108 =>
241           table_value := 123;
242         when 109 =>
243           table_value := 124;
244         when 110 =>
245           table_value := 124;
246         when 111 =>
247           table_value := 124;
248         when 112 =>
249           table_value := 125;
250         when 113 =>
251           table_value := 125;
252         when 114 =>
253           table_value := 125;
254         when 115 =>
255           table_value := 126;
256         when 116 =>
257           table_value := 126;
258         when 117 =>
```

```
259           table_value := 126;
260         when 118 =>
261           table_value := 126;
262         when 119 =>
263           table_value := 126;
264         when 120 =>
265           table_value := 126;
266         when 121 =>
267           table_value := 127;
268         when 122 =>
269           table_value := 127;
270         when 123 =>
271           table_value := 127;
272         when 124 =>
273           table_value := 127;
274         when 125 =>
275           table_value := 127;
276         when 126 =>
277           table_value := 127;
278         when 127 =>
279           table_value := 127;
280       end case;
281       return table_value;
282     end;
283
284 end;
```

A.11 debounce.vhd

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  entity debounce is
5      GENERIC(
6          clk_freq    : INTEGER := 100_000_000;  --system clock frequency in Hz
7          stable_time : INTEGER := 10);         --time button must remain stable in ms
8      port(
9          mclk    : in std_logic;
10         mrst    : in std_logic;
11         button_inp  : in std_logic;
12         button_stable   : out std_logic
13     );
14 end entity;
15
16 architecture arch of debounce is
17     constant WAIT_TIME : integer := clk_freq*stable_time/1000;
18
19     signal flipflops   : STD_LOGIC_VECTOR(1 DOWNTO 0); --input flip flops
20     signal counter_set : STD_LOGIC;                    --sync reset to zero
21   begin
22
23     counter_set <= flipflops(0) xor flipflops(1);  --determine when to start/reset counter
24
25     process(mclk, mrst)
26         variable counter :  integer := 0;  --counter for timing
27         begin
28             if(mrst = '1') then                        --reset
29                 flipflops(1 downto 0) <= "00";                 --clear input flipflops
30                 button_stable <= '0';                                --clear result
                    ↪ register
31             elsif rising_edge(mclk) then           --rising clock edge
32                 flipflops(0) <= button_inp;                        --store button value in
                    ↪ 1st flipflop
33                 flipflops(1) <= flipflops(0);                  --store 1st flipflop value
                    ↪ in 2nd flipflop
34                 if(counter_set = '1') then                     --reset counter because
                    ↪ input is changing
35                     counter := 0;                                      --clear the counter
36                 elsif(counter < WAIT_TIME) then  --stable input time is not yet met
37                     counter := counter + 1;                             --increment counter
38                 else                                           --stable input time is met
39                     button_stable <= flipflops(1);                          --output the
                        ↪ stable value
40                 end if;
41             end if;
42     END PROCESS;
43
44
45 end arch;
```

Appendix B

State machine Diagrams

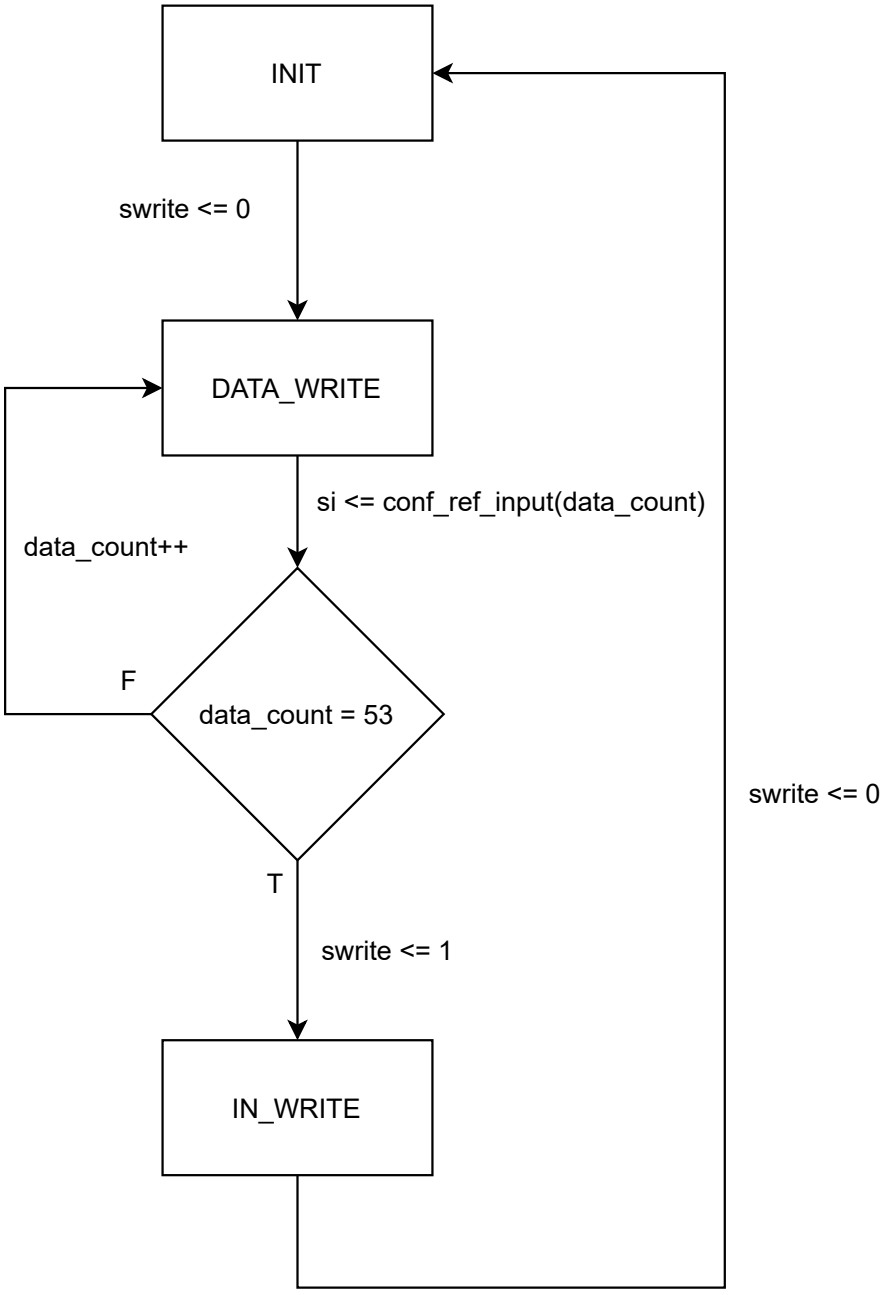


Figure B.1: State machine controlling the m-NIC2 serial register, writing

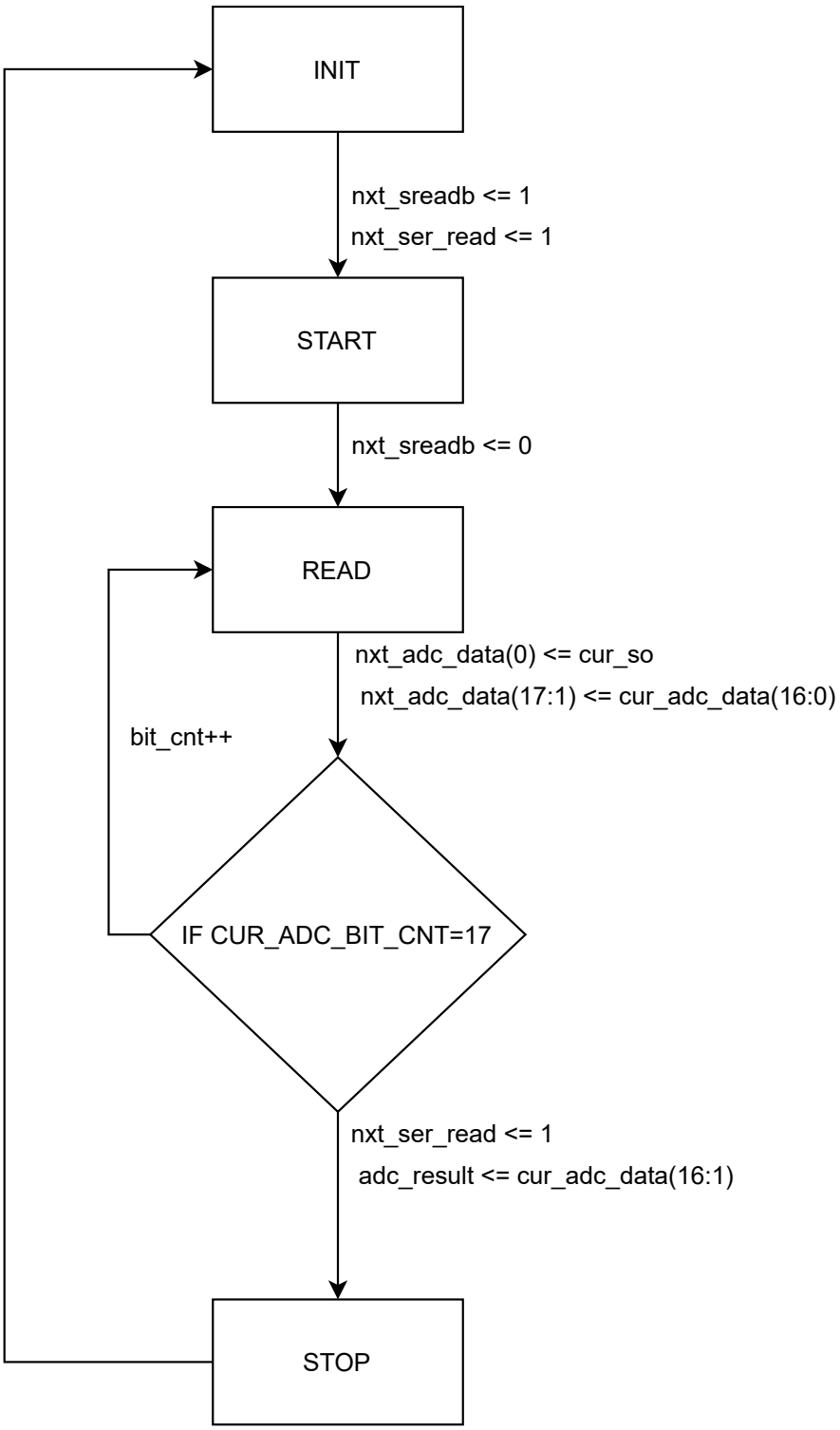


Figure B.2: State machine controlling the m-NIC2 serial register, reading

Appendix C

Pin assignment

```
1
2  set_property -dict {PACKAGE_PIN D19 IOSTANDARD LVCMOS33} [get_ports mrst_0]
               #BTN0
3  set_property -dict {PACKAGE_PIN M20 IOSTANDARD LVCMOS33} [get_ports {mode_0[0]}]
               #SW0
4  set_property -dict {PACKAGE_PIN M19 IOSTANDARD LVCMOS33} [get_ports {mode_0[1]}]
               #SW1
5  set_property -dict {PACKAGE_PIN R14 IOSTANDARD LVCMOS33} [get_ports {led_0[0]}]
               #LED0
6  set_property -dict {PACKAGE_PIN P14 IOSTANDARD LVCMOS33} [get_ports {led_0[1]}]
               #LED1
7  set_property -dict {PACKAGE_PIN M14 IOSTANDARD LVCMOS33} [get_ports dbg_led_0]
               #LED3
8  set_property -dict {PACKAGE_PIN L20 IOSTANDARD LVCMOS33} [get_ports adc_switch_0
       ]       #BTN2
9  set_property -dict {PACKAGE_PIN N16 IOSTANDARD LVCMOS33} [get_ports
       adc_selected_0]     #LED2
10 set_property -dict {PACKAGE_PIN N17 IOSTANDARD LVCMOS33} [get_ports trigger_1]
               #AR13
11 #set_property -dict {PACKAGE_PIN P18 IOSTANDARD LVCMOS33} [get_ports TEST_POINT]
               #AR12
12
13
14
15
16 set_property -dict {PACKAGE_PIN Y8 IOSTANDARD LVCMOS33} [get_ports adc_en_0]
               #RP35
17 set_property -dict {PACKAGE_PIN W19 IOSTANDARD LVCMOS33} [get_ports adc_eoc_0]
               #RP5
18 set_property -dict {PACKAGE_PIN A20 IOSTANDARD LVCMOS33} [get_ports
       int_adc_clk_0]      #RP38
19 set_property -dict {PACKAGE_PIN Y16 IOSTANDARD LVCMOS33} [get_ports adc_comp_0]
               #RP27
20
```

```
21 set_property -dict {PACKAGE_PIN Y18 IOSTANDARD LVCMOS33} [get_ports {mresb_1
       [0]}]        #RP8
22
23 set_property -dict {PACKAGE_PIN B19 IOSTANDARD LVCMOS33} [get_ports sclk_0]
                      #RP36
24 set_property -dict {PACKAGE_PIN W18 IOSTANDARD LVCMOS33} [get_ports adc_selch1_0
       ]        #RP3
25 set_property -dict {PACKAGE_PIN C20 IOSTANDARD LVCMOS33} [get_ports
       adc_ext_sel_0]       #RP12
26 set_property -dict {PACKAGE_PIN W6 IOSTANDARD LVCMOS33} [get_ports scr_enbias_0]
              #RP16
27 set_property -dict {PACKAGE_PIN V7 IOSTANDARD LVCMOS33} [get_ports {mresb_1[1]}]
              #RP13
28 set_property -dict {PACKAGE_PIN F19 IOSTANDARD LVCMOS33} [get_ports sreadb_0]
              #RP24
29 set_property -dict {PACKAGE_PIN U19 IOSTANDARD LVCMOS33} [get_ports swrite_0]
              #RP26
30 set_property -dict {PACKAGE_PIN Y9 IOSTANDARD LVCMOS33} [get_ports so_0]
                      #RP40
31 set_property -dict {PACKAGE_PIN B20 IOSTANDARD LVCMOS33} [get_ports si_0]
                      #RP32
32
33
34 set_property -dict {PACKAGE_PIN U18 IOSTANDARD LVCMOS33} [get_ports
       max1132_clk_0]       #RP18
35 set_property -dict {PACKAGE_PIN V10 IOSTANDARD LVCMOS33} [get_ports
       max1132_dout_0]      #RP21
36 set_property -dict {PACKAGE_PIN V8 IOSTANDARD LVCMOS33} [get_ports
       max1132_sstrb_0]      #RP19
37 set_property -dict {PACKAGE_PIN U7 IOSTANDARD LVCMOS33} [get_ports max1132_din_0
       ]        #RP11
38 set_property -dict {PACKAGE_PIN U8 IOSTANDARD LVCMOS33} [get_ports max1132_rst_0
       ]        #RP15
39 set_property -dict {PACKAGE_PIN V6 IOSTANDARD LVCMOS33} [get_ports
       max1132_shdn_0]       #RP7
40 set_property -dict {PACKAGE_PIN W10 IOSTANDARD LVCMOS33} [get_ports max1132_cs_0
       ]        #RP23
41
42 set_property -dict {PACKAGE_PIN V18 IOSTANDARD LVCMOS33} [get_ports {A_0[0]}]
              #AR9
43 set_property -dict {PACKAGE_PIN V17 IOSTANDARD LVCMOS33} [get_ports {A_0[1]}]
              #AR8
44
45 set_property -dict {PACKAGE_PIN T16 IOSTANDARD LVCMOS33} [get_ports {wr_0}]
                      #AR10
46 set_property -dict {PACKAGE_PIN T14 IOSTANDARD LVCMOS33} [get_ports {dac_out_0
       [0]}] #AR0
47 set_property -dict {PACKAGE_PIN U12 IOSTANDARD LVCMOS33} [get_ports {dac_out_0
       [1]}] #AR1
48 set_property -dict {PACKAGE_PIN U13 IOSTANDARD LVCMOS33} [get_ports {dac_out_0
       [2]}] #AR2
```

```
49 set_property -dict {PACKAGE_PIN V13 IOSTANDARD LVCMOS33} [get_ports {dac_out_0
       [3]}] #AR3
50 set_property -dict {PACKAGE_PIN V15 IOSTANDARD LVCMOS33} [get_ports {dac_out_0
       [4]}]     #AR4
51 set_property -dict {PACKAGE_PIN T15 IOSTANDARD LVCMOS33} [get_ports {dac_out_0
       [5]}]     #AR5
52 set_property -dict {PACKAGE_PIN R16 IOSTANDARD LVCMOS33} [get_ports {dac_out_0
       [6]}]     #AR6
53 set_property -dict {PACKAGE_PIN U17 IOSTANDARD LVCMOS33} [get_ports {dac_out_0
       [7]}]     #AR7
54 set_property -dict {PACKAGE_PIN R17 IOSTANDARD LVCMOS33} [get_ports sweep_sync_0
       ]       #AR11
55 set_property -dict {PACKAGE_PIN W8 IOSTANDARD LVCMOS33} [get_ports input_sync_0]
             #RP33
56
57
58 set_property DRIVE 12 [get_ports {A_0[1]}]
59 set_property DRIVE 12 [get_ports {A_0[0]}]
60
61 set_operating_conditions -process maximum
62 set_operating_conditions -heatsink low
63 set_property C_CLK_INPUT_FREQ_HZ 300000000 [get_debug_cores dbg_hub]
64 set_property C_ENABLE_CLK_DIVIDER false [get_debug_cores dbg_hub]
65 set_property C_USER_SCAN_CHAIN 1 [get_debug_cores dbg_hub]
66 connect_debug_port dbg_hub/clk [get_nets clk]
```

Appendix D

Python code

D.1 Readout code

```
# Imports
from pynq import Overlay
from pynq import Clocks
from pynq import GPIO
from pynq.lib import AxiGPIO
import pynq.lib.dma
from pynq import Xlnk
import numpy as np
import time
import matplotlib.pyplot as plt
import csv
from datetime import datetime
import os

# Programs the FPGA
# This loads the .bit (bitstream), as well as a .tcl and .hwh file. These files must have the same name at the .bit file
overlay = Overlay('/home/xilinx/pynq/overlays/pcb_control/plasma_chamber/final_x2/x2_plasma.bit')
dma = overlay.axi_dma_adc
xlnk = Xlnk()

def save(path, filename):
    with open(path + filename, mode='w') as file:
        writer = csv.writer(file)
        row = ["VH=" + str(VH),"VM=1.6498"]
        writer.writerow("Current=1500")
        #writer.writerow(row)
        row = ["time [s]", "adc_data [16-bit]"]
        writer.writerow(row)
        index = 0
        for n in range(N):
            for i in range(data_size):
```

```
                if(index == 0):
                    counttime[index] = 0
                else:
                    counttime[index] = counttime[index-1] + period
                try:
                    adc[index] = int(bin(int(data[n,i]))[18:34],2)*to_volts
                    #dac[index] = int(bin(int(data[n,i]))[3:12],2)*to_volts_dac
                    #adc_sync[index] = int(bin(int(data[n,i]))[17:18],2)
                    #sweep_sync[index] = int(bin(int(data[n,i]))[16:17],2)
                except Exception as e:
                    adc[index] = adc[index-1]
                    error_count += 1


                row = [counttime[index], adc[index]]
                #row = [counttime[i],adc[i],adc_sync[i]]
                #row = [counttime[i], adc[i]]
                writer.writerow(row)
                index += 1
            #print(n)

def capture(N):
    for i in range(N):
        dma.recvchannel.transfer(buffer) # Gets 1 buffer from the DMA, 1 buffer = 1.5s ish
        dma.recvchannel.wait()
        data[i] = buffer

# Path and filename, make sure path folder exists
path = '/home/xilinx/pynq/data/'
filename = 'test.csv'

# Captures data
frequency = 20.008 * 10**3 # To create time axis
period = 1/frequency
data_size =  32768
N = 2 # Amount of buffers, 130 = 3.5 min capture

print("Capture time: ", N*32768/frequency, "s")

data = np.ndarray(shape=(N,data_size)) # (N x data_size) array
buffer = xlnk.cma_array(shape=(data_size,), dtype=np.uint32)
np_buffer = np.zeros(data_size)
# Clearing last buffer
dma.recvchannel.transfer(buffer)
dma.recvchannel.wait()
start_time = time.time()
print("Starting with" , filename)

capture(N) # Receives N buffers from the DMA

print("Done")
stop_time = time.time()
```

```
84  exec_time = stop_time-start_time
85  print('Execution time: ',exec_time)
86
87  # Data capture is finished, moves to saving data
88
89  index = 0
90  error_count = 0
91  adc = np.zeros(data_size*N)
92  dac = np.zeros(data_size*N)
93  counttime = np.zeros(data_size*N)
94  adc_sync = np.zeros(data_size*N)
95  osc_trigger = np.zeros(data_size*N)
96  sweep_sync = np.zeros(data_size*N)
97  adc_trigger = np.zeros(data_size*N)
98  VH = 3.2989 # Measured VH referance
99  to_volts = VH/65536.0
100 to_volts_dac = 5/256.0
101
102 save(path, filename) # Saves data to path/filename.csv
103
104 print("Error count: " , error_count)
105
106 stop_time = time.time()
107 exec_time = stop_time-start_time
108 print("Finished with" , filename, "Total time: ", exec_time)
```

D.2 Deviation calculation code

```
import numpy as np
import matplotlib.pyplot as plt
import csv
import time

time = []
adc_data = []
start = 5500
stop = int(2.4*10**6) # To create a regress line from 50 mV to 2.7 V

with open('4mhz_ver3.csv', 'r') as data:
    csv_reader = csv.reader(data)
    line_count = 0
    for row in csv_reader:
        if(line_count < 2):
            line_count += 1
        else:
            time.append(float(row[0]))
            adc_data.append(float(row[1]))
            line_count += 1
            #plt.plot(time,adc_data)
            #plt.show()

time_lin = np.array(time[start:stop])
plt.plot(adc_data[start:stop])

lsb = 3.3/2**(16)
adc_data_lin = adc_data[start:stop]
reg = np.polyfit(time_lin, adc_data_lin,1)
reg_eq = reg[0]*time_lin + reg[1]
error = adc_data[start:stop] - reg_eq

#plt.plot(reg_eq,error/lsb, label='Deviation from straight line')
plt.xlabel("Input voltage [V]")
plt.ylabel("Error [mV]")
plt.legend()
#plt.show()

n = 64
label = str(n) + ' length moving average'
# Moving average calculation
ret = np.cumsum(error, dtype=float)
ret[n:] = ret[n:] - ret[:-n]
averaged_error = ret[n - 1:]/n

averaged_error_bits = averaged_error/lsb
std = np.std(error/lsb)
print("Average error (LSB) = " , np.average(np.abs(error)/lsb))
print("Standard deviation (LSB) = " , np.std(error/lsb))
print("Max deviation:" , np.max(np.abs(error/lsb)))
```

```
51
52  plt.plot(reg_eq[:len(averaged_error_bits)], averaged_error_bits, label=label)
53  plt.xlabel("Regress line voltage [V]")
54  plt.ylabel("Deviation from regress line [LSB]")
55  plt.grid('on')
56  plt.legend()
57  plt.show()
58
```

Appendix E

Vivado block diagram

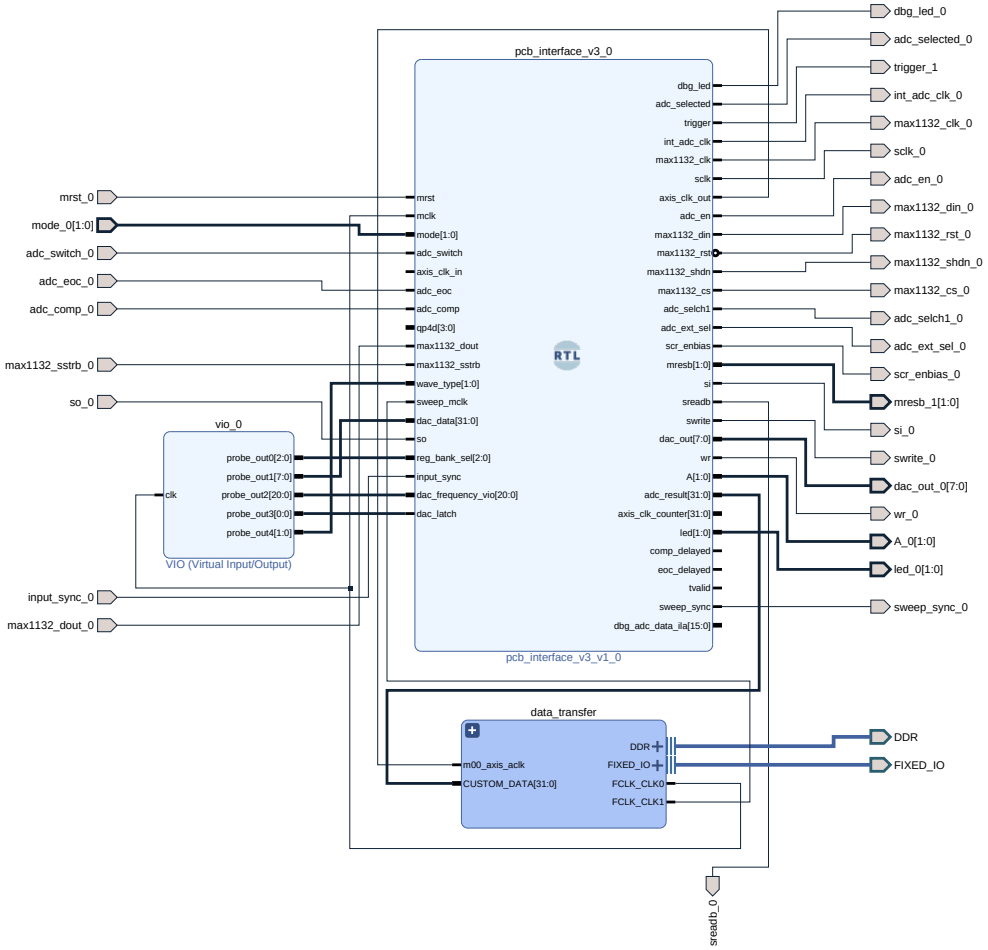


Figure E.1: Block diagram of the full system being run on the PYNQ board

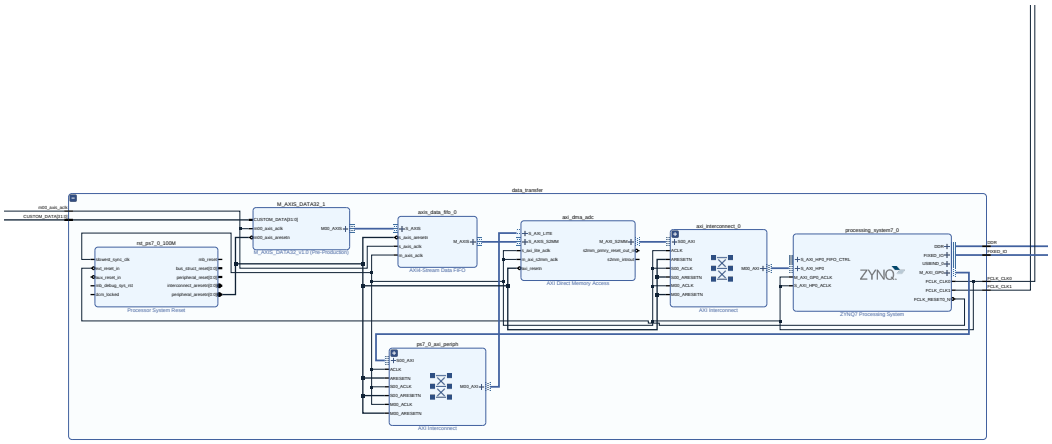
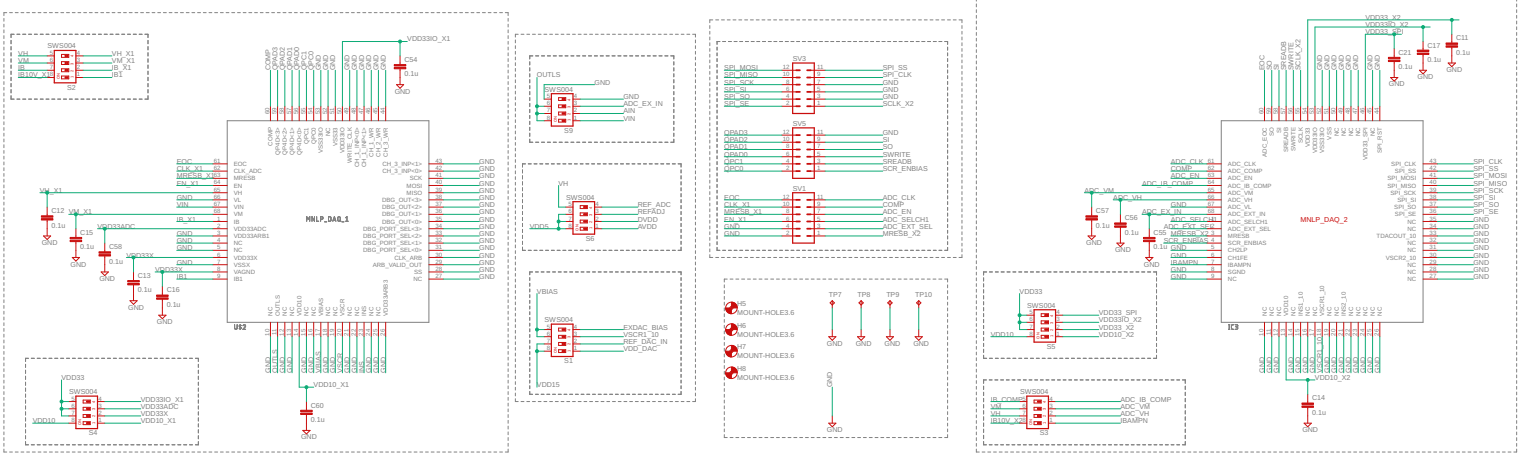
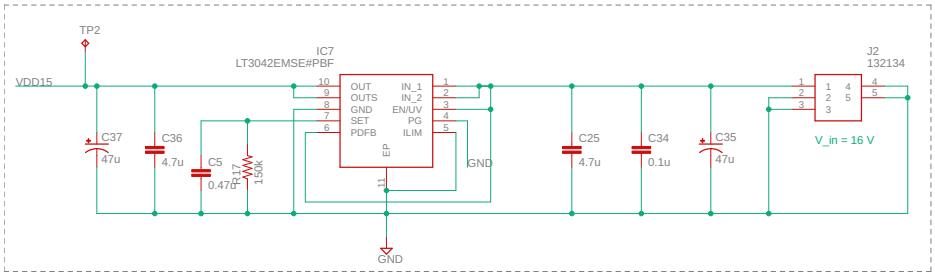


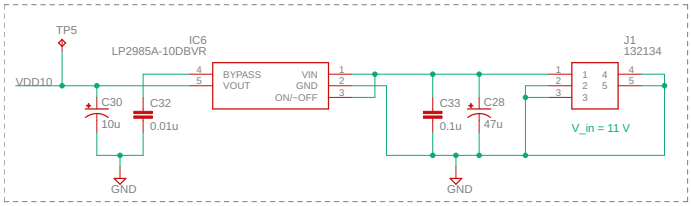
Figure E.2: Block diagram of the data transfer module

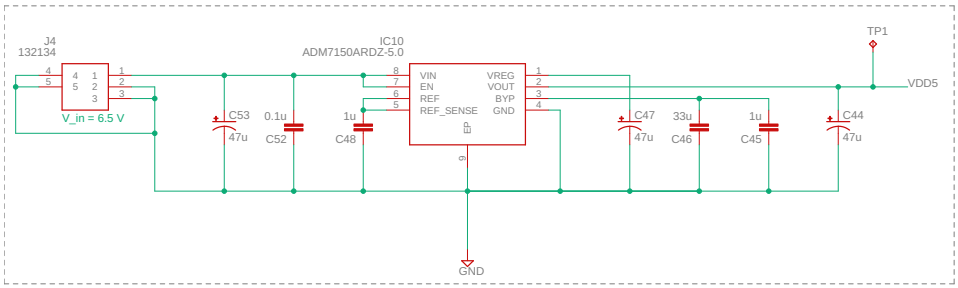
Appendix F

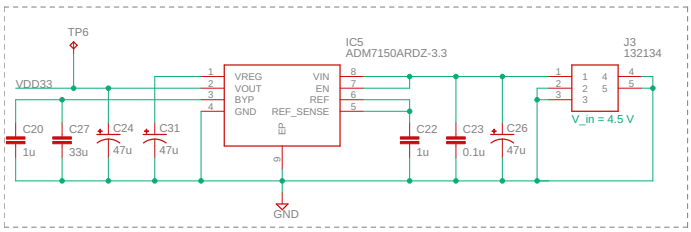
m-NIC PCB Schematics

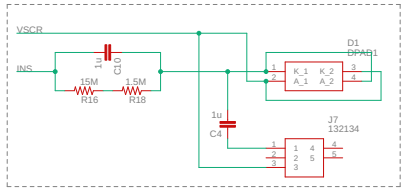

MNLP_DAQ_1
U62
MNLP_DAQ_2
IC9
SW5004
SV3
SV5
SV1
VDD33IO_X1
VDD10_X1
VDD33_X2
VDD33_SPI
VH_X1
VM_X1
IB_X1
IB1
IB10V_X1
ADC_EX_IN
AIN
VIN
REF_ADC
REFADJ
AVDD
FXDAC_BIAS
VSCR1_10
REF_DAC_IN
VDD_DAC
ADC_IB_COMP
ADC_VM
ADC_VH
IBAMPN
SPI_SS
SPI_MOSI
SPI_MISO
SPI_SCK
SPI_SI
SPI_SO
SPI_SE
SCLK_X2
ADC_CLK
ADC_EN
ADC_SELCH1
ADC_EXT_SEL
MRESB_X2
SWRITE
SREADB
SCR_ENBIAS
MOUNT-HOLE3.6
TP7
TP8
TP9
TP10
GND

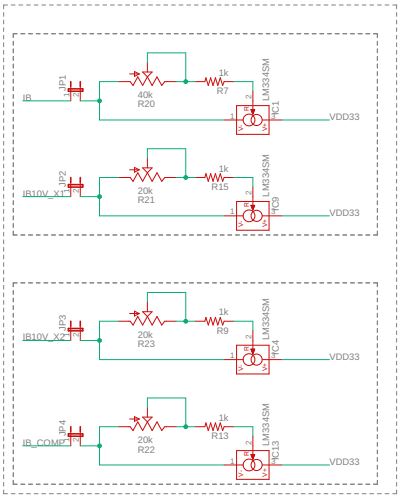

TP2
VDD15
IC7
LT3042EMSE#PBF
OUT
OUTS
GND
SET
PDFB
IN_1
IN_2
EN/UV
PG
ILIM
EP
C37
47u
C36
4.7u
C5
0.47u
R17
150k
GND
C25
4.7u
C34
0.1u
C35
47u
J2
132134
V_in = 16 V
GND

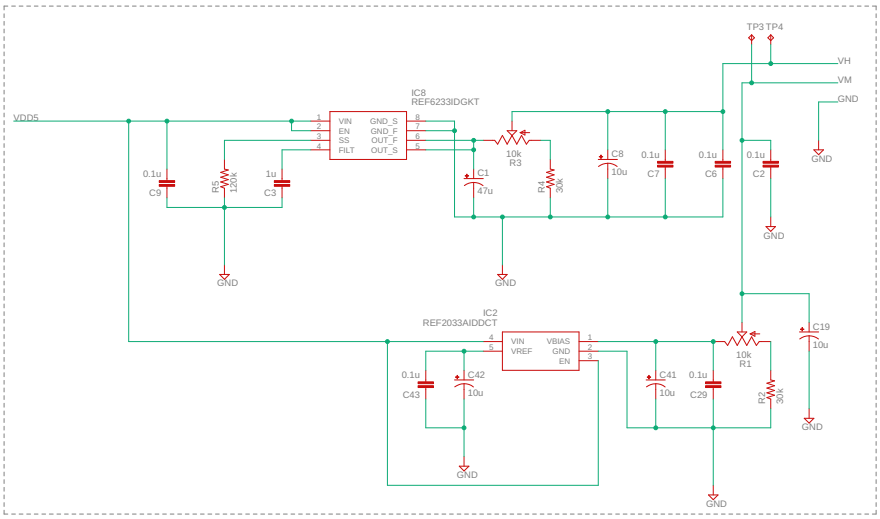

TP5
VDD10
IC6
LP2985A-10DBVR
BYPASS
VOUT
VIN
GND
ON/~OFF
C30
10u
C32
0.01u
GND
C33
0.1u
C28
47u
GND
J1
132134
V_in = 11 V

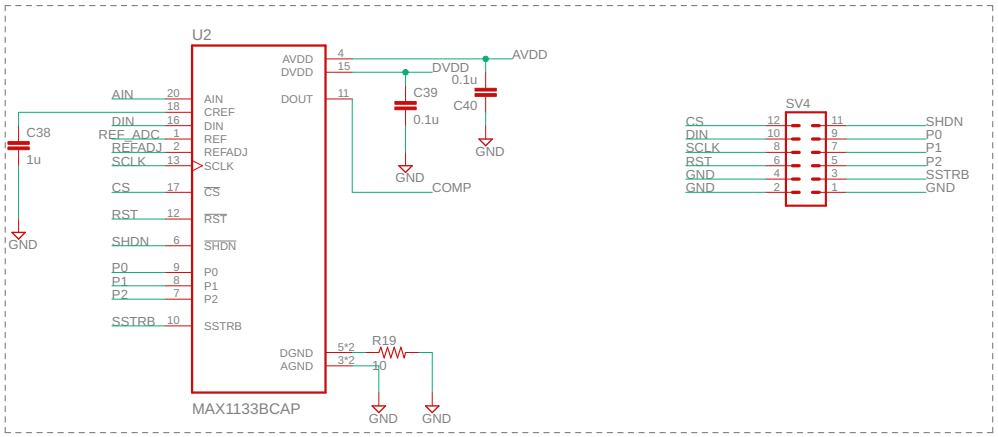

J4
132134
V_in = 6.5 V
IC10
ADM7150ARDZ-5.0
VIN
EN
REF
REF_SENSE
VREG
VOUT
BYP
GND
EP
C53
47u
C52
0.1u
C48
1u
C47
47u
C46
33u
C45
1u
C44
47u
TP1
VDD5
GND

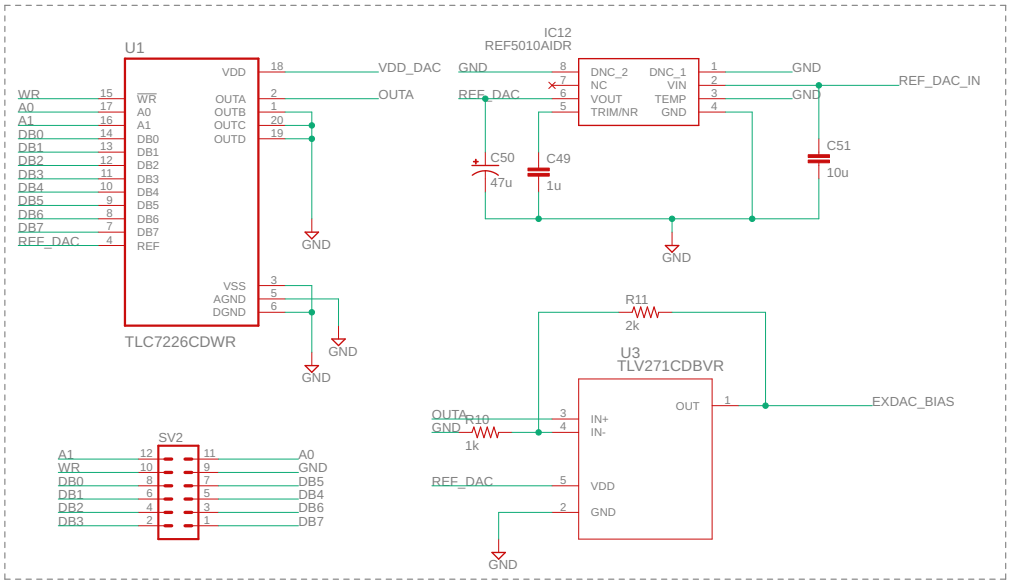

TP6
VDD33
IC5
ADM7150ARDZ-3.3
VREG
VOUT
BYP
GND
VIN
EN
REF
REF_SENSE
EP
C20
1u
C27
33u
C24
47u
C31
47u
C22
1u
C23
0.1u
C26
47u
J3
132134
V_in = 4.5 V
GND


VSCR
INS
C10
15M
R16
1.5M
R18
1u
C4
D1
DPAB1
K_1
A_1
K_2
A_2
J7
132134


IB
JP1
40k
R20
1k
R7
LM334SM
HC1
VDD33
IB10V_X1
JP2
20k
R21
1k
R15
LM334SM
HC9
VDD33
IB10V_X2
JP3
20k
R23
1k
R9
LM334SM
HC4
VDD33
IB_COMP
JP4
20k
R22
1k
R13
LM334SM
HC13
VDD33


TP3 TP4
VH
VM
GND
VDD5
IC8
REF6233IDGKT
VIN
EN
SS
FILT
GND_S
GND_F
OUT_F
OUT_S
0.1u
C9
R5
120k
1u
C3
C1
47u
10k
R3
R4
38k
C8
10u
0.1u
C7
0.1u
C6
0.1u
C2
GND
IC2
REF2033AIDDCT
VIN
VREF
VBIAS
GND
EN
C19
10u
10k
R1
0.1u
C43
C42
10u
C41
10u
0.1u
C29
R2
39k


U2
AVDD
DVDD
AIN
CREF
DIN
REF
REFADJ
SCLK
CS
RST
SHDN
P0
P1
P2
SSTRB
DOUT
DGND
AGND
MAX1133BCAP
AVDD
DVDD
0.1u
C39
0.1u
C40
GND
COMP
C38
1u
GND
AIN
DIN
REF_ADC
REFADJ
SCLK
CS
RST
SHDN
P0
P1
P2
SSTRB
R19
5*2
3*2
10
GND
GND
SV4
CS
DIN
SCLK
RST
GND
GND
SHDN
P0
P1
P2
SSTRB
GND


U1
VDD
WR
A0
A1
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
REF
OUTA
OUTB
OUTC
OUTD
VSS
AGND
DGND
TLC7226CDWR
VDD_DAC
OUTA
GND
GND
GND
IC12
REF5010AIDR
DNC_2
NC
VOUT
TRIM/NR
DNC_1
VIN
TEMP
GND
REF_DAC_IN
C50
47u
C49
1u
C51
10u
GND
R11
2k
U3
TLV271CDBVR
IN+
IN-
OUT
VDD
GND
OUTA
R10
1k
REF_DAC
EXDAC_BIAS
GND
SV2
A1
WR
DB0
DB1
DB2
DB3
A0
GND
DB5
DB4
DB6
DB7

Bibliography

[1] T. A. Bekkeng, "Prototype development of a multi-needle langmuir probe system," Master's thesis, University of Oslo, 2009.

[2] R. J. S. C. T. Russell, J. G. Luhmann, *Space Physics: An Introduction*. Cambridge University Press, 2016.

[3] I. R. Ionosphere, "International reference ionosphere." `http://irimodel.org/`. Accessed: 25-05-2021.

[4] U. center for Science Education, "The ionosphere, may. 22, 2021." `https://scied.ucar.edu/learning-zone/atmosphere/ionosphere`.

[5] NASA, "The earth's plasmasphere." `https://plasmasphere.nasa.gov/`. 22-05-2021.

[6] A. Berthelier, J. Cerisier, D. Lagoutte, and C. Béghin, "The small-scale turbulent structure of the high latitude ionosphere: Arcad-aureol-3 observations," *Annales Geophysicae*, vol. 9, pp. 725–737, 01 1991.

[7] N. Linty, A. Farasin, A. Favenza, and F. Dovis, "Detection of gnss ionospheric scintillations based on machine learning decision tree," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 55, pp. 303–317, 2019.

[8] S. W. P. Center, "Total electron content." `https://www.swpc.noaa.gov/phenomena/total-electron-content`. Accessed: 2021-05-26.

[9] L. H. Brace, R. F. Theis, and A. Dalgarno, "The cylindrical electrostatic probes for atmosphere explorer -c,-d, and -e," *Radio Science*, vol. 8, no. 4, pp. 341–348, 1973.

[10] K. S. Jacobsen, A. Pedersen, J. I. Moen, and T. A. Bekkeng, "A new langmuir probe concept for rapid sampling of space plasma electron density," *Measurement Science and Technology*, vol. 21, p. 085902, jul 2010.

[11] E. Electronics, "multi-needle langmuir probe." `https://eidel.no/product/multi-needle-langmuir-probe/`. Accessed: 05-05-2021.

[12] QB50, "Qb50." `https://www.qb50.eu/index.php/project-description-obj/mission-objectives.html`. 23-05-2021.

[13] Airbus, "Esa and airbus sign contract for bartolomeo platform on the international space station." https://www.airbus.com/newsroom/press-releases/en/2020/01/esa-and-airbus-sign-contract-for-bartolomeo-platform-on-the-international-space-stati html. 22-05-2021.

[14] H. M. Hoang, *High-Spatial-resolution electron density measurements using the needle Langmuir system on different space platforms*. PhD thesis, 2019.

[15] J. M. Østby, "Front-end and adc asic for langmuir probes." 2017.

[16] J. M. Østby, "4dspace xfab mpw1 measurements." 2018.

[17] Zen-In. https://commons.wikimedia.org/wiki/File:TIA_simple.svg. Accessed: 21-05-2021.

[18] M. Khaled, *Enhancing the Performance of Digital Controllers using Distributed Multicore/Heterogeneous Embedded Systems*. PhD thesis, 01 2014.

[19] Maxim Integrated, *16-Bit ADC, 200ksps, 5V Single-Supply with Reference*. 19-2083; Rev 0; 8/01.

[20] Texas Instruments, *8-Bit, 5 us Quad DAC, Parallel Input, Single / Dual Supply*. Rev. F.

[21] Texas Instrument, *Low-Noise,Very Low Drift, Precision Voltage Reference*. Rev 02-2020.

[22] "Ieee standard for terminology and test methods of digital-to-analog converter devices," *IEEE Std 1658-2011*, pp. 1–126, 2012.

[23] "Ieee standard for terminology and test methods for analog-to-digital converters," *IEEE Std 1241-2010 (Revision of IEEE Std 1241-2000)*, pp. 1–139, 2011.

[24] R. A. T. Pedersen, "Risc-v processor core customization and verification for m-nlp applications," Master's thesis, University of Oslo, 2020.