Multi-channel Electrical Impedance Tomography with Field Programmable Gate Array

Kjetil Vermundsen Madsen

Thesis submitted for the degree of
Master in Electrical Engineering, Informatics and Technology
(Microelectronics and Sensor technology)
60 credits

Department of Physics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2021

Multi-channel Electrical Impedance Tomography with Field Programmable Gate Array

Kjetil Vermundsen Madsen

Multi-channel Electrical Impedance Tomography with Field Programmable Gate Array

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

Printed: Reprosentralen, University of Oslo

Preface

Before we start, I would like to offer my heartfelt thanks to my supervisors during this thesis. Both for the incredibly useful help and the wondrous insight into the field of biomedical instrumentation.

Additionaly, I would like to express my gratitude to everyone else at the Department of Physics for helping me realize my thesis, be it technical, theoretical or administrative. I appreciate every one of you.

I cannot forget to thank everyone at V333 and SEF for great conversations, both academical and recreational, as well as all the good times during coffee breaks. You truly made this time special.

Thanks

Abstract

In this thesis a digital system for biomedical applications is proposed. A framework synthesizing analog waveforms and digital processing of measured analog signals is implemented. This is relevant for biomedical instrumentation because electrical signals measured on tissue are often small and noisy due to limitations with applied currents.

A Field Programmable Gate Array (FPGA) is chosen for this role, and the task is performed by generating reference signals with an implemented Direct Digital Synthesizer (DDS). The Digital Signal Processing (DSP) on the sampled data were done with a Lock-In Amplifier (LIA) and Coordination Rotational Digital Computer (CORDIC). Because of this the system is able to calculate the real and imaginary parts of a measured system, as well as the phase difference and magnitude.

Potential applications of the digital system includes, but are not limited to Electrical Impedance Tomography (EIT), impedance measurements over fixed frequencies and frequency sweeps. EIT is a non-invasive, non-ionized form of imaging, typically used in medical medical applications, and is often used in intensive care units. For EIT imaging, the open source EIDORS libraries can be used.

We propose a user-friendly, freely configurable and portable system for easy transfer immitance measurements. Potentially opening the door for different types of continuous measurements.

Contents

List of Figures

List of Tables

List of Listings

Abbreviations

ADC - Analog to Digital Converter
ALU - Arithmetic Logic Unit
AXI - Advanced eXtensible Interface
AXIS - AXI4-Stream
BP - Band-pass
BRAM - Block Random Access Memory
CORDIC - COordination Rotational DIgital Computer
COTS - Commercial Of The Shelf
CRC - Cyclic Redundancy Check
DAC - Digital to Analog Converter
DAQ - Data Acquisition
DDS - Direct Digital Synthesis
DSP - Digital Signal Processing
DUT - Device Under Test
EIDORS - Electrical Impedance Tomography and Diffuse Optical Tomography Reconstruction Software
EIT - Electrical Impedance Tomography
FIFO - First In First Out
FIR - Finite Impulse Response
FPGA - Field Programmable Gate Array
GPIO - General Purpose Input Output
I²C - Inter-Integrated Circuit
I/O - Input/Output
IP - Intellectual Property
LFSR - Linear-feedback shift register
LP - Low-pass
LUT - Look-Up Table
MAC - Multiply and ACcumulate
MUT - Material Under Test
MUX - Multiplexer
NCO - Numerically Controlled Oscillator
PL - Programmable Logic
PLL - Phase Locked Loop
PMOD - Peripheral module
PS - Processing System
PGA - Programmable Gain Amplifier
SoC - System on Chip
STA - Static Timing Analyzis
SPI - Serial Peripheral Interface
UUT - Unit Under Test
VHDL - Very high speed integrated circuit Hardware Description Language

Chapter 1

Introduction

This chapter explains the background for the project and relevant theory, including Bioimmitance and the electrical properties of cells, system requirements for performing bioimmitance calculations and how to implement these in Field Programmable Gate Arrays (FPGA).

Electrical Impedance Tomography (EIT) is a non-invasive, non-ionized form of imaging, typically used in medical medical applications. It is based on the differences in electrical properties in a volume of material, and is measured by applying a current and measuring the excited voltage as $Z = \frac{V}{I}$, thus giving the transimpedance, or inversely, the transadmittance.

A common use of EIT is imaging of organs and tissue, like continuous measurements of lung-function in intensive care units as it does not apply any harmful radiation or intrusive effects [1]. Cardiac imaging, measurements of brain function and gastric emptying, to name a few examples [2].

EIT does however have a few notable challenges. One of these is that it is measured on the outside of the object, without knowing the detailed state of the insides. This means that the path of the current is not known either, and is called the *inverse problem* [2]. Another necessity is knowledge of the mathematical position of all of the electrodes in relation to each other, and movement can deteriorate the quality of the imaging, which is especially prevalent in medical EIT as people and organs tend to move. The spatial resolution of the images are also often lower than with Computer Tomography (CT) and Magnetic Resonance Imaging (MRI), on the other hand it's shown for having the possibility of higher temporal resolution [2].

FPGA is, as the name suggests, programmable logic implemented on a chip. An effective and cheap tool for prototyping, because of its reconfigurability, meaning it can be reprogrammed as much as the developer wants, as opposed to ASICs. FPGAs can not compete in size, but in development time and price, especially for prototyping[1] [3].

[1] FPGAs excel in price for prototyping, and when something is not to be mass produced.

They also allow *real parallel processing*[2] making them extremely effective for processing large amounts of data in a short amount of time. This can be useful in real time systems such as medical imaging.

While bioimmitance measurement tools are often large and expensive, FPGAs can be relatively small and inexpensive.

Specifications of the system

The digital system developed in this thesis needed to be able to generate analog sines in frequencies up to 10 kHz with at least 12 samples per period, to avoid aliasing of the signal [4]. This is a frequency range before cell membranes are shorted and starts acting like conductors [1].

A benefit would be if the system was also able to parallel process for fast Digital Signal Processing (DSP). This signal processing would have to be able to lock-in amplify the measured signal and return the DC and phase shifted angle of the measurement in relation to the reference. Buttons and/or switches would also a benefit for easy prototyping.

In order to measure over analog circuits and process in digital, analog and digital converters are needed. In order to compare with resolution of commercial EIT, these converters need to have an amplitude resolution of 12-bit. To service the frequency needs they need to have a throughput higher than

$$\texttt{throughput} = f_{max} \cdot 12 = 10\,\text{kHz} \cdot 12 = 120\,\text{kHz} \tag{1.1}$$

The system should also be able to control multiplexers (MUX), for switching between different electrode channels, as well as power an analog front-end. These MUXes would ideally have at least 8 channels, giving a good spatial resolution for EIT measurements over a tub filled with saline solution.

Outline

Chapter 1: Introduction - Describes necessary theory for the digital system. This includes the background theory behind bioimittance measurements, electrical properties, DSP operations used in this thesis, data transfer and FPGA design theory.
Chapter 2: Methods - Describes the system design, including all the HDL submodules, as well as the testbench setup.
Chapter 3: Results - Simulated and measured results achieved from the system.
Chapter 4: Discussion - Did the measurements behave as expected, characterization of system, comparison with other similar projects, future work and conclusion.

[2]Processing where different operations are performed at the same time with dedicated hardware resources as opposed to using shared resources as in a CPU

1.1 Bioimmitance theory

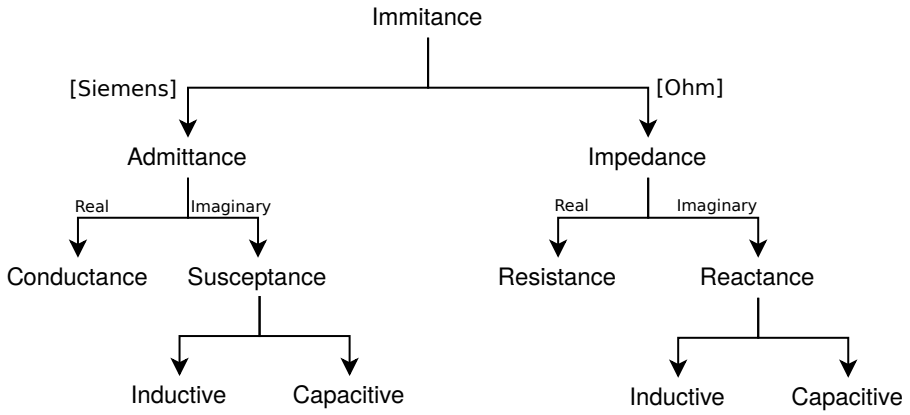


Figure 1.1: Immitance deviated characteristics

Bioimmitance is the common term of admittance and impedance of biological matter. The admittance is the cell's ability to let current pass, and is the inverse of impedance, which is the ability to limit current flow. In this section, different electrical properties of cells and how to measure them will be discussed. Bioimpedance is living cell's ability to impede current flow both because of the material and the geometry of the cell [1]. Properties ending with `-ivity` are insintric properties of the material while `-ance` are properties which also depend on the geometry of the material. A cell can be viewed as a dielectric, as it has several dielectric properties. A dielectric is an insulator that can be polarized applying an electrical field, where the ability to polarize is defined by the material's permittivity [1].

1.1.1 Transfer Immitance

Transfer immitance is the backbone of EIT measurements [1]. By applying a current, I_i, and measuring a voltage, V_o, one can calculate the impedance of the material the current was applied over. It is the transfer function for impedance.

$$Z = \frac{V_o}{I_i} = \frac{1}{Y} \tag{1.2}$$

Where Z is the impedance, measured in ohm, and Y is the admittance measured in Siemens. The impedance is the magnitude of a material's resistive and reactive components

$$Z = \sqrt{R^2 + X^2} \tag{1.3}$$

1.1.2 Frequency dependant impedance

Impedance consists of two components, resistance `R`, and reactance `X`, respectively the real and imaginary part. With direct current there is only resistive losses and no phase shift between the voltage and current, meaning $X = 0$. With

alternating current, part of the energy can be stored either as charge buildup (capacitor) or as magnetic fields (inductor), in addition to the resistive losses. Thus the reactance can either be positive (inductive, X_L) or negative (capacitive, X_C). Due to the mechanisms behind energy storage, reactance is frequency dependant and that is why the current is out of phase with the voltage, because energy storage is not instant [1]. Similarly the inductive and capacitive properties of admittance is described by susceptance `B`.

$$X_C = \frac{1}{2\pi f C} \tag{1.4a}$$

$$X_L = 2\pi f L \tag{1.4b}$$

$$X = X_L - X_C \tag{1.4c}$$

The inductive reactance of cells is almost non-existent, thus the only reactive part of the cells are capacitive and impose a negative phase shift on the applied signal.

$$X_{cell} = X_c \tag{1.5a}$$

$$Z_{cell} = \sqrt{R_{cell}^2 + X_{cell}^2} \tag{1.5b}$$

1.1.3 Capacitance

Capacitance is the materials ability to store electrical charge [4].

$$C = \epsilon \frac{A}{d} \tag{1.6}$$

Where C is the capacitance in Farad (F), A is the area of the capacitor plate, in this case the cell wall, in m^2 and d is the distance between capacitor plates, inversely proportional with the cell wall, in m and the permitivity ϵ in F/m, where ϵ is the product of absolute permittivity in vacuum and the relative permittivity.

The cell membrane is what seperates the intra-cellular liquid (ICL) from the extra-cellular liquid (ECL), in other words, it's what separates, and thus protects, the inside of the cell from the environment. However, it is semipermeable, meaning certain molecules or ions can flow through. The membrane has two layers, and behaves as a capacitor, where each layer is considered to be a capacitor plate. Because of the nanometer thin cell wall [1] the distance (d) between capacitor plates is rather small, making the capacitance large, as can be seen from eq. (1.6).
This means that low frequency currents will flow around the cells and the dominating impedance will be a result of the liquids on the outside of the cells, while a high frequency current will short the cell and go straight through it, acting as a dielectric and conductor respectively. In practice the measured impedance will be a result of the relationship between liquids both on the outside and inside of the cells, as well as on the celltypes and tissue structures in the region where

the current flows. The frequency dependent change between current passing in the liquid between the cells and the current starting to short across the cell membranes typically start to occur around 50 kHz [1].

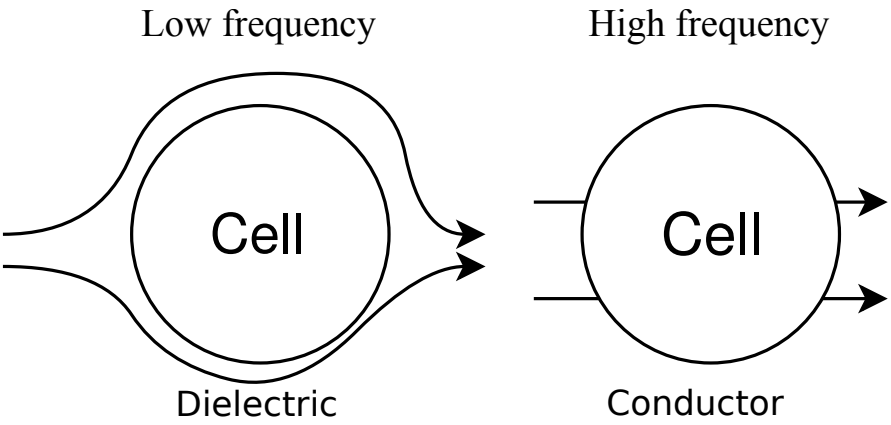


Figure 1.2: Frequency response on a cell membrane

1.1.4 Electrical double layer

An electrical double layer (EDL) is when ions are attracted to the surface of an object because of electrical charge. This will create a layer of either positive or negative ions, this is called the surface charge. EDL can be a source of noise in bioimmitance measurements if we measure at frequencies where the additional impedance from the EDL dominates over the electrical properties of the tissue we are measuring on [1].

1.1.5 Electrical Impedance Tomography

EIT is a form of imaging based on the electrical properties of materials. A higher transimpedance for one channel will imply that a material of higher impedance will be in the way of the current between the input and output electrodes.
2-dimensional cross-section images are made with EIT. This is a problem as current flows 3-dimensionally, meaning there can be more than one solution for image reconstruction. Layers of cross-sections can be stacked as several rings of electrodes, returning 3-D images. If an image is made for several different frequencies the types of materials might be inferred from empirical data, by knowing their conductivity or resistivity. This circumvents the inverse problem if the a-priori data are accurate enough [2]. EIT is dependent on the relative change in electrical properties, hence the larger difference there is between electrical properties, the better the image gets.

The first commercially available EIT lung imaging device was the Sheffield Mark 3.5 with 16 electrodes, and was produced by Maltron International, their

current device has 8 channels. They have been producing EIT equipment for over 20 years [5].
Generation of EIT images can be done through the usage of external third party open source libraries such as EIDORS [6].

1.1.6 Reciprocity

A system is reciprocal if the function of the system is independent of the direction of the input and output. This means that the response of the system is equal for excitation and response [1]. Reciprocity is beneficial especially for multi-probed measurement systems, due to the reduced amount of measurements required.

$$Z = \frac{V_o}{I_i} = \frac{-V_o}{-I_i} \tag{1.7}$$

1.2 Trigonometry background theory

In this section trigonometric theory behind calculations for bioimmitance measurements is discussed. Later this theory is applied for digitally deciding the phase difference between signals.

1.2.1 Arctangent

For calculating the angle between sides of a triangle, arctan can be used.

Arctangent is the inverse tangent and thus converts the ratio between sides into the angle between them.
`y` $= side_{opposite}$ and `x` $= side_{adjacent}$

$$tan\theta = \frac{y}{x}$$

$$tan^{-1}\frac{y}{x} = \theta$$

With this formula, only the ratio between sides matter for the angle, and the polar quadrant is thus not specified. This is why in programming, atan2 is more often used. The formula of atan2 takes into account both the ratio and the sign of the sides [7]. This increases the range from $[-\pi/2, \pi/2]$ to $[-\pi, \pi]$, and is then able to depict the entire unit circle.

2-argument arctangent

Instead of only using the ratio to compute the angle, atan2 needs both of the cartesian coordinates, `(x, y)` and converts them to polar coordinates, `(r,` θ`)`. r is the `magnitude` of `(x, y)`, while θ is the angle between the point `(x, y)` and the plane.

$$x = rcos\theta \tag{1.8a}$$

$$y = rsin\theta \tag{1.8b}$$

$$r = \sqrt{x^2 + y^2} \tag{1.8c}$$

$$\theta = arctan\left(\frac{y}{x}\right) \tag{1.9}$$

Equation (1.9) is not always accurate as it does not return the quadrant of the phase angle. For this reason it can be extended to atan2, covering more cases.

$$\text{atan2(y, x)} = \begin{cases} arctan\left(\frac{y}{x}\right) & \text{for } x > 0 \quad \texttt{- Quadrant 1 if } y > 0 \texttt{ or 4 if } y < 0 \\ arctan\left(\frac{y}{x}\right) + \pi & \text{for } x < 0 \text{ and } y \geq 0 \quad \texttt{- Quadrant 2} \\ arctan\left(\frac{y}{x}\right) - \pi & \text{for } x < 0 \text{ and } y < 0 \quad \texttt{- Quadrant 3} \\ \frac{\pi}{2} & \text{for } x = 0 \text{ and } y > 0 \quad \texttt{- Between quadrant 1 and 2} \\ -\frac{\pi}{2} & \text{for } x = 0 \text{ and } y < 0 \quad \texttt{- Between quadrant 3 and 4} \\ \text{Undefined} & \text{for } x = y = 0 \end{cases}$$

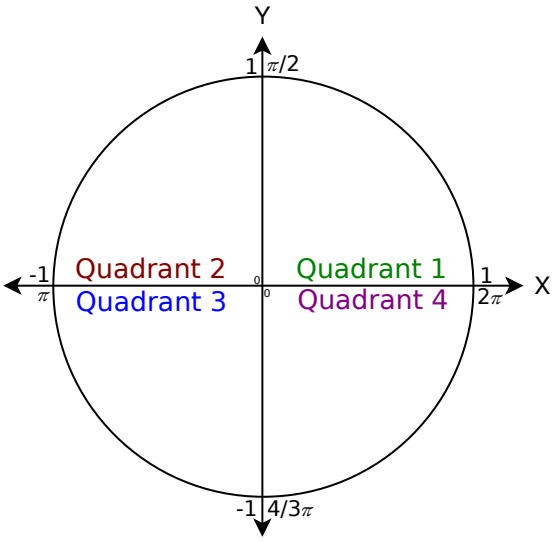


Figure 1.3: Unit circle

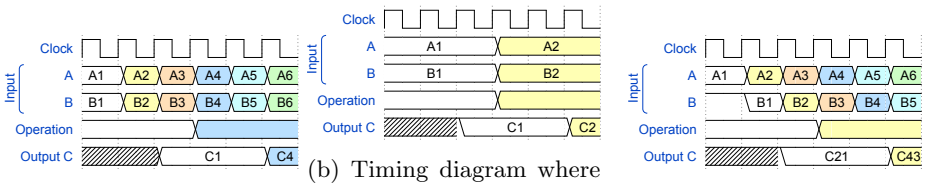
1.3 FPGA design theory

This section discusses the background theory and design for several of the FPGA modules designed in this project as well as communication protocols, along with the reason for choosing these techniques.

1.3.1 Operations in FPGA

Certain operations spend more than one clock cycle to be performed [12]. This can cause bottlenecks in operating throughput or even loss of data.

If a system needs data from multiple sources this can cause them to 'arrive' at different times, this will cause errors in the calculations. In this subsection multiplication and division and how to align them in larger systems is discussed.



(a) Timing diagram of overflow

(b) Timing diagram where input is buffered beforehand

(c) Timing diagram where data is not aligned

Figure 1.4: Timing diagrams of different dataflow problems

For many of these operations, turning *floating point* numbers to *fixed point* is necessary [12]. Reason being an integer's lowest non-zero value is $|1|$. Fixed point means turning a decimal number to an integer and is done by multiplying the decimal by a power of 2. This can greatly increase resolution without needing decimal numbers, where the resolution of the fixed point number is the inverse of 2^N. Lastly, the product will have to be divided by the same power of 2 in order to get the correct scale.

$$\texttt{integer} = \texttt{decimal} \cdot 2^N \tag{1.10a}$$

$$\texttt{resolution} = 1/2^N \tag{1.10b}$$

Common operation alternatives

Mathematical operations in FPGA are often resource or time intensive, because they often need many logic ports and intermediate results. Multiplication is performed in three clock cycles, while boolean operations such as AND, OR and XOR are implemented using combinatorial logic, which is asynchronous and is such considered to be instantaneous[3]. Division with an arbitrary number is especially difficult in FPGA and is thus not recommended. A common

[3]There is an Resistive-Capacitive (RC) timing delay between gate and interface, but for our purpose this is not considered as it does not increase the amount of clock cycles needed.

technique to achieve multiplication and division are left- and right- bit-shifting, respectively. However, one is constrained to a resolution of factor 2.

$$\frac{x}{2^N} = x >> N \quad \texttt{Right shift for division} \tag{1.11a}$$

$$y \cdot 2^N = y << N \quad \texttt{Left shift for multiplication} \tag{1.11b}$$

This means that if at all possible one should opt to divide or multiply with a power of 2. This is not always possible and it can then be an alternative for division to instead multiply with the fixed point number of the inverse value of the divisor.

Pipelining

Bottlenecks in a system can cause either loss of data or loss of performance, this is why pipelining is used. To ensure everything is synchronized, data is buffered in registers, making every path as long as the critical timing path. This results in higher throughput and a more stable system, this does however increase start latency, which often is negligible in a large or continuous system. Pipelining does not increase the operation speed of a single operation, but allows for higher throughput

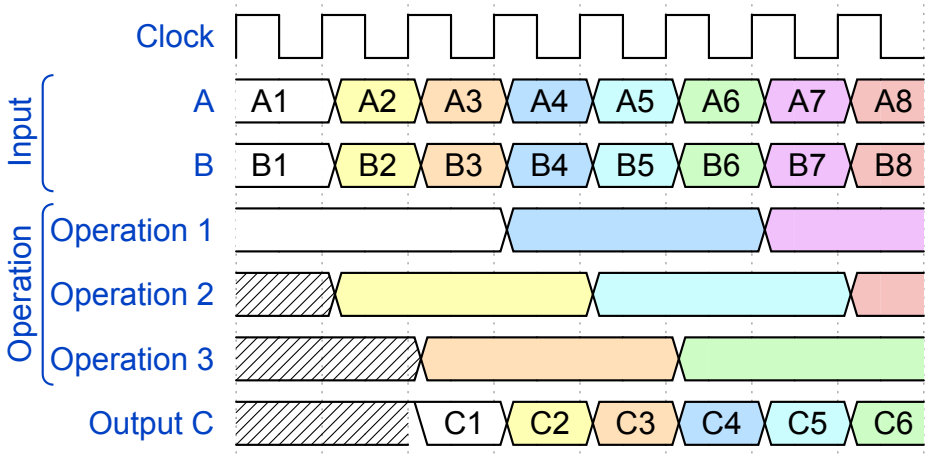


Figure 1.5: Pipelining timing diagram

In order to quantify the performance of a pipelined design, the metric Mean Throughput per Cycle (MTPC) is used. `MTPC` is the amount of finished results per cycle on average, and the ideal pipelined system would have an `MTPC` of 1. The variable Cycles to Result (CtR) is the needed clock cycles from start to finished result. Number of pipelined results is `N`. This gives us the equation

$$\texttt{MTPC} = \frac{1}{\texttt{CtR} + \texttt{CtR/N}} \tag{1.12}$$

For the first operation a result will take as long time as without pipelining, but every subsequent cycle there will be a new value. Thus pipelining will see

most benefit in long running systems with high latency. This means that a low latency system will have an increase of resource usage for little to no throughput increase.

1.3.2 Direct Digital Synthesis

In order to perform measurements, analog waveforms need to be generated. To achieve this from the digital domain Direct Digital Synthesis (DDS) can be used. Different types of waves can be generated, such as sines, triangle- or square waves.

DDS can be used for analog waveform generation from digital logic and a DAC, its analog equivalent would be the Phase-Locked Loop (PLL). DDS is divided into three parts; deciding frequency, deciding phase and deciding amplitude.

Frequency Control Word

The Frequency Control Word (FCW) controls the frequency by deciding how much the Numerically Controlled Oscillator (NCO) will increment for each iteration. The control words are dependant on the system clock frequency, f_{sys}, the desired output frequency, f_{out} and the NCO vector length `N`.

$$f_{out} = \frac{FCW \cdot f_{sys}}{2^N} \tag{1.13a}$$

$$FCW = \frac{f_{out}}{f_{sys}} \cdot 2^N \tag{1.13b}$$

Numerically Controlled Oscillator

The NCO is the phase incrementer of the DDS. For each iteration the value of the FCW is added to the NCO, thus deciding how large phase jumps per step. The phase goes from 0 to 2π.

$$\texttt{nco} += \texttt{FCW} \tag{1.14}$$

Because the NCO is an oscillator `nco` will loop around to 0 when it overflows.

$$\theta \in [0,\ 2\pi] \tag{1.15a}$$

$$\texttt{nco}(\theta) = \texttt{nco}(\theta + 2\pi \cdot \text{N}) \tag{1.15b}$$

If this relation is not satisfied a phase drift will be introduced. Phase drift is an unwanted offset in the waves expected phase.

Phase-to-Amplitude Conversion

The phase from the NCO is used as an address for the desired sine amplitude from a table with sine values.

$$sin_{Reg} = \texttt{sine\_table[nco]}$$

Advantages and improvements

Advantages DDS has over its analog counterpart is better frequency agility, it can change frequency the instant the FCW is changed, both the frequency and phase is highly tunable, the last point also applies for the phase [13]. Increased vector length of NCO will increase the resolution, without having to increase the length of the output vector as only the top bits are used. This will however cause spurious truncation.

Disadvantages and how to avoid them

DDS also has a few disadvantages, because the frequency changes the instant the FCW changes, there is no smooth transition between the frequencies during a change, resulting in a period where the wave has to different frequencies. This change can happen at any time during the period.
Also, because of truncation in the `nco`, each period will not necessarily look the same.
A possibility for removing these problems are to manually reset the `nco` to 0 when the frequency changes and when the `nco` overflows.

1.3.3 CORDIC

COordination Rotating DIgital Computer (CORDIC) is a method of approximating various mathematical methods, such as generating sines, calculating amplitude [14]. For this system CORDIC was used to calculate the phase angle by calculating the atan2 (see eq. (1.9) for more details). Calculating the atan2 also gives us the magnitude, as a by-product.

$$\phi, \|Z\| = atan2(X_c, R) \tag{1.16}$$

This is performed by doing a binary search of the opposite side of the triangle with bit shifting, which results in easier calculations for the FPGA as explained in section 1.3.1.

This binary search works with a table of tan values where the angles range from 45° to 0°, meaning the maximum angle calculated can be $\approx 90°$, if all angles are summed. These angles are not half the previous angle, but the inverse tan of the right shifted value. The most important part of the angle table is that the next value is between the previous and half of the previous.

$$\texttt{ang\_table[i]} = tan^{-1}(2^{-i}) \cdot \frac{180}{\pi} \tag{1.17}$$

Angle resolution is decided by the amount of elements in the table. These are turned to fixed point.

Calculating the new values of the sides and angle

$$X = X + (|Y| >> \texttt{loop\_cnt}) \tag{1.18}$$

$$\mathrm{Y} = \begin{cases} Y - (X >> \texttt{loop\_count}) & \text{for } Y > 0 \\ Y + (X >> \texttt{loop\_count}) & \text{for } Y < 0 \end{cases}$$

$$\theta = \begin{cases} \theta + \texttt{ang\_table[loop\_count]} & \text{for } Y > 0 \\ \theta - \texttt{ang\_table[loop\_count]} & \text{for } Y < 0 \end{cases}$$

Where `loop_count` is the loops of calculations performed. The max value of this is dependent on the vector length of the sides. When `Y` is 0 the rotations are complete. These operations are pseudo-rotations of the sides, where `X` increase in length for each iteration with a gain of

$$A_n = \Pi_n cos(ang\_table[i]) = \Pi_n \sqrt{1 + 2^{-2i}} \tag{1.19}$$

`X` needs to be multiplied with the inverse gain in order to get the magnitude.

$$Magnitude = \texttt{X} \cdot 1/A_n \tag{1.20}$$

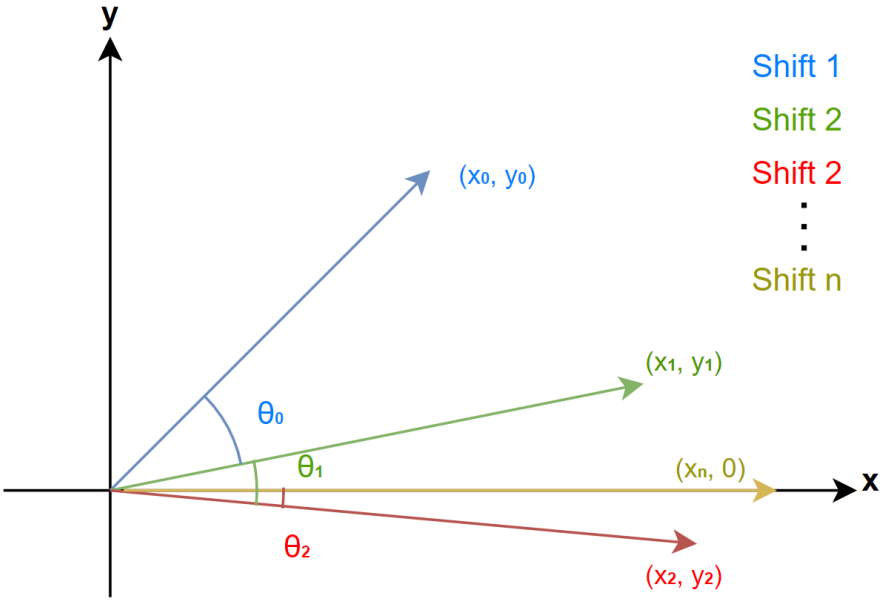


Figure 1.6: Binary search vector shift, where `Shift i` is the pseudo-rotation from i-1 to i

1.3.4 Lock-in Amplifier

Lock-in amplification is a method of extracting a modulated signal of known frequency from a noisy environment [1]. It demodulates the signals by multiplying with a reference and filtering away the unwanted frequencies. This results in an amplifier with extremely narrow bandwidth, and is thus perfect for isolating/filtering out the wanted signal in a noisy environment [15], such as living tissue.

Lock-in Amplifiers (LIA) can be digital and analog, where the former are more flexible and precise, while the latter have lower power consumption and

higher frequency of operation [1]. Digital LIAs are composed of a demodulator and an averager.
The in-phase and quadrature (I/Q) parts are found by multiplying the signal with a reference signal of the same frequency or a 90° phase shifted reference signal respectively, and then averaged. These are the real and imaginary parts of the signal [1].

$$Inphase = \frac{V_{in}(\theta + \phi) \cdot sin\theta}{2} \tag{1.21a}$$

$$Quadrature = \frac{V_{in}(\theta + \phi) \cdot cos\theta}{2} \tag{1.21b}$$

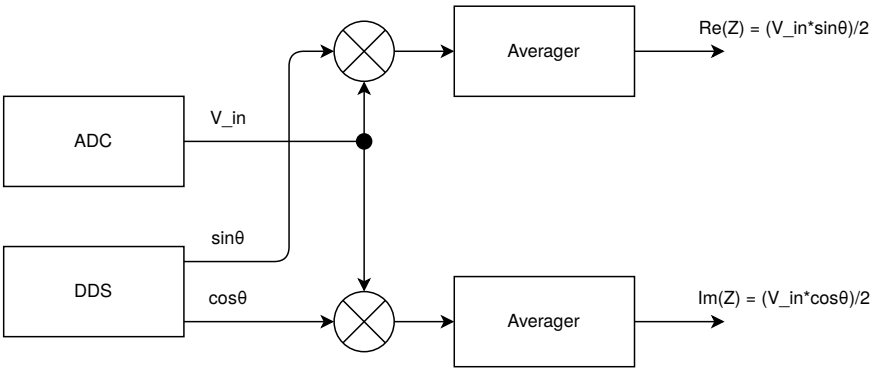


Figure 1.7: Digital LIA block diagram

An input signal with a modulated frequency equal to the reference frequency will always yield a non-zero result for one of the reference phases, the other will be an output with double the frequency swinging around 0, therefore being filtered away, either with an averager because the average is 0, or a low-pass filter, typically implemented as a Finite Impulse Respone (FIR) filter. Zeroing of signal will also happen with all other frequencies than the reference. This means that if the measured signal is perfectly in phase with either of the references, this average will also be the magnitude, and the phase shift will be either 0° or 90°. Transfer impedance and -conductance are the complex transfer functions between voltage and input, and thus needs both the magnitude and phase shift. Unless measuring over purely resistive systems this will not be the case and both components will have a contribution.

Quadrature amplitude modulation is also used in Quadrature Phase Shift Keying (QPSK) in digital communication [16]. It transfers data by phase shifting a constant frequency.

1.3.5 FIFO

A First In First Out (FIFO) buffer is a type of buffer behaving like a queue. It stores elements which are then able to be extracted later. Elements are written from one part and read from another, where pointers control where the next

element can be written to or read from. Like a queue new elements are written to the back and read from the front. In FPGA this can be used as a synchronization buffer or storage for vectors. For synchronizing between different clock speeds, the FIFO use the different clocks for read and write. The write and read pointers loop around and if one catches the other the buffer is either full or empty. The pointers are used as addresses for pulling or pushing data. Control inputs are read and write flag, and output flags are full and empty. Some systems also make use of almost full and almost empty, which signal when they are one step away. A FIFO can not be written to when full, and not read from when empty, as in the first case data would be overwritten before it is read, and in the latter nothing new would be read. This is to prevent overflow.

Table 1.1: FIFO control signals truth table

Full	Empty	Write enable	Read enable	Operation
0	0	0	0	Nothing.
0	0	0	1	An element is read, and read pointer is incremented.
0	0	1	1	An element is read and another is written, both read- and write pointers are incremented.
0	1	0	1	Nothing, as there is nothing to be read.
0	1	1	0	An element is written and the empty flag goes low.
0	1	1	1	An element is written and the empty flag goes low.
1	0	0	1	An element is read and the full flag goes low.
1	0	1	0	Nothing, as the buffer can not be written to when full.
1	0	1	1	An element is read and the full flag goes low.

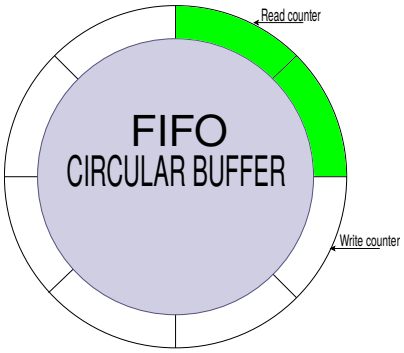


Figure 1.8: FIFO Circular buffer concept

1.3.6 SPI

In order to control and communicate with the peripherals of the FPGA, a communication protocol is needed. In this subsection SPI is introduced and compared to a few other serial communication protocols, and we explain why SPI was ultimately selected.

Introduction

SPI is an interface for low-overhead, fast and parallel transfer of data to and from slave and master.

SPI is a standard popularized by Motorola in the 1980s. It is traditionally used with four wires; `sclk`, `ss/cs`, `miso` and `mosi`, but can be used with fewer. The `sclk` is the clock from master driving the slave. The `ss/cs` are respectively the slave- and chip select. `Miso` and `mosi` are the data transfer bits serially, meaning data is transferred by shift register. If data is only sent one way, such as in a Digital-to-Analog Converter (DAC), only one of the data lines are needed, making it a three-wire system.

Table 1.2: SPI wires

Wire	Use
`sclk`	System clock
`miso`	Master-in Slave-out Serial data from slave to master
`mosi`	Master-out Slave-in Serial data from master to slave
`ss`	Slave Select Select what slave(s) are active.

SPI has four modes of operation, depending on the two bits; `CPOL` and `CPHA`. `CPOL` is the SPI Clock Polarity Bit and selects either active high or active low clock.
`CPHA` is the bit used to select SPI clock format.

Table 1.3: CPHA and CPOL functionality

Value	**CPHA**	**CPOL**
0	Sampling occurs at odd edges of the SCK	Active-high clock. Idle state SCK is low
1	Sampling occurs at even edges of the SCK	Active-low clock. Idle state SCK is high

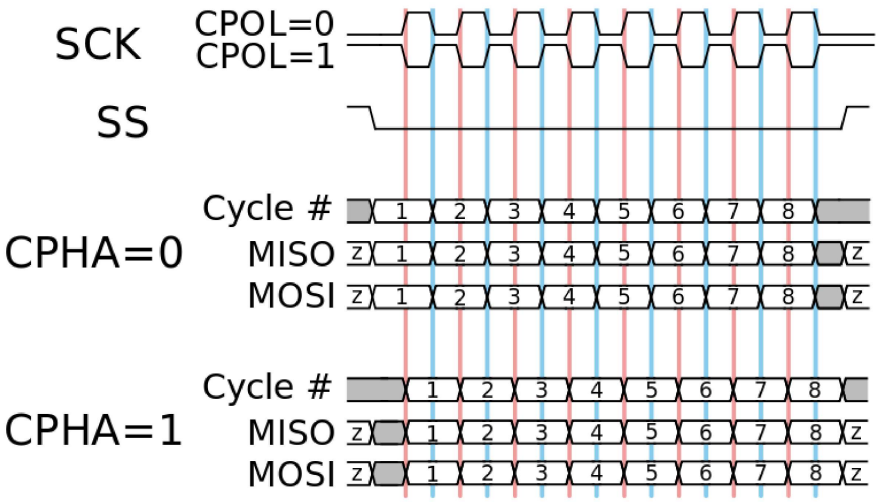


Figure 1.9: Timing diagram of SPI protocol from [9]

I²C

Inter-Integrated Circuit (1²C) is a serial communication bus with multi-master and multi-slave support. It has one data- (sda) and one clock-line (scl). Because it only has one data line, the adress of slave needs to be part of the package, making it larger than SPI overhead and thus slower.

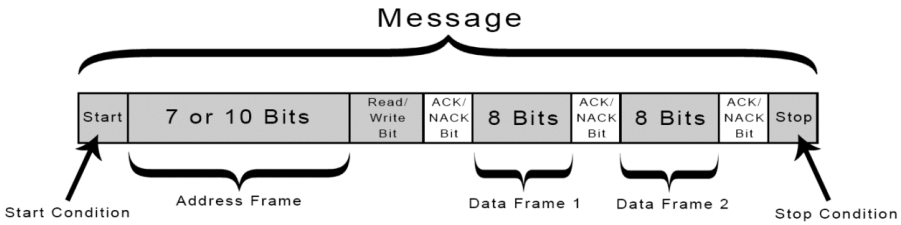


Figure 1.10: I²C package, figure from [10]

UART

Universal Asynchronous Recieve Transfer (UART) protocol is a serial interface for full-dulex asynchronous transfer of data. It has 2 wires, one for transfer (tx) and one for receiving (rx). Because it has no clock line a start and stop of transaction is signalled with start and stop bits. This tells the receiver when to start and stop reading. However because of this, the tranceiver and receiver need a common clock rate. It also has a parity bit which checks if the sent packet is equal to the received packet by counting the numbers of 1's in the packet [11].

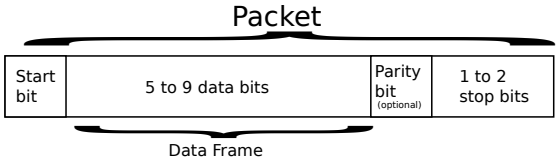


Figure 1.11: UART packet data [11]

Comparison

Out of these interfaces, SPI is the fastest, though it has no acknowledge or parity bit. For output of a continuous wave this does not matter as streaming does not benefit from re-sending lost or broken data at a later date.

Table 1.4: SPI, I²C and UART comparison

Properties	SPI	I²C	UART
Wires	4(downto 2)	2	2
Masters	Single	Multiple	Single
Slaves	Multiple	Multiple	Single
Duplex	Full	Half	Full
Synchronicity	Synchronized from master	Synchronized from master	Asynchronous
Acknowledge	No	Yes	Parity bit
Slave select	Enable signal	Address on data line	N/A
Data frame	Custom	8 bits, but can have several frames for each message	9 bits

1.3.7 Linear-feedback shift register

For simulation, randomization is used to efficiently test unpredicted cases. A Linear-Feedback Shift Register (LFSR) is a technique often used for pseudo-randomization, typically for numbers or noise sequences [17]. It is a shift register with an input that is a linear function of the previous state, typically done by XORing several of the bits, these bits are called the *taps* of the function. Not all taps provide the maximum amount of functions, i.e. $2^N - 1$ [17]. A seed of all 0's will never change the output as $0 \hat{} 0 = 0$. An LFSR can easily be implemented in both hardware and software. In hardware an operation is performed each clock cycle as both shift and XOR are simple operations. As previously mentioned, a bit shift is a simple wire change in hardware, and an XOR operation is performed with the XOR port, making it fast and resource light.

Chapter 2

Methods

In this chapter the different FPGA modules in the digital system and their purpose, as well as the analog front-end and the setup of testbenches, is described.

2.1 FPGA Implementation

In this section the different FPGA modules made for this project is explained, in order of dataflow.

The board used in this project is TUL Pynq-Z2 with the Xilinx Zynq-7020 FPGA. Among the Input-Output (I/O) peripheral types it contains are Ethernet, Peripheral Module (PMOD) and Raspberry Pi headers to name a few. This makes the board very useful for connectivity. Pynq boards are unique in that they can run Python code, and even the bitstream can be loaded, meaning digital logic on the FPGA can be programmed through Python. Another advantage of this board was the PMOD ports. These are generic peripherals for many different types of connections and interfaces, making prototyping more trivial. In the case of this project they were used for connecting Digital-to-Analog Converter (DAC) and Analog-to-Digital Converter(ADC) with the Serial Peripheral Interface (SPI).
With it's 3.3 V, 5 V and ground ports, it is able to ground and supply an analog front-end.

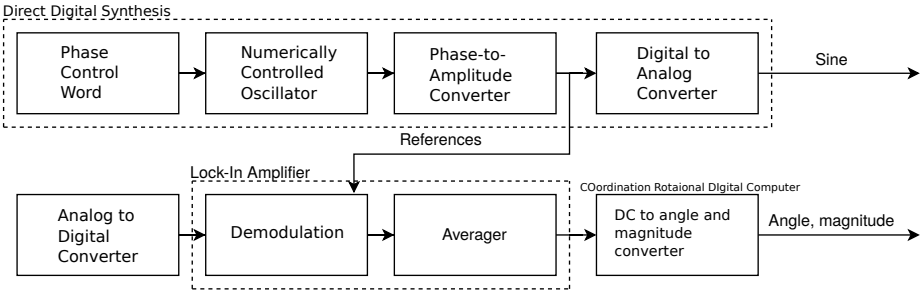


Figure 2.1: Abstracted block diagram of DDS and DSP system

The FPGA system is made as modular as possible to enable adding of new functionality and changing variables such as frequencies, electrodes etc. This makes testing different setups easier and it also allows for others to use the same system in the future.

2.1.1 Clocking

The SPI `sclk` for the converters are derived from the PL clock. The DAC can operate up to 50 MHz while the ADC can operate up to 20 MHz. In both these cases the clock divider is

$$\texttt{sclk} = \frac{\texttt{clk}}{2 \cdot N_{div}} \tag{2.1}$$

Where $\texttt{clk} = PL_{clk} = sys\_clk$ The 2 from the equation comes from the fact that the division counter can only increase on each rising edge of the `clk`, this means that a clock can only be divided by an even number, if not the ADC `sclk` would be divided by 5 instead of 6, thus making use of the entire frequency capacity of the ADC.
The Processing System (PS) clock is a variable clock which can be changed at any time in Jupyter Notebook.

Table 2.1: Clock division

	N_div	`sclk`
System Clock	N/A	100 MHz
DAC	1	50 MHz
ADC	3	16.67 MHz

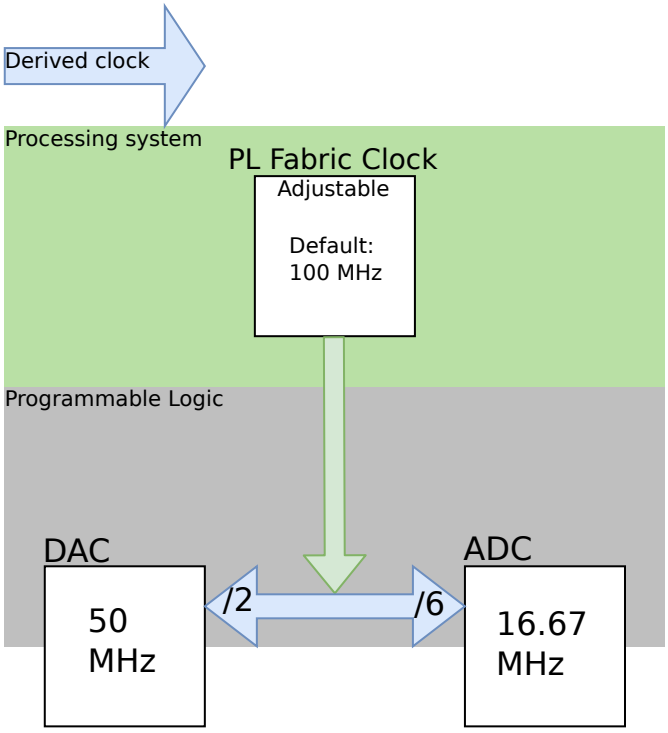


Figure 2.2: Clock tree

2.1.2 Electrode Master

The task of the `Electrode Master` module is to perform an electrode sweep for all frequencies. It is a two-electrode measurement with one input channel and another output channel. Because of reciprocity for each new input channel the max channel of the output channel is decreased by one, where the max channel is the number of electrodes, saving half of the measurements. After the electrode sweep for a frequency is done it increments to the next frequency.

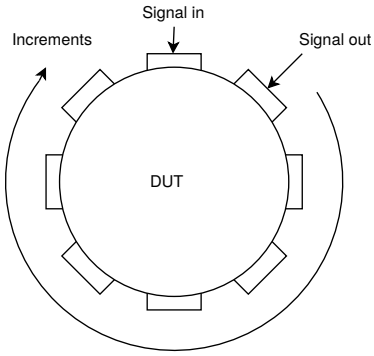


Figure 2.3: Electrode change

Table 2.2: Simplified Electrode Master I/O

Direction	Signal	To/from
Output	`FCW`	Phase Accumulator
Output	MUX control signals	Analog front-end

Algorithmic State Machine (ASM) diagram can be found in Appendix, fig. A.1 Timing waveform of electrode and frequency iteration can be found in Appendix, fig. B.1

2.1.3 Phase Accumulator

The `Phase Accumulator` outputs the in-phase (sine) and quadrature-phase (cosine) based on the frequency from the `Electrode Master`. It does this by getting the FCW from a table using the frequency as the index and adding it to the `nco` as explained in eq. (1.14)
The SPI system clock for the DAC is 50 MHz, the length of the NCO is 32 bits and the desired output frequencies are 1 kHz to 10 kHz. According to eq. (1.13) this yields

$$FCW = \frac{\texttt{f\_sine}}{\texttt{f\_DAC}} \cdot 2^{N_{NCO}}$$

Where `f_sine` ranges from 1 kHz to 10 kHz, `f_DAC` is set to 50 MHz and the number of bits in the NCO is set to 32 bits.

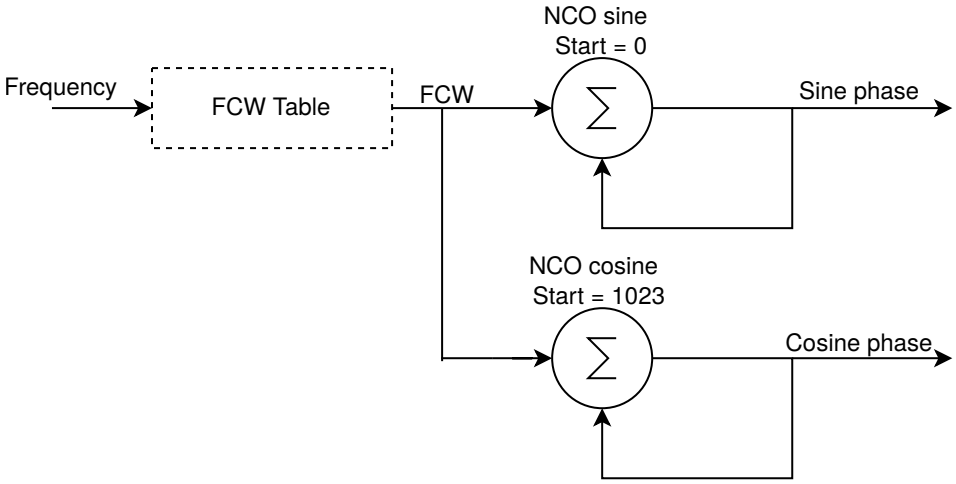


Figure 2.4: Diagram of implemented Phase Accumulator

The `nco` has 2^{12} values, giving a phase length of $2\pi = 4095$. It is converted to a fixed point vector of 32 bits for increased phase resolution. This means that at certain frequencies one step does not necessarily change the phase.
The `nco` of the sine starts at phase 0, while the cosine starts at one fourth of the highest phase value, $4096/4 - 1 = 1023$. This creates the phase shift of 90°.

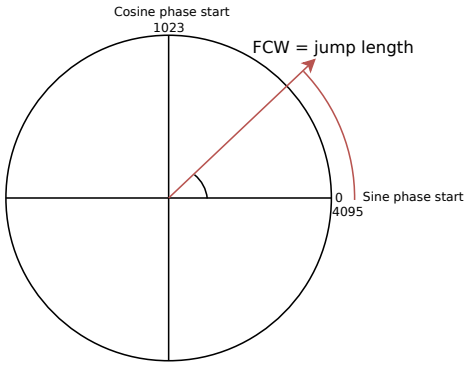


Figure 2.5: NCO jump length

Table 2.3: Simplified Phase Accumulator I/O

Direction	Signal	To/from
Input	`FCW`	Electrode Master
Output	`nco` of sine	Sine table
Output	`nco` of cosine	Sine table

2.1.4 Sine table

The sine table is a PAC and takes the phase from the phase accumulator and uses this as address in the SIN LUT, and outputs the sine amplitude. The LUT has $2^{12} = 4096$ indices, where this number is the full phase, it is also possible to have one half phase, or just one quadrant if one uses more logic, this could have been done with CORDIC. As the exchange of memory space versus logic would be negligible in this system the more trivial solution was chosen.

Table 2.4: Simplified Sine table I/O

Direction	Signal	To/from
Input	`nco` of sine	Phase Accumulator
Input	`nco` of cosine	Phase Accumulator
Output	`sin`	DAC and DSP
Output	`cos`	DSP

2.1.5 SPI

The two SPI controllers used have a few key differences. Both only use one-way dataflow, but the SPI to the DAC has one data-channel, and the ADC has two data-channels. Otherwise only the timing and sclk frequencies are different.

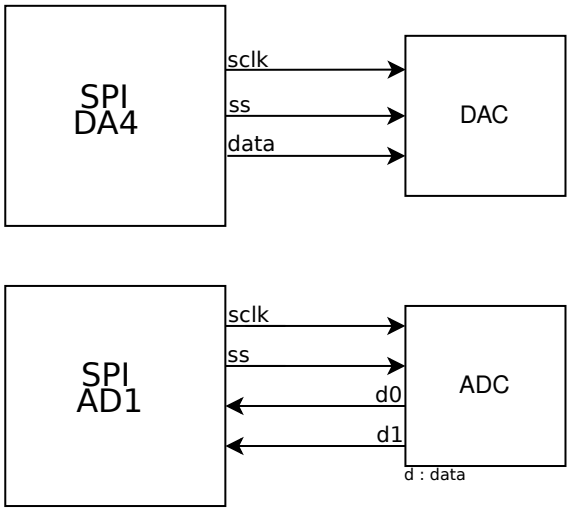


Figure 2.6: SPI diagrams

DA4 and AD1 are the PMODS for the DAC and ADC, respectively.

2.1.6 Lock-in Amplifier

The lock-in amplifier is implemented in this system to extract the real and imaginary parts of the applied signal over a load.

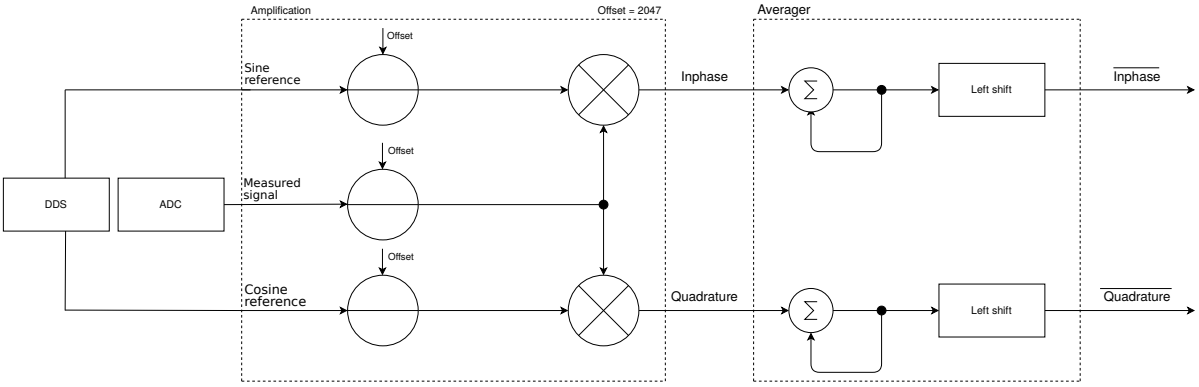


Figure 2.7: Diagram of LIA implementation

Implementation

Since the signals sent from and sampled in the DAC and ADC can only be positive values, both the reference frequencies and measured signal will be an unsigned value from 0 to 4095. To ensure that other frequencies have an average of 0, the signals are digitally level-shifted from the middle-value of $4096/2-1 = 2047$, down to being centered around 0, giving us an output swing of 0 ± 2047

The averager accumulates the calculated signals, and after 10^5 clock cycles the sums are multiplied with a gain of 1.31[1] before being right bit-shifted with 17 bits, i.e. divided by 2^{17}. The amount of clock cycles are derived from the relation between the (lowest) frequency of interest, $f_{sig} = 1\,\text{kHz}$, and the system frequency, $f_{clk} = 100\,\text{MHz}$.

$$\texttt{mean} = (\texttt{sum} \cdot 1.31) >> 17$$

The gain is the ratio between bit shift and the amount of summations per averaging.

$$A = \frac{2^{17}}{10^5} = 1.3107$$

This essentially divides the sum over 10^5.

Table 2.5: Simplified Lock-In Amplifier I/O

Direction	Signal	To/from
Input	Measured signal	ADC
Input	`sin`	Sin table
Input	`cos`	Sin table
Output	Averaged in-phase	CORDIC
Output	Averaged quadrature	CORDIC

[1]This gain is converted to fixed point with a resolution of 2^{14}

2.1.7 CORDIC

The CORDIC calculates the angle and magnitude of the averaged `inphase` and `quadrature` signals from the LIA, referred to as `X` and `Y` for the rest of the section, respectively.
Section 1.3.3 explains how the pseudo-rotations are performed, as well as the gain of the magnitude and calculation of `ang_table`. The main difference in implementation, in relation to the theory, is that the signs of inputs are stored and only the absolute values of them are used for rotations. These Most Significant Bit (MSB) signs are used after the rotations in order to offset the angle to correct quadrant. It also starts by checking if either `X` or Y are 0. If `X` is 0 the angle is $|90°|$, and if Y is 0 the angle is either 0° or 180° according to the sign. The magnitude is the non-zero side, with no gain as it has not been rotated.

Table 2.6: CORDIC FSM states

State	Operation
Idle	Waits for start strobe before buffering inputs
Load	Stores the MSB of the inputs before removing them from the inputs
Check	Checks and handles corner cases, that is if `X` of `Y` are 0 jumps to Scale if this is the case.
Calc	Performs the pseudo-rotations as described in section 1.3.3
Scale	Scales the magnitude with a gain A_n according to the amount of rotations performed. The angle is offset according to the sign of the inputs, stored from the Load state
Result	Outputs the angle and magnitude and sets done strobe high.

Table 2.7: Simplified CORDIC I/O

Direction	Signal	To/from
Input	Averaged in-phase	Lock-in Amplifier
Input	Averaged quadrature	Lock-in Amplifier
Output	Angle	Written to file
Output	Magnitude	Written to file

The ASM diagram is in Appendix, fig. A.2

Resolution

The angles in `ang_table` are turned to fixed point numbers by a factor of 2^7. Angle resolution is then according to eq. (1.10b)

$$2^{-7} = 7.8125° \cdot 10^{-3} \tag{2.2}$$

2.1.8 IP

This subsection describes the different Xilinx Intellectual Properties (IP) used. An IP is proprietary code from the distributor. Vivado[2] block diagram of the system can be found in fig. A.3

[2]Vivado is a design suite used for generating FPGA hardware files from HDL

Direct Memory Access

Uses a Xilinx Direct Memory Access (DMA) IP in the block diagram to connect to Jupyter and transferring the data. It has write-only channels.
The FPGA only has 4 DMAs for the PL so multiple signals are sent through the same DMAs. As with the rest of the block design, the DMA runs at 10 MHz To make the most out of each DMA different signals are concatenated and sent together.

Table 2.8: DMA data overhead

DMA				
0	`Button`	`Electrode in`	`cos reference`	`sin reference`
1	`FCW`	`Electrode out`	`xn[1]`	`xn[0]`
2			`Quadrature mean`	`Inphase mean`
3			`Angle`	`Magnitude`

Integrated Logic Analyzer

For internal probing of the signals, the Vivado Integrated Logic Analyzer (ILA) IP was used. This is a probe that is physically implemented on the FPGA and thus has a certain resource usage [18]. These probes are useful for debugging. ILA is mostly beneficial during early prototyping and development stages, however when the system is near finish, the IP can, and should, be removed if the additional (logic) are is required for other digital blocks in the FPGA.

2.2 FPGA Hierarchy

This section focuses on the connection between the different modules.

The three main parts of the system connected by the top module is the `DDS` module for generating the reference waves, the `DSP` for sampling and performing operations on the measured data and the `packages`, which store commonly used data.

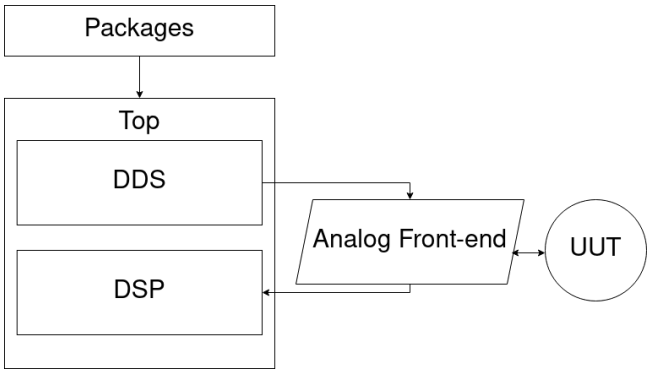


Figure 2.8: Coarse partition of system hierarchy

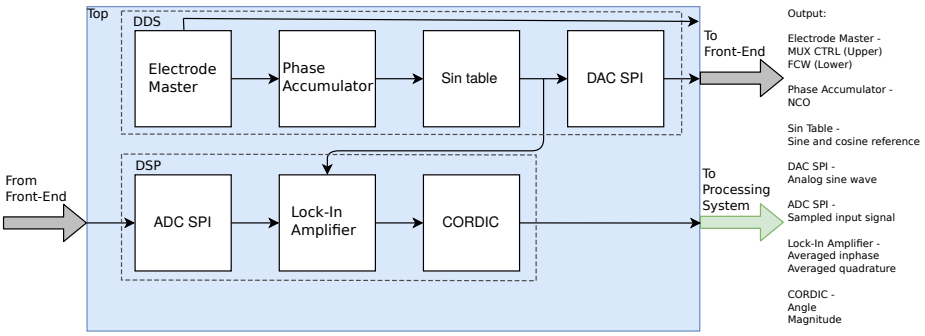


Figure 2.9: System hierarchy of implemented PL modules

2.2.1 Packages

In VHDL one can make special package files for packaging data which is used in several modules, it can be compared to a library or header in other languages. These packages can among other things include constants, types functions and procedures [3]. In the case of this system the two first are used. Constants that are used in multiple modules are packaged together. This allows for changing these values without editing the source code. The sine wave is stored in an array in a package. One package stores all the constants for sizes, such as number of electrodes, array length and vector length. There is also a timer package for comparing counter to the amount of cycles taken for each frequency to reach to periods.

Table 2.9: Packaged constants

Package	Constant	Size	Description	Use
`array_pkg`	`vector_length`	12	Most used vector length	Used for generated sines, sampled signals and more
`array_pkg`	`Electrodes`	8	Amount of electrode channels	Used in the `electrode master` module, for knowing the maximum amount of electrodes as well as for generating more `DSP` components
`array_pkg`	`adc_array`	12x2 array	2-dimensional array for adc samples	Array for the sampled signals from each ADC channel
`sine_wave_pkg`	`C_sin_table`	4096x1 Array	Array of amplitude values of the sine for one period	Used in Phase to Amplitude convertion

2.2.2 DDS

The DDS module connects the Phase Accumulator with the sine table, where the NCO output from the Phase Accumulator is the phase address in Sin table. The PA increments the NCO by an amount from the frequency table, which is chosen from the Electrode Master. This frequency table is calculated from the clock frequency and desired frequency.

2.2.3 DSP

Because of the FPGA's ability to parallel process, each of the electrode channels have their own instanciated DSP component. This makes it possible to increase the amount of electrodes without increasing processing time at the cost of more resource usage. Changing the number of electrodes in the code is as easy as changing the number in the package.
The LIA multiplies the input signal with sine and cosine references according to desired frequency. DC and phase shift is calculated in the CORDIC before the measured and processed signals are written to files in Jupyter Notebook.

[3] Everything that can be stored in packages; constants, types, subtypes, functions, procedures, attributes, components, aliases and files [12]

2.2.4 I/O

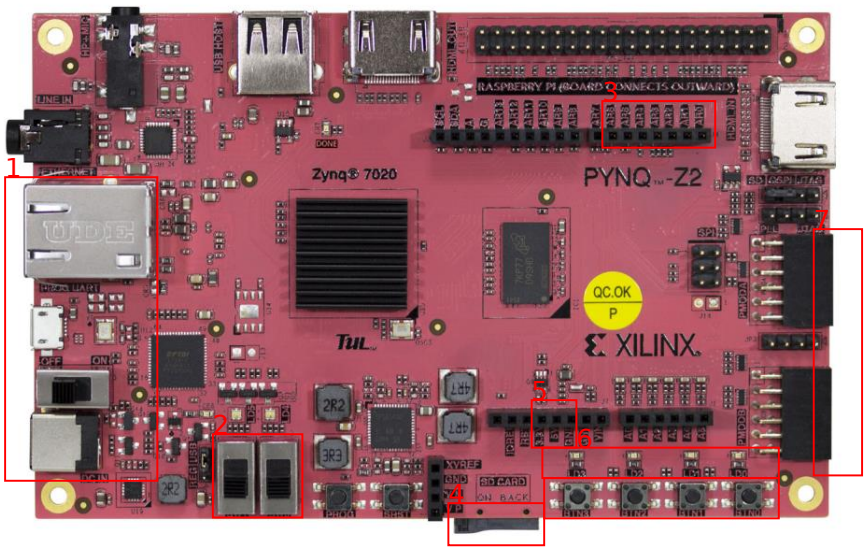


Figure 2.10: PYNQ I/O

Table 2.10: I/O table, function from left to right

Route	Type	Function
1	Phy	Ethernet PHY Micro usb connection Power switch Power supply
2	Switch	`tx` `reset`
3	Arduino ports	MUX controls
4	SD	SD
5	LEDs	LEDs
6	Buttons	Start measurement Keep frequency Increase frequency Decrease frequency
7	PMOD	PMOD ports

2.2.5 Jupyter notebook

From Pynq the overlay and DMA libraries are used to program the FPGA and gather the data. `bit`, `tcl` and `hwh` files are used for this. A function is made to read the data from the DMA buffers, splitting these into the appropriate vectors and then writing them to a `csv` file for EIDORS imaging in MATLAB.

2.3 Analog front-end

In order to measure on a unit under test (UUT), like a tissue, an analog front-end is needed. This is for deciding what channels to use for in-signal and measurements, for amplification of the signal and for getting a proportional voltage reading of the current.

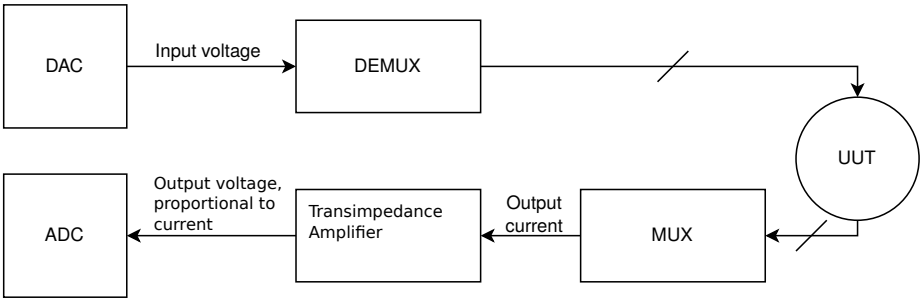


Figure 2.11: Simplified analog Front-end

2.3.1 Transimpedance Amplifier

A transimpedance amplifier is used in the analog front end for measuring the current over a circuit. It is a current to voltage amplifier, meaning the resulting voltage will be proportional to the current accross the UUT multiplied by the gain from the feedback resistor [19]. This gain can be too large compared to the input range the Operational Amplifier(OpAmp) or the spesifications of the DAC, if the resistance of the UUT is sufficiently small. For this reason an input resistor is used, turning the gain from R_f to $\frac{R_f}{R_{in}}$.

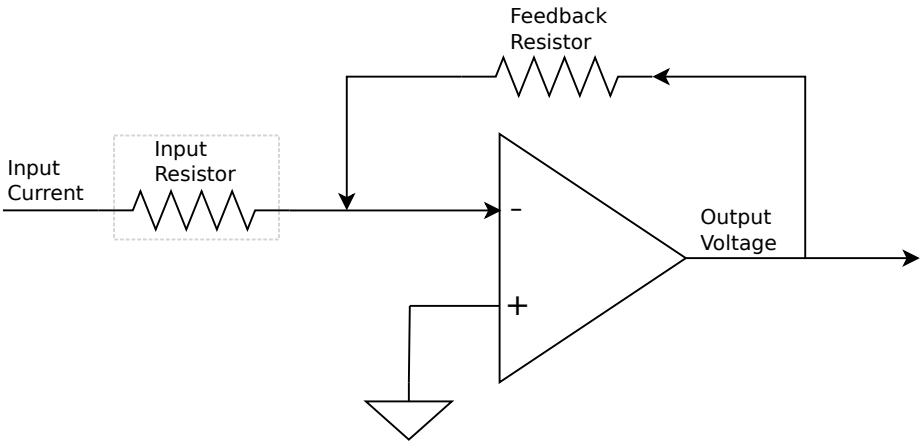


Figure 2.12: Transimpedance Amplifier

2.3.2 MUX

The MUXes used are SN74CB3Q3251 [20] from Texas Instruments. They are soldered on the TSSOP-DIP socket adapters for jumper pin connections. Output signal from the DAC is demuxed from one channel to eight. The Electrode Master module counts which electrodes are used and the control signals are sent to both the DEMUX and MUX. The MUX takes the signals from all of the electrodes and sends the signal from the active one to the ADC. They have a switching speed between channels of 20 MHz.

2.3.3 DAC

The DAC consists of a PMOD DA4 with the AD5628-1 chip. It uses a three-wire SPI protocol with an `sclk` of 50 MHz. It has 8 output channels, which are chosen by `curr_in` from `electrode_master` and used as `addr` in the spi controller. Only one channel can be used at a time. The voltage output ranges from 0 V to 2.5 V.

2.3.4 ADC

The ADC consists of a PMOD AD1 with two AD7476A chips, making it dual-channel, meaning the SPI-protocol has to be adjusted for two channel MISO. `Sclk` is $\frac{100\,\text{MHz}}{6} = 16.67\,\text{MHz}$ as the maximum frequency of the converters are 20 MHz [23]. This sampling rate could be achieved with a system frequency of 120 MHz, but for simplicity's sake it is kept at 100 MHz. As the DAC would then have a clock speed of 60 MHz, exceeding the max frequency of 50 MHz. The voltage input ranges from 0 V to 3.3 V.

2.3.5 Resolution of converters

The sine output over the electrodes and the measured signal from the electrodes are both 12 bits.

Voltage resolution
DAC: 1 LSB = $\frac{2.5\,\text{V}}{4095} = 610\,\mu\text{V}$ [22]
ADC: 1 LSB = $\frac{3.3\,\text{V}}{4095} = 806\,\mu\text{V}$ [23]

Temporal resolution
DAC: sclk = 50.00 MHz @ 33 cycles per transaction $\Rightarrow$ 1.515 MHz
ADC: sclk = 16.67 MHz @ 17 cycles per transaction $\Rightarrow$ 981 kHz

From eq. (1.1), highest theoretical frequency measured without aliasing can be found as

$$\texttt{samples per period} = \frac{\texttt{lowest temporal resolution}}{f_{max}} \tag{2.3a}$$

$$f_{max} = \frac{\texttt{lowest temporal resolution}}{12} = \frac{981\,\text{kHz}}{12} = 81\,750\,\text{Hz} \tag{2.3b}$$

Throughput
Per channel:
DAC: $12 \cdot 1.515\,\text{MHz} = 18.18$ MBPS

ADC: $12 \cdot 981.0\,\text{kHz} = 11.77$ MBPS

Table 2.11: Specifications of converters

Property	DAC	ADC
Clock	50 MHz	16.67 MHz
Channels	1	2
Max Voltage (V)	2.5	3.3
LSB (µV)	610	806
Cycles per Transaction	33	17
Temporal resolution (MHz)	1.515	0.981
Throughput per channel (MBPS)	18.18	11.7

The ADC has two channels, effectively doubling throughput, but only one is used for measurements, the other is for the reference output from the DAC.

2.4 Test setup

Testing of the digital system is done with simulation, verification in ILA, and analog measurements over UUT. The data are also post-processed in MATLAB. To ensure proper behaviour of the system a few things need to be in place. The DDS generated waves need to have deterministic phase, i.e. no noticeable drift, and need to output the expected frequency. Demodulation input and averaging to zero the unwanted parts of the signal needs to be performed in LIA.
The CORDIC need to calculate DC and phase shift with an acceptable resolution.

2.4.1 Simulation

When verifying signals internally under ideal circumstances, simulation is used as a way to ensure the testing is only based on the system itself and no external interference.

Simulating does however need some time for creating a testbench, and the simulations themselves are slow and resource intensive, meaning one can not simulate everything when dealing with larger systems. This is where randomization and edge cases come in. Usually, for verification, it is most interesting to simulate where the system is most likely to fail. This saves on simulation time, and can be done by randomizing input data around edge cases, such as minimum and maximum values.

For compiling and simulating VHDL, the software used was **GHDL**. **GTKWave** was used for displaying waveforms from simulations. They are both open-source software, as opposed to the proprietary **ModelSim**.
The Makefile for compiling and running simulations can be found in listing E.1

To test the system without analog measurements a testbench was made with LFSRs for randomizing input for the simulated electrodes. A weighted test is

made for randomized gain, phase shift and noise of the input. One can choose to omit the randomization of any of these parameters, and instead set them manually. The testbench writes the output results to tables for imaging in MATLAB.

LFSR

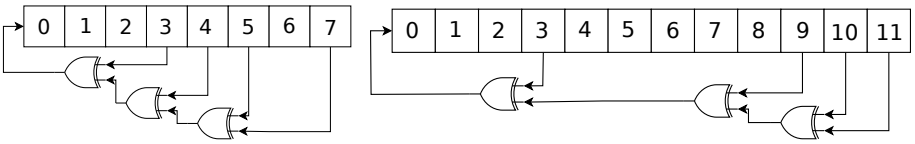
The LFSRs used for randomization was 8 and 12 bits. The taps used for full length was

$$\texttt{8-bit:} \quad x^8 + x^6 + x^5 + x^4 + 1$$

and

$$\texttt{12-bit:} \quad x^{12} + x^{11} + x^{10} + x^4 + 1$$

They are referred to as LFSR8 and LFSR12.



(a) LFSR8 (b) LFSR12

Figure 2.13: Implementation of LFSR with taps

Seeds

A table of 8 seeds is made and used as start values for the different LFSRs. The values are $12-$bit. The first value is 27 and they increase by one for each value.

Noise

For adding noise to the signal two LFSR8 where used; one for adding to the sine and one to subtract from the sine. To reduce logic for avoiding overflow, the subtraction was done first, then the addition is done. The maximum and minimum values of the result is 4095, and 0, respectively, if the noise were to exceed this range, the values are truncated.

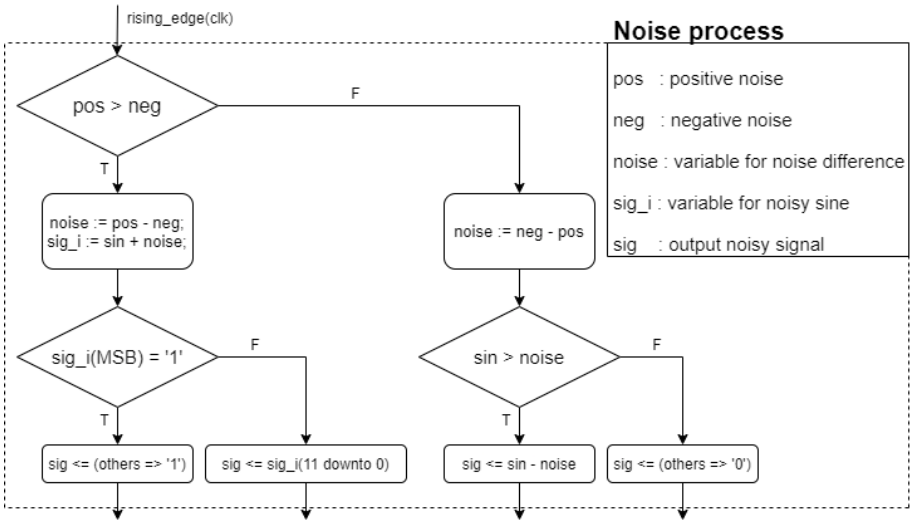


Figure 2.14: Noise process

Phase shift

A LFSR12 is used to randomize the phase shift of the sine, this phase is added to the phase from the Phase Accumulumator and sent to the PAC.

Dampening

After the noise and phase shift, the signal is dampened, to simulate loss of signal. A LFSR12 is used to randomize this value. Subtracting the dampening value from the signal is the final step of the randomization.

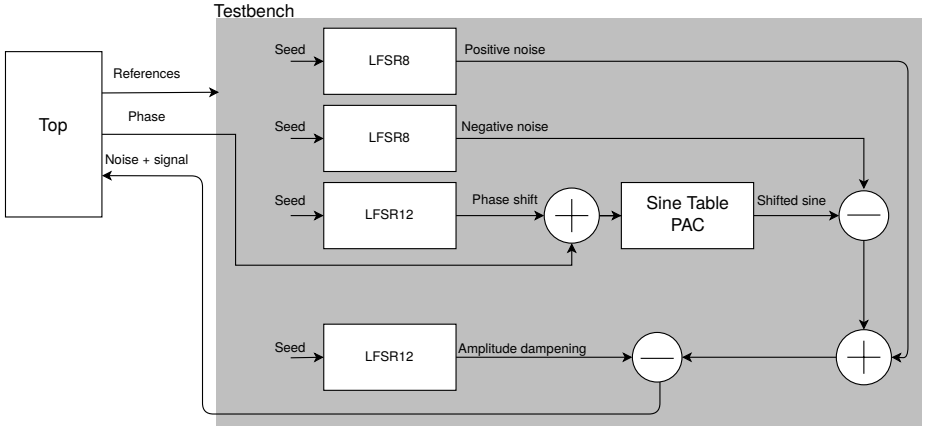


Figure 2.15: Diagram of randomized testbench

2.4.2 Verification in ILA

An ILA IP was used for benchmarking the different characteristics of the system. In order to get the phase offset from a wave is generated to it is processed in the Lock-in Amplifier. The DAC output is sent directly to the ADC and both the reference and input to Lock-in Amplifier is sampled in the ILA. From this the achievable LSB resolution and voltage range of the samples can also be calculated.

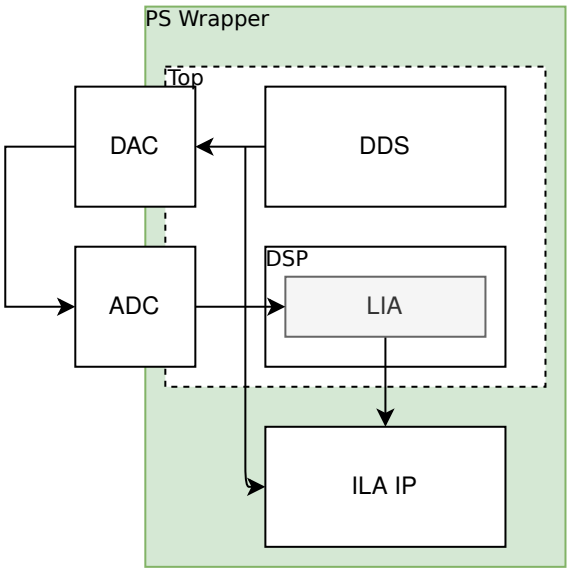


Figure 2.16: Block design of ILA measurements

2.5 Measurement setup

Generated signal from DAC is sent to a resistor in series with the TIA with a gain of

$$|A_v| = \frac{R_f}{R_{in}} = 1997/1949 = 1.0246$$

This is to measure on a more ideal setup, as a step between verification and "real" measurements.

2.5.1 Calibration

For testing and calibrations measurements were made over known resistors. These were at 10 Ω to 3000 Ω This was done by simply applying the voltage from the DAC to a resistor of known resistance and then amplifiying the resulting current in the transimpedance amplifier before the results are sampled in the ADC.

2.6 Post-processing

The different characteristics are written from the FPGA to csv tables. These are read in MATLAB for further processing and imaging. Such processing include scaling and turning fixed point numbers into floating point. Scaling include gain from TIA, and the ratio between LSB of the converters.

$$\frac{3.3}{2.5} = 1.32$$

For generating EIT images, the EIDORS library were used.

2.6.1 EIDORS

To set up the EIT imaging in EIDORS one first need to make a model of the data. This includes shape of the system, number of electrodes, position and stimulation pattern, i.e. which electrodes (channels) are used for stimulation and for measuring. Images are then made by solving this model for the reference data (a homogeneous model) and the stimulation data.

In this thesis two different types of measurements are used, where one is modulated in post, and the other by the analog front-end, from now on referred to as *post modulated* and *analog modulated*, respectively.

The homogenous model of the post modulated measurements is the internal DDS sine reference, and the stimulated results are the ADC sampled sine multiplied with 4 different gains to simulate 4 channels.

For the analog modulated, the homogeneous model was the measurements from TIA with no load, and the stimulated data channels were the different resistors.

Chapter 3

Results

This chapter describes measurements and calculations made with the digital system, along with timing and resource usage.

3.1 Verification and simulation

3.1.1 DDS

To quantify the determinism of the sines generated from the DDS, the phase drift (described in section 1.3.2) was calculated. This was performed by generating continuous sines for the different frequencies, windowing each of the periods, and then finding the sample where the amplitude crosses the swing DC, in this case `Amplitude`$/2 = 2047$.

A drift of 2 samples per 500 kS is measured, which is every 5 ms. At 1 kHz this is every 5th period, and for 10 Hz it is for every 50th period. The drift is

$$\frac{2}{500000} \cdot 360^\circ = 1.44^\circ \cdot 10^{-3}$$

3.1.2 CORDIC

To get the correct DC and phase shift values from the Lock-In Amplifier it was important to make sure the CORDIC behaved properly. The simulated data was compared with MATLAB results of the same input

Table 3.1: Table over CORDIC edge cases

Inputs		**CORDIC outputs**		**MATLAB calculations**	
X	Y	Magnitude	Angle	$\sqrt{X^2 + Y^2}$	Atan2(Y, X)
0	0	0	0°	0	0°
2048	0	2048	0°	2048	0°
1024	0	1024	0°	1024	0°
-256	0	256	180°	256	180°
-127	0	127	180°	127	180°
0	2048	2048	90°	2048	90°
2048	2048	2896	45°	2896.309376	45°
1024	2048	2290	63.476 562 5°	2289.733609	63.4349°
-256	2048	2063	97.140 625°	2063.937984	97.1250°
-127	2048	2051	93.531 25°	2051.933966	93.5485°
0	1024	1024	90°	1024	90°
2048	1024	2290	26.523 437 5°	2289.733609	26.5651°
1024	1024	1448	45°	1448.154688	45°
-256	1024	1057	104.015 625°	1055.51504	104.0362°
-127	1024	1030	97.0625°	1031.845434	97.0699°
0	-256	256	−90°	256	−90°
2048	-256	2065	−7.140 625°	2063.937984	−7.1250°
1024	-256	1055	−14.015 625°	1055.51504	−14.0362°
-256	-256	323	−135°	362.038672	−135°
-127	-256	285	−116.234 375°	285.7708873	−116.3857°
0	-127	127	−90°	127	−90°
2048	-127	2055	−3.570 312 5°	2051.933966	−3.5485°
1024	-127	1032	−7.0625°	1031.845434	−7.0699°
-256	-127	287	−153.804 687 5°	285.7708873	−153.6143°
-127	-127	160	−135°	179.6051224	−135°

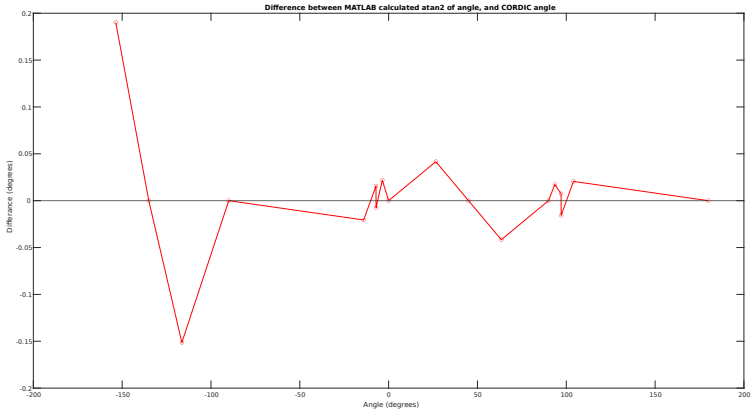


Figure 3.1: The difference between angle calculated with atan2 in MATLAB and with CORDIC

3.1.3 Known parameters

In this subsection signals with known frequency and phase is sent as inputs for the DSP system. This is done in order to verify that the digital system is able to reject the unwanted frequencies without affecting the wanted frequency.
The following tables describe the CORDIC calculated phase shift and magnitude of I/Q signals from the digial LIA in relation to the reference frequency. The magnitude is a purely numerical value in simulations. Frequencies from 1 kHz to 10 kHz are tested.

Table 3.2: 1 kHz sine input

Reference frequency (kHz)	Magnitude	Phase difference
1	259	0.007 812 5°
2	1	−135°
3	1	−135°
4	1	−135°
5	1	−135°

Table 3.3: 2 kHz sine input

Reference frequency (kHz)	Magnitude	Phase difference
2	259	0.007 812 5°
4	1	−135°
6	1	−135°
8	1	−135°
10	1	−135°

Table 3.4: 4 kHz sine input

Reference frequency (kHz)	Magnitude	Phase difference
2	-1	−135°
3	-1	−135°
4	255	88.695 312 5°
5	-1	−135°
6	-1	−135°
8	-1	−135°
10	-1	−135°

Table 3.5: 8 kHz sine input

Reference frequency (kHz)	Magnitude	Phase difference
2	-1	−135°
4	-1	−135°
6	-1	−135°
8	256	17.132 812 5°
10	-1	−135°

Table 3.6: 10 kHz sine input

Reference frequency (kHz)	Magnitude	Phase difference
2	-1	$-135°$
4	-1	$-135°$
6	-1	$-135°$
8	-1	$-135°$
10	261	$158.179\,687\,5°$

Magnitude deviation of known parameters: 2.2989 %

Input signal of 1 kHz phase shifted with 45°

Table 3.7: 1 kHz phase shifted 45° input

Reference frequency (kHz)	Magnitude	Phase difference
1	255	$45°$
Others	1	$-135.898\,437\,5°$

3.1.4 Randomization

In this subsection the result from different randomized parameters is shown, where the testbench is setup according to fig. 2.15.

The LFSR randomized noise repeats after 4096 clock cycles. This noise is added to an input signal before being processed in the LIA.

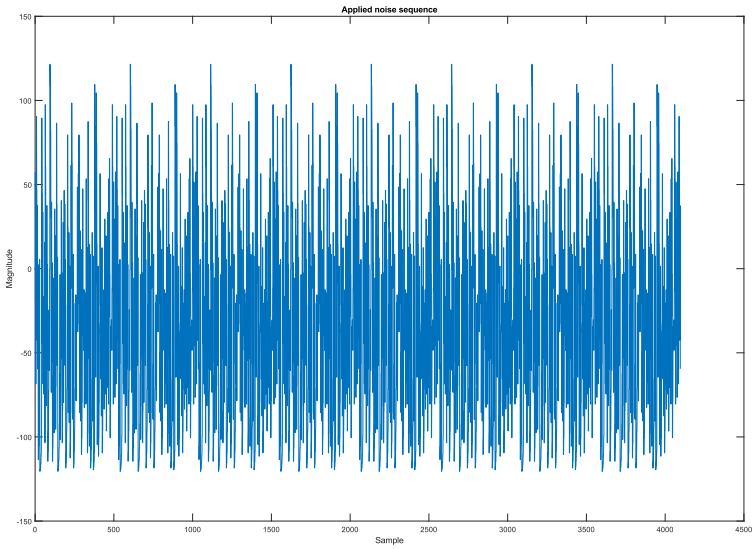


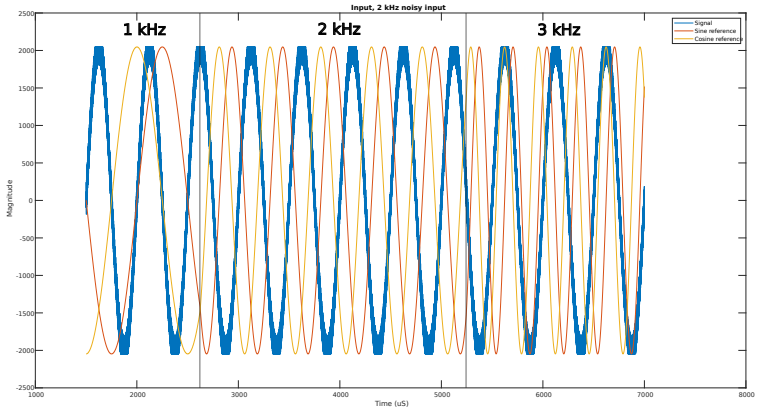
Figure 3.2: Applied pseudo-noise from LFSR

Sine of 2 kHZ with randomized noise

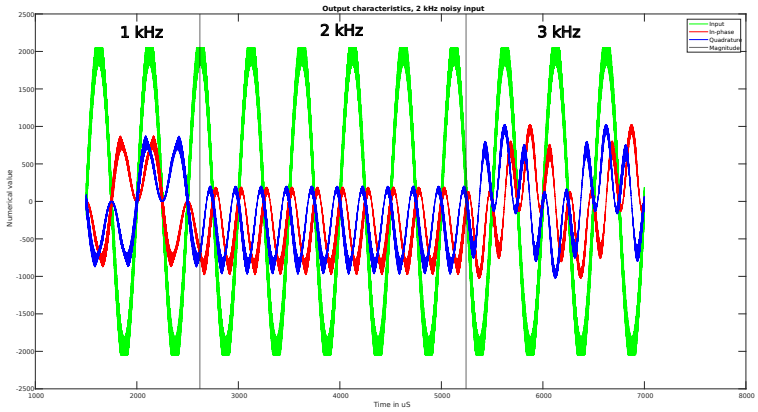
A 2 kHz input sine with applied randomized noise was sent to the DSP block.

Table 3.8: 2 kHz noisy sine input

Reference frequency (kHz)	Magnitude	Phase difference
1	1	$-135°$
2	258	$-135.898\,437\,5°$
3	1	$-135°$
4	1	$-135°$
5	1	$-135°$



(a) 2 kHZ input sine along with reference signals from 1 kHz to 3 kHz. These are the inputs of the LIA



(b) 2 kHZ input sine along with output waves demodulated from 1 kHz to 3 kHz reference. The I/Q waves are the inputs to the averager in the LIA

Figure 3.3: 2 kHz noisy input sine

These figures show a frequency sweep of the reference from 1 kHz to 3 kHz. The input is a 2 kHz sine with applied noise. The first figure show the input in relation to the reference, and the second show the input in relation to the I/Q output products from the demodulation in LIA. When the reference does not have equal frequency to the input, the I/Q signals have a magnitude of 0.

3.1.5 ILA

This subsection discusses the characterization of signals sampled with the ILA. Results are obtained by sampling the output from the DAC directly in the ADC, as shown in fig. 2.16 The frequencies used are 1 kHz, 5 kHz and 10 kHz

3.1.5.1 Period

Checks the period of the reference and synthesised waves, finding the sample where the amplitude crosses the swing DC with the same technique as in section 3.1.1, but for a slightly different purpose, finding the frequency instead of the phase.

Table 3.9: Samples per period of wave, with a sampling rate of 100 MHz

Wave	Samples per period	Actual frequency
Reference (1 kHz)	100000	1000 Hz
Synthesized (1 kHz)	100065	999.35 Hz
Reference (5 kHz)	20000	5000 Hz
Synthesized (5 kHz)	Every other 20055 or 19950	4986.29 Hz to 5012.53 Hz
Reference (10 kHz)	10000	10 000 Hz
Synthesized (10 kHz)	9975 and every fourth is 10080	9920.63 Hz to 10 025.06 Hz

Converters

The ADC samples ground,an inactive channel from the DAC as well outputs of 0 and 4095, where these numerical values represent 0 V and 2.5 V. Because the DAC and ADC have different LSB values (described in section 2.3.5), the sampled values are scaled by 1.32. These measurements are done in order to measure noise floor.

Table 3.10: ADC measurements

Input	Sampled	Scaled sample (1.32)	DC (V)
Ground	0	0	0
DAC: off	0	0	0
DAC: 0	4 or 5	5.28 or 6.6	3.2 m or 4 m
DAC: 4095	3097 or 3098	4088.04 or 4089.36	2.4961 or 2.4970

Voltage range

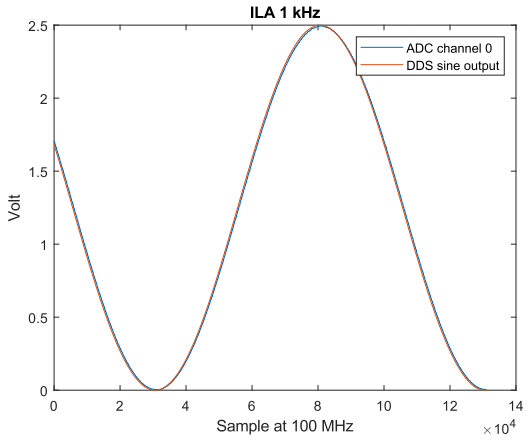
This table shows the lowest and highest voltages of the sampled signal, as well as the root-mean square of them, in order to compare with DC noise measurements.

Table 3.11: Sampled input characteristics

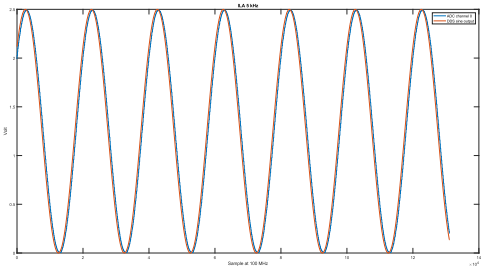
Frequency (kHz)	Min value (V)	Max value (V)	RMS (V)
1	3.2 m	2.4941	0.8807
5	4.8 m	2.4949	0.8804
10	6.4 m	2.4941	0.8795

Sampled signal

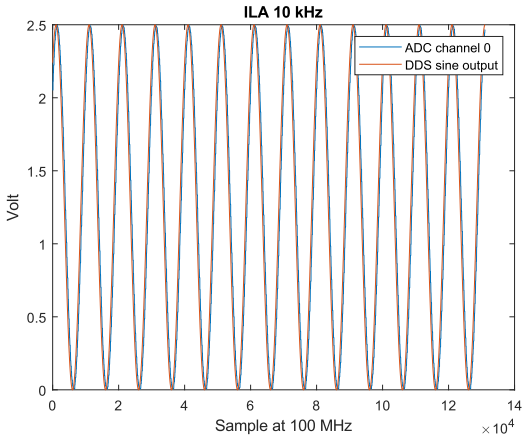
The following figures show the difference between the digital sine reference and the sampled output wave.



(a) 1 kHz



(b) 5 kHz



(c) 10 kHz

Figure 3.4: Digital DDS sine ref, and DAC to ADC sampled reference. Sampled for 1.31 ms

3.2 Measurements

3.2.1 Transimpedance Amplifier

Measuring over the TIA without any load and with 10 Ω, 984 Ω and 3 kΩ, set up as fig. 2.12. The measured signals' phase shift are calculated in CORDIC. Ideal resistive circuits do not apply any phase shift, and the calculated phase will then only be from the system itself.

Table 3.12: Phase shift at different loads

Frequency (kHz))	Phase (°), no load	Phase (°), 10 Ω	Phase (°),984 Ω	Phase (°), 3 kΩ
1	-27.7266	-26.5234	-27.4141	-27.66414
2	-27.4141	-27.414	-27.4141	-27.6641
3	-27.4141	-27.4141	-27.4141	-27.6641
4	-27.6641	-27.6641	-27.6641	-27.3984
5	-27.6641	-27.6641	-27.6641	-27.3984
6	-27.6641	-27.6641	-27.6641	-27.7266
7	-27.7266	-27.7266	-27.7266	-27.7266
8	-27.7266	-27.7266	-27.7266	-27.7266
9	-27.7266	-27.7266	-27.7266	-27.7266
10	-27.7266	-27.7031	-27.7266	-27.7266
Standard deviation (σ)	0.01252	0.3711	0.1222	0.1316
Variance (σ^2)	0.0157	0.1377	0.0149	0.173

3.3 EIDORS

In this section different EIDORS generated images is shown. These are generated from the measured voltage of each channel, and show the resistivity in measurements.

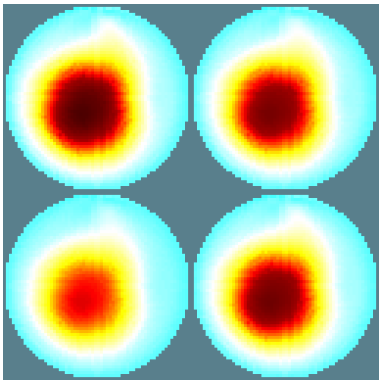
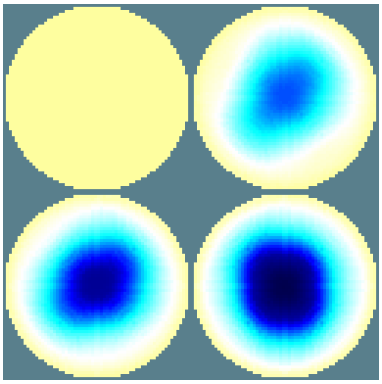
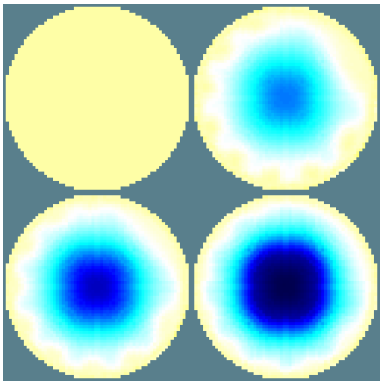


Figure 3.5: EIDORS generated images of the post modulated data

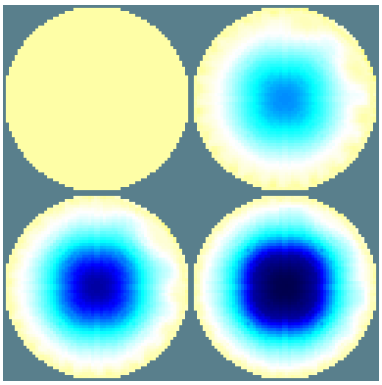
Where the images from top left to bottom right are the waves sampled and multiplied with a gain of 0.5, gain of 0.6, 0.78, 0.58. They are measured in relation to the DDS reference.



(a) 1 kHz



(b) 5 kHz



(c) 10 kHz

Figure 3.6: EIDORS generated images of the analog modulated channels with, from top left to bottom right, no load, 10 ohm, 984 Ω and 3 kΩ at different frequencies

Chapter 4

Discussion

In this chapter the results from the previous chapter is discussed. This includes notable flaws and or advantages with the system as well as comparing it to similar projects. Finally there is a conclusion of the project describing future work and remaining challenges.

4.1 Verification and measurements

4.1.1 DDS

After every 5 ms the phase drifts an insignificant amount. This does not matter both because since the reference are synchronous the relative phase between them will stay the same. Also because it is smaller than the theoretical angular resolution of CORDIC it is then negligible.

$$\frac{7.8125^\circ \cdot 10^{-3}}{1.44^\circ \cdot 10^{-3}} = 5.4253 \tag{4.1a}$$

$$\texttt{time until noticeable drift} = 5\,\text{ms} \cdot 5.4253 = 27.1267\,\text{ms} \tag{4.1b}$$

The phase has to drift over 5 times before the CORDIC is able to notice the difference.
If however, this difference was a problem it could be resolved by resetting the phase between each period. The reason behind this phase drift is probably because of truncation in the `nco`. Small rounding errors eventually accumulates enough to be noticeable.

The sampled frequencies are not accurate with the reference frequency. This error increases along with the frequency, and is likely created by the converters.

The ADC has a throughput of 98 samples at 10 kHz, thus satisfying the anti-aliasing sampling requirement of 12 samples.

4.1.2 CORDIC

For edge cases, both the magnitude and angle is the same as the reference calculations. This is because this values are set directly and not calculated, as

explained in section 2.1.7.

Phase

The CORDIC angle had a theoretical resolution of $1/2^7 = 7.8125^\circ \cdot 10^{-3}$ as explained in eq. (2.2). This resolution is not always feasable, but the resolution is already limited by the converters and LIA so the achieved angular resolution is more than enough for the purpose of this system.

Magnitude

Round-off errors are prevalent in the magnitude calculations. Due to hardware limitations the magnitude is converted from fixed point to float in the FPGA, by a power of 12, thus introducing a lower resolution. A solution for this could be to only right bit-shift the magnitude enough to still be within limitations, a power of 2 is enough. This will greatly increase magnitude resolution.

4.1.3 Known parameters

For the pure sine wave only the exact frequency yields any result. All the other frequencies are averaged away.
It is however interesting to notice that for averaged inphase and quadrature data from the LIA of zero mean waves the results are often -1. This is probably an artifact of the method used for division in the module. Because they are not divided, but multiplied with a gain and right bit-shifted, the most significant bit (MSB) will remain. This is the sign of the number, and the result will then be -1 instead of 0. This affects the angle results by being -135° instead of 0°. Two equal sides of a triangle will always return 45°, when only accounting for the first quadrant, but the CORDIC takes polarity into account and the angle is then calculated as

$$45^\circ - \pi = -135^\circ$$

This also affects the correct frequency. The quadrature side is -1. Because of this the CORDIC will not notice this as an edge case, and will rotate until the quadrature side is 0, where it will then apply the quadrant shift from the MSB of the quadrature mean. This could be fixed with checking if the averaged absolute value is 0, and if this is the case 0 is returned instead of -1.

4.1.4 Noisy simulation

Because the frequencies of the reference are changed sequentially, the input sine is not in phase with the reference sine, but lags behind with about -135°. The reason for this is when the frequencies change this can be in the middle of a phase and the reference will become phase shifted. This would not happen in the system where the applied frequency changes at the same time as the reference frequency, they would then keep the same relation between phase, and is therefore not a problem for the system in practice.
As we can see, the system still extracts the desired frequency even in a noisy environment.

4.1.5 EIDORS

Both EIDORS measurements behave similar with the more impeded measurements showing as darker in the images. Darker color means higher relative difference, i.e. more impedance, than the reference model.
Each of these channels only have one fixed resistance, but the circular model was made to illustrate the potential of EIT. If a system where the impedance was geometry dependent was used the images would be less symmetrical.

4.1.6 Measurements

It seems that $\approx -27°$ is the phase difference from the sine is generated digitally in the DDS to the sampled signal reaches the LIA. This becomes the phase offset, due to converting time and data path in the system.

4.1.7 Analog Front-End

The I/O of the DAC and ADC behaved as expected with correct resolution and range. Noise floor was measured to be between 3.2 mV and 4 mV. This difference is due to the ADC having an LSB of 806 µV, while the DAC has 610 µV. LSB_{DAC} is different than LSB_{ADC} meaning a quantization error will occur[1].
The noise floor increases with higher frequency, where at 5 kHz it is multiplied with 1.5 and at 10 kHz it is doubled.

The OpAmp however was unstable and should probably have a decoupling capacitor [24]. This was noticed when probed signals did not satisfy Ohm's law [2]. There was not enough time to fix this due to corona restrictions, thus not allowing for all wanted measurements, such as measuring phase shift from a capacitive circuit.
It is preferable with a controllable current source instead of a TIA, but unfortunately this circuit was not finished in time for testing.

Instead of a fixed gain, an implementation of programmable gain could be used. Variable gain would allow the system to measure on a larger range of impedances, as small and large signals require different gains to not be buried in noise and also avoid saturating either the OpAmp or the ADC.

Programmable gain can be implemented in many different ways, but the design philosophy is to dynamically choose which resistor is to be used as feedback. One of the simpler way to to this is to have a network of different feedback resistors connected to the input with switches. A way to incorporate both the FPGA and MUXes would be to instead of switches connect the resistors to the different channels, and choose which one is used with control signals from the FPGA. They could be shifted through either manually, or by detecting the

[1] Quantization error is rounding error which happens in analog to digital conversion when the analog value is between LSB of the converter. The value of the quantization error is the difference between analog value and LSB.
[2] U = RI

sampled amplitude from the ADC. If the sampled signal is either too high or low, the gain is chosen accordingly.

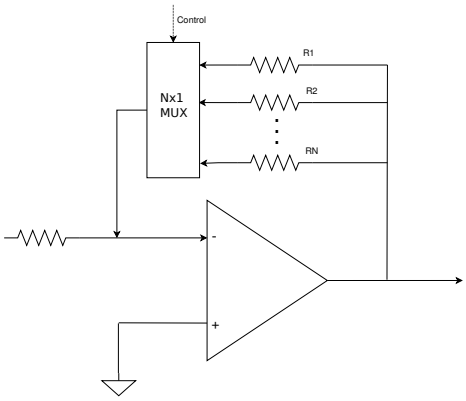


Figure 4.1: Programmable gain implemented with a MUX

4.2 Comparison

This section discusses similarities and differences between this and similar projects.

One study implemented a LIA system with frequency detection. The detected frequency was sent through a DDS, generating square wave references. This was done in order to prevent amplitude drift of the reference, and thus reducing the precision of the LIA [25]. Because of harmonics the output noise is increased with amplitude drift. Another difference was the averaging of signals. A digital FIR filter for low-pass filtering was used.

Another study implemented a LIA in FPGA in order to make a low-cost portable system for measuring dielectric properties of cells [26]. The NCO generates a frequency of 810 kHz for stimulating the cells. It accomplished this by using a 14-bit NCO as frequency generator, both for the LIA multiplication and as the input for synthetization in DAC. This NCO runs at 65 MHz, as opposed to the 50 MHz in this thesis, and is decimated by a factor of 6500, down to 10 KSPS, before being filtered in a FIR filter.

Table 4.1: Different filtering techniques used after LIA

Paper	Filtering
This thesis	Averager
[25]	FIR low-pass
[26]	Decimation and FIR

The reason for using an Averager in this thesis as opposed to a FIR filter, was because the magnitude and phase shift were the interesting parameters for the purpose of this thesis.

4.3 Challenges

During the measurements, a lot of time was allocated to debugging parts of the analog front-end. Having the analog front-end characterized, and measured as a standalone block before integration with the rest of the system would be beneficial. Ideally, the analog front-end should've been a complete standalone printed circuit board (PCB), verified as a single block. Alternatively, the analog front-end could've been acquired in full through the usage of commercial off-the shelf (COTS) components, as opposed to using standalone blocks.

While a breadboard is trivial to prototype on, there's several disadvantages to it. Especially with regards to ground- and power-distribution, as well as noise performance. Making the analog front end, as part of a custom PCB would most likely achieve better performance all around, as well as making the entire testbench setup a lot less cluttered.

These difficulties hindered the characterization of the implemented design, and should probably have been tackled a lot earlier in the design process.

4.4 Future work

In this section different improvements and steps forward for the system is discussed.

First of all a functioning analog front-end needs to be used with the system for proper measurements and characterization. Preferably with the possibility of applying a programmable current. The next step would be to measure on tubs of saline solutions with different electrode channels. To implement a changing impedance, a plastic object could be placed in the water and moved around, as shown in fig. 2.11 The ultimate goal would be to eventually be able to generate EIT images from living tissue.

The converters have lots of room for higher frequency generation and measurements. With a higher frequency range, more types of materials can be measured on, and different behavior of tissue will also be observed. This does however demand a front-end which can handle these frequencies without stray capacitance becoming a problem.

For the digital part of the design, it could be improved with dynamic frequency averaging, meaning changing the summation period for higher frequencies. Further characterization, both simulated and analogue, of the system also needs to be done, making it easier to compare to other solutions and deciding further improvements of the design. DDS has been characterized in regards to phase and frequency, but only with the PMOD DAC and ADC, meaning either of the converters could introduce frequency errors. CORDIC performs calculations as expected and LIA extracts the wanted frequency with 1 kHz steps and in noisy environments.

For further characterization of the LIA a more noise environment is wanted in order to test the bandwidth, which should hopefully be around 0.1 Hz.

4.5 Conclusion

In this thesis a FPGA-based system for digital synthethication of analog waveforms of different frequencies, as well as recovery the magnitude and phase shift of modulated signals in the digital domain has been developed. The digital system can perform electrode and frequency sweeps in the range of 1 kHz to 10 kHz with reliable phase and high temporal resolution.
Phase drift is accounted for by phase being relative to the reference, and not absolute.
Temporal resolution ranges from 980 to 98 at 1 kHz to 10 kHz.

The system can also control analog muxes for multi-probe measurements. All measurements and results are written to tables for viewing and/or further processing.

As other papers also have shown, FPGA lends itself well to these types of continuous operations, making it an excellent choice for bioimmitance measurements, such as EIT.

Bibliography

[1] Sverre Grimnes, Orjan G. Martinsen *Bioimpedance and Bioelectricity Basics, Third Edition*
Academic Press, 2015

[2] David S Holder *Electrical Impedance Tomography: Methods, History and Applications*
Taylor Francis, 2004

[3] FPGA Advantages and Most Common Applications[website] Retrieced 13th June, 2021m from
`https://hardwarebee.com/fpga-advantages-common-applications-today/`

[4] Robert T. Paynter, B.J. Toby Boydell *Electronics Technology Fundamentals: Electron Flow Version, Third Edition*
Pearson Education, 2009

[5] Maltron International Ltd: New EIT Under Development Electrical Impedance Tomography (EIT) Maltron Sheffield MK 3.5[website] Retrieved May 2nd, 2021
`https://maltronint.com/electrical-impedance-tomography/`

[6] EIDORS: Electrical Impedance Tomography and Diffuse Optical Tomography Reconstruction Software [website] Retrieved June 13th, 2021
`http://eidors3d.sourceforge.net/docs.shtml`

[7] Wilhelm Burger Mark J. Burge *Principles of Digital Image Processing: Fundamental Techniques*
Springer London, 2011

[8] Motorola, Inc.: SPI Block Guide
`[pdf]https://web.archive.org/web/20150413003534/`
`http://www.ee.nmt.edu/ teare/ee308l/datasheets/S12SPIV3.pdf`
(2003)

[9] Omegatron (5 December 2018): Digital timing diagram
`https://www.wikiwand.com/en/Digital_timing_diagram`

[10] Scott Campbell: Basics of the I2C Communication Protocol[website] Retrieved May 5th, 2021, from
`https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/`

[11] Scott Campbell: Basics of UART Communication
`[pdf]https://cs140e.sergio.bz/notes/lec4/uart-basics.pdf`

[12] Peter J. Ashenden *The Designer's Guide to VHDL, Third Edition*
Morgan Kaufmann, 2008

[13] Jim Surber and Leo McHugh: Single-Chip Direct Digital Synthesis vs. the Analog PLL
[pdf]https://www.analog.com/media/en/analog-dialogue/volume-30/number-3/articles/volume30-number3.pdfpage=12 (1996)

[14] CORDIC For Dummies
[pdf]https://www.eit.lth.se/fileadmin/eit/courses/eitf35/2017/CORDIC_For_Dummies.pdf

[15] Stanford Research Systems: About Lock-In Amplifiers
[pdf]https://www.thinksrs.com/downloads/pdfs/applicationnotes/AboutLIAs.pdf

[16] Behzad Ravazi *RF Microelectronics, Second Edition*
Prentice Hall, 2012

[17] Solomon Wolf Golomb *Shift register sequences*
Holden-Day, Inc. 1967

[18] Xilinx: Integrated Logic Analyzer v6.1
[pdf]https://www.xilinx.com/support/documentation/ip_documentation/ila/v6_1/pg172-ila.pdf
(2016)

[19] Tony Chan Carusone, David Johns, Kenneth Martin *Analog Integrated Circuit Design: International Student Version, Second edition*
John Wiley and Sons, 2013

[20] Texas Instruments: SN74CB3Q3251 1-OF-8 FET MULTIPLEXER/-DEMULTIPLEXER 2.5-V/3.3-V LOW-VOLTAGE HIGH-BANDWIDTH BUS SWITCH
[pdf]https://www.ti.com/lit/ds/symlink/sn74cb3q3251.pdf?HQS=dis-dk-null-digikeymode-dsf-
(2005)

[21] Mindseye Biomedical: Spectra Bioimpedance and EIT Complete Kit
https://mindseyebiomedical.com/products/spectra-deluxe-kit
(2021)

[22] Analog Devices, Inc.: Octal, 12-/14-/16-Bit, SPI Voltage Output denseDAC with 5 ppm/°C, On-Chip Reference
[pdf]https://www.analog.com/media/en/technical-documentation/data-sheets/AD5628_5648_5668

[23] Analog Devices, Inc.: 2.35 V to 5.25 V, 1 MSPS,12-/10-/8-Bit ADCs in 6-Lead SC70
[pdf]https://www.analog.com/media/cn/technical-documentation/evaluation-documentation/AD

[24] Henry Ott Consultants (8 January, 2007)
http://www.hottconsultants.com/techtips/decoupling.html

[25] Cheng Zhang, Huan Liu, Jian Ge Haobin Dong: FPGA-Based Digital Lock-in Amplifier With High-Precision Automatic Frequency Tracking
[pdf]https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=arnumber=9129659,
(2020)

[26] D. Divakar, K. Mahesh, M. M. Varma P. Sen (2018) FPGA-Based Lock-In Amplifier for Measuring the Electrical Properties of Individual Cells. *2018 IEEE 13th Annual International Conference on Nano/Micro Engineered and Molecular Systems (NEMS)*, pp. 1-5, doi: 10.1109/NEMS.2018.8556987

Appendix A

Diagrams

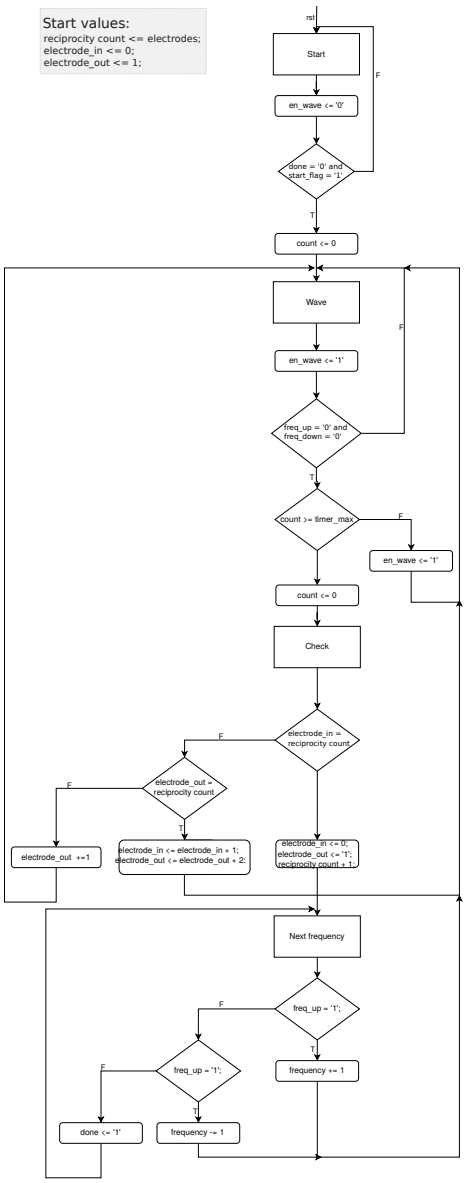


Figure A.1: Electrode Master ASM

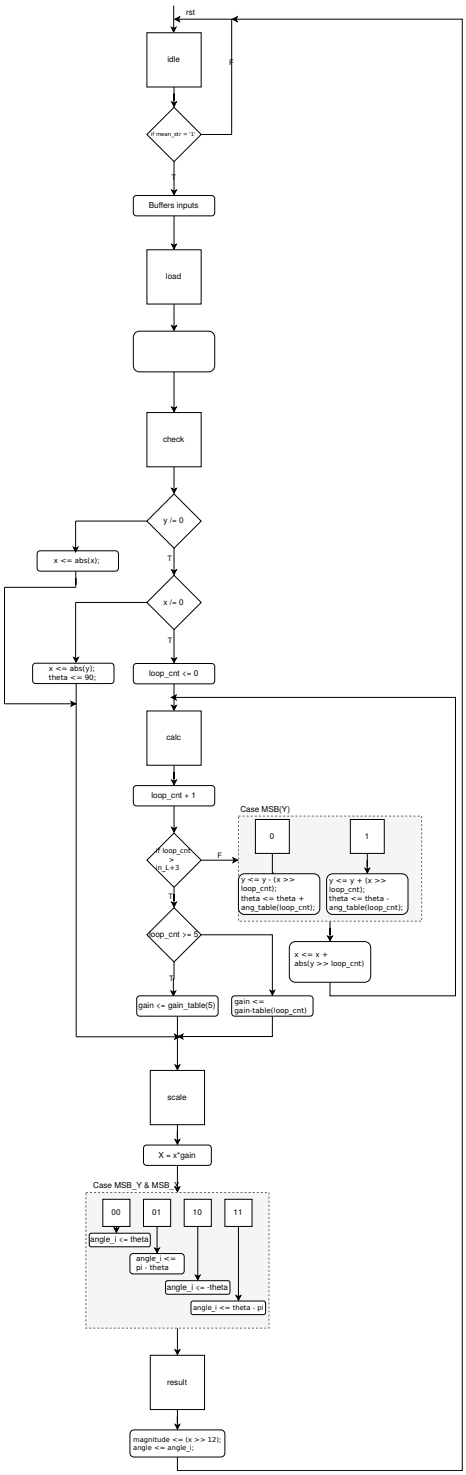


Figure A.2: CORDIC ASM

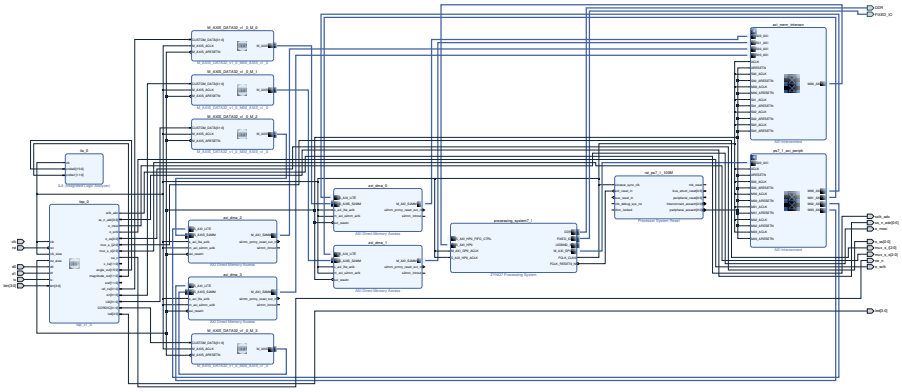


Figure A.3: Vivado Block Diagram of system

Appendix B

Simulations

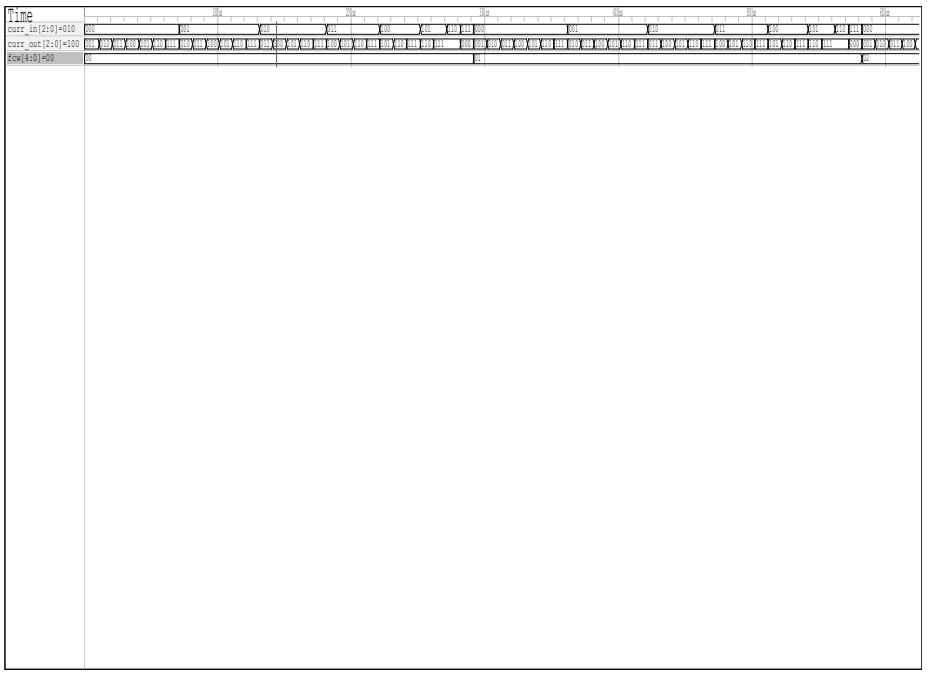


Figure B.1: Waveform of electrode and frequency loop in `electrode_master`

Appendix C

VHDL code

Source Code C.1: Connection between pmod adc and system

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use work.array_pkg.all;
5
6  entity adc_imp is
7    port (
8        clk        : IN      STD_LOGIC;                        --system clock
9        reset_n    : IN      STD_LOGIC;                        --active low
         ↪  asynchronous reset
10
11       -- Data in
12       d0   :  IN      STD_LOGIC;                       --channel 0 serial data
         ↪  from ADC
13       d1   :  IN      STD_LOGIC;                       --channel 1 serial data
         ↪  from ADC
14
15       -- SPI
16       sclk_o       : out  STD_LOGIC;                       --SPI bus from ADC:
         ↪  serial clock (SCLK)
17       ss_n_o       : out  STD_LOGIC_VECTOR(0 DOWNTO 0);    --SPI bus from ADC:
         ↪  slave select (~SYNC)
18
19       -- Sampled data
20       electrode0   : out std_logic_vector(vector_length-1 downto 0);
21       electrode1   : out std_logic_vector(vector_length-1 downto 0)
22   );
23 end entity;
24
25 architecture rtl of adc_imp is
26
27   signal busy_i : std_logic;
28
29   -- FIFO
30   signal full_i0, full_i1, empty_i0, empty_i1   : std_logic; -- Status signals
31   signal adc_rw_str : std_logic; -- Read/write strobe
32   signal adc_0_data_i, adc_1_data_i, adc_0_data_id, adc_1_data_id  :
     ↪  std_logic_vector(vector_length-1 downto 0); -- Data
33
34   signal cnt : unsigned(4 downto 0);
35
36   signal sclk_o_i  : std_logic;
```

```
37    signal ss_n_o_i  : std_logic_vector(0 downto 0);
38    alias  write_bit : std_logic is ss_n_o_i(0);
39    signal adc_read0, adc_read1 : std_logic;
40
41    component pmod_adc_ad7476a IS
42      GENERIC(
43        clk_freq    :  INTEGER := 100; --system clock frequency in MHz
44        spi_clk_div :  INTEGER := 3);  --spi_clk_div = clk_freq/40 (answer rounded
          ↪ up)
45      PORT(
46        clk         :  IN      STD_LOGIC;                      --system clock
47        reset_n     :  IN      STD_LOGIC;                      --active low reset
48        data_in_0   :  IN      STD_LOGIC;                      --channel 0 serial
          ↪ data from ADC
49        data_in_1   :  IN      STD_LOGIC;                      --channel 1 serial
          ↪ data from ADC
50        sck         :  BUFFER  STD_LOGIC;                      --serial clock
51        cs_n        :  BUFFER  STD_LOGIC_VECTOR(0 DOWNTO 0);   --chip select
52        adc_0_data  :  OUT     STD_LOGIC_VECTOR(11 DOWNTO 0);  --channel 0 ADC
          ↪ result
53        adc_1_data  :  OUT     STD_LOGIC_VECTOR(11 DOWNTO 0)); --channel 1 ADC
          ↪ result
54    END component pmod_adc_ad7476a;
55
56    component fifo is
57      generic (
58        inout_length : integer := vector_length-1;
59        new_array_length : integer := array_length-1
60      );
61      port (
62        clk      :  in std_logic;
63        rst      :  in std_logic;
64        -- Read/write
65        wr_en    :  in std_logic;
66        rd_en    :  in std_logic;
67        full     : out std_logic;
68        empty    : out std_logic;
69        -- Data
70        sin_wr   :  in std_logic_vector(vector_length-1 downto 0);
71        sin_rd   : out std_logic_vector(vector_length-1 downto 0)
72      );
73    end component fifo;
74
75  begin
76
77    FIFO_electrode0: fifo
78    port map (
79      clk              => clk,
80      rst              => not(reset_n),
81      wr_en            => adc_rw_str,
82      rd_en            => adc_read0,--not(empty_i0),--adc_rw_str,
83      full             => full_i0,
84      empty            => empty_i0,
85      sin_wr           => adc_0_data_i,
86      sin_rd           => adc_0_data_id
87    );
88
89    FIFO_electrode1: fifo
90    port map (
91      clk              => clk,
92      rst              => not(reset_n),
93      wr_en            => adc_rw_str,
```

```
94      rd_en            => adc_read1,--not(empty_i1),--adc_rw_str,
95      full             => full_i1,
96      empty            => empty_i1,
97      sin_wr           => adc_1_data_i,
98      sin_rd           => adc_1_data_id
99    );
100
101
102   ADC: pmod_adc_ad7476a
103   port map (
104       clk         => clk,
105       reset_n     => reset_n,
106       data_in_0   => d0,
107       data_in_1   => d1,
108       sck         => sclk_o,
109       cs_n        => ss_n_o_i,
110       adc_0_data  => electrode0,--adc_0_data_i,
111       adc_1_data  => electrode1--adc_1_data_i
112   );
113
114   --sclk_o <= sclk_o_i;
115   ss_n_o <= ss_n_o_i;
116   --adc_read0 <= not(empty_i0);
117   --adc_read1 <= not(empty_i1);
118
119   process(clk)
120   begin
121     if rising_edge(clk) then
122       adc_rw_str <= write_bit;
123       adc_read0   <= adc_rw_str;
124       adc_read1   <= adc_rw_str;
125       --adc_0_data_id <= adc_0_data_i;
126       --adc_1_data_id <= adc_0_data_i;
127       --if ((adc_0_data_i /= adc_0_data_id) or (adc_1_data_i /= adc_1_data_id))
          ↪  then
128       --  adc_rw_str <= '1';
129         --cnt <= (others => '0');
130       --end if;
131     end if;
132   end process;
133
134 end architecture rtl;
```

Source Code C.2: Array package

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  package array_pkg is
6    constant array_length  : integer := 4096;
7    --constant    ds_length  : integer := 8192; -- /2
8    --constant    us_length  : integer := 24573; -- 3/2
9    constant vector_length : integer := 12;
10   --constant ds_factor     : integer := 2;
11   --constant us_factor     : integer := 3;
12   constant electrodes    : integer := 8;
13   constant measurements  : integer := 28;
14
15   --type t_wave   is array  (0 to array_length-1)     of
     ↪  std_logic_vector(vector_length-1 downto 0);
16
17   --type ds_wave  is array  (0 to ds_length-1)        of
     ↪  std_logic_vector(vector_length-1 downto 0);
18   --type us_wave  is array  (0 to us_length-1)        of
     ↪  std_logic_vector(2*vector_length-1 downto 0);
19
20   --type lia_wave is array  (0 to ds_length-1)        of
     ↪  std_logic_vector(2*vector_length-1 downto 0);
21
22   type adc_array    is array (0 to 2-1)  of std_logic_vector(vector_length-1
     ↪  downto 0);
23   type dsp_array    is array(0 to electrodes-1)   of
     ↪  std_logic_vector(vector_length-1 downto 0);
24   type eidors_array is array(0 to electrodes-1)   of
     ↪  std_logic_vector(2*vector_length-1 downto 0);
25 end array_pkg;
```

Source Code C.3: CORDIC

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
--use work.array_pkg.all;

entity cordic is
  generic (
    in_L     : integer
  );
  port (
    clk       :  in std_logic;
    rst       :  in std_logic;
    mean_str  :  in std_logic;
    Y_in      :  in signed(in_L+1 downto 0);
    X_in      :  in signed(in_L+1 downto 0);
    magnitude : out std_logic_vector(in_L downto 0);
    angle     : out std_logic_vector(in_L+4 downto 0)
  );
end entity cordic;

architecture rtl of cordic is

  --type t_angle_table is array(0 to in_L-1) of unsigned(2*vector_length-2 downto
  ↪  0); -- 24b res/rea
  type t_angle_table is array(0 to in_L+2) of signed(in_L+4 downto 0);
  constant ang_table   : t_angle_table  :=
         --("10110100000000000000000", "01011010000000000000000",
         ↪  "00101101000000000000000",
         -- "00010110100000000000000", "00001011010000000000000",
         ↪  "00000101101000000000000",
         -- "00000010110100000000000", "00000001011010000000000",
         ↪  "00000000101101000000000",
         -- "00000000010110100000000", "00000000001011010000000",
         ↪  "00000000000101101000000",
         -- "00000000000010110100000", "00000000000001011010000",
         ↪  "00000000000000101101000",
         -- "00000000000000010110100", "00000000000000001011010",
         ↪  "00000000000000000101101",
         -- "00000000000000000010110", "00000000000000000001011",
         ↪  "00000000000000000000101",
         -- "00000000000000000000010", "00000000000000000000001"); --sfixed =
         ↪  131072
        --("10110100000", "01011010000", "00101101000",
        --  "00010110100", "00001011010", "00000101101",
        --  "00000010110", "00000001011", "00000000101",
        --  "00000000010", "00000000001"); --sfixed = 2^5 = 32

        -- 14 bits => 2^8
       ("0001011010100000",--0000000000",
        "0000110101001000",--000000000",
        "0000011100000100",--000000000",
        "0000001110010000",--000000000",
        "0000000111001001",--000000000",
        "0000000011100101",--000000000",
        "0000000001110010",
        "0000000000111001",
        "0000000000011100",
        "0000000000001100",
        "0000000000000111",
```

```
51          "0000000000000011",
52          "0000000000000001",
53          "0000000000000001");
54        --("10110100000000000000", "01011010000000000000",
55        --"00101101000000000000", "00010110100000000000",
          ↪  "00001011010000000000",
56        --"00000101101000000000", "00000010110100000000",
          ↪  "00000001011010000000",
57        --"00000000101101000000", "00000000010110100000",
          ↪  "00000000001011010000",
58        --"00000000000101101000", "00000000000010110100",
          ↪  "00000000000001011010",
59        --"00000000000000101101", "00000000000000010111",
          ↪  "00000000000000001100",
60        --"00000000000000000110", "00000000000000000011",
          ↪  "00000000000000000001"); --sfixed = 65536
61
62    type t_gain is array(0 to 5) of unsigned(12 downto 0);
63    constant gain_table : t_gain :=
64      -- 2^12
65      (
66      "0101101010000", -- 0
67      "0101000011110", -- 1
68      "0100111010001", -- 2
69      "0100110111101", -- 3
70      "0100110111001", -- 4
71      "0100110110111"); --5
72
73    constant pi : signed(in_L+4 downto 0) := "0101101000000000";
74    constant zero_c : signed(in_L+4 downto 0) := "0000000000000000";
75    signal   loop_cnt    : integer range -1 to in_L+4;
76    signal   x_new       : unsigned(in_L+1 downto 0);
77    signal   X           : unsigned(in_L+12+2+1 downto 0);
78    signal   X_sign      :   signed(in_L+1+2 downto 0);
79    signal   uns_x : unsigned(in_L+3 downto 0);
80    signal   y_new, Y    :   signed(in_L+1+2 downto 0);
81    signal   theta       : signed(in_L+4 downto 0);
82    signal   done        : std_logic;
83    signal   theta_rst     : std_logic;
84
85    signal angle_i, angle_sign       : signed(in_L+4 downto 0);
86    signal mag_i         : unsigned(in_L downto 0);
87
88    signal mean_str_i    : std_logic;
89    signal gain          : unsigned(12 downto 0);
90
91    signal X_msb, Y_msb  : std_logic;
92    signal sign_state : std_logic_vector(1 downto 0);
93
94    type state_t is (idle, load, check, calc, scale, result);
95    signal state : state_t;
96
97    -- Internal input signals
98    signal x_in_i, y_in_i : signed(in_L+1 downto 0);
99
100 begin
101
102   P_CORDIC: process(clk, rst)
103   variable uns_y : unsigned(in_L+3 downto 0);
104   --variable uns_x : unsigned(in_L+3 downto 0);
105
```

```
106     begin
107       if rst = '1' then
108         theta <= (others => '0');
109         loop_cnt <= -1;
110         theta_rst <= '0';
111         x_new <= (others => '0');
112         y_new <= (others => '0');
113         x <= (others => '0');
114         y <= (others => '0');
115         x_sign <= (others => '0');
116         mean_str_i <= '0';
117 
118         X_msb <= '0';
119         Y_msb <= '0';
120 
121         angle_i    <= (others => '0');
122         angle_sign <= (others => '0');
123         sign_state <= "00";
124         gain   <= "1000000000000";
125 
126         x_in_i <= (others => '0');
127         y_in_i <= (others => '0');
128         uns_x  <= (others => '0');
129 
130       elsif rising_edge(clk) then
131         done <= '0';
132         sign_state <= Y_msb & X_msb;
133         --loop_cnt <= loop_cnt + 1;
134         mean_str_i <= mean_str;
135 
136         case state is
137 
138           when idle => -- Waiting for start strobe from Lock-In Amplifier
139             if mean_str_i = '1' then
140               x_in_i <= X_in;
141               y_in_i <= Y_in;
142               state <= load;
143             else
144               loop_cnt <= -1;
145             end if;
146 
147           when load => -- Loading inputs
148             X_msb  <= x_in_i(in_L+1);
149             Y_msb  <= y_in_i(in_L+1);
150             x_sign <= ("00" & abs(x_in_i));
151             y      <= ("00" & abs(y_in_i));
152             theta  <= (others => '0');
153             gain   <= "1000000000000";
154 
155             state  <= check;
156 
157           when check => -- Checking corner cases
158             if to_integer(y) = 0 then
159               uns_X <= unsigned(abs(x_sign));
160               state <= scale;
161             elsif to_integer(x_sign) = 0 then
162               uns_X <= unsigned(abs(y));
163               theta <= ("0010110100000000");
164               state <= scale;
165             else
166               loop_cnt <= 0;
167               state    <= calc;
```

```
168             end if;
169
170           when calc => -- Performing rotations
171             loop_cnt <= loop_cnt + 1;
172
173             if (loop_cnt >= in_L+3) or (to_integer(Y) = 0) then
174               if (loop_cnt >= 5) then
175                 gain <= gain_table(5);
176               else
177                 gain <= gain_table(loop_cnt);
178               end if;
179               uns_X <= unsigned(abs(x_sign));
180               loop_cnt <= -1;
181               state <= scale;
182             else
183               case Y(in_L+3) is--Y'high) is
184                 when '0' =>
185                   y <= y - x_sign(in_L+3 downto loop_cnt);
186                   theta <= theta + ang_table(loop_cnt);
187                 when '1' =>
188                   y <= y + x_sign(in_L+3 downto loop_cnt);
189                   theta <= theta - ang_table(loop_cnt);
190                 when others =>
191               end case;
192               x_sign <= x_sign + abs(y(in_L+3 downto loop_cnt));
193             end if;
194
195           when scale => -- Scaling.
196             done <= '1';
197             -- Magnitude gain
198             X <= (uns_x(in_L+2 downto 0)*gain);
199             -- Quadrant shift
200             case sign_state is
201               when "00" =>
202                 angle_i <= (theta);           -- Q1
203               when "01" =>
204                 angle_i <= (pi - theta);      -- Q2
205               when "10" =>
206                 angle_i <= (zero_c - theta);  -- Q4
207               when "11" =>
208                 angle_i <= (theta - pi);      -- Q3
209             when others =>
210             end case;
211
212             state <= result;
213
214           when result => -- Magnitude and angle results
215             magnitude <= std_logic_vector(X(in_L+12 downto 12));
216             angle     <= std_logic_vector(angle_i);
217             state     <= idle;
218
219           when others =>
220             done  <= '0';
221             state <= idle;
222           end case;
223       end if;
224     end process;
225
226   end architecture rtl;
```

Source Code C.4: Connection between pmod dac and system

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use work.array_pkg.all;
5
6  entity dac_imp is
7    port (
8        clk        : IN      STD_LOGIC;                      --system clock
9        reset_n    : IN      STD_LOGIC;                      --active low
         ↪ asynchronous reset
10       dac_tx_ena : IN      STD_LOGIC;                      --enable transaction
         ↪ with DAC
11       addr       : in std_logic_vector(3 downto 0);
12       data       : in std_logic_vector(11 downto 0);
13       --busy_o       : OUT     STD_LOGIC;                     --indicates when
         ↪ transactions with DAC can be initiated
14       mosi_o       : OUT     STD_LOGIC;                     --SPI bus to DAC:
         ↪ master out, slave in (DIN)
15       sclk_o       : out  STD_LOGIC;                      --SPI bus to DAC:
         ↪ serial clock (SCLK)
16       ss_n_o       : out  STD_LOGIC_VECTOR(0 DOWNTO 0)    --SPI bus to DAC: slave
         ↪ select (~SYNC)
17   );
18 end entity;
19
20 architecture rtl of dac_imp is
21   signal cmd : std_logic_vector(3 downto 0);
22   --signal addr : std_logic_vector(3 downto 0);
23   --signal data : unsigned(vector_length-1 downto 0);
24   signal busy_i : std_logic;
25   --signal cos_i : std_logic_vector(vector_length-1 downto 0);
26   --signal sin_addr : unsigned(vector_length-1 downto 0);
27   --signal pause : unsigned(5 downto 0);
28   signal full_i   : std_logic;
29   signal empty_i  : std_logic;
30   signal data_rd  : std_logic_vector(vector_length-1 downto 0);
31
32   component pmod_dac_ad5628 IS
33     GENERIC(
34       clk_freq    : INTEGER := 50;  --system clock frequency in MHz
35       spi_clk_div : INTEGER := 1);  --spi_clk_div = clk_freq/100 (answer rounded
         ↪ up)
36     PORT(
37       clk        : IN      STD_LOGIC;                      --system clock
38       reset_n    : IN      STD_LOGIC;                      --active low
         ↪ asynchronous reset
39       dac_tx_ena : IN      STD_LOGIC;                      --enable transaction
         ↪ with DAC
40       dac_cmd    : IN      STD_LOGIC_VECTOR(3 DOWNTO 0);   --command to send to
         ↪ DAC
41       dac_addr   : IN      STD_LOGIC_VECTOR(3 DOWNTO 0);   --address to send to
         ↪ DAC
42       dac_data   : IN      STD_LOGIC_VECTOR(11 DOWNTO 0);  --data value to send
         ↪ to DAC
43       busy       : OUT     STD_LOGIC;                      --indicates when
         ↪ transactions with DAC can be initiated
44       mosi       : OUT     STD_LOGIC;                      --SPI bus to DAC:
         ↪ master out, slave in (DIN)
```

```
45        sclk       : BUFFER  STD_LOGIC;                       --SPI bus to DAC:
          ↪  serial clock (SCLK)
46        ss_n       : BUFFER  STD_LOGIC_VECTOR(0 DOWNTO 0));  --SPI bus to DAC:
          ↪  slave select (~SYNC)
47    END component pmod_dac_ad5628;
48
49
50    component fifo is
51      generic (
52        inout_length : integer := vector_length-1;
53        new_array_length : integer := 4*array_length--1262143
54      );
55      port (
56        clk      :  in std_logic;
57        rst      :  in std_logic;
58        -- Read/write
59        wr_en    :  in std_logic;
60        rd_en    :  in std_logic;
61        full     : out std_logic;
62        empty    : out std_logic;
63        -- Data
64        sin_wr   :  in std_logic_vector(vector_length-1 downto 0);
65        sin_rd   : out std_logic_vector(vector_length-1 downto 0)
66      );
67    end component fifo;
68
69  begin
70
71    cmd  <= "0011";
72    --addr <= "0000";--(others => '0');
73
74
75    --FIFO_sin: fifo
76    --port map (
77    --  clk              => clk,
78    --  rst              => not(reset_n),
79    -- wr_en            => en_wave,
80    --  rd_en            => not(busy_i),
81    --  full             => full_i,
82    --  empty            => empty_i,
83    --  sin_wr           => data,
84    --  sin_rd           => data_rd
85    --);
86
87
88    DAC: pmod_dac_ad5628
89    port map (
90      clk              => clk,
91      reset_n          => reset_n,
92      dac_tx_ena       => dac_tx_ena,
93      dac_cmd          => cmd,
94      dac_addr         => addr,
95      dac_data         => data,
96      busy             => busy_i,
97      mosi             => mosi_o,
98      sclk             => sclk_o,
99      ss_n             => ss_n_o
100   );
101
102 end architecture rtl;
```

Source Code C.5: DDS

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use work.array_pkg.all;
5  use work.sine_wave_pkg.all;
6  use work.timer_pkg.all;
7
8  entity dds_rand is
9    port(
10     clk           :  in std_logic;
11     rst           :  in std_logic;
12     fcw           :  in unsigned(3 downto 0);
13     en_wave       :  in std_logic;
14     sin           : out std_logic_vector(vector_length-1 downto 0);
15     cos           : out std_logic_vector(vector_length-1 downto 0)
16   );
17 end entity dds_rand;
18
19 architecture rtl of dds_rand is
20
21   component phase_acc is
22     generic(
23       cos_start     : integer := 1023
24     );
25     port (
26       clk           :  in std_logic;
27       rst           :  in std_logic;
28       fcw           :  in unsigned(3 downto 0);
29       en_wave       :  in std_logic;
30       nco           : out std_logic_vector(vector_length-1 downto 0);
31       nco_cos       : out std_logic_vector(vector_length-1 downto 0)
32     );
33   end component phase_acc;
34
35   component sin_table is
36     port (
37       clk           :  in std_logic;
38       sin_addr      :  in unsigned(vector_length-1 downto 0);
39       cos_addr      :  in unsigned(vector_length-1 downto 0);
40       sin           : out std_logic_vector(vector_length-1 downto 0);
41       cos           : out std_logic_vector(vector_length-1 downto 0)
42     );
43   end component sin_table;
44
45   signal sin_i : std_logic_vector(vector_length-1 downto 0);
46   signal cos_i : std_logic_vector(vector_length-1 downto 0);
47
48 begin
49
50   NCO: phase_acc
51     port map(
52       clk           => clk,
53       rst           => rst,
54       fcw           => fcw,
55       en_wave       => en_wave,
56       nco           => sin_i,
57       nco_cos       => cos_i
58     );
59
```

```
60      LUT: sin_table
61        port map(
62          clk           => clk,
63          sin_addr      => unsigned(sin_i),
64          cos_addr      => unsigned(cos_i),
65          sin           => sin,
66          cos           => cos
67        );
68
69    end architecture;
```

Source Code C.6: DSP

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use work.array_pkg.all;
5
6  entity dsp is
7    port (
8      clk        :  in std_logic;
9      rst        :  in std_logic;
10
11     -- ADC
12     sin_in     :  in std_logic_vector(vector_length-1 downto 0);
13     o_sclk     :  in std_logic;
14
15     -- DDS
16     sin_ref         : in std_logic_vector(vector_length-1 downto 0);
17     cos_ref         : in std_logic_vector(vector_length-1 downto 0);
18     fcw             :  in std_logic_vector(3 downto 0);
19
20     -- LIA
21     x_out            : out signed(vector_length+1 downto 0);
22     s_out            : out signed(vector_length+1 downto 0);
23     c_out            : out signed(vector_length+1 downto 0);
24     inph_out         : out signed(2*vector_length+3 downto 0);
25     quad_out         : out signed(2*vector_length+3 downto 0);
26
27     period_str      :  in std_logic;
28     magnitude       : out std_logic_vector(11 downto 0);--2*vector_length-2
       ↪  downto 0);
29     angle           : out std_logic_vector(11+4 downto 0);--2*vector_length-2
       ↪  downto 0);
30     xin_mean        : out std_logic_vector(11 downto 0);
31     inphase_mean    : out std_logic_vector(11+1 downto 0);
32     quadrature_mean : out std_logic_vector(11+1 downto 0)
33
34     --eidors_rea : out std_logic_vector(2*vector_length-2 downto 0);
35     --eidors_res : out std_logic_vector(2*vector_length-2 downto 0)
36   );
37 end entity dsp;
38
39 architecture rtl of dsp is
40
41   --signal res_lia   :  std_logic_vector(2*vector_length-2 downto 0);
42   --signal rea_lia   :  std_logic_vector(2*vector_length-2 downto 0);
43
44   --LIA mean
45   signal period_str_i : std_logic;
46   signal x_mean : std_logic_vector(11 downto 0);
47   signal inph_mean, quad_mean : signed(12 downto 0);
48
49   signal sin_ref_i, cos_ref_i : std_logic_vector(vector_length-1 downto 0);
50   signal mean_str_i : std_logic;
51
52   -- CORDIC signals
53   type cordic_arr_t is array (0 to 22) of std_logic_vector(2*vector_length-2
     ↪  downto 0);
54   signal res_arr, rea_arr, mag_arr, ang_arr : cordic_arr_t;
55
56   signal inph_cor, quad_cor : std_logic_vector(22 downto 0);
```

```
57    signal cnt : integer range 0 to 22;
58
59
60    -- FIFO signals
61    signal wr_en_i, dma_rd : std_logic;
62    signal full_i, full_rea, full_res, empty_i, empty_rea, empty_res : std_logic;
63    signal angle_i : std_logic_vector(11+4 downto 0);
64    signal mag_i : std_logic_vector(11 downto 0);
65    signal res_dma, rea_dma : std_logic_vector(2*vector_length-2 downto 0);
66
67    signal sin_slow  : std_logic_vector(vector_length-1 downto 0);
68    signal cos_slow  : std_logic_vector(vector_length-1 downto 0);
69    signal str_50MHz : std_logic;
70    -- DMA
71    signal rd_en_i : std_logic;
72    signal no_i    : unsigned(4 downto 0);
73
74
75    component lockinamp is
76      port (
77        clk              :  in std_logic;
78        rst              :  in std_logic;
79        period_str       :  in std_logic;
80        fcw              :  in std_logic_vector(3 downto 0);
81        sin_ref          :  in std_logic_vector(vector_length-1 downto 0);
82        cos_ref          :  in std_logic_vector(vector_length-1 downto 0);
83        x_in             :  in std_logic_vector(vector_length-1 downto 0);
84
85            x_out            : out signed(vector_length+1 downto 0);
86                s_out            : out signed(vector_length+1 downto 0);
87      c_out            : out signed(vector_length+1 downto 0);
88      inph_out         : out signed(2*vector_length+3 downto 0);
89      quad_out         : out signed(2*vector_length+3 downto 0);
90
91        mean_str         : out std_logic;
92        xin_mean         : out std_logic_vector(11 downto 0);
93        inphase_mean     : out signed(12 downto 0);
94        quadrature_mean  : out signed(12 downto 0)
95      );
96    end component lockinamp;
97
98    component eidors_sync is
99      port (
100       clk         :  in std_logic;
101       rst         :  in std_logic;
102       rea_in      :  in std_logic_vector(2*vector_length-1 downto 0);
103       res_in      :  in std_logic_vector(2*vector_length-1 downto 0);
104       rea_out     : out std_logic_vector(2*vector_length-1 downto 0);
105       res_out     : out std_logic_vector(2*vector_length-1 downto 0)
106     );
107   end component eidors_sync;
108
109   component fifo is
110     generic (
111       inout_length : integer := (vector_length-1);
112       new_array_length : integer := 150000--262143--(4*array_length-1)
113     );
114     port (
115       clk      :  in std_logic;
116       rst      :  in std_logic;
117       -- Read/write
118       wr_en    :  in std_logic;
```

```
119         rd_en    :  in std_logic;
120         full     : out std_logic;
121         empty    : out std_logic;
122         -- Data
123         sin_wr   :  in std_logic_vector(inout_length downto 0);
124         sin_rd   : out std_logic_vector(inout_length downto 0)
125       );
126     end component fifo;
127 
128     component cordic is
129       generic (
130         in_L     : integer := vector_length-1
131       );
132     port (
133       clk       :  in std_logic;
134       rst       :  in std_logic;
135       mean_str  :  in std_logic;
136       Y_in      :  in signed(in_L+1 downto 0);
137       X_in      :  in signed(in_L+1 downto 0);
138       magnitude : out std_logic_vector(in_L downto 0);
139       angle     : out std_logic_vector(in_L+4 downto 0)
140     );
141     end component cordic;
142 
143     component dds_rand is
144       port(
145         clk           :  in std_logic;
146         rst           :  in std_logic;
147         fcw           :  in unsigned(3 downto 0);
148         en_wave       :  in std_logic;
149         sin           : out std_logic_vector(vector_length-1 downto 0);
150         cos           : out std_logic_vector(vector_length-1 downto 0)
151       );
152     end component dds_rand;
153 
154   begin
155 
156   --  DDS: dds_rand
157   --    port map(
158   --      clk           => str_50MHz,
159   --      rst           => rst,
160   --      fcw           => unsigned(fcw),
161   --      en_wave       => '1',
162   --      sin           => sin_slow,
163   --      cos           => cos_slow
164   --  );
165 
166     LIA: lockinamp
167       port map(
168         clk        => clk,
169         rst        => rst,
170         period_str => period_str_i,
171         fcw        => fcw,
172         sin_ref    => sin_ref,--sin_slow,
173         cos_ref    => cos_ref,--cos_slow,
174         x_in       => sin_in,
175         x_out      => x_out,
176         s_out      => s_out,
177         c_out      => c_out,
178         mean_str   => mean_str_i,
179         inph_out   => inph_out,
180         quad_out   => quad_out,
```

```
181         xin_mean   => x_mean,
182         inphase_mean => inph_mean,
183         quadrature_mean => quad_mean
184       );
185
186       --inph_cor <= inph_mean & "00000000000";
187       --quad_cor <= quad_mean & "00000000000";
188     --CORDIC_gen: for I in 0 to 22 generate
189      CORDIC_p: cordic
190         port map (
191           clk       => clk,
192           rst       => rst,
193           mean_str  => mean_str_i,
194           X_in      => inph_mean,--std_logic_vector(std_logic_vector(inph_mean) &
              ↪  "00000000000"),--arr(I),
195           Y_in      => quad_mean,--std_logic_vector(std_logic_vector(quad_mean) &
              ↪  "00000000000"),--arr(I),
196           magnitude => mag_i,--arr(I),
197           angle     => angle_i--_arr(I)
198         );
199     --end generate CORDIC_gen;
200
201       rd_en_i <= '0' when rst = '1' else
202                  '1';
203       wr_en_i <= '0' when rst = '1' else
204                  '1';
205       dma_rd  <= '0' when rst = '1' else
206                  '1';
207
208       process(clk, rst)
209       begin
210         if rst = '1' then
211           str_50MHz <= '0';
212         elsif rising_edge(clk) then
213           str_50MHz <= not(str_50MHz);
214         end if;
215       end process;
216       --process(clk, rst)
217       --begin
218       --  if rst = '1' then
219       --    angle_i <= (others => '0');
220       --    mag_i   <= (others => '0');
221       --    cnt     <= 0;
222       --  elsif rising_edge(clk) then
223       --    res_arr(cnt) <= res_lia;
224       --    rea_arr(cnt) <= rea_lia;
225       --    mag_i      <= mag_arr(cnt);
226       --    angle_i    <= ang_arr(cnt);
227       --    if cnt >= 22 then
228       --      cnt <= 0;
229       --    else
230       --      cnt <= cnt + 1;
231       --    end if;
232       --  end if;
233       --end process;
234
235       --eidors_rea <= rea_lia;
236       --eidors_res <= res_lia;
237       angle      <= angle_i;
238       magnitude  <= mag_i;
239
240       period_str_i <= period_str;
```

```
241       xin_mean        <= x_mean;
242       inphase_mean    <= std_logic_vector(inph_mean);
243       quadrature_mean <= std_logic_vector(quad_mean);
244   end architecture rtl;
```

Source Code C.7: Electrode Master

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use work.array_pkg.all;
5  use work.timer_pkg.all;
6
7  entity electrode_master is
8    port(
9      clk         :  in std_logic;
10     rst         :  in std_logic;
11     btn         :  in std_logic_vector(3 downto 0);
12     n_electrode : out unsigned(5 downto 0);
13     measurement : out unsigned(4 downto 0);
14     fcw         : out unsigned(3 downto 0);
15     period_str  : out std_logic;
16     en_wave     : out std_logic;
17     en_delay    : out std_logic;
18     done_mux    : out std_logic
19   );
20 end entity electrode_master;
21
22 architecture rtl of electrode_master is
23
24   type t_state is (start, wave, check, next_freq);
25
26   constant c_freqs : integer := 9;
27   signal state : t_state;
28   signal count : unsigned(21 downto 0);
29   signal timer : integer range 0 to c_freqs;
30   signal curr_in, curr_out : unsigned(2 downto 0);
31   constant elec_max : unsigned(2 downto 0) := "111";
32   signal meas_cnt : unsigned(4 downto 0);
33   signal done_i : std_logic;
34   signal wait_i : std_logic;
35   signal timer_max : integer range tmin to tmax;
36   constant c_timer_max : integer := 65536;--0;
37   signal period_str_i : std_logic;
38
39   -- Buttons
40   signal start_btn, freq_up, freq_down : std_logic;
41
42   signal start_flag : std_logic;
43   signal start_cnt : integer range 0 to 101;
44
45   signal   delay_cnt : unsigned(11 downto 0);
46   constant c_delay : integer := 1510;
47
48   signal timer_delayed : integer range 0 to c_freqs;
49   signal en_delay_i, en_wave_i : std_logic;
50   signal delay_en : std_logic_vector(1 downto 0);
51
52 begin
53
54   process(clk, rst)
55   begin
56     if rst = '1' then
57       timer <= 0;
58       count <= (others => '0');
59       curr_in   <= "000";
```

```
60          curr_out  <= "001";
61          meas_cnt  <= (others => '0');
62          done_i <= '0';
63          wait_i <= '0';
64          state <= start;
65          delay_cnt <= (others => '0');
66          timer_delayed <= 0;
67          en_delay_i <= '0';
68          en_wave_i  <= '0';
69          period_str_i <= '1';
70        elsif rising_edge(clk) then
71          en_wave_i <= '1';
72          done_i <= '0';
73          --if btn = '1' then
74          --  delay_cnt   <= (others => '0');
75          --  done_i      <= '0';
76          --  timer <= 0;--timer + 1;
77          --  electrode_i <= electrode_i + 1;
78          --  count <= (others => '0');
79          --  state <= wave;
80          --else
81            case state is
82              when wave =>
83                en_wave_i  <= '1';
84                en_delay_i <= '1';
85                if freq_up = '0' and freq_down = '0' then
86                  if (to_integer(count) >= c_timer_max) then
87                    count <= (others => '0');
88                    period_str_i <= '1';
89                    state   <= check;
90                  else
91                    period_str_i <= '0';
92                    count <= count + 1;
93                    --if (to_integer(delay_cnt) >= c_delay) then
94                    --  case delay_en is
95                    --    when "00" =>
96                    --      en_delay_i <= '0';
97                    --      delay_en  <= "01";
98                    --      delay_cnt <= (others => '0');
99                    --    when "01" =>
100                   --      timer_delayed <= timer;
101                   --      en_delay_i    <= '1';
102                   --      delay_en <= "10";
103                   --    when others =>
104                   --      en_delay_i <= '1';
105                   --  end case;
106                   --else
107                   --  delay_cnt <= delay_cnt + 1;
108                   --end if;
109                 end if;
110               end if;
111
112             when check =>
113               state <= wave;
114               meas_cnt <= meas_cnt + 1;
115               if curr_in = elec_max then
116                 state  <= next_freq;
117                   --delay_cnt <= (others => '0');
118                   --delay_en  <= "00";
119                 curr_in   <= "000";
120                 curr_out  <= "001";
121                 meas_cnt  <= (others => '0');
```

```
122               elsif curr_out = elec_max then
123                 curr_in  <= curr_in  + 1;
124                 curr_out <= curr_in  + 2;
125                 --meas_cnt <= meas_cnt + 1;
126               else
127                 curr_out <= curr_out + 1;
128                 --meas_cnt <= meas_cnt + 1;
129               end if;
130
131             when next_freq =>
132               state    <= wave;
133               if freq_up = '1' then
134                 timer    <= timer + 1;
135               elsif freq_down = '1' then
136                 timer    <= timer - 1;
137               else
138                 state  <= next_freq;
139                 done_i <= '1';
140               end if;
141
142             when others => --start
143               en_wave_i <= '0';
144               if done_i = '0' and start_flag = '1' then
145                 count   <= (others => '0');
146                 state   <= wave;
147               end if;
148           end case;
149         --end if;
150       end if;
151     end process;
152
153     done_mux    <= done_i;
154     timer_max   <= C_timer_table(timer);
155     n_electrode <= curr_out & curr_in;
156
157     measurement <= meas_cnt;
158
159     fcw         <= to_unsigned(timer, fcw'length);
160     --fcw_delayed <= to_unsigned(timer_delayed, fcw_delayed'length);
161     en_delay    <= en_delay_i;
162
163     start_btn <= btn(3);
164
165     freq_up   <= btn(1);
166     freq_down <= btn(0);
167
168     start_cnt   <= 0 when rst = '1' else
169                    start_cnt + 1 when (rising_edge(clk)) and (start_cnt /= 100) else
170                    start_cnt;
171
172     start_flag  <= '1' when start_cnt = 100 and start_btn = '1' else '0';
173
174     en_wave     <= en_wave_i;
175
176
177   end architecture rtl;
```

Source Code C.8: FIFO

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use work.array_pkg.all;
5
6  entity fifo is
7    generic (
8      inout_length : integer := vector_length-1;
9      new_array_length : integer := array_length-1
10   );
11   port (
12     clk      :  in std_logic;
13     rst      :  in std_logic;
14     -- Read/write
15     wr_en    :  in std_logic;
16     rd_en    :  in std_logic;
17     full     : out std_logic;
18     empty    : out std_logic;
19     -- Data
20     sin_wr   :  in std_logic_vector(inout_length downto 0);
21     sin_rd   : out std_logic_vector(inout_length downto 0)
22   );
23 end entity fifo;
24
25 architecture rtl of fifo is
26
27   --constant divider : integer := 32;
28
29   type buff       is array (0 to new_array_length) of
     ↪  std_logic_vector(inout_length downto 0);
30   signal buff_arr : buff;
31
32   signal wr_index   : integer range 0 to new_array_length;
33   signal rd_index   : integer range 0 to new_array_length;
34   signal fifo_count : integer range -1 to new_array_length+2;
35
36   signal full_i  : std_logic;
37   signal empty_i : std_logic;
38
39 begin
40
41   process(clk)
42   begin
43     if rising_edge(clk) then
44       if rst = '1' then
45         fifo_count <= 0;
46         wr_index   <= 0;
47         rd_index   <= 0;
48         --reset: for I in 0 to new_array_length-1 loop
49         --  buff_arr(I) <= (others => '0');
50         --end loop;
51       else
52
53         if (wr_en = '1' and full_i = '0') then
54           fifo_count <= fifo_count + 1;
55           if wr_index = new_array_length then
56             wr_index <= 0;
57           else
58             wr_index <= wr_index + 1;
```

```
59            end if;
60          elsif (wr_en = '0' and rd_en = '1') then
61            fifo_count <= fifo_count - 1;
62          end if;
63
64          if (rd_en = '1' and empty_i = '0') then
65            if rd_index = new_array_length then
66              rd_index <= 0;
67            else
68              rd_index <= rd_index + 1;
69            end if;
70          end if;
71
72          --if wr_en = '1' then
73          --  buff_arr(wr_index) <= sin_wr;
74          --end if;
75        end if;
76      end if;
77    end process;
78
79    PROC_RAM : process(clk)
80    begin
81      if rising_edge(clk) then
82        buff_arr(wr_index) <= sin_wr;
83        sin_rd <= buff_arr(rd_index);
84      end if;
85    end process;
86
87    --sin_rd <= buff_arr(rd_index);
88
89    full_i   <= '1' when fifo_count = new_array_length else '0';
90    empty_i  <= '1' when fifo_count =  0 else '0';
91
92    full  <= full_i;
93    empty <= empty_i;
94
95  end rtl;
```

Source Code C.9: LFSR8

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity lfsr8 is
6    port (
7      clk        :  in std_logic;
8      rst        :  in std_logic;
9      start_seed :  in std_logic;
10     seed       :  in std_logic_vector(7 downto 0);
11     random_out : out std_logic_vector(7 downto 0)
12   );
13 end entity;
14
15 architecture rtl of lfsr8 is
16
17   signal random_i, seed_i   : std_logic_vector(7 downto 0);
18   signal start_i, start_id  : std_logic;
19
20 begin
21   process(clk, rst)
22   begin
23     if rst = '1' then
24       random_i <= (others => '0');
25       seed_i   <= (others => '0');
26       start_i  <= '1';
27     elsif rising_edge(clk) then
28       seed_i   <= seed;
29       start_i  <= start_seed;
30       start_id <= start_i;
31       case start_id is
32         when '0' =>
33           random_i(7 downto 1) <= random_i(6 downto 0);
34           random_i(0) <= random_i(3) xor random_i(4) xor random_i(5) xor
             ↪  random_i(7);
35         when '1' =>
36           random_i <= seed_i;
37         when others =>
38       end case;
39       random_out <= random_i;
40     end if;
41   end process;
42 end rtl;
```

Source Code C.10: LFSR12

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity lfsr12 is
6    port (
7      clk        :  in std_logic;
8      rst        :  in std_logic;
9      start_seed :  in std_logic;
10     seed       :  in std_logic_vector(11 downto 0);
11     random_out : out std_logic_vector(11 downto 0)
12   );
13 end entity;
14
15 architecture rtl of lfsr12 is
16
17   signal random_i, seed_i : std_logic_vector(11 downto 0);
18   signal start_i, start_id  : std_logic;
19
20 begin
21   process(clk, rst)
22   begin
23     if rst = '1' then
24       random_i <= (others => '0');
25       seed_i   <= (others => '0');
26       start_i  <= '1';
27     elsif rising_edge(clk) then
28       seed_i   <= seed;
29       start_i  <= start_seed;
30       start_id <= start_i;
31       case start_id is
32         when '0' =>
33           random_i(11 downto 1) <= random_i(10 downto 0);
34           random_i(0) <= random_i(3) xor random_i(9) xor random_i(10) xor
             ↪  random_i(11);
35         when '1' =>
36           random_i <= seed_i;
37         when others =>
38       end case;
39       random_out <= random_i;
40     end if;
41   end process;
42 end rtl;
```

Source Code C.11: Lock-in Amplifier

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

entity lockinamp is
  port (
    clk              :  in std_logic;
    rst              :  in std_logic;
    period_str       :  in std_logic;
    fcw              :  in std_logic_vector(3 downto 0);
    sin_ref          :  in std_logic_vector(vector_length-1 downto 0);
    cos_ref          :  in std_logic_vector(vector_length-1 downto 0);
    x_in             :  in std_logic_vector(vector_length-1 downto 0);

    x_out            : out signed(vector_length+1 downto 0);
    s_out            : out signed(vector_length+1 downto 0);
    c_out            : out signed(vector_length+1 downto 0);

    inph_out         : out signed(2*vector_length+3 downto 0);
    quad_out         : out signed(2*vector_length+3 downto 0);


    mean_str         : out std_logic;
    xin_mean         : out std_logic_vector(11 downto 0);
    inphase_mean     : out signed(12 downto 0);
    quadrature_mean  : out signed(12 downto 0)
    --valid            : out std_logic;
    --resist_out       : out std_logic_vector(2*vector_length-2 downto 0);
    --react_out        : out std_logic_vector(2*vector_length-2 downto 0)
  );
end entity;

architecture rtl of lockinamp is

signal sin_ref_d, sin_ref_dd  : signed(vector_length+1 downto 0);
signal cos_ref_d, cos_ref_dd  : signed(vector_length+1 downto 0);
signal x_in_d, x_in_dd        : signed(vector_length+1 downto 0);
signal inphase_i              : signed(2*vector_length+3 downto 0);
signal inph_d, inph_dd        : signed(2*vector_length+3 downto 0);
signal quadrature_i           : signed(2*vector_length+3 downto 0);
signal quad_d, quad_dd        : signed(2*vector_length+3 downto 0);
signal arst_flag    : std_logic;
signal valid_i      : std_logic;
constant offset     : signed(12 downto 0) := "0011111111111";

signal sin_ref_offset, cos_ref_offset, x_in_offset : signed(vector_length+1
↪  downto 0);

-- Averager
signal fcw_i : integer range 0 to 9;
signal xin_sum, inph_sum, quad_sum : signed(2*vector_length+4 downto 0); --
↪  Accumulated sum for averager
signal inph_scaled, quad_scaled : signed(44 downto 0);
signal x_mean, inph_mean, quad_mean : signed(12+9 downto 0);
signal div_mean : integer range 25 downto 16;
constant zero_constant : unsigned(20 downto 0) := "00000000000000000000";
signal n : unsigned(17 downto 0);
signal mean_str_i : std_logic;
```

```
58
59  signal period_str_i, period_str_d : std_logic;
60
61  constant div_gain : signed(15 downto 0) := "0101001111010111";
62
63  type sample_length_t is array (0 to 9) of integer range 9999 to 100000;
64  constant sample_length_C : sample_length_t := (100000-1, 50000-1, 33333, 25000-1,
    ↪  20000-1, 16666, 14285, 12500-1, 11111, 10000-1);
65  signal max_sample : integer range 9999 to 99999;
66
67  begin
68
69    LIA_proc: process(clk, rst)
70    begin
71      if (rst = '1') then
72        sin_ref_d      <= (others => '0');
73        sin_ref_dd     <= (others => '0');
74        cos_ref_d      <= (others => '0');
75        cos_ref_dd     <= (others => '0');
76        x_in_d         <= (others => '0');
77        x_in_dd        <= (others => '0');
78        sin_ref_offset <= (others => '0');
79        cos_ref_offset <= (others => '0');
80        x_in_offset    <= (others => '0');
81        --valid_i <= '0';
82      elsif rising_edge(clk) then
83        --if valid_i = '0' the9n
84        sin_ref_d  <= signed("00" & unsigned(sin_ref));
85        sin_ref_dd <= sin_ref_d;
86        sin_ref_offset <= (sin_ref_dd - offset);
87
88        cos_ref_d  <= signed("00" & unsigned(cos_ref));
89        cos_ref_dd <= cos_ref_d;
90        cos_ref_offset <= (cos_ref_dd - offset);
91
92        x_in_d      <= signed("00" & unsigned(x_in));
93        x_in_dd     <= x_in_d;
94        x_in_offset <= (x_in_dd - offset);
95
96        inphase_i    <= (sin_ref_offset*x_in_offset);
97        quadrature_i <= (cos_ref_offset*x_in_offset);
98
99        inph_d       <= inphase_i;--(21 downto 0);
100       inph_dd      <= inph_d;
101
102       quad_d       <= quadrature_i;--(21 downto 0);
103       quad_dd      <= quad_d;
104
105     end if;
106   end process;
107
108   x_out    <= x_in_offset;
109   s_out    <= sin_ref_offset;
110   c_out    <= cos_ref_offset;
111   inph_out <= inph_dd;
112   quad_out <= quad_dd;
113
114   AVERAGE_proc: process(clk, rst)
115   begin
116     if rst = '1' then
117       xin_sum    <=  (others => '0');
118       inph_sum   <=  (others => '0');
```

```
119         quad_sum   <=  (others => '0');
120         div_mean   <=  25;
121 
122         n <= (others => '0');
123 
124         x_mean    <= (others => '0');
125         inph_mean <= (others => '0');
126         quad_mean <= (others => '0');
127 
128         inph_scaled <= (others => '0');
129         quad_scaled <= (others => '0');
130         --n          <= 0;
131       elsif rising_edge(clk) then
132         period_str_i <= period_str;
133         period_str_d <= period_str_i;
134         fcw_i <= to_integer(unsigned(fcw));
135         --fcw_d <= fcw_i;
136 
137         max_sample <= sample_length_C(fcw_i);
138 
139         if to_integer(n) = 99999 then
140           --x_mean     <= signed(xin_sum(2*vector_length+4 downto 16));
141           inph_scaled  <= inph_sum*div_gain;
142           quad_scaled  <= quad_sum*div_gain;
143           --x_mean     <= xin_sum/n;
144           --inph_mean  <= inph_sum/n;
145           --quad_mean  <= quad_sum/n;
146           xin_sum    <= (others => '0');
147           inph_sum   <= (others => '0');
148           quad_sum   <= (others => '0');
149           n          <= (others => '0');
150           mean_str_i <= '1';
151         elsif period_str_d = '0' then
152           xin_sum    <= xin_sum + x_in_offset;
153           inph_sum   <= inph_sum + inph_dd(2*vector_length+3 downto 12);
154           quad_sum   <= quad_sum + quad_dd(2*vector_length+3 downto 12);
155           --div_mean   <= 25 - to_integer(unsigned(fcw));
156           n <= n + 1;
157           mean_str_i <= '0';
158         end if;
159         --xin_mean_i        <= (zero_constant +    x_mean(31 downto div_mean));
160         --inphase_mean_i    <= (zero_constant + inph_mean(31 downto div_mean));
161         --quadrature_mean_i <= (zero_constant + quad_mean(31 downto div_mean));
162       end if;
163     end process;
164 
165     mean_str <= mean_str_i;
166 
167     --inph_mean    <= (others => '1') when rst = '1' else
168     --              signed("00000000000000000000000" + (inph_scaled(44 downto
      ↪  32-fcw_i))) when rising_edge(clk) else
169     --              inph_mean;--(43 downto 23));
170     --quad_mean     <= (others => '1') when rst = '1' else
171     --                  signed("00000000000000000000000" + (quad_scaled(44 downto
      ↪  32-fcw_i))) when rising_edge(clk) else
172                   --    quad_mean;--(43 downto 23));
173     inphase_mean <= inph_scaled(44 downto 32);
174     quadrature_mean    <= quad_scaled(44 downto 32);
175 
176   end architecture rtl;
```

Source Code C.12: Vivado generated native to 32 bit AXI-Stream

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity M_AXIS_DATA32_v1_0_M00_AXIS is
6          generic (
7                  -- Users to add parameters here
8
9                  -- User parameters ends
10                 -- Do not modify the parameters beyond this line
11
12                 -- Width of S_AXIS address bus. The slave accepts the read and
                   ↪  write addresses of width C_M_AXIS_TDATA_WIDTH.
13                 C_M_AXIS_TDATA_WIDTH        : integer        := 32;
14                 -- Start count is the number of clock cycles the master will wait
                   ↪  before initiating/issuing any transaction.
15                 C_M_START_COUNT         : integer        := 32
16         );
17         port (
18                 -- Users to add ports here
19                 CUSTOM_DATA    : in std_logic_vector(31 downto 0);
20                 -- User ports ends
21                 -- Do not modify the ports beyond this line
22
23                 -- Global ports
24                 M_AXIS_ACLK         : in std_logic;
25                 --
26                 M_AXIS_ARESETN         : in std_logic;
27                 -- Master Stream Ports. TVALID indicates that the master is
                   ↪  driving a valid transfer, A transfer takes place when both
                   ↪  TVALID and TREADY are asserted.
28                 M_AXIS_TVALID        : out std_logic;
29                 -- TDATA is the primary payload that is used to provide the data
                   ↪  that is passing across the interface from the master.
30                 M_AXIS_TDATA         : out std_logic_vector(C_M_AXIS_TDATA_WIDTH-1
                   ↪  downto 0);
31                 -- TSTRB is the byte qualifier that indicates whether the content
                   ↪  of the associated byte of TDATA is processed as a data byte
                   ↪  or a position byte.
32                 M_AXIS_TSTRB         : out
                   ↪  std_logic_vector((C_M_AXIS_TDATA_WIDTH/8)-1 downto 0);
33                 -- TLAST indicates the boundary of a packet.
34                 M_AXIS_TLAST         : out std_logic;
35                 -- TREADY indicates that the slave can accept a transfer in the
                   ↪  current cycle.
36                 M_AXIS_TREADY         : in std_logic
37         );
38 end M_AXIS_DATA32_v1_0_M00_AXIS;
39
40 architecture implementation of M_AXIS_DATA32_v1_0_M00_AXIS is
41         -- Total number of output data
           ↪
42         constant NUMBER_OF_OUTPUT_WORDS : integer := 65535;
43
44          -- function called clogb2 that returns an integer which has the
45          -- value of the ceiling of the log base 2.
46         function clogb2 (bit_depth : integer) return integer is
47                  variable depth  : integer := bit_depth;
48                  variable count  : integer := 1;
```

```
49          begin
50                  for clogb2 in 1 to bit_depth loop  -- Works for up to 32 bit
                    ↪  integers
51                if (bit_depth <= 2) then
52                  count := 1;
53                else
54                  if(depth <= 1) then
55                          count := count;
56                        else
57                          depth := depth / 2;
58                    count := count + 1;
59                        end if;
60                     end if;
61            end loop;
62            return(count);
63          end;
64
65          -- WAIT_COUNT_BITS is the width of the wait counter.
            ↪
66          constant  WAIT_COUNT_BITS  : integer := clogb2(C_M_START_COUNT-1);
67
68        -- In this example, Depth of FIFO is determined by the greater of
          ↪
69        -- the number of input words and output words.
          ↪
70        constant depth : integer := NUMBER_OF_OUTPUT_WORDS;
71
72        -- bit_num gives the minimum number of bits needed to address 'depth' size
          ↪  of FIFO
73        constant bit_num : integer := clogb2(depth);
74
75        -- Define the states of state machine
          ↪
76        -- The control state machine oversees the writing of input streaming data
          ↪  to the FIFO,
77        -- and outputs the streaming data from the FIFO
          ↪
78        type state is ( IDLE,         -- This is the initial/idle state
          ↪
79                        INIT_COUNTER,  -- This state initializes the counter,
                          ↪  once
80                                       -- the counter reaches C_M_START_COUNT
                                         ↪  count,
81                                       -- the state machine changes state to
                                         ↪  SEND_STREAM
82                        SEND_STREAM);  -- In this state the
                          ↪
83                                     -- stream data is output through
                                       ↪  M_AXIS_TDATA
84        -- State variable
          ↪
85        signal  mst_exec_state : state;
86        -- Example design FIFO read pointer
          ↪
87        signal read_pointer : integer range 0 to depth-1;
88
89        -- AXI Stream internal signals
90        --wait counter. The master waits for the user defined number of clock
          ↪  cycles before initiating a transfer.
91        signal count        : std_logic_vector(WAIT_COUNT_BITS-1 downto 0);
92        --streaming data valid
93        signal axis_tvalid        : std_logic;
```

```
94          --streaming data valid delayed by one clock cycle
95          signal axis_tvalid_delay        : std_logic;
96          --Last of the streaming data
97          signal axis_tlast        : std_logic;
98          --Last of the streaming data delayed by one clock cycle
99          signal axis_tlast_delay        : std_logic;
100         --FIFO implementation signals
101         signal stream_data_out        : std_logic_vector(C_M_AXIS_TDATA_WIDTH-1
            ↪  downto 0);
102         signal tx_en        : std_logic;
103         --The master has issued all the streaming data stored in FIFO
104         signal tx_done        : std_logic;
105
106
107 begin
108         -- I/O Connections assignments
109
110         M_AXIS_TVALID        <= axis_tvalid_delay;
111         M_AXIS_TDATA        <= stream_data_out;
112         M_AXIS_TLAST        <= axis_tlast_delay;
113         M_AXIS_TSTRB        <= (others => '1');
114
115
116         -- Control state machine implementation
            ↪
117         process(M_AXIS_ACLK)
118         begin
119           if (rising_edge (M_AXIS_ACLK)) then
120             if(M_AXIS_ARESETN = '0') then
121               -- Synchronous reset (active low)
                  ↪
122               mst_exec_state      <= IDLE;
123               count <= (others => '0');
124             else
125               case (mst_exec_state) is
126                 when IDLE     =>
127                   -- The slave starts accepting tdata when
                      ↪
128                   -- there tvalid is asserted to mark the
                      ↪
129                   -- presence of valid streaming data
                      ↪
130                   --if (count = "0")then
                      ↪
131                     mst_exec_state <= INIT_COUNTER;
132                   --else
                      ↪
133                   --  mst_exec_state <= IDLE;
                      ↪
134                   --end if;
                      ↪
135
136                   when INIT_COUNTER =>
137                     -- This state is responsible to wait for user defined
                        ↪  C_M_START_COUNT
138                     -- number of clock cycles.
                        ↪
139                     if ( count = std_logic_vector(to_unsigned((C_M_START_COUNT -
                        ↪  1), WAIT_COUNT_BITS))) then
140                       mst_exec_state  <= SEND_STREAM;
141                     else
142                       count <= std_logic_vector (unsigned(count) + 1);
```

```
143                     mst_exec_state  <= INIT_COUNTER;
144                   end if;
145
146                 when SEND_STREAM  =>
147                   -- The example design streaming master functionality starts
                      ↪
148                   -- when the master drives output tdata from the FIFO and the
                      ↪  slave
149                   -- has finished storing the S_AXIS_TDATA
                      ↪
150                   if (tx_done = '1') then
151                     mst_exec_state <= IDLE;
152                   else
153                     mst_exec_state <= SEND_STREAM;
154                   end if;
155
156                 when others    =>
157                   mst_exec_state <= IDLE;
158
159               end case;
160             end if;
161           end if;
162         end process;
163
164
165         --tvalid generation
166         --axis_tvalid is asserted when the control state machine's state is
            ↪  SEND_STREAM and
167         --number of output streaming data is less than the
            ↪  NUMBER_OF_OUTPUT_WORDS.
168         axis_tvalid <= '1' when ((mst_exec_state = SEND_STREAM) and (read_pointer
            ↪  < NUMBER_OF_OUTPUT_WORDS)) else '0';
169
170         -- AXI tlast generation
            ↪
171         -- axis_tlast is asserted number of output streaming data is
            ↪  NUMBER_OF_OUTPUT_WORDS-1
172         -- (0 to NUMBER_OF_OUTPUT_WORDS-1)
            ↪
173         axis_tlast <= '1' when (read_pointer = NUMBER_OF_OUTPUT_WORDS-1) else
            ↪  '0';
174
175         -- Delay the axis_tvalid and axis_tlast signal by one clock cycle
            ↪
176         -- to match the latency of M_AXIS_TDATA
            ↪
177         process(M_AXIS_ACLK)
178         begin
179           if (rising_edge (M_AXIS_ACLK)) then
180             if(M_AXIS_ARESETN = '0') then
181               axis_tvalid_delay <= '0';
182               axis_tlast_delay <= '0';
183             else
184               axis_tvalid_delay <= axis_tvalid;
185               axis_tlast_delay <= axis_tlast;
186             end if;
187           end if;
188         end process;
189
190
191         --read_pointer pointer
192
```

```
193         process(M_AXIS_ACLK)
194         begin
195           if (rising_edge (M_AXIS_ACLK)) then
196             if(M_AXIS_ARESETN = '0') then
197               read_pointer <= 0;
198               tx_done  <= '0';
199             else
200               if (read_pointer <= NUMBER_OF_OUTPUT_WORDS-1) then
201                 if (tx_en = '1') then
202                   -- read pointer is incremented after every read from the FIFO
                      ↪
203                   -- when FIFO read signal is enabled.
                      ↪
204                   read_pointer <= read_pointer + 1;
205                   tx_done <= '0';
206                 end if;
207               elsif (read_pointer = NUMBER_OF_OUTPUT_WORDS) then
208                 -- tx_done is asserted when NUMBER_OF_OUTPUT_WORDS numbers of
                      ↪  streaming data
209                 -- has been out.
                      ↪
210                 tx_done <= '1';
211               end  if;
212             end  if;
213           end  if;
214         end process;
215
216
217         --FIFO read enable generation
218
219         tx_en <= M_AXIS_TREADY and axis_tvalid;
220
221         -- FIFO Implementation
            ↪
222
223         -- Streaming output data is read from FIFO
            ↪
224         process(M_AXIS_ACLK)
225         variable  sig_one : integer := 1;
226         begin
227           if (rising_edge (M_AXIS_ACLK)) then
228                 if(M_AXIS_ARESETN = '0') then
229                   stream_data_out <=
                        ↪  std_logic_vector(to_unsigned(sig_one,C_M_AXIS_TDATA_WIDTH));
230                 elsif (tx_en = '1') then -- && M_AXIS_TSTRB(byte_index)
                      ↪
231                   stream_data_out <= CUSTOM_DATA; --std_logic_vector(
                        ↪  to_unsigned(read_pointer,C_M_AXIS_TDATA_WIDTH) +
                        ↪  to_unsigned(sig_one,C_M_AXIS_TDATA_WIDTH));
232                 end if;
233            end if;
234          end process;
235
236         -- Add user logic here
237       --stream_data_out <= M_AXIS_STREAM_DATA;
238         -- User logic ends
239
240   end implementation;
```

Source Code C.13: Phase Accumulator

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

entity phase_acc is
  generic(
    cos_start     : integer := (array_length/4)
  );
  port (
    clk           :  in std_logic;
    rst           :  in std_logic;
    fcw           :  in unsigned(3 downto 0);
    en_wave       :  in std_logic;
    nco           : out std_logic_vector(vector_length-1 downto 0);
    nco_cos       : out std_logic_vector(vector_length-1 downto 0)
  );
end phase_acc;

architecture rtl of phase_acc is

  signal fcw_i            : unsigned( 3 downto 0);
  signal nco_i            : unsigned(31 downto 0);
  signal nco_cos_i        : unsigned(31 downto 0);
  signal FTW_i            : unsigned(31 downto 0);

  type t_freq_table is array(0 to 9) of unsigned(31 downto 0);
  constant C_freq_table : t_freq_table := (
  "00000000000000010100111110001011",
  "00000000000000010100111110010110",
  "00000000000000011111010110100010",
  "00000000000001010011111000101101",
  "00000000000001101000110110111000",
  "00000000000011111011010100000100",
  "00000000000100100101100110011111",
  "00000000000101001111100010110010",
  "00000000000101111001011110100110",
  "00000000000110100011011011100001"); -- 1k-10k

begin

  p_nco8 : process(clk, rst, en_wave)
  begin
    if (rst = '1') or en_wave = '0' then
      nco_i             <= (others => '0');
      nco_cos_i         <= to_unsigned(cos_start, nco_cos'length) &
      ↪  "00000000000000000000";
    elsif (rising_edge(clk)) then
      if (en_wave = '1') then
        fcw_i           <= fcw;
        nco_i           <= nco_i + FTW_i;
        nco_cos_i       <= nco_cos_i + FTW_i;--fcw_i;
      end if;
    end if;
  end process;

  FTW_i   <= (C_freq_table(to_integer(fcw_i)));
  nco     <= std_logic_vector(nco_i(nco_i'high downto
  ↪  nco_i'high-vector_length+1));
```

```
58      nco_cos <= std_logic_vector(nco_cos_i(nco_cos_i'high downto
        ↪  nco_cos_i'high-vector_length+1));
59
60  end rtl;
```

Source Code C.14: ADC

```
1  ------------------------------------------------------------------------------
2  --
3  --   FileName:         pmod_adc_ad7476a.vhd
4  --   Dependencies:     spi_master_dual_miso.vhd
5  --   Design Software:  Quartus Prime Version 17.0.0 Build 595 SJ Lite Edition
6  --
7  --   HDL CODE IS PROVIDED "AS IS."  DIGI-KEY EXPRESSLY DISCLAIMS ANY
8  --   WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT
9  --   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
10 --   PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL DIGI-KEY
11 --   BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL
12 --   DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF
13 --   PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS
14 --   BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
15 --   ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.
16 --
17 --   Version History
18 --   Version 1.0 12/19/2019 Scott Larson
19 --     Initial Public Release
20 --
21 ------------------------------------------------------------------------------
22
23 LIBRARY ieee;
24 USE ieee.std_logic_1164.all;
25
26 ENTITY pmod_adc_ad7476a IS
27   GENERIC(
28     clk_freq    :  INTEGER := 100; --system clock frequency in MHz
29     spi_clk_div :  INTEGER := 3);  --spi_clk_div = clk_freq/40 (answer rounded
       ↪ up)
30   PORT(
31     clk         :  IN      STD_LOGIC;                     --system clock
32     reset_n     :  IN      STD_LOGIC;                     --active low reset
33     data_in_0   :  IN      STD_LOGIC;                     --channel 0 serial
       ↪ data from ADC
34     data_in_1   :  IN      STD_LOGIC;                     --channel 1 serial
       ↪ data from ADC
35     sck         :  BUFFER  STD_LOGIC;                     --serial clock
36     cs_n        :  BUFFER  STD_LOGIC_VECTOR(0 DOWNTO 0);  --chip select
37     adc_0_data  :  OUT     STD_LOGIC_VECTOR(11 DOWNTO 0); --channel 0 ADC result
38     adc_1_data  :  OUT     STD_LOGIC_VECTOR(11 DOWNTO 0)); --channel 1 ADC result
39 END pmod_adc_ad7476a;
40
41 ARCHITECTURE behavior OF pmod_adc_ad7476a IS
42   SIGNAL  spi_rx_data_0  : STD_LOGIC_VECTOR(15 DOWNTO 0);  --latest channel 0
     ↪ data received
43   SIGNAL  spi_rx_data_1  : STD_LOGIC_VECTOR(15 DOWNTO 0);  --latest channel 1
     ↪ data received
44   SIGNAL  spi_ena        : STD_LOGIC;                      --enable for spi bus
45   SIGNAL  spi_busy       : STD_LOGIC;                      --busy signal from spi
     ↪ bus
46
47   --declare SPI Master component
48   COMPONENT spi_master_dual_miso IS
49     GENERIC(
50       slaves  : INTEGER := 1;   --number of spi slaves
51       d_width : INTEGER := 16); --data bus width
52     PORT(
53       clock     : IN     STD_LOGIC;                             --system clock
```

```
54        reset_n   : IN     STD_LOGIC;                             --asynchronous
          ↪ reset
55        enable    : IN     STD_LOGIC;                             --initiate
          ↪ transaction
56        cpol      : IN     STD_LOGIC;                             --spi clock
          ↪ polarity
57        cpha      : IN     STD_LOGIC;                             --spi clock phase
58        cont      : IN     STD_LOGIC;                             --continuous mode
          ↪ command
59        clk_div   : IN     INTEGER;                               --system clock
          ↪ cycles per 1/2 period of sclk
60        addr      : IN     INTEGER;                               --address of
          ↪ slave
61        tx_data   : IN     STD_LOGIC_VECTOR(d_width-1 DOWNTO 0);  --data to
          ↪ transmit
62        miso_0    : IN     STD_LOGIC;                             --master in,
          ↪ slave out, channel 0
63        miso_1    : IN     STD_LOGIC;                             --master in,
          ↪ slave out, channel 1
64        sclk      : BUFFER STD_LOGIC;                             --spi clock
65        ss_n      : BUFFER STD_LOGIC_VECTOR(slaves-1 DOWNTO 0);   --slave select
66        mosi      : OUT    STD_LOGIC;                             --master out,
          ↪ slave in
67        busy      : OUT    STD_LOGIC;                             --busy / data
          ↪ ready signal
68        rx_data_0 : OUT    STD_LOGIC_VECTOR(d_width-1 DOWNTO 0);  --data received,
          ↪ channel 0
69        rx_data_1 : OUT    STD_LOGIC_VECTOR(d_width-1 DOWNTO 0)); --data received,
          ↪ channel 1
70      END COMPONENT spi_master_dual_miso;
71
72    BEGIN
73
74      --instantiate and configure the SPI Master component
75      spi_master_dual_miso_0:  spi_master_dual_miso
76        GENERIC MAP(slaves => 1, d_width => 16)
77        PORT MAP(clock => clk, reset_n => reset_n, enable => spi_ena, cpol => '1',
78                 cpha => '0', cont => '0', clk_div => spi_clk_div, addr => 0,
79                 tx_data => (OTHERS => '0'), miso_0 => data_in_0, miso_1 =>
                   ↪ data_in_1,
80                 sclk => sck, ss_n => cs_n, mosi => open, busy => spi_busy,
81                 rx_data_0 => spi_rx_data_0, rx_data_1 => spi_rx_data_1);
82
83      PROCESS(clk)
84        VARIABLE count :  INTEGER := 0;
85      BEGIN
86        IF(reset_n = '0') THEN                --asynchronous reset
87          count := 0;                           --clear clock counter
88          spi_ena <= '0';                       --clear enable signal for serial
          ↪ interface
89        ELSIF(clk'EVENT AND clk = '1') THEN   --rising system clock edge
90          IF(spi_busy = '0') THEN               --serial transaction with ADC not in
          ↪ process
91            IF(count < clk_freq/20-2) THEN        --wait at least 50ns between serial
              ↪ transactions
92              count := count + 1;                   --increment clock counter
93              spi_ena <= '0';                       --do not enable serial
                ↪ transaction
94            ELSE                                  --50ns wait time met
95              spi_ena <= '1';                       --enable next serial transaction
                ↪ to get data
96            END IF;
```

```
97          ELSE                                --serial transaction with ADC in
            ↪ process
98            count := 0;                         --clear clock counter
99            spi_ena <= '0';                     --clear enable signal for next
              ↪ transaction
100         END IF;
101       END IF;
102     END PROCESS;
103     -- 12 downto 1
104     adc_0_data <= spi_rx_data_0(11 DOWNTO 0); --assign channel 0 ADC data bits to
        ↪ output
105     adc_1_data <= spi_rx_data_1(11 DOWNTO 0); --assign channel 1 ADC data bits to
        ↪ output
106
107   END behavior;
```

Source Code C.15: DAC

```
1  --------------------------------------------------------------------------------
2  --
3  --   FileName:         pmod_dac_ad5628.vhd
4  --   Dependencies:     spi_master.vhd
5  --   Design Software:  Quartus Prime Version 17.0.0 Build 595 SJ Lite Edition
6  --
7  --   HDL CODE IS PROVIDED "AS IS."  DIGI-KEY EXPRESSLY DISCLAIMS ANY
8  --   WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT
9  --   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
10 --   PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL DIGI-KEY
11 --   BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL
12 --   DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF
13 --   PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS
14 --   BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
15 --   ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.
16 --
17 --   Version History
18 --   Version 1.0 06/17/2020 Scott Larson
19 --     Initial Public Release
20 --
21 --------------------------------------------------------------------------------
22
23 LIBRARY ieee;
24 USE ieee.std_logic_1164.all;
25
26 ENTITY pmod_dac_ad5628 IS
27   GENERIC(
28     clk_freq    : INTEGER := 50;  --system clock frequency in MHz
29     spi_clk_div : INTEGER := 1);  --spi_clk_div = clk_freq/100 (answer rounded
       ↪ up)
30   PORT(
31     clk        : IN      STD_LOGIC;                      --system clock
32     reset_n    : IN      STD_LOGIC;                      --active low
       ↪ asynchronous reset
33     dac_tx_ena : IN      STD_LOGIC;                      --enable transaction
       ↪ with DAC
34     dac_cmd    : IN      STD_LOGIC_VECTOR(3 DOWNTO 0);   --command to send to DAC
35     dac_addr   : IN      STD_LOGIC_VECTOR(3 DOWNTO 0);   --address to send to DAC
36     dac_data   : IN      STD_LOGIC_VECTOR(11 DOWNTO 0);  --data value to send to
       ↪ DAC
37     busy       : OUT     STD_LOGIC;                      --indicates when
       ↪ transactions with DAC can be initiated
38     mosi       : OUT     STD_LOGIC;                      --SPI bus to DAC: master
       ↪ out, slave in (DIN)
39     sclk       : BUFFER  STD_LOGIC;                      --SPI bus to DAC: serial
       ↪ clock (SCLK)
40     ss_n       : BUFFER  STD_LOGIC_VECTOR(0 DOWNTO 0));  --SPI bus to DAC: slave
       ↪ select (~SYNC)
41 END pmod_dac_ad5628;
42
43 ARCHITECTURE behavior OF pmod_dac_ad5628 IS
44   TYPE machine IS(start, configure, pause, ready, send_data); --needed states
45   SIGNAL state         : machine := start;                    --state machine
46   SIGNAL spi_busy_prev : STD_LOGIC;                           --previous value of
       ↪ the SPI component's busy signal
47   SIGNAL spi_busy      : STD_LOGIC;                           --busy signal from
       ↪ SPI component
48   SIGNAL spi_ena       : STD_LOGIC;                           --enable for SPI
       ↪ component
```

```
49    SIGNAL spi_tx_data   : STD_LOGIC_VECTOR(31 DOWNTO 0);       --transmit data for
      ↪  SPI component
50
51    --declare SPI Master component
52    COMPONENT spi_master IS
53      GENERIC(
54        slaves  : INTEGER := 1;   --number of spi slaves
55        d_width : INTEGER := 32); --data bus width
56      PORT(
57        clock   : IN     STD_LOGIC;                             --system clock
58        reset_n : IN     STD_LOGIC;                             --asynchronous
          ↪  reset
59        enable  : IN     STD_LOGIC;                             --initiate
          ↪  transaction
60        cpol    : IN     STD_LOGIC;                             --spi clock
          ↪  polarity
61        cpha    : IN     STD_LOGIC;                             --spi clock phase
62        cont    : IN     STD_LOGIC;                             --continuous mode
          ↪  command
63        clk_div : IN     INTEGER;                               --system clock
          ↪  cycles per 1/2 period of sclk
64        addr    : IN     INTEGER;                               --address of slave
65        tx_data : IN     STD_LOGIC_VECTOR(d_width-1 DOWNTO 0);  --data to transmit
66        miso    : IN     STD_LOGIC;                             --master in, slave
          ↪  out
67        sclk    : BUFFER STD_LOGIC;                             --spi clock
68        ss_n    : BUFFER STD_LOGIC_VECTOR(slaves-1 DOWNTO 0);   --slave select
69        mosi    : OUT    STD_LOGIC;                             --master out, slave
          ↪  in
70        busy    : OUT    STD_LOGIC;                             --busy / data ready
          ↪  signal
71        rx_data : OUT    STD_LOGIC_VECTOR(d_width-1 DOWNTO 0)); --data received
72    END COMPONENT spi_master;
73
74  BEGIN
75
76    --instantiate the SPI Master component
77    spi_master_0:  spi_master
78      GENERIC MAP(slaves => 1, d_width => 32)
79      PORT MAP(clock => clk, reset_n => reset_n, enable => spi_ena, cpol => '1',
        ↪  cpha => '0',
80               cont => '0', clk_div => spi_clk_div, addr => 0, tx_data =>
                 ↪  spi_tx_data, miso => '0',
81               sclk => sclk, ss_n => ss_n, mosi => mosi, busy => spi_busy, rx_data
                 ↪  => open);
82
83    PROCESS(clk, reset_n)
84      VARIABLE count : INTEGER RANGE 0 TO clk_freq*100 := 0; --counter
85    BEGIN
86
87      IF(reset_n = '0') THEN               --reset activated
88        spi_ena <= '0';                      --clear SPI component enable
89        spi_tx_data <= (OTHERS => '0');      --clear SPI component transmit data
90        busy <= '1';                         --indication component is unavailable
91        state <= start;                      --restart state machine
92      ELSIF(clk'EVENT AND clk = '1') THEN --rising edge of system clock
93
94        spi_busy_prev <= spi_busy;           --collect previous spi_busy
95
96        CASE state IS                        --state machine
97
98          --entry state, give DAC 100us to power up before communicating
```

```
99          WHEN start =>
100           busy <= '1';                      --component is busy, DAC not yet
              ↪ available
101           IF(count < clk_freq*100) THEN     --100us not yet reached
102             count := count + 1;               --increment counter
103           ELSE                              --100us reached
104             count := 0;                       --clear counter
105             state <= configure;               --advance to configure the DAC
106           END IF;
107
108         --perform SPI transaction to turn on internal voltage reference
109         WHEN configure =>
110           IF(spi_busy = '0' AND spi_busy_prev = '0') THEN      --no command sent
111             spi_ena <= '1';                                      --enable
                ↪ transaction with DAC
112             spi_tx_data <= "00001000000000000000000000000001";   --send data to
                ↪ turn on internal voltage reference
113           ELSIF(spi_busy = '1') THEN                           --transaction
              ↪ underway
114             spi_ena <= '0';                                      --clear
                ↪ transaction enable
115           ELSE                                                 --transaction
              ↪ complete
116             state <= pause;                                      --advance to
                ↪ pause state
117           END IF;
118
119         --pauses 20ns between SPI transactions
120         WHEN pause =>
121           IF(count < clk_freq/50) THEN   --less than 20ns
122             count := count + 1;            --increment counter
123           ELSE                           --20ns has elapsed
124             count := 0;                    --clear counter
125             busy <= '0';                   --indicate component is ready for a
                ↪ transaction
126             state <= ready;                --advance to ready state
127           END IF;
128
129         --wait for a new transaction and latch it in
130         WHEN ready =>
131           IF(dac_tx_ena = '1') THEN
              ↪ --transaction to DAC requested
132             spi_tx_data <= "0000" & dac_cmd & dac_addr & dac_data & "00000000";
                ↪ --latch in data stream to send
133             busy <= '1';
                ↪ --indicate transaction is in progress
134             state <= send_data;
                ↪ --advance to sending transaction
135           END IF;
136
137         --performs SPI transaction to DAC
138         WHEN send_data =>
139           IF(spi_busy = '0' AND spi_busy_prev = '0') THEN  --transaction not
              ↪ started
140             spi_ena <= '1';                                  --enable SPI
                ↪ transaction
141           ELSIF(spi_busy = '1') THEN                       --transaction underway
142             spi_ena <= '0';                                  --clear enable
                ↪
143           ELSE                                             --transaction complete
144             state <= pause;                                  --return to pause
                ↪ state
```

```
145             END IF;
146 
147           --default to start state
148           WHEN OTHERS =>
149             state <= start;
150 
151         END CASE;
152       END IF;
153     END PROCESS;
154   END behavior;
```

Source Code C.16: Signal Loop

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use work.array_pkg.all;
5  use work.sine_wave_pkg.all;
6  use work.timer_pkg.all;
7
8  entity signal_loop_rand is
9      port (
10       clk         :  in std_logic;
11       rst         :  in std_logic;
12       btn         :  in std_logic_vector(3 downto 0);
13       period_str  : out std_logic;
14       done_ld     : out std_logic;
15       fcw_out     : out std_logic_vector(3 downto 0);
16       sin         : out std_logic_vector(vector_length-1 downto 0);
17       cos         : out std_logic_vector(vector_length-1 downto 0);
18       measurement : out unsigned(4 downto 0);
19       n_electrode : out unsigned(5 downto 0)
20     );
21 end entity signal_loop_rand;
22
23 architecture rtl of signal_loop_rand is
24
25   signal fcw_i, fcw_delayed :   unsigned(3 downto 0);
26   signal sin_i, sin_ref_i   :   std_logic_vector(vector_length-1 downto 0);
27   signal cos_i, cos_ref_i   :   std_logic_vector(vector_length-1 downto 0);
28   signal en_delay, en_wave  :   std_logic;
29   signal busy               :   std_logic;
30
31
32   component dds_rand is
33     port(
34       clk           :  in std_logic;
35       rst           :  in std_logic;
36       fcw           :  in unsigned(3 downto 0);
37       en_wave       :  in std_logic;
38       sin           : out std_logic_vector(vector_length-1 downto 0);
39       cos           : out std_logic_vector(vector_length-1 downto 0)
40     );
41   end component dds_rand;
42
43
44   component electrode_master is
45     port(
46       clk         :  in std_logic;
47       rst         :  in std_logic;
48       btn         :  in std_logic_vector(3 downto 0);
49       n_electrode : out unsigned(5 downto 0);
50       measurement : out unsigned(4 downto 0);
51       fcw         : out unsigned(3 downto 0);
52       period_str  : out std_logic;
53       en_wave     : out std_logic;
54       en_delay    : out std_logic;
55       done_mux    : out std_logic
56     );
57   end component electrode_master;
58
59
```

```
60  begin
61
62    DDS_out: dds_rand
63    port map(
64      clk         =>      clk,
65      rst         =>      rst,
66      fcw         =>      fcw_i,
67      en_wave     =>      en_wave,
68      sin         =>      sin_i,
69      cos         =>      cos_i
70    );
71
72    MUX: electrode_master
73    port map(
74      clk          =>      clk,
75      rst          =>      rst,
76      btn          =>      btn,
77      n_electrode  =>      n_electrode,
78      measurement  =>      measurement,
79      fcw          =>      fcw_i,
80      period_str   =>      period_str,
81      en_wave      =>      en_wave,
82      en_delay     =>      en_delay,
83      done_mux     =>      done_ld
84    );
85
86    --DDS_ref: dds_rand
87    --port map(
88    --  clk         =>      clk,
89    --  rst         =>      rst,
90    --  en_wave     =>      en_delay,
91    --  sin         =>      sin_ref_i,
92    --  cos         =>      cos_ref_i
93    --);
94
95  sin <= sin_i;
96  --sin_ref <= sin_ref_i;
97  cos <= cos_i;
98  fcw_out <= std_logic_vector(fcw_i);
99
100 end architecture rtl;
```

Source Code C.17: Sine table

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use work.array_pkg.all;
5  use work.sine_wave_pkg.all;
6
7  entity sin_table is
8    port (
9      clk           :  in std_logic;
10     sin_addr      :  in unsigned(vector_length-1 downto 0);
11     cos_addr      :  in unsigned(vector_length-1 downto 0);
12     sin           : out std_logic_vector(vector_length-1 downto 0);
13     cos           : out std_logic_vector(vector_length-1 downto 0)
14   );
15 end entity sin_table;
16
17 architecture rtl of sin_table is
18
19 begin
20
21     p_sin : process(clk)
22     begin
23       if(rising_edge(clk)) then
24         sin <= C_SIN_TABLE(to_integer(sin_addr));
25         cos <= C_SIN_TABLE(to_integer(cos_addr));
26       end if;
27     end process p_sin;
28
29 end rtl;
```

Source Code C.18: Sine wave package

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use work.array_pkg.all;
5
6  package sine_wave_pkg is
7
8    type t_sin_table is array(0 to array_length-1) of
     ↪  std_logic_vector(vector_length-1 downto 0);
9    constant C_sin_table : t_sin_table := (
10   "011111111111", "100000000010", "100000000101", "100000001000", "100000001100",
     ↪  "100000001111", "100000010010", "100000010101", "100000011000",
     ↪  "100000011011", "100000011110", "100000100010", "100000100101",
     ↪  "100000101000", "100000101011", "100000101110", "100000110001",
     ↪  "100000110100", "100000111000", "100000111011", "100000111110",
     ↪  "100001000001", "100001000100", "100001000111", "100001001010",
     ↪  "100001001101", "100001010001", "100001010100", "100001010111",
     ↪  "100001011010", "100001011101", "100001100000", "100001100011",
     ↪  "100001100111", "100001101010", "100001101101", "100001110000",
     ↪  "100001110011", "100001110110", "100001111001", "100001111101",
     ↪  "100010000000", "100010000011", "100010000110", "100010001001",
     ↪  "100010001100", "100010001111", "100010010010", "100010010110",
     ↪  "100010011001", "100010011100", "100010011111", "100010100010",
     ↪  "100010100101", "100010101000", "100010101011", "100010101111",
     ↪  "100010110010", "100010110101", "100010111000", "100010111011",
     ↪  "100010111110", "100011000001", "100011000101", "100011001000",
     ↪  "100011001011", "100011001110", "100011010001", "100011010100",
     ↪  "100011010111", "100011011010", "100011011110", "100011100001",
     ↪  "100011100100", "100011100111", "100011101010", "100011101101",
     ↪  "100011110000", "100011110011", "100011110110", "100011111010",
     ↪  "100011111101", "100100000000", "100100000011", "100100000110",
     ↪  "100100001001", "100100001100", "100100001111", "100100010010",
     ↪  "100100010110", "100100011001", "100100011100", "100100011111",
     ↪  "100100100010", "100100100101", "100100101000", "100100101011",
     ↪  "100100101110", "100100110010", "100100110101",
```

```
11      "1001001111000", "1001001110111", "1001001111110", "1001010000001", "1001010001000",
        ↪  "1001010000111", "1001010010100", "1001010010111", "1001010011010", "1001010011101",
        ↪  "1001010100000", "1001010100011", "1001010100110", "1001010101001",
        ↪  "1001010110100", "1001010110111", "1001010111001", "1001010111100",
        ↪  "1001011000101", "1001011001000", "1001011001011", "1001011001110",
        ↪  "1001011010001", "1001011010101", "1001011011000", "1001011011011",
        ↪  "1001011011110", "1001011100001", "1001011100100", "1001011100111",
        ↪  "1001011101010", "1001011101101", "1001011110000", "1001011110011",
        ↪  "1001011110110", "1001011111001", "1001011111100", "1001100000000",
        ↪  "1001100000011", "1001100000110", "1001100001001", "1001100001100",
        ↪  "1001100001111", "1001100010010", "1001100010101", "1001100011000",
        ↪  "1001100011011", "1001100011110", "1001100100001", "1001100100100",
        ↪  "1001100100111", "1001100101010", "1001100101101", "1001100110000",
        ↪  "1001100110011", "1001100110110", "1001100111001", "1001100111100",
        ↪  "1001101000000", "1001101000011", "1001101000110", "1001101001001",
        ↪  "1001101001100", "1001101001111", "1001101010010", "1001101010101",
        ↪  "1001101011000", "1001101011011", "1001101011110", "1001101100001",
        ↪  "1001101100100", "1001101100111", "1001101101010", "1001101101101",
        ↪  "1001101110000", "1001101110011", "1001101110110", "1001101111001",
        ↪  "1001101111100", "1001101111111", "1001110000010", "1001110000101",
        ↪  "1001110001000", "1001110001011", "1001110001110", "1001110010001",
        ↪  "1001110010100", "1001110010111", "1001110011010", "1001110011101",
        ↪  "1001110100000", "1001110100011", "1001110100110",
12      "1001110101001", "1001110101100", "1001110101111", "1001110110010", "1001110110101",
        ↪  "1001110111000", "1001110111011", "1001110111110", "1001111000001",
        ↪  "1001111000100", "1001111000111", "1001111001010", "1001111001101",
        ↪  "1001111010000", "1001111010011", "1001111010110", "1001111011001",
        ↪  "1001111011100", "1001111011111", "1001111100010", "1001111100101",
        ↪  "1001111101000", "1001111101011", "1001111101110", "1001111110001",
        ↪  "1001111110100", "1001111110111", "1001111111001", "1001111111100",
        ↪  "1001111111111", "1010000000010", "1010000000101", "1010000001000",
        ↪  "1010000001011", "1010000001110", "1010000010001", "1010000010100",
        ↪  "1010000010111", "1010000011010", "1010000011101", "1010000100000",
        ↪  "1010000100011", "1010000100110", "1010000101001", "1010000101100",
        ↪  "1010000101111", "1010000110010", "1010000110101", "1010000111000",
        ↪  "1010000111011", "1010000111110", "1010001000001", "1010001000100",
        ↪  "1010001000111", "1010001001010", "1010001001101", "1010001010000",
        ↪  "1010001010011", "1010001010110", "1010001011001", "1010001011100",
        ↪  "1010001011111", "1010001100010", "1010001100101", "1010001101000",
        ↪  "1010001101011", "1010001101110", "1010001110001", "1010001110100",
        ↪  "1010001110111", "1010001111010", "1010001111101", "1010010000000",
        ↪  "1010010000011", "1010010000110", "1010010001001", "1010010001100",
        ↪  "1010010001111", "1010010010010", "1010010010101", "1010010011000",
        ↪  "1010010011011", "1010010011110", "1010010100001", "1010010100100",
        ↪  "1010010100111", "1010010101010", "1010010101101", "1010010110000",
        ↪  "1010010110011", "1010010110110", "1010010111001", "1010010111100",
        ↪  "1010010111111", "1010011000010", "1010011000101", "1010011001000",
        ↪  "1010011001011", "1010011001110", "1010011010001", "1010011010100",
        ↪  "1010011010111", "1010011011010", "1010011011101", "1010011100000",
        ↪  "1010011100011", "1010011100110", "1010011101000", "1010011101011",
        ↪  "1010011101110", "1010011110001", "1010011110100", "1010011110111",
        ↪  "1010011111010", "1010011111101", "1011000000000", "1011000000011",
        ↪  "1011000000110", "1011000001001", "1011000001011", "1011000001110",
        ↪  "1011000010001", "1011000010100", "1011000010111", "1011000011010",
        ↪  "1011000011101", "1011000100000", "1011000100011", "1011000100110",
        ↪  "1011000101000", "1011000101011", "1011000101110", "1011000110001",
        ↪  "1011000110100", "1011000110111", "1011000111010", "1011000111101",
        ↪  "1011000111111", "1011010000010", "1011010000101", "1011010001000",
        ↪  "1011010001011", "1011010001110", "1011010010001", "1011010010011",
        ↪  "1011010010110", "1011010011001", "1011010011100", "1011010011111",
        ↪  "1011011000010", "1011011000101", "1011011001000", "1011011001011",
        ↪  "1011011001110", "1011011010001", "1011011010100", "1011011010111",
        ↪  "1011011011010", "1011011011101", "1011011100000", "1011011100011",
        ↪  "1011011100110", "1011011101001", "1011011101100", "1011011101111",
        ↪  "1011011110010", "1011011110101", "1011011111000", "1011011111011",
        ↪  "1011011111110", "1011100000001", "1011100000100", "1011100000110",
        ↪  "1011100001001", "1011100001100",
```

```
13      "101110001100", "101110001111", "101110010010", "101110010101", "101110010111",
        ↪  "101110011010", "101110011101", "101110100000", "101110100011",
        ↪  "101110100101", "101110101000", "101110101011", "101110101110",
        ↪  "101110110001", "101110110011", "101110110110", "101110111001",
        ↪  "101110111100", "101110111110", "101111000001", "101111000100",
        ↪  "101111000111", "101111001001", "101111001100", "101111001111",
        ↪  "101111010010", "101111010101", "101111010111", "101111011010",
        ↪  "101111011101", "101111100000", "101111100010", "101111100101",
        ↪  "101111101000", "101111101010", "101111101101", "101111110000",
        ↪  "101111110011", "101111110101", "101111111000", "101111111011",
        ↪  "101111111110", "110000000000", "110000000011", "110000000110",
        ↪  "110000001000", "110000001011", "110000001110", "110000010001",
        ↪  "110000010011", "110000010110", "110000011001", "110000011011",
        ↪  "110000011110", "110000100001", "110000100011", "110000100110",
        ↪  "110000101001", "110000101011", "110000101110", "110000110001",
        ↪  "110000110100", "110000110110", "110000111001", "110000111100",
        ↪  "110000111110", "110001000001", "110001000011", "110001000110",
        ↪  "110001001001", "110001001011", "110001001110", "110001010001",
        ↪  "110001010011", "110001010110", "110001011001", "110001011011",
        ↪  "110001011110", "110001100001", "110001100011", "110001100110",
        ↪  "110001101000", "110001101011", "110001101110", "110001110000",
        ↪  "110001110011", "110001110101", "110001111000", "110001111011",
        ↪  "110001111101", "110010000000", "110010000010", "110010000101",
        ↪  "110010001000", "110010001010", "110010001101", "110010001111",
        ↪  "110010010010", "110010010101", "110010010111",
14      "110010011010", "110010011100", "110010011111", "110010100001", "110010100100",
        ↪  "110010100110", "110010101001", "110010101100", "110010101110",
        ↪  "110010110001", "110010110011", "110010110110", "110010111000",
        ↪  "110010111011", "110010111101", "110011000000", "110011000010",
        ↪  "110011000101", "110011000111", "110011001010", "110011001100",
        ↪  "110011001111", "110011010001", "110011010100", "110011010110",
        ↪  "110011011001", "110011011011", "110011011110", "110011100000",
        ↪  "110011100011", "110011100101", "110011101000", "110011101010",
        ↪  "110011101101", "110011101111", "110011110010", "110011110100",
        ↪  "110011110111", "110011111001", "110011111100", "110011111110",
        ↪  "110100000001", "110100000011", "110100000101", "110100001000",
        ↪  "110100001010", "110100001101", "110100001111", "110100010010",
        ↪  "110100010100", "110100010110", "110100011001", "110100011011",
        ↪  "110100011110", "110100100000", "110100100011", "110100100101",
        ↪  "110100100111", "110100101010", "110100101100", "110100101111",
        ↪  "110100110001", "110100110011", "110100110110", "110100111000",
        ↪  "110100111010", "110100111101", "110100111111", "110101000010",
        ↪  "110101000100", "110101000110", "110101001001", "110101001011",
        ↪  "110101001101", "110101010000", "110101010010", "110101010100",
        ↪  "110101010111", "110101011001", "110101011011", "110101011110",
        ↪  "110101100000", "110101100010", "110101100101", "110101100111",
        ↪  "110101101001", "110101101100", "110101101110", "110101110000",
        ↪  "110101110010", "110101110101", "110101110111", "110101111001",
        ↪  "110101111100", "110101111110", "110110000000", "110110000010",
        ↪  "110110000101", "110110000111", "110110001001",
```

```
15      "1101100001100", "1101100001110", "1101100010000", "1101100010010", "1101100010101",
        ↪  "1101100010111", "1101100011001", "1101100011011", "1101100011110",
        ↪  "1101100100000", "1101100100010", "1101100100100", "1101100100110",
        ↪  "1101100101001", "1101100101011", "1101100101101", "1101100101111",
        ↪  "1101100110010", "1101100110100", "1101100110110", "1101100111000",
        ↪  "1101100111010", "1101100111100", "1101100111111", "1101111000001",
        ↪  "1101111000011", "1101111000101", "1101111000111", "1101111001010",
        ↪  "1101111001100", "1101111001110", "1101111010000", "1101111010010",
        ↪  "1101111010100", "1101111010110", "1101111011001", "1101111011011",
        ↪  "1101111011101", "1101111011111", "1101111100001", "1101111100011",
        ↪  "1101111100101", "1101111101000", "1101111101010", "1101111101100",
        ↪  "1101111101110", "1101111110000", "1101111110010", "1101111110100",
        ↪  "1101111110110", "1101111111000", "1101111111010", "1101111111100",
        ↪  "1101111111111", "1111000000001", "1111000000011", "1111000000101",
        ↪  "1111000000111", "1111000001001", "1111000001011", "1111000001101",
        ↪  "1111000001111", "1111000010001", "1111000010011", "1111000010101",
        ↪  "1111000010111", "1111000011001", "1111000011011", "1111000011101",
        ↪  "1111000011111", "1111000100001", "1111000100011", "1111000100101",
        ↪  "1111000100111", "1111000101001", "1111000101011", "1111000101101",
        ↪  "1111000101111", "1111000110001", "1111000110011", "1111000110101",
        ↪  "1111000110111", "1111000111001", "1111000111011", "1111000111101",
        ↪  "1111000111111", "1111001000001", "1111001000011", "1111001000101",
        ↪  "1111001000111", "1111001001001", "1111001001011", "1111001001101",
        ↪  "1111001001111", "1111001010001", "1111001010011", "1111001010100",
        ↪  "1111001010110", "1111001011000", "1111001011010",
16      "1111001011100", "1111001011110", "1111001100000", "1111001100010", "1111001100100",
        ↪  "1111001100110", "1111001100111", "1111001101001", "1111001101011",
        ↪  "1111001101101", "1111001101111", "1111001110001", "1111001110011",
        ↪  "1111001110100", "1111001110110", "1111001111000", "1111001111010",
        ↪  "1111001111100", "1111001111110", "1111010000000", "1111010000001",
        ↪  "1111010000011", "1111010000101", "1111010000111", "1111010001001",
        ↪  "1111010001010", "1111010001100", "1111010001110", "1111010010000",
        ↪  "1111010010010", "1111010010011", "1111010010101", "1111010010111",
        ↪  "1111010011001", "1111010011010", "1111010011100", "1111010011110",
        ↪  "1111010100000", "1111010100010", "1111010100011", "1111010100101",
        ↪  "1111010100111", "1111010101000", "1111010101010", "1111010101100",
        ↪  "1111010101110", "1111010101111", "1111010110001", "1111010110011",
        ↪  "1111010110101", "1111010110110", "1111010111000", "1111010111010",
        ↪  "1111010111011", "1111010111101", "1111010111111", "1111011000000",
        ↪  "1111011000010", "1111011000100", "1111011000101", "1111011000111",
        ↪  "1111011001001", "1111011001010", "1111011001100", "1111011001110",
        ↪  "1111011001111", "1111011010001", "1111011010011", "1111011010100",
        ↪  "1111011010110", "1111011011000", "1111011011001", "1111011011011",
        ↪  "1111011011100", "1111011011110", "1111011100000", "1111011100001",
        ↪  "1111011100011", "1111011100100", "1111011100110", "1111011101000",
        ↪  "1111011101001", "1111011101011", "1111011101100", "1111011101110",
        ↪  "1111011101111", "1111011110001", "1111011110011", "1111011110100",
        ↪  "1111011110110", "1111011110111", "1111011111001", "1111011111010",
        ↪  "1111011111100", "1111011111101", "1111011111111", "1111100000000",
        ↪  "1111100000010", "1111100000011", "1111100000101",
```

```
17      "1111000000110", "1111000001000", "1111000001001", "1111000001011", "1111000001100",
        ↪  "1111000001110", "1111000001111", "1111000010001", "1111000010010",
        ↪  "1111000010100", "1111000010101", "1111000010111", "1111000011000",
        ↪  "1111000011001", "1111000011011", "1111000011100", "1111000011110",
        ↪  "1111000011111", "1111001000001", "1111001000010", "1111001000011",
        ↪  "1111001001001", "1111001001100", "1111001001101", "1111001001111",
        ↪  "1111001010010", "1111001010011", "1111001010100", "1111001010101",
        ↪  "1111001010110", "1111001010111", "1111001011000", "1111001011001",
        ↪  "1111001011011", "1111001011100", "1111001011101", "1111001011110",
        ↪  "1111001011111", "1111001100000", "1111001100001", "1111001100010",
        ↪  "1111001100011", "1111001100101", "1111001100110", "1111001100111",
        ↪  "1111001101000", "1111001101001", "1111001101010", "1111001101100",
        ↪  "1111001101101", "1111001101110", "1111001101111", "1111001110000",
        ↪  "1111001110001", "1111001110011", "1111001110100", "1111001110101",
        ↪  "1111001110110", "1111001110111", "1111001111000", "1111001111001",
        ↪  "1111001111010", "1111001111100", "1111001111101", "1111001111110",
        ↪  "1111001111111", "1111010000000", "1111010000001", "1111010000010",
        ↪  "1111010000011", "1111010000100", "1111010000101",
18      "1111010000110", "1111010000111", "1111010001000", "1111010001001", "1111010001011",
        ↪  "1111010001100", "1111010001101", "1111010001110", "1111010001111",
        ↪  "1111010010000", "1111010010001", "1111010010010", "1111010010011",
        ↪  "1111010010100", "1111010010101", "1111010010110", "1111010010111",
        ↪  "1111010011000", "1111010011001", "1111010011010", "1111010011011",
        ↪  "1111010011100", "1111010011101", "1111010011101", "1111010011110",
        ↪  "1111010011111", "1111010100000", "1111010100001", "1111010100010",
        ↪  "1111010100011", "1111010100100", "1111010100101", "1111010100110",
        ↪  "1111010100111", "1111010101000", "1111010101001", "1111010101001",
        ↪  "1111010101010", "1111010101011", "1111010101100", "1111010101101",
        ↪  "1111010101110", "1111010101111", "1111010110000", "1111010110000",
        ↪  "1111010110001", "1111010110010", "1111010110011", "1111010110100",
        ↪  "1111010110101", "1111010110110", "1111010110110", "1111010110111",
        ↪  "1111010111000", "1111010111001", "1111010111010", "1111010111010",
        ↪  "1111010111011", "1111010111100", "1111010111101", "1111010111110",
        ↪  "1111010111110", "1111010111111", "1111011000000", "1111011000001",
        ↪  "1111011000001", "1111011000010", "1111011000011", "1111011000100",
        ↪  "1111011000100", "1111011000101", "1111011000110", "1111011000111",
        ↪  "1111011000111", "1111011001000", "1111011001001", "1111011001001",
        ↪  "1111011001010", "1111011001011", "1111011001100", "1111011001100",
        ↪  "1111011001101", "1111011001110", "1111011001110", "1111011001111",
        ↪  "1111011010000", "1111011010000", "1111011010001", "1111011010010",
        ↪  "1111011010010", "1111011010011", "1111011010100", "1111011010100",
        ↪  "1111011010101", "1111011010101", "1111011010110", "1111011010111",
        ↪  "1111011010111", "1111011011000", "1111011011000",
```

```
19    "1111111011001", "1111111011010", "1111111011010", "1111111011011", "1111111011011",
      ↪  "1111111011100", "1111111011101", "1111111011101", "1111111011110",
      ↪  "1111111011110", "1111111011111", "1111111011111", "1111111100000",
      ↪  "1111111100000", "1111111100001", "1111111100001", "1111111100010",
      ↪  "1111111100010", "1111111100011", "1111111100100", "1111111100100",
      ↪  "1111111100101", "1111111100101", "1111111100101", "1111111100110",
      ↪  "1111111100110", "1111111100111", "1111111100111", "1111111101000",
      ↪  "1111111101000", "1111111101001", "1111111101001", "1111111101010",
      ↪  "1111111101010", "1111111101011", "1111111101011", "1111111101011",
      ↪  "1111111101100", "1111111101100", "1111111101101", "1111111101101",
      ↪  "1111111101101", "1111111101110", "1111111101110", "1111111101111",
      ↪  "1111111101111", "1111111101111", "1111111110000", "1111111110000",
      ↪  "1111111110000", "1111111110001", "1111111110001", "1111111110010",
      ↪  "1111111110010", "1111111110010", "1111111110011", "1111111110011",
      ↪  "1111111110011", "1111111110100", "1111111110100", "1111111110100",
      ↪  "1111111110100", "1111111110101", "1111111110101", "1111111110101",
      ↪  "1111111110110", "1111111110110", "1111111110110", "1111111110110",
      ↪  "1111111110111", "1111111110111", "1111111110111", "1111111110111",
      ↪  "1111111111000", "1111111111000", "1111111111000", "1111111111000",
      ↪  "1111111111001", "1111111111001", "1111111111001", "1111111111001",
      ↪  "1111111111010", "1111111111010", "1111111111010", "1111111111010",
      ↪  "1111111111010", "1111111111011", "1111111111011", "1111111111011",
      ↪  "1111111111011", "1111111111011", "1111111111011", "1111111111100",
      ↪  "1111111111100", "1111111111100", "1111111111100", "1111111111100",
      ↪  "1111111111100", "1111111111100", "1111111111100",
20    "1111111111101", "1111111111101", "1111111111101", "1111111111101", "1111111111101",
      ↪  "1111111111101", "1111111111101", "1111111111101", "1111111111101",
      ↪  "1111111111101", "1111111111110", "1111111111110", "1111111111110",
      ↪  "1111111111110", "1111111111110", "1111111111110", "1111111111110",
      ↪  "1111111111110", "1111111111110", "1111111111110", "1111111111110",
      ↪  "1111111111110", "1111111111110", "1111111111110", "1111111111110",
      ↪  "1111111111110", "1111111111110", "1111111111110", "1111111111110",
      ↪  "1111111111110", "1111111111110", "1111111111101", "1111111111101",
      ↪  "1111111111101", "1111111111101", "1111111111101", "1111111111101",
      ↪  "1111111111101", "1111111111101", "1111111111101", "1111111111101",
      ↪  "1111111111100", "1111111111100", "1111111111100", "1111111111100",
      ↪  "1111111111100", "1111111111100", "1111111111100", "1111111111100",
      ↪  "1111111111011", "1111111111011", "1111111111011", "1111111111011",
      ↪  "1111111111011", "1111111111011", "1111111111010", "1111111111010",
      ↪  "1111111111010", "1111111111010", "1111111111010", "1111111111001",
      ↪  "1111111111001", "1111111111001", "1111111111001", "1111111111000",
      ↪  "1111111111000", "1111111111000", "1111111111000", "1111111110111",
      ↪  "1111111110111", "1111111110111", "1111111110111", "1111111110110",
      ↪  "1111111110110", "1111111110110", "1111111110110", "1111111110101",
      ↪  "1111111110101", "1111111110101", "1111111110100", "1111111110100",
      ↪  "1111111110100", "1111111110100", "1111111110011", "1111111110011",
      ↪  "1111111110011", "1111111110010", "1111111110010", "1111111110010",
      ↪  "1111111110001", "1111111110001", "1111111110000",
```

```
21      "1111111110000", "1111111110000", "1111111101111", "1111111101111", "1111111101111",
        ↪  "1111111101110", "1111111101110", "1111111101101", "1111111101101",
        ↪  "1111111101101", "1111111101100", "1111111101100", "1111111101011",
        ↪  "1111111101011", "1111111101011", "1111111101010", "1111111101010",
        ↪  "1111111101001", "1111111101001", "1111111101000", "1111111101000",
        ↪  "1111111100111", "1111111100111", "1111111100110", "1111111100110",
        ↪  "1111111100101", "1111111100101", "1111111100101", "1111111100100",
        ↪  "1111111100100", "1111111100011", "1111111100010", "1111111100010",
        ↪  "1111111100001", "1111111100001", "1111111100000", "1111111100000",
        ↪  "1111111011111", "1111111011111", "1111111011110", "1111111011110",
        ↪  "1111111011101", "1111111011101", "1111111011100", "1111111011011",
        ↪  "1111111011011", "1111111011010", "1111111011010", "1111111011001",
        ↪  "1111111011000", "1111111011000", "1111111010111", "1111111010111",
        ↪  "1111111010110", "1111111010101", "1111111010101", "1111111010100",
        ↪  "1111111010100", "1111111010011", "1111111010010", "1111111010010",
        ↪  "1111111010001", "1111111010000", "1111111010000", "1111111001111",
        ↪  "1111111001110", "1111111001110", "1111111001101", "1111111001100",
        ↪  "1111111001100", "1111111001011", "1111111001010", "1111111001001",
        ↪  "1111111001001", "1111111001000", "1111111000111", "1111111000111",
        ↪  "1111111000110", "1111111000101", "1111111000100", "1111111000100",
        ↪  "1111111000011", "1111111000010", "1111111000001", "1111111000001",
        ↪  "1111111000000", "1111110111111", "1111110111110", "1111110111110",
        ↪  "1111110111101", "1111110111100", "1111110111011", "1111110111010",
        ↪  "1111110111010", "1111110111001", "1111110111000", "1111110110111",
        ↪  "1111110110110", "1111110110110", "1111110110101",
22      "1111110110100", "1111110110011", "1111110110010", "1111110110001", "1111110110000",
        ↪  "1111110110000", "1111110101111", "1111110101110", "1111110101101",
        ↪  "1111110101100", "1111110101011", "1111110101010", "1111110101001",
        ↪  "1111110101001", "1111110101000", "1111110100111", "1111110100110",
        ↪  "1111110100101", "1111110100100", "1111110100011", "1111110100010",
        ↪  "1111110100001", "1111110100000", "1111110011111", "1111110011110",
        ↪  "1111110011101", "1111110011101", "1111110011100", "1111110011011",
        ↪  "1111110011010", "1111110011001", "1111110011000", "1111110010111",
        ↪  "1111110010110", "1111110010101", "1111110010100", "1111110010011",
        ↪  "1111110010010", "1111110010001", "1111110010000", "1111110001111",
        ↪  "1111110001110", "1111110001101", "1111110001100", "1111110001011",
        ↪  "1111110001001", "1111110001000", "1111110000111", "1111110000110",
        ↪  "1111110000101", "1111110000100", "1111110000011", "1111110000010",
        ↪  "1111110000001", "1111110000000", "1111101111111", "1111101111110",
        ↪  "1111101111101", "1111101111100", "1111101111010", "1111101111001",
        ↪  "1111101111000", "1111101110111", "1111101110110", "1111101110101",
        ↪  "1111101110100", "1111101110011", "1111101110001", "1111101110000",
        ↪  "1111101101111", "1111101101110", "1111101101101", "1111101101100",
        ↪  "1111101101010", "1111101101001", "1111101101000", "1111101100111",
        ↪  "1111101100110", "1111101100101", "1111101100011", "1111101100010",
        ↪  "1111101100001", "1111101100000", "1111101011111", "1111101011101",
        ↪  "1111101011100", "1111101011011", "1111101011010", "1111101011000",
        ↪  "1111101010111", "1111101010110", "1111101010101", "1111101010011",
        ↪  "1111101010010", "1111101010001", "1111101010000", "1111101001110",
        ↪  "1111101001101", "1111101001100", "1111101001011",
```

```
23      "111101001001", "111101001000", "111101000111", "111101000101", "111101000100",
        ↪  "111101000011", "111101000001", "111101000000", "111100111111",
        ↪  "111100111101", "111100111100", "111100111011", "111100111001",
        ↪  "111100111000", "111100110111", "111100110101", "111100110100",
        ↪  "111100110011", "111100110001", "111100110000", "111100101111",
        ↪  "111100101101", "111100101100", "111100101010", "111100101001",
        ↪  "111100101000", "111100100110", "111100100101", "111100100011",
        ↪  "111100100010", "111100100001", "111100011111", "111100011110",
        ↪  "111100011100", "111100011011", "111100011001", "111100011000",
        ↪  "111100010111", "111100010101", "111100010100", "111100010010",
        ↪  "111100010001", "111100001111", "111100001110", "111100001100",
        ↪  "111100001011", "111100001001", "111100001000", "111100000110",
        ↪  "111100000101", "111100000011", "111100000010", "111100000000",
        ↪  "111011111111", "111011111101", "111011111100", "111011111010",
        ↪  "111011111001", "111011110111", "111011110110", "111011110100",
        ↪  "111011110011", "111011110001", "111011101111", "111011101110",
        ↪  "111011101100", "111011101011", "111011101001", "111011101000",
        ↪  "111011100110", "111011100100", "111011100011", "111011100001",
        ↪  "111011100000", "111011011110", "111011011100", "111011011011",
        ↪  "111011011001", "111011011000", "111011010110", "111011010100",
        ↪  "111011010011", "111011010001", "111011001111", "111011001110",
        ↪  "111011001100", "111011001010", "111011001001", "111011000111",
        ↪  "111011000101", "111011000100", "111011000010", "111011000000",
        ↪  "111010111111", "111010111101", "111010111011", "111010111010",
        ↪  "111010111000", "111010110110", "111010110101",
24      "111010110011", "111010110001", "111010101111", "111010101110", "111010101100",
        ↪  "111010101010", "111010101000", "111010100111", "111010100101",
        ↪  "111010100011", "111010100010", "111010100000", "111010011110",
        ↪  "111010011100", "111010011010", "111010011001", "111010010111",
        ↪  "111010010101", "111010010011", "111010010010", "111010010000",
        ↪  "111010001110", "111010001100", "111010001010", "111010001001",
        ↪  "111010000111", "111010000101", "111010000011", "111010000001",
        ↪  "111010000000", "111001111110", "111001111100", "111001111010",
        ↪  "111001111000", "111001110110", "111001110100", "111001110011",
        ↪  "111001110001", "111001101111", "111001101101", "111001101011",
        ↪  "111001101001", "111001100111", "111001100110", "111001100100",
        ↪  "111001100010", "111001100000", "111001011110", "111001011100",
        ↪  "111001011010", "111001011000", "111001010110", "111001010100",
        ↪  "111001010011", "111001010001", "111001001111", "111001001101",
        ↪  "111001001011", "111001001001", "111001000111", "111001000101",
        ↪  "111001000011", "111001000001", "111000111111", "111000111101",
        ↪  "111000111011", "111000111001", "111000110111", "111000110101",
        ↪  "111000110011", "111000110001", "111000101111", "111000101101",
        ↪  "111000101011", "111000101001", "111000100111", "111000100101",
        ↪  "111000100011", "111000100001", "111000011111", "111000011101",
        ↪  "111000011011", "111000011001", "111000010111", "111000010101",
        ↪  "111000010011", "111000010001", "111000001111", "111000001101",
        ↪  "111000001011", "111000001001", "111000000111", "111000000101",
        ↪  "111000000011", "111000000001", "110111111111", "110111111100",
        ↪  "110111111010", "110111111000", "110111110110",
```

```
25      "1101111110100", "1101111110010", "1101111110000", "1101111101110", "1101111101100",
        ↪  "1101111101010", "1101111101000", "1101111100101", "1101111100011",
        ↪  "1101111100001", "1101111011111", "1101111011101", "1101111011011",
        ↪  "1101111011001", "1101111010110", "1101111010100", "1101111010010",
        ↪  "1101111010000", "1101111001110", "1101111001100", "1101111001010",
        ↪  "1101111000111", "1101111000101", "1101111000011", "1101111000001",
        ↪  "1101110111111", "1101110111100", "1101110111010", "1101110111000",
        ↪  "1101110110110", "1101110110100", "1101110110010", "1101110101111",
        ↪  "1101110101101", "1101110101011", "1101110101001", "1101110100110",
        ↪  "1101110100100", "1101110100010", "1101110100000", "1101110011110",
        ↪  "1101110011011", "1101110011001", "1101110010111", "1101110010101",
        ↪  "1101110010010", "1101110010000", "1101110001110", "1101110001100",
        ↪  "1101110001001", "1101110000111", "1101110000101", "1101110000010",
        ↪  "1101110000000", "1101101111110", "1101101111100", "1101101111001",
        ↪  "1101101110111", "1101101110101", "1101101110010", "1101101110000",
        ↪  "1101101101110", "1101101101100", "1101101101001", "1101101100111",
        ↪  "1101101100101", "1101101100010", "1101101100000", "1101101011110",
        ↪  "1101101011011", "1101101011001", "1101101010111", "1101101010100",
        ↪  "1101101010010", "1101101010000", "1101101001101", "1101101001011",
        ↪  "1101101001001", "1101101000110", "1101101000100", "1101101000010",
        ↪  "1101100111111", "1101100111101", "1101100111010", "1101100111000",
        ↪  "1101100110110", "1101100110011", "1101100110001", "1101100101111",
        ↪  "1101100101100", "1101100101010", "1101100100111", "1101100100101",
        ↪  "1101100100011", "1101100100000", "1101100011110", "1101100011011",
        ↪  "1101100011001", "1101100010110", "1101100010100",
26      "1101100010010", "1101100001111", "1101100001101", "1101100001010", "1101100001000",
        ↪  "1101100000101", "1101100000011", "1101100000001", "1100111111110",
        ↪  "1100111111100", "1100111111001", "1100111110111", "1100111110100",
        ↪  "1100111110010", "1100111101111", "1100111101101", "1100111101010",
        ↪  "1100111101000", "1100111100101", "1100111100011", "1100111100000",
        ↪  "1100111011110", "1100111011011", "1100111011001", "1100111010110",
        ↪  "1100111010100", "1100111010001", "1100111001111", "1100111001100",
        ↪  "1100111001010", "1100111000111", "1100111000101", "1100111000010",
        ↪  "1100111000000", "1100110111101", "1100110111011", "1100110111000",
        ↪  "1100110110110", "1100110110011", "1100110110001", "1100110101110",
        ↪  "1100110101100", "1100110101001", "1100110100110", "1100110100100",
        ↪  "1100110100001", "1100110011111", "1100110011100", "1100110011010",
        ↪  "1100110010111", "1100110010101", "1100110010010", "1100110001111",
        ↪  "1100110001101", "1100110001010", "1100110001000", "1100110000101",
        ↪  "1100110000010", "1100110000000", "1100001111101", "1100001111011",
        ↪  "1100001111000", "1100001110101", "1100001110011", "1100001110000",
        ↪  "1100001101110", "1100001101011", "1100001101000", "1100001100110",
        ↪  "1100001100011", "1100001100001", "1100001011110", "1100001011011",
        ↪  "1100001011001", "1100001010110", "1100001010011", "1100001010001",
        ↪  "1100001001110", "1100001001011", "1100001001001", "1100001000110",
        ↪  "1100001000011", "1100001000001", "1100000111110", "1100000111100",
        ↪  "1100000111001", "1100000110110", "1100000110100", "1100000110001",
        ↪  "1100000101110", "1100000101011", "1100000101001", "1100000100110",
        ↪  "1100000100011", "1100000100001", "1100000011110", "1100000011011",
        ↪  "1100000011001", "1100000010110", "1100000010011",
```

```
27      "1100000010001", "1100000001110", "1100000001011", "1100000001000", "1100000000110",
        ↪  "1100000000011", "1100000000000", "1011111111110", "1011111111011",
        ↪  "1011111111000", "1011111110101", "1011111110011", "1011111110000",
        ↪  "1011111101101", "1011111101010", "1011111101000", "1011111100101",
        ↪  "1011111100010", "1011111100000", "1011111011101", "1011111011010",
        ↪  "1011111010111", "1011111010101", "1011111010010", "1011111001111",
        ↪  "1011111001100", "1011111001001", "1011111000111", "1011111000100",
        ↪  "1011111000001", "1011110111110", "1011110111100", "1011110111001",
        ↪  "1011110110110", "1011110110011", "1011110110001", "1011110101110",
        ↪  "1011110101011", "1011110101000", "1011110100101", "1011110100011",
        ↪  "1011110100000", "1011110011101", "1011110011010", "1011110010111",
        ↪  "1011110010101", "1011110010010", "1011110001111", "1011110001100",
        ↪  "1011110001001", "1011110000110", "1011110000100", "1011110000001",
        ↪  "1011101111110", "1011101111011", "1011101111000", "1011101110110",
        ↪  "1011101110011", "1011101110000", "1011101101101", "1011101101010",
        ↪  "1011101100111", "1011101100101", "1011101100010", "1011101011111",
        ↪  "1011101011100", "1011101011001", "1011101010110", "1011101010011",
        ↪  "1011101010001", "1011101001110", "1011101001011", "1011101001000",
        ↪  "1011101000101", "1011101000010", "1011100111111", "1011100111101",
        ↪  "1011100111010", "1011100110111", "1011100110100", "1011100110001",
        ↪  "1011100101110", "1011100101011", "1011100101000", "1011100100110",
        ↪  "1011100100011", "1011100100000", "1011100011101", "1011100011010",
        ↪  "1011100010111", "1011100010100", "1011100010001", "1011100001110",
        ↪  "1011100001011", "1011100001001", "1011100000110", "1011100000011",
        ↪  "1011100000000", "1010111111101", "1010111111010",
28      "1010111110111", "1010111110100", "1010111110001", "1010111101110", "1010111101011",
        ↪  "1010111101000", "1010111100110", "1010111100011", "1010111100000",
        ↪  "1010111011101", "1010111011010", "1010111010111", "1010111010100",
        ↪  "1010111010001", "1010111001110", "1010111001011", "1010111001000",
        ↪  "1010111000101", "1010111000010", "1010110111111", "1010110111100",
        ↪  "1010110111001", "1010110110111", "1010110110100", "1010110110001",
        ↪  "1010110101110", "1010110101011", "1010110101000", "1010110100101",
        ↪  "1010110100010", "1010110011111", "1010110011100", "1010110011001",
        ↪  "1010110010110", "1010110010011", "1010110010000", "1010110001101",
        ↪  "1010110001010", "1010110000111", "1010110000100", "1010110000001",
        ↪  "1010001111110", "1010001111011", "1010001111000", "1010001110101",
        ↪  "1010001110010", "1010001101111", "1010001101100", "1010001101001",
        ↪  "1010001100110", "1010001100011", "1010001100000", "1010001011101",
        ↪  "1010001011010", "1010001010111", "1010001010100", "1010001010001",
        ↪  "1010001001110", "1010001001011", "1010001001000", "1010001000101",
        ↪  "1010001000010", "1010000111111", "1010000111100", "1010000111001",
        ↪  "1010000110110", "1010000110011", "1010000110000", "1010000101101",
        ↪  "1010000101010", "1010000100111", "1010000100100", "1010000100001",
        ↪  "1010000011110", "1010000011011", "1010000011000", "1010000010101",
        ↪  "1010000010010", "1010000001111", "1010000001100", "1010000001001",
        ↪  "1010000000110", "1010000000011", "1010000000000", "1001111111101",
        ↪  "1001111111010", "1001111110110", "1001111110011", "1001111110000",
        ↪  "1001111101101", "1001111101010", "1001111100111", "1001111100100",
        ↪  "1001111100001", "1001111011110", "1001111011011", "1001111011000",
        ↪  "1001111010101", "1001111010010", "1001111001111",
```

```
29      "100111001100", "100111001001", "100111000110", "100111000011", "100111000000",
        ↪  "100110111100", "100110111001", "100110110110", "100110110011",
        ↪  "100110110000", "100110101101", "100110101010", "100110100111",
        ↪  "100110100100", "100110100001", "100110011110", "100110011011",
        ↪  "100110011000", "100110010101", "100110010001", "100110001110",
        ↪  "100110001011", "100110001000", "100110000101", "100110000010",
        ↪  "100101111111", "100101111100", "100101111001", "100101110110",
        ↪  "100101110011", "100101110000", "100101101100", "100101101001",
        ↪  "100101100110", "100101100011", "100101100000", "100101011101",
        ↪  "100101011010", "100101010111", "100101010100", "100101010001",
        ↪  "100101001101", "100101001010", "100101000111", "100101000100",
        ↪  "100101000001", "100100111110", "100100111011", "100100111000",
        ↪  "100100110101", "100100110010", "100100101110", "100100101011",
        ↪  "100100101000", "100100100101", "100100100010", "100100011111",
        ↪  "100100011100", "100100011001", "100100010110", "100100010010",
        ↪  "100100001111", "100100001100", "100100001001", "100100000110",
        ↪  "100100000011", "100100000000", "100011111101", "100011111010",
        ↪  "100011110110", "100011110011", "100011110000", "100011101101",
        ↪  "100011101010", "100011100111", "100011100100", "100011100001",
        ↪  "100011011110", "100011011010", "100011010111", "100011010100",
        ↪  "100011010001", "100011001110", "100011001011", "100011001000",
        ↪  "100011000101", "100011000001", "100010111110", "100010111011",
        ↪  "100010111000", "100010110101", "100010110010", "100010101111",
        ↪  "100010101011", "100010101000", "100010100101", "100010100010",
        ↪  "100010011111", "100010011100", "100010011001",
30      "100010010110", "100010010010", "100010001111", "100010001100", "100010001001",
        ↪  "100010000110", "100010000011", "100010000000", "100001111101",
        ↪  "100001111001", "100001110110", "100001110011", "100001110000",
        ↪  "100001101101", "100001101010", "100001100111", "100001100011",
        ↪  "100001100000", "100001011101", "100001011010", "100001010111",
        ↪  "100001010100", "100001010001", "100001001101", "100001001010",
        ↪  "100001000111", "100001000100", "100001000001", "100000111110",
        ↪  "100000111011", "100000111000", "100000110100", "100000110001",
        ↪  "100000101110", "100000101011", "100000101000", "100000100101",
        ↪  "100000100010", "100000011110", "100000011011", "100000011000",
        ↪  "100000010101", "100000010010", "100000001111", "100000001100",
        ↪  "100000001000", "100000000101", "100000000010", "011111111111",
        ↪  "011111111100", "011111111001", "011111110110", "011111110010",
        ↪  "011111101111", "011111101100", "011111101001", "011111100110",
        ↪  "011111100011", "011111100000", "011111011100", "011111011001",
        ↪  "011111010110", "011111010011", "011111010000", "011111001101",
        ↪  "011111001010", "011111000110", "011111000011", "011111000000",
        ↪  "011110111101", "011110111010", "011110110111", "011110110100",
        ↪  "011110110001", "011110101101", "011110101010", "011110100111",
        ↪  "011110100100", "011110100001", "011110011110", "011110011011",
        ↪  "011110010111", "011110010100", "011110010001", "011110001110",
        ↪  "011110001011", "011110001000", "011110000101", "011110000001",
        ↪  "011101111110", "011101111011", "011101111000", "011101110101",
        ↪  "011101110010", "011101101111", "011101101100", "011101101000",
        ↪  "011101100101", "011101100010", "011101011111",
```

```
31      "011101011100", "011101011001", "011101010110", "011101010011", "011101001111",
        ↪  "011101001100", "011101001001", "011101000110", "011101000011",
        ↪  "011101000000", "011100111101", "011100111001", "011100110110",
        ↪  "011100110011", "011100110000", "011100101101", "011100101010",
        ↪  "011100100111", "011100100100", "011100100000", "011100011101",
        ↪  "011100011010", "011100010111", "011100010100", "011100010001",
        ↪  "011100001110", "011100001011", "011100001000", "011100000100",
        ↪  "011100000001", "011011111110", "011011111011", "011011111000",
        ↪  "011011110101", "011011110010", "011011101111", "011011101100",
        ↪  "011011101000", "011011100101", "011011100010", "011011011111",
        ↪  "011011011100", "011011011001", "011011010110", "011011010011",
        ↪  "011011010000", "011011001100", "011011001001", "011011000110",
        ↪  "011011000011", "011011000000", "011010111101", "011010111010",
        ↪  "011010110111", "011010110100", "011010110001", "011010101101",
        ↪  "011010101010", "011010100111", "011010100100", "011010100001",
        ↪  "011010011110", "011010011011", "011010011000", "011010010101",
        ↪  "011010010010", "011010001110", "011010001011", "011010001000",
        ↪  "011010000101", "011010000010", "011001111111", "011001111100",
        ↪  "011001111001", "011001110110", "011001110011", "011001110000",
        ↪  "011001101101", "011001101001", "011001100110", "011001100011",
        ↪  "011001100000", "011001011101", "011001011010", "011001010111",
        ↪  "011001010100", "011001010001", "011001001110", "011001001011",
        ↪  "011001001000", "011001000101", "011001000010", "011000111110",
        ↪  "011000111011", "011000111000", "011000110101", "011000110010",
        ↪  "011000101111", "011000101100", "011000101001",
32      "011000100110", "011000100011", "011000100000", "011000011101", "011000011010",
        ↪  "011000010111", "011000010100", "011000010001", "011000001110",
        ↪  "011000001011", "011000001000", "011000000100", "011000000001",
        ↪  "010111111110", "010111111011", "010111111000", "010111110101",
        ↪  "010111110010", "010111101111", "010111101100", "010111101001",
        ↪  "010111100110", "010111100011", "010111100000", "010111011101",
        ↪  "010111011010", "010111010111", "010111010100", "010111010001",
        ↪  "010111001110", "010111001011", "010111001000", "010111000101",
        ↪  "010111000010", "010110111111", "010110111100", "010110111001",
        ↪  "010110110110", "010110110011", "010110110000", "010110101101",
        ↪  "010110101010", "010110100111", "010110100100", "010110100001",
        ↪  "010110011110", "010110011011", "010110011000", "010110010101",
        ↪  "010110010010", "010110001111", "010110001100", "010110001001",
        ↪  "010110000110", "010110000011", "010110000000", "010101111101",
        ↪  "010101111010", "010101110111", "010101110100", "010101110001",
        ↪  "010101101110", "010101101011", "010101101000", "010101100101",
        ↪  "010101100010", "010101011111", "010101011100", "010101011001",
        ↪  "010101010110", "010101010011", "010101010000", "010101001101",
        ↪  "010101001010", "010101000111", "010101000101", "010101000010",
        ↪  "010100111111", "010100111100", "010100111001", "010100110110",
        ↪  "010100110011", "010100110000", "010100101101", "010100101010",
        ↪  "010100100111", "010100100100", "010100100001", "010100011110",
        ↪  "010100011011", "010100011000", "010100010110", "010100010011",
        ↪  "010100010000", "010100001101", "010100001010", "010100000111",
        ↪  "010100000100", "010100000001", "010011111110",
```

```
33      "0100111111011", "0100111111000", "0100111110101", "0100111110011", "0100111110000",
        ↪  "0100111101101", "0100111101010", "0100111100111", "0100111100100",
        ↪  "0100111100001", "0100111011110", "0100111011011", "0100111011000",
        ↪  "0100111010110", "0100111010011", "0100111010000", "0100111001101",
        ↪  "0100111001010", "0100111000111", "0100111000100", "0100111000001",
        ↪  "0100110111111", "0100110111100", "0100110111001", "0100110110110",
        ↪  "0100110110011", "0100110110000", "0100110101101", "0100110101011",
        ↪  "0100110101000", "0100110100101", "0100110100010", "0100110011111",
        ↪  "0100110011100", "0100110011001", "0100110010111", "0100110010100",
        ↪  "0100110010001", "0100110001110", "0100110001011", "0100110001000",
        ↪  "0100110000110", "0100110000011", "0100110000000", "0100001111101",
        ↪  "0100001111010", "0100001111000", "0100001110101", "0100001110010",
        ↪  "0100001101111", "0100001101100", "0100001101001", "0100001100111",
        ↪  "0100001100100", "0100001100001", "0100001011110", "0100001011011",
        ↪  "0100001011001", "0100001010110", "0100001010011", "0100001010000",
        ↪  "0100001001101", "0100001001011", "0100001001000", "0100001000101",
        ↪  "0100001000010", "0100001000000", "0100000111101", "0100000111010",
        ↪  "0100000110111", "0100000110101", "0100000110010", "0100000101111",
        ↪  "0100000101100", "0100000101001", "0100000100111", "0100000100100",
        ↪  "0100000100001", "0100000011110", "0100000011100", "0100000011001",
        ↪  "0100000010110", "0100000010100", "0100000010001", "0100000001110",
        ↪  "0100000001011", "0100000001001", "0100000000110", "0100000000011",
        ↪  "0100000000000", "0011111111110", "0011111111011", "0011111111000",
        ↪  "0011111110110", "0011111110011", "0011111110000", "0011111101101",
        ↪  "0011111101011", "0011111101000", "0011111100101",
34      "0011111100011", "0011111100000", "0011111011101", "0011111011011", "0011111011000",
        ↪  "0011111010101", "0011111010011", "0011111010000", "0011111001101",
        ↪  "0011111001010", "0011111001000", "0011111000101", "0011111000010",
        ↪  "0011111000000", "0011110111101", "0011110111011", "0011110111000",
        ↪  "0011110110101", "0011110110011", "0011110110000", "0011110101101",
        ↪  "0011110101011", "0011110101000", "0011110100101", "0011110100011",
        ↪  "0011110100000", "0011110011101", "0011110011011", "0011110011000",
        ↪  "0011110010110", "0011110010011", "0011110010000", "0011110001110",
        ↪  "0011110001011", "0011110001001", "0011110000110", "0011110000011",
        ↪  "0011110000001", "0011101111110", "0011101111100", "0011101111001",
        ↪  "0011101110110", "0011101110100", "0011101110001", "0011101101111",
        ↪  "0011101101100", "0011101101001", "0011101100111", "0011101100100",
        ↪  "0011101100010", "0011101011111", "0011101011101", "0011101011010",
        ↪  "0011101011000", "0011101010101", "0011101010010", "0011101010000",
        ↪  "0011101001101", "0011101001011", "0011101001000", "0011101000110",
        ↪  "0011101000011", "0011101000001", "0011100111110", "0011100111100",
        ↪  "0011100111001", "0011100110111", "0011100110100", "0011100110010",
        ↪  "0011100101111", "0011100101101", "0011100101010", "0011100101000",
        ↪  "0011100100101", "0011100100011", "0011100100000", "0011100011110",
        ↪  "0011100011011", "0011100011001", "0011100010110", "0011100010100",
        ↪  "0011100010001", "0011100001111", "0011100001100", "0011100001010",
        ↪  "0011100000111", "0011100000101", "0011100000010", "0011100000000",
        ↪  "0010111111101", "0010111111011", "0010111111001", "0010111110110",
        ↪  "0010111110100", "0010111110001", "0010111101111", "0010111101100",
        ↪  "0010111101010", "0010111101000", "0010111100101",
```

```
35      "0010111100011", "0010111100000", "0010111011110", "0010111011011", "0010111011001",
        ↪  "0010111010111", "0010111010100", "0010111010010", "0010111001111",
        ↪  "0010111001101", "0010111001011", "0010111001000", "0010111000110",
        ↪  "0010111000100", "0010111000001", "0010110111111", "0010110111100",
        ↪  "0010110111010", "0010110111000", "0010110110101", "0010110110011",
        ↪  "0010110110001", "0010110101110", "0010110101100", "0010110101010",
        ↪  "0010110100111", "0010110100101", "0010110100011", "0010110100000",
        ↪  "0010110011110", "0010110011100", "0010110011001", "0010110010111",
        ↪  "0010110010101", "0010110010010", "0010110010000", "0010110001110",
        ↪  "0010110001100", "0010110001001", "0010110000111", "0010110000101",
        ↪  "0010110000010", "0010110000000", "0010011111110", "0010011111100",
        ↪  "0010011111001", "0010011110111", "0010011110101", "0010011110010",
        ↪  "0010011110000", "0010011101110", "0010011101100", "0010011101001",
        ↪  "0010011100111", "0010011100101", "0010011100011", "0010011100000",
        ↪  "0010011011110", "0010011011100", "0010011011010", "0010011011000",
        ↪  "0010011010101", "0010011010011", "0010011010001", "0010011001111",
        ↪  "0010011001100", "0010011001010", "0010011001000", "0010011000110",
        ↪  "0010011000100", "0010011000010", "0010010111111", "0010010111101",
        ↪  "0010010111011", "0010010111001", "0010010110111", "0010010110100",
        ↪  "0010010110010", "0010010110000", "0010010101110", "0010010101100",
        ↪  "0010010101010", "0010010101000", "0010010100101", "0010010100011",
        ↪  "0010010100001", "0010010011111", "0010010011101", "0010010011011",
        ↪  "0010010011001", "0010010010110", "0010010010100", "0010010010010",
        ↪  "0010010010000", "0010010001110", "0010010001100", "0010010001010",
        ↪  "0010010001000", "0010010000110", "0010010000100",
36      "0010000000010", "0001111111111", "0001111111101", "0001111111011", "0001111111001",
        ↪  "0001111110111", "0001111110101", "0001111110011", "0001111110001",
        ↪  "0001111101111", "0001111101101", "0001111101011", "0001111101001",
        ↪  "0001111100111", "0001111100101", "0001111100011", "0001111100001",
        ↪  "0001111011111", "0001111011101", "0001111011011", "0001111011001",
        ↪  "0001111010111", "0001111010101", "0001111010011", "0001111010001",
        ↪  "0001111001111", "0001111001101", "0001111001011", "0001111001001",
        ↪  "0001111000111", "0001111000101", "0001111000011", "0001111000001",
        ↪  "0001110111111", "0001110111101", "0001110111011", "0001110111001",
        ↪  "0001110110111", "0001110110101", "0001110110011", "0001110110001",
        ↪  "0001110101111", "0001110101101", "0001110101011", "0001110101010",
        ↪  "0001110101000", "0001110100110", "0001110100100", "0001110100010",
        ↪  "0001110100000", "0001110011110", "0001110011100", "0001110011010",
        ↪  "0001110011000", "0001110010111", "0001110010101", "0001110010011",
        ↪  "0001110010001", "0001110001111", "0001110001101", "0001110001011",
        ↪  "0001110001010", "0001110001000", "0001110000110", "0001110000100",
        ↪  "0001110000010", "0001110000000", "0001101111110", "0001101111101",
        ↪  "0001101111011", "0001101111001", "0001101110111", "0001101110101",
        ↪  "0001101110100", "0001101110010", "0001101110000", "0001101101110",
        ↪  "0001101101100", "0001101101011", "0001101101001", "0001101100111",
        ↪  "0001101100101", "0001101100100", "0001101100010", "0001101100000",
        ↪  "0001101011110", "0001101011100", "0001101011011", "0001101011001",
        ↪  "0001101010111", "0001101010110", "0001101010100", "0001101010010",
        ↪  "0001101010000", "0001101001111", "0001101001101", "0001101001011",
        ↪  "0001101001001", "0001101001000", "0001101000110",
```

```
37      "0001010000100", "0001010000011", "0001010000001", "0001000111111", "0001000111110",
        ↪  "0001000111100", "0001000111010", "0001000111001", "0001000110111",
        ↪  "0001000110101", "0001000110100", "0001000110010", "0001000110000",
        ↪  "0001000101111", "0001000101101", "0001000101011", "0001000101010",
        ↪  "0001000101000", "0001000100110", "0001000100101", "0001000100011",
        ↪  "0001000100010", "0001000100000", "0001000011110", "0001000011101",
        ↪  "0001000011011", "0001000011010", "0001000011000", "0001000010110",
        ↪  "0001000010101", "0001000010011", "0001000010010", "0001000010000",
        ↪  "0001000001111", "0001000001101", "0001000001011", "0001000001010",
        ↪  "0001000001000", "0001000000111", "0001000000101", "0001000000100",
        ↪  "0001000000010", "0001000000001", "0000111111111", "0000111111110",
        ↪  "0000111111100", "0000111111011", "0000111111001", "0000111111000",
        ↪  "0000111110110", "0000111110101", "0000111110011", "0000111110010",
        ↪  "0000111110000", "0000111101111", "0000111101101", "0000111101100",
        ↪  "0000111101010", "0000111101001", "0000111100111", "0000111100110",
        ↪  "0000111100101", "0000111100011", "0000111100010", "0000111100000",
        ↪  "0000111011111", "0000111011101", "0000111011100", "0000111011011",
        ↪  "0000111011001", "0000111011000", "0000111010110", "0000111010101",
        ↪  "0000111010100", "0000111010010", "0000111010001", "0000111001111",
        ↪  "0000111001110", "0000111001101", "0000111001011", "0000111001010",
        ↪  "0000111001001", "0000111000111", "0000111000110", "0000111000101",
        ↪  "0000111000011", "0000111000010", "0000111000001", "0000110111111",
        ↪  "0000110111110", "0000110111101", "0000110111011", "0000110111010",
        ↪  "0000110111001", "0000110110111", "0000110110110", "0000110110101",
        ↪  "0000110110011", "0000110110010", "0000110110001",
38      "0000110110000", "0000110101110", "0000110101101", "0000110101100", "0000110101011",
        ↪  "0000110101001", "0000110101000", "0000110100111", "0000110100110",
        ↪  "0000110100100", "0000110100011", "0000110100010", "0000110100001",
        ↪  "0000110011111", "0000110011110", "0000110011101", "0000110011100",
        ↪  "0000110011011", "0000110011001", "0000110011000", "0000110010111",
        ↪  "0000110010110", "0000110010101", "0000110010100", "0000110010010",
        ↪  "0000110010001", "0000110010000", "0000110001111", "0000110001110",
        ↪  "0000110001101", "0000110001011", "0000110001010", "0000110001001",
        ↪  "0000110001000", "0000110000111", "0000110000110", "0000110000101",
        ↪  "0000110000100", "0000110000010", "0000110000001", "0000110000000",
        ↪  "0000101111111", "0000101111110", "0000101111101", "0000101111100",
        ↪  "0000101111011", "0000101111010", "0000101111001", "0000101111000",
        ↪  "0000101110111", "0000101110110", "0000101110101", "0000101110011",
        ↪  "0000101110010", "0000101110001", "0000101110000", "0000101101111",
        ↪  "0000101101110", "0000101101101", "0000101101100", "0000101101011",
        ↪  "0000101101010", "0000101101001", "0000101101000", "0000101100111",
        ↪  "0000101100110", "0000101100101", "0000101100100", "0000101100011",
        ↪  "0000101100010", "0000101100001", "0000101100001", "0000101100000",
        ↪  "0000101011111", "0000101011110", "0000101011101", "0000101011100",
        ↪  "0000101011011", "0000101011010", "0000101011001", "0000101011000",
        ↪  "0000101010111", "0000101010110", "0000101010101", "0000101010101",
        ↪  "0000101010100", "0000101010011", "0000101010010", "0000101010001",
        ↪  "0000101010000", "0000101001111", "0000101001110", "0000101001110",
        ↪  "0000101001101", "0000101001100", "0000101001011", "0000101001010",
        ↪  "0000101001001", "0000101001000", "0000101001000",
```

```
39      "0000001000111", "0000001000110", "0000001000101", "0000001000100", "0000001000100",
        ↪  "0000001000011", "0000001000010", "0000001000001", "0000001000000",
        ↪  "0000001000000", "0000000111111", "0000000111110", "0000000111101",
        ↪  "0000000111101", "0000000111100", "0000000111011", "0000000111010",
        ↪  "0000000111010", "0000000111001", "0000000111000", "0000000110111",
        ↪  "0000000110111", "0000000110110", "0000000110101", "0000000110101",
        ↪  "0000000110100", "0000000110011", "0000000110010", "0000000110010",
        ↪  "0000000110001", "0000000110000", "0000000110000", "0000000101111",
        ↪  "0000000101110", "0000000101110", "0000000101101", "0000000101100",
        ↪  "0000000101100", "0000000101011", "0000000101010", "0000000101010",
        ↪  "0000000101001", "0000000101001", "0000000101000", "0000000100111",
        ↪  "0000000100111", "0000000100110", "0000000100110", "0000000100101",
        ↪  "0000000100100", "0000000100100", "0000000100011", "0000000100011",
        ↪  "0000000100010", "0000000100001", "0000000100001", "0000000100000",
        ↪  "0000000100000", "0000000011111", "0000000011111", "0000000011110",
        ↪  "0000000011110", "0000000011101", "0000000011101", "0000000011100",
        ↪  "0000000011100", "0000000011011", "0000000011010", "0000000011010",
        ↪  "0000000011001", "0000000011001", "0000000011001", "0000000011000",
        ↪  "0000000011000", "0000000010111", "0000000010111", "0000000010110",
        ↪  "0000000010110", "0000000010101", "0000000010101", "0000000010100",
        ↪  "0000000010100", "0000000010011", "0000000010011", "0000000010011",
        ↪  "0000000010010", "0000000010010", "0000000010001", "0000000010001",
        ↪  "0000000010001", "0000000010000", "0000000010000", "0000000001111",
        ↪  "0000000001111", "0000000001111", "0000000001110", "0000000001110",
        ↪  "0000000001110", "0000000001101", "0000000001101",
40      "0000000001100", "0000000001100", "0000000001100", "0000000001011", "0000000001011",
        ↪  "0000000001011", "0000000001010", "0000000001010", "0000000001010",
        ↪  "0000000001010", "0000000001001", "0000000001001", "0000000001001",
        ↪  "0000000001000", "0000000001000", "0000000001000", "0000000001000",
        ↪  "0000000000111", "0000000000111", "0000000000111", "0000000000111",
        ↪  "0000000000110", "0000000000110", "0000000000110", "0000000000110",
        ↪  "0000000000101", "0000000000101", "0000000000101", "0000000000101",
        ↪  "0000000000100", "0000000000100", "0000000000100", "0000000000100",
        ↪  "0000000000100", "0000000000011", "0000000000011", "0000000000011",
        ↪  "0000000000011", "0000000000011", "0000000000011", "0000000000010",
        ↪  "0000000000010", "0000000000010", "0000000000010", "0000000000010",
        ↪  "0000000000010", "0000000000010", "0000000000010", "0000000000001",
        ↪  "0000000000001", "0000000000001", "0000000000001", "0000000000001",
        ↪  "0000000000001", "0000000000001", "0000000000001", "0000000000001",
        ↪  "0000000000001", "0000000000000", "0000000000000", "0000000000000",
        ↪  "0000000000000", "0000000000000", "0000000000000", "0000000000000",
        ↪  "0000000000000", "0000000000000", "0000000000000", "0000000000000",
        ↪  "0000000000000", "0000000000000", "0000000000000", "0000000000000",
        ↪  "0000000000000", "0000000000000", "0000000000000", "0000000000000",
        ↪  "0000000000000", "0000000000000", "0000000000001", "0000000000001",
        ↪  "0000000000001", "0000000000001", "0000000000001", "0000000000001",
        ↪  "0000000000001", "0000000000001", "0000000000001", "0000000000001",
        ↪  "0000000000010", "0000000000010", "0000000000010",
```

```
41      "000000000010", "000000000010", "000000000010", "000000000010", "000000000010",
        ↪  "000000000011", "000000000011", "000000000011", "000000000011",
        ↪  "000000000011", "000000000011", "000000000100", "000000000100",
        ↪  "000000000100", "000000000100", "000000000100", "000000000101",
        ↪  "000000000101", "000000000101", "000000000101", "000000000110",
        ↪  "000000000110", "000000000110", "000000000110", "000000000111",
        ↪  "000000000111", "000000000111", "000000000111", "000000001000",
        ↪  "000000001000", "000000001000", "000000001000", "000000001001",
        ↪  "000000001001", "000000001001", "000000001010", "000000001010",
        ↪  "000000001010", "000000001010", "000000001011", "000000001011",
        ↪  "000000001011", "000000001100", "000000001100", "000000001100",
        ↪  "000000001101", "000000001101", "000000001110", "000000001110",
        ↪  "000000001110", "000000001111", "000000001111", "000000001111",
        ↪  "000000010000", "000000010000", "000000010001", "000000010001",
        ↪  "000000010001", "000000010010", "000000010010", "000000010011",
        ↪  "000000010011", "000000010011", "000000010100", "000000010100",
        ↪  "000000010101", "000000010101", "000000010110", "000000010110",
        ↪  "000000010111", "000000010111", "000000011000", "000000011000",
        ↪  "000000011001", "000000011001", "000000011001", "000000011010",
        ↪  "000000011010", "000000011011", "000000011100", "000000011100",
        ↪  "000000011101", "000000011101", "000000011110", "000000011110",
        ↪  "000000011111", "000000011111", "000000100000", "000000100000",
        ↪  "000000100001", "000000100001", "000000100010", "000000100011",
        ↪  "000000100011", "000000100100", "000000100100", "000000100101",
        ↪  "000000100110", "000000100110", "000000100111",
42      "000000100111", "000000101000", "000000101001", "000000101001", "000000101010",
        ↪  "000000101010", "000000101011", "000000101100", "000000101100",
        ↪  "000000101101", "000000101110", "000000101110", "000000101111",
        ↪  "000000110000", "000000110000", "000000110001", "000000110010",
        ↪  "000000110010", "000000110011", "000000110100", "000000110101",
        ↪  "000000110101", "000000110110", "000000110111", "000000110111",
        ↪  "000000111000", "000000111001", "000000111010", "000000111010",
        ↪  "000000111011", "000000111100", "000000111101", "000000111101",
        ↪  "000000111110", "000000111111", "000001000000", "000001000000",
        ↪  "000001000001", "000001000010", "000001000011", "000001000100",
        ↪  "000001000100", "000001000101", "000001000110", "000001000111",
        ↪  "000001001000", "000001001000", "000001001001", "000001001010",
        ↪  "000001001011", "000001001100", "000001001101", "000001001110",
        ↪  "000001001110", "000001001111", "000001010000", "000001010001",
        ↪  "000001010010", "000001010011", "000001010100", "000001010101",
        ↪  "000001010101", "000001010110", "000001010111", "000001011000",
        ↪  "000001011001", "000001011010", "000001011011", "000001011100",
        ↪  "000001011101", "000001011110", "000001011111", "000001100000",
        ↪  "000001100001", "000001100001", "000001100010", "000001100011",
        ↪  "000001100100", "000001100101", "000001100110", "000001100111",
        ↪  "000001101000", "000001101001", "000001101010", "000001101011",
        ↪  "000001101100", "000001101101", "000001101110", "000001101111",
        ↪  "000001110000", "000001110001", "000001110010", "000001110011",
        ↪  "000001110101", "000001110110", "000001110111", "000001111000",
        ↪  "000001111001", "000001111010", "000001111011",
```

```
43      "0000001111100", "0000001111101", "0000001111110", "0000001111111", "0000010000000",
        ↪  "0000010000001", "0000010000010", "0000010000100", "0000010000101",
        ↪  "0000010000110", "0000010000111", "0000010001000", "0000010001001",
        ↪  "0000010001010", "0000010001011", "0000010001101", "0000010001110",
        ↪  "0000010001111", "0000010010000", "0000010010001", "0000010010010",
        ↪  "0000010010100", "0000010010101", "0000010010110", "0000010010111",
        ↪  "0000010011000", "0000010011001", "0000010011011", "0000010011100",
        ↪  "0000010011101", "0000010011110", "0000010011111", "0000010100001",
        ↪  "0000010100010", "0000010100011", "0000010100100", "0000010100110",
        ↪  "0000010100111", "0000010101000", "0000010101001", "0000010101011",
        ↪  "0000010101100", "0000010101101", "0000010101110", "0000010110000",
        ↪  "0000010110001", "0000010110010", "0000010110011", "0000010110101",
        ↪  "0000010110110", "0000010110111", "0000010111001", "0000010111010",
        ↪  "0000010111011", "0000010111101", "0000010111110", "0000010111111",
        ↪  "0000011000001", "0000011000010", "0000011000011", "0000011000101",
        ↪  "0000011000110", "0000011000111", "0000011001001", "0000011001010",
        ↪  "0000011001011", "0000011001101", "0000011001110", "0000011001111",
        ↪  "0000011010001", "0000011010010", "0000011010100", "0000011010101",
        ↪  "0000011010110", "0000011011000", "0000011011001", "0000011011011",
        ↪  "0000011011100", "0000011011101", "0000011011111", "0000011100000",
        ↪  "0000011100010", "0000011100011", "0000011100101", "0000011100110",
        ↪  "0000011100111", "0000011101001", "0000011101010", "0000011101100",
        ↪  "0000011101101", "0000011101111", "0000011110000", "0000011110010",
        ↪  "0000011110011", "0000011110101", "0000011110110", "0000011111000",
        ↪  "0000011111001", "0000011111011", "0000011111100",
44      "0000011111110", "0000011111111", "0001000000001", "0001000000010", "0001000000100",
        ↪  "0001000000101", "0001000000111", "0001000001000", "0001000001010",
        ↪  "0001000001011", "0001000001101", "0001000001111", "0001000010000",
        ↪  "0001000010010", "0001000010011", "0001000010101", "0001000010110",
        ↪  "0001000011000", "0001000011010", "0001000011011", "0001000011101",
        ↪  "0001000011110", "0001000100000", "0001000100010", "0001000100011",
        ↪  "0001000100101", "0001000100110", "0001000101000", "0001000101010",
        ↪  "0001000101011", "0001000101101", "0001000101111", "0001000110000",
        ↪  "0001000110010", "0001000110100", "0001000110101", "0001000110111",
        ↪  "0001000111001", "0001000111010", "0001000111100", "0001000111110",
        ↪  "0001000111111", "0001010000001", "0001010000011", "0001010000100",
        ↪  "0001010001000", "0001010001001", "0001010001011", "0001010001101",
        ↪  "0001010001111", "0001010010000", "0001010010010", "0001010010011",
        ↪  "0001010010100", "0001010010110", "0001010010111", "0001010011001",
        ↪  "0001010011011", "0001010011100", "0001010011110", "0001011000000",
        ↪  "0001011000010", "0001011000011", "0001011000101", "0001011000110",
        ↪  "0001011001000", "0001011001010", "0001011001011", "0001011001101",
        ↪  "0001011001110", "0001011001111", "0001011010000", "0001011010010",
        ↪  "0001011010011", "0001011010101", "0001011010110", "0001011010111",
        ↪  "0001011011001", "0001011011011", "0001011011100", "0001011011110",
        ↪  "0001011100000", "0001011100010", "0001011100011", "0001011100101",
        ↪  "0001011100110", "0001011100111", "0001011101001", "0001011101011",
        ↪  "0001011101100", "0001011101110", "0001011101111", "0001011110001",
        ↪  "0001011110010", "0001011110011", "0001011110101", "0001011110110",
        ↪  "0001011110111", "0001011111001", "0001011111011", "0001011111101",
        ↪  "0001011111110", "0001100000000", "0001100000010", "0001100000100",
        ↪  "0001100000110", "0001100001000", "0001100001010", "0001100001011",
        ↪  "0001100001101", "0001100001111", "0001100010001", "0001100010011",
        ↪  "0001100010101", "0001100010111", "0001100011000", "0001100011010",
        ↪  "0001100011100", "0001100011110", "0001100100000", "0001100100010",
        ↪  "0001100100100", "0001100100110", "0001100101000",
```

```
45      "000110101010", "000110101011", "000110101101", "000110101111", "000110110001",
        ↪  "000110110011", "000110110101", "000110110111", "000110111001",
        ↪  "000110111011", "000110111101", "000110111111", "000111000001",
        ↪  "000111000011", "000111000101", "000111000111", "000111001001",
        ↪  "000111001011", "000111001101", "000111001111", "000111010001",
        ↪  "000111010011", "000111010101", "000111010111", "000111011001",
        ↪  "000111011011", "000111011101", "000111011111", "000111100001",
        ↪  "000111100011", "000111100101", "000111100111", "000111101001",
        ↪  "000111101011", "000111101101", "000111101111", "000111110001",
        ↪  "000111110011", "000111110101", "000111110111", "000111111001",
        ↪  "000111111011", "000111111101", "000111111111", "001000000010",
        ↪  "001000000100", "001000000110", "001000001000", "001000001010",
        ↪  "001000001100", "001000001110", "001000010000", "001000010010",
        ↪  "001000010100", "001000010110", "001000011001", "001000011011",
        ↪  "001000011101", "001000011111", "001000100001", "001000100011",
        ↪  "001000100101", "001000101000", "001000101010", "001000101100",
        ↪  "001000101110", "001000110000", "001000110010", "001000110100",
        ↪  "001000110111", "001000111001", "001000111011", "001000111101",
        ↪  "001000111111", "001001000010", "001001000100", "001001000110",
        ↪  "001001001000", "001001001010", "001001001100", "001001001111",
        ↪  "001001010001", "001001010011", "001001010101", "001001011000",
        ↪  "001001011010", "001001011100", "001001011110", "001001100000",
        ↪  "001001100011", "001001100101", "001001100111", "001001101001",
        ↪  "001001101100", "001001101110", "001001110000", "001001110010",
        ↪  "001001110101", "001001110111", "001001111001",
46      "001001111100", "001001111110", "001010000000", "001010000010", "001010000101",
        ↪  "001010000111", "001010001001", "001010001100", "001010001110",
        ↪  "001010010000", "001010010010", "001010010101", "001010010111",
        ↪  "001010011001", "001010011100", "001010011110", "001010100000",
        ↪  "001010100011", "001010100101", "001010100111", "001010101010",
        ↪  "001010101100", "001010101110", "001010110001", "001010110011",
        ↪  "001010110101", "001010111000", "001010111010", "001010111100",
        ↪  "001010111111", "001011000001", "001011000100", "001011000110",
        ↪  "001011001000", "001011001011", "001011001101", "001011001111",
        ↪  "001011010010", "001011010100", "001011010111", "001011011001",
        ↪  "001011011011", "001011011110", "001011100000", "001011100011",
        ↪  "001011100101", "001011101000", "001011101010", "001011101100",
        ↪  "001011101111", "001011110001", "001011110100", "001011110110",
        ↪  "001011111001", "001011111011", "001011111101", "001100000000",
        ↪  "001100000010", "001100000101", "001100000111", "001100001010",
        ↪  "001100001100", "001100001111", "001100010001", "001100010100",
        ↪  "001100010110", "001100011001", "001100011011", "001100011110",
        ↪  "001100100000", "001100100011", "001100100101", "001100101000",
        ↪  "001100101010", "001100101101", "001100101111", "001100110010",
        ↪  "001100110100", "001100110111", "001100111001", "001100111100",
        ↪  "001100111110", "001101000001", "001101000011", "001101000110",
        ↪  "001101001000", "001101001011", "001101001101", "001101010000",
        ↪  "001101010010", "001101010101", "001101011000", "001101011010",
        ↪  "001101011101", "001101011111", "001101100010", "001101100100",
        ↪  "001101100111", "001101101001", "001101101100",
```

```
47      "001101101111", "001101110001", "001101110100", "001101110110", "001101111001",
        ↪  "001101111100", "001101111110", "001110000001", "001110000011",
        ↪  "001110000110", "001110001001", "001110001011", "001110001110",
        ↪  "001110010000", "001110010011", "001110010110", "001110011000",
        ↪  "001110011011", "001110011101", "001110100000", "001110100011",
        ↪  "001110100101", "001110101000", "001110101011", "001110101101",
        ↪  "001110110000", "001110110011", "001110110101", "001110111000",
        ↪  "001110111011", "001110111101", "001111000000", "001111000010",
        ↪  "001111000101", "001111001000", "001111001010", "001111001101",
        ↪  "001111010000", "001111010011", "001111010101", "001111011000",
        ↪  "001111011011", "001111011101", "001111100000", "001111100011",
        ↪  "001111100101", "001111101000", "001111101011", "001111101101",
        ↪  "001111110000", "001111110011", "001111110110", "001111111000",
        ↪  "001111111011", "001111111110", "010000000000", "010000000011",
        ↪  "010000000110", "010000001001", "010000001011", "010000001110",
        ↪  "010000010001", "010000010100", "010000010110", "010000011001",
        ↪  "010000011100", "010000011110", "010000100001", "010000100100",
        ↪  "010000100111", "010000101001", "010000101100", "010000101111",
        ↪  "010000110010", "010000110101", "010000110111", "010000111010",
        ↪  "010000111101", "010001000000", "010001000010", "010001000101",
        ↪  "010001001000", "010001001011", "010001001101", "010001010000",
        ↪  "010001010011", "010001010110", "010001011001", "010001011011",
        ↪  "010001011110", "010001100001", "010001100100", "010001100111",
        ↪  "010001101001", "010001101100", "010001101111", "010001110010",
        ↪  "010001110101", "010001111000", "010001111010",
48      "010001111101", "010010000000", "010010000011", "010010000110", "010010001000",
        ↪  "010010001011", "010010001110", "010010010001", "010010010100",
        ↪  "010010010111", "010010011001", "010010011100", "010010011111",
        ↪  "010010100010", "010010100101", "010010101000", "010010101011",
        ↪  "010010101101", "010010110000", "010010110011", "010010110110",
        ↪  "010010111001", "010010111100", "010010111111", "010011000001",
        ↪  "010011000100", "010011000111", "010011001010", "010011001101",
        ↪  "010011010000", "010011010011", "010011010110", "010011011000",
        ↪  "010011011011", "010011011110", "010011100001", "010011100100",
        ↪  "010011100111", "010011101010", "010011101101", "010011110000",
        ↪  "010011110011", "010011110101", "010011111000", "010011111011",
        ↪  "010011111110", "010100000001", "010100000100", "010100000111",
        ↪  "010100001010", "010100001101", "010100010000", "010100010011",
        ↪  "010100010110", "010100011000", "010100011011", "010100011110",
        ↪  "010100100001", "010100100100", "010100100111", "010100101010",
        ↪  "010100101101", "010100110000", "010100110011", "010100110110",
        ↪  "010100111001", "010100111100", "010100111111", "010101000010",
        ↪  "010101000101", "010101000111", "010101001010", "010101001101",
        ↪  "010101010000", "010101010011", "010101010110", "010101011001",
        ↪  "010101011100", "010101011111", "010101100010", "010101100101",
        ↪  "010101101000", "010101101011", "010101101110", "010101110001",
        ↪  "010101110100", "010101110111", "010101111010", "010101111101",
        ↪  "010110000000", "010110000011", "010110000110", "010110001001",
        ↪  "010110001100", "010110001111", "010110010010", "010110010101",
        ↪  "010110011000", "010110011011", "010110011110",
```

```
49      "0101101000001", "0101101001000", "0101101001111", "0101101010010", "0101101011101",
        ↪  "0101101100000", "0101101100011", "0101101100110", "0101101101001",
        ↪  "0101101110000", "0101101110011", "0101101110110", "0101101111001",
        ↪  "0101110000010", "0101110000101", "0101110001000", "0101110001011",
        ↪  "0101110001110", "0101110010001", "0101110010100", "0101110010111",
        ↪  "0101110011010", "0101110011101", "0101110100000", "0101110100011",
        ↪  "0101110100110", "0101110101001", "0101110101100", "0101110101111",
        ↪  "0101110110010", "0101110110101", "0101110111000", "0101110111011",
        ↪  "0101110111110", "0101111000001", "0101111000100", "0101111000111",
        ↪  "0101111001010", "0101111001101", "0101111010000", "0101111010011",
        ↪  "0101111010110", "0101111011001", "0101111011100", "0101111011111",
        ↪  "0101111100010", "0101111100101", "0101111101000", "0101111101011",
        ↪  "0101111101110", "0101111110001", "0101111110100", "0101111110111",
        ↪  "0101111111010", "0101111111101", "0110000000000", "0110000000011",
        ↪  "0110000000110", "0110000001001", "0110000001100", "0110000001111",
        ↪  "0110000010010", "0110000010101", "0110000011000", "0110000011011",
        ↪  "0110000011110", "0110000100001", "0110000100100", "0110000100111",
        ↪  "0110000101010", "0110000101101", "0110000110000", "0110000110011",
        ↪  "0110000110110", "0110000111001", "0110000111100", "0110000111111",
        ↪  "0110001000010", "0110001000101", "0110001001000", "0110001001011",
        ↪  "0110001001110", "0110001010001", "0110001010100", "0110001010111",
        ↪  "0110001011010", "0110001011101", "0110001100000", "0110001100011",
        ↪  "0110001100110", "0110001101001", "0110001101101", "0110001110000",
        ↪  "0110001110011", "0110001110110", "0110001111001", "0110001111100",
        ↪  "0110001111111", "0110010000010", "0110010000101", "0110010001000",
        ↪  "0110010001011", "0110010001110", "0110010010010", "0110010010101",
        ↪  "0110010011000", "0110010011011", "0110010011110", "0110010100001",
        ↪  "0110010100100", "0110010100111", "0110010101010", "0110010101101",
        ↪  "0110010110001", "0110010110100", "0110010110111", "0110010111010",
        ↪  "0110010111101", "0110011000000", "0110011000011", "0110011000110",
        ↪  "0110011001001", "0110011001100", "0110011010000",
50      "0110011010011", "0110011010110", "0110011011001", "0110011011100", "0110011011111",
        ↪  "0110011100010", "0110011100101", "0110011101000", "0110011101100",
        ↪  "0110011101111", "0110011110010", "0110011110101", "0110011111000",
        ↪  "0110011111011", "0110011111110", "0111000000001", "0111000000100",
        ↪  "0111000001000", "0111000001011", "0111000001110", "0111000010001",
        ↪  "0111000010100", "0111000010111", "0111000011010", "0111000011101",
        ↪  "0111000100000", "0111000100100", "0111000100111", "0111000101010",
        ↪  "0111000101101", "0111000110000", "0111000110011", "0111000110110",
        ↪  "0111000111001", "0111000111101", "0111001000000", "0111001000011",
        ↪  "0111001000110", "0111001001001", "0111001001100", "0111001001111",
        ↪  "0111001010011", "0111001010110", "0111001011001", "0111001011100",
        ↪  "0111001011111", "0111001100010", "0111001100101", "0111001101000",
        ↪  "0111001101100", "0111001101111", "0111001110010", "0111001110101",
        ↪  "0111001111000", "0111001111011", "0111001111110", "0111010000001",
        ↪  "0111010000101", "0111010001000", "0111010001011", "0111010001110",
        ↪  "0111010010001", "0111010010100", "0111010010111", "0111010011011",
        ↪  "0111010011110", "0111010100001", "0111010100100", "0111010100111",
        ↪  "0111010101010", "0111010101101", "0111010110001", "0111010110100",
        ↪  "0111010110111", "0111010111010", "0111010111101", "0111011000000",
        ↪  "0111011000011", "0111011000110", "0111011001010", "0111011001101",
        ↪  "0111011010000", "0111011010011", "0111011010110", "0111011011001",
        ↪  "0111011011100", "0111011100000", "0111011100011", "0111011100110",
        ↪  "0111011101001", "0111011101100", "0111011101111", "0111011110010",
        ↪  "0111011110110", "0111011111001", "0111011111100"
51      );
52
53    end package sine_wave_pkg;
```

Source Code C.19: SPI Master

```
1  ----------------------------------------------------------------------------------
2  --
3  --   FileName:         spi_master.vhd
4  --   Dependencies:     none
5  --   Design Software:  Quartus II Version 9.0 Build 132 SJ Full Version
6  --
7  --   HDL CODE IS PROVIDED "AS IS."  DIGI-KEY EXPRESSLY DISCLAIMS ANY
8  --   WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT
9  --   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
10 --   PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL DIGI-KEY
11 --   BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL
12 --   DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF
13 --   PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS
14 --   BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
15 --   ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.
16 --
17 --   Version History
18 --   Version 1.0 7/23/2010 Scott Larson
19 --     Initial Public Release
20 --   Version 1.1 4/11/2013 Scott Larson
21 --     Corrected ModelSim simulation error (explicitly reset clk_toggles signal)
22 --
23 ----------------------------------------------------------------------------------
24
25 LIBRARY ieee;
26 USE ieee.std_logic_1164.all;
27 USE ieee.std_logic_arith.all;
28 USE ieee.std_logic_unsigned.all;
29
30 ENTITY spi_master IS
31   GENERIC(
32     slaves  : INTEGER := 4;  --number of spi slaves
33     d_width : INTEGER := 2); --data bus width
34   PORT(
35     clock   : IN     STD_LOGIC;                             --system clock
36     reset_n : IN     STD_LOGIC;                             --asynchronous reset
37     enable  : IN     STD_LOGIC;                             --initiate
       ↪ transaction
38     cpol    : IN     STD_LOGIC;                             --spi clock polarity
39     cpha    : IN     STD_LOGIC;                             --spi clock phase
40     cont    : IN     STD_LOGIC;                             --continuous mode
       ↪ command
41     clk_div : IN     INTEGER;                               --system clock cycles
       ↪ per 1/2 period of sclk
42     addr    : IN     INTEGER;                               --address of slave
43     tx_data : IN     STD_LOGIC_VECTOR(d_width-1 DOWNTO 0);  --data to transmit
44     miso    : IN     STD_LOGIC;                             --master in, slave
       ↪ out
45     sclk    : BUFFER STD_LOGIC;                             --spi clock
46     ss_n    : BUFFER STD_LOGIC_VECTOR(slaves-1 DOWNTO 0);   --slave select
47     mosi    : OUT    STD_LOGIC;                             --master out, slave
       ↪ in
48     busy    : OUT    STD_LOGIC;                             --busy / data ready
       ↪ signal
49     rx_data : OUT    STD_LOGIC_VECTOR(d_width-1 DOWNTO 0)); --data received
50 END spi_master;
51
52 ARCHITECTURE logic OF spi_master IS
```

```
53    TYPE machine IS(ready, execute);                            --state machine data
      ↪ type
54    SIGNAL state       : machine;                               --current state
55    SIGNAL slave       : INTEGER;                               --slave selected for
      ↪ current transaction
56    SIGNAL clk_ratio   : INTEGER;                               --current clk_div
57    SIGNAL count       : INTEGER;                               --counter to trigger
      ↪ sclk from system clock
58    SIGNAL clk_toggles : INTEGER RANGE 0 TO d_width*2 + 1;      --count spi clock
      ↪ toggles
59    SIGNAL assert_data : STD_LOGIC;                             --'1' is tx sclk
      ↪ toggle, '0' is rx sclk toggle
60    SIGNAL continue    : STD_LOGIC;                             --flag to continue
      ↪ transaction
61    SIGNAL rx_buffer   : STD_LOGIC_VECTOR(d_width-1 DOWNTO 0); --receive data
      ↪ buffer
62    SIGNAL tx_buffer   : STD_LOGIC_VECTOR(d_width-1 DOWNTO 0); --transmit data
      ↪ buffer
63    SIGNAL last_bit_rx : INTEGER RANGE 0 TO d_width*2;          --last rx data bit
      ↪ location
64  BEGIN
65    PROCESS(clock, reset_n)
66    BEGIN
67
68      IF(reset_n = '0') THEN        --reset system
69        busy <= '1';                --set busy signal
70        ss_n <= (OTHERS => '1');    --deassert all slave select lines
71        mosi <= 'Z';                --set master out to high impedance
72        rx_data <= (OTHERS => '0'); --clear receive data port
73        state <= ready;             --go to ready state when reset is exited
74
75      ELSIF(clock'EVENT AND clock = '1') THEN
76        CASE state IS               --state machine
77
78          WHEN ready =>
79            busy <= '0';             --clock out not busy signal
80            ss_n <= (OTHERS => '1'); --set all slave select outputs high
81            mosi <= 'Z';             --set mosi output high impedance
82            continue <= '0';         --clear continue flag
83
84            --user input to initiate transaction
85            IF(enable = '1') THEN
86              busy <= '1';             --set busy signal
87              IF(addr < slaves) THEN   --check for valid slave address
88                slave <= addr;         --clock in current slave selection if valid
89              ELSE
90                slave <= 0;            --set to first slave if not valid
91              END IF;
92              IF(clk_div = 0) THEN     --check for valid spi speed
93                clk_ratio <= 1;        --set to maximum speed if zero
94                count <= 1;            --initiate system-to-spi clock counter
95              ELSE
96                clk_ratio <= clk_div;  --set to input selection if valid
97                count <= clk_div;      --initiate system-to-spi clock counter
98              END IF;
99              sclk <= cpol;            --set spi clock polarity
100             assert_data <= NOT cpha; --set spi clock phase
101             tx_buffer <= tx_data;    --clock in data for transmit into buffer
102             clk_toggles <= 0;        --initiate clock toggle counter
103             last_bit_rx <= d_width*2 + conv_integer(cpha) - 1; --set last rx data
                ↪ bit
104             state <= execute;        --proceed to execute state
```

```
105             ELSE
106               state <= ready;          --remain in ready state
107             END IF;
108
109           WHEN execute =>
110             busy <= '1';        --set busy signal
111             ss_n(slave) <= '0'; --set proper slave select output
112
113             --system clock to sclk ratio is met
114             IF(count = clk_ratio) THEN
115               count <= 1;                     --reset system-to-spi clock counter
116               assert_data <= NOT assert_data; --switch transmit/receive indicator
117               IF(clk_toggles = d_width*2 + 1) THEN
118                 clk_toggles <= 0;               --reset spi clock toggles counter
119               ELSE
120                 clk_toggles <= clk_toggles + 1; --increment spi clock toggles
                    ↪  counter
121               END IF;
122
123               --spi clock toggle needed
124               IF(clk_toggles <= d_width*2 AND ss_n(slave) = '0') THEN
125                 sclk <= NOT sclk; --toggle spi clock
126               END IF;
127
128               --receive spi clock toggle
129               IF(assert_data = '0' AND clk_toggles < last_bit_rx + 1 AND
                  ↪  ss_n(slave) = '0') THEN
130                 rx_buffer <= rx_buffer(d_width-2 DOWNTO 0) & miso; --shift in
                    ↪  received bit
131               END IF;
132
133               --transmit spi clock toggle
134               IF(assert_data = '1' AND clk_toggles < last_bit_rx) THEN
135                 mosi <= tx_buffer(d_width-1);                    --clock out data
                    ↪  bit
136                 tx_buffer <= tx_buffer(d_width-2 DOWNTO 0) & '0'; --shift data
                    ↪  transmit buffer
137               END IF;
138
139               --last data receive, but continue
140               IF(clk_toggles = last_bit_rx AND cont = '1') THEN
141                 tx_buffer <= tx_data;                     --reload transmit
                    ↪  buffer
142                 clk_toggles <= last_bit_rx - d_width*2 + 1; --reset spi clock
                    ↪  toggle counter
143                 continue <= '1';                          --set continue flag
144               END IF;
145
146               --normal end of transaction, but continue
147               IF(continue = '1') THEN
148                 continue <= '0';      --clear continue flag
149                 busy <= '0';          --clock out signal that first receive data is
                    ↪  ready
150                 rx_data <= rx_buffer; --clock out received data to output port
151               END IF;
152
153               --end of transaction
154               IF((clk_toggles = d_width*2 + 1) AND cont = '0') THEN
155                 busy <= '0';             --clock out not busy signal
156                 ss_n <= (OTHERS => '1'); --set all slave selects high
157                 mosi <= 'Z';             --set mosi output high impedance
158                 rx_data <= rx_buffer;    --clock out received data to output port
```

```
159                 state <= ready;          --return to ready state
160               ELSE                       --not end of transaction
161                 state <= execute;        --remain in execute state
162               END IF;
163
164             ELSE        --system clock to sclk ratio not met
165               count <= count + 1; --increment counter
166               state <= execute;   --remain in execute state
167             END IF;
168
169         END CASE;
170       END IF;
171     END PROCESS;
172   END logic;
```

Source Code C.20: SPI Master Dual MISO

```
1  ---------------------------------------------------------------------------------
2  --
3  --   FileName:         spi_master_dual_miso.vhd
4  --   Dependencies:     none
5  --   Design Software:  Quartus Prime Version 17.0.0 Build 595 SJ Lite Edition
6  --
7  --   HDL CODE IS PROVIDED "AS IS."  DIGI-KEY EXPRESSLY DISCLAIMS ANY
8  --   WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT
9  --   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
10 --   PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL DIGI-KEY
11 --   BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL
12 --   DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF
13 --   PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS
14 --   BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
15 --   ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.
16 --
17 --   Version History
18 --   Version 1.0 12/19/2019 Scott Larson
19 --     Initial Public Release
20 --
21 ---------------------------------------------------------------------------------
22
23 LIBRARY ieee;
24 USE ieee.std_logic_1164.all;
25 USE ieee.std_logic_arith.all;
26 USE ieee.std_logic_unsigned.all;
27
28 ENTITY spi_master_dual_miso IS
29   GENERIC(
30     slaves  : INTEGER := 1;   --number of spi slaves
31     d_width : INTEGER := 16); --data bus width
32   PORT(
33     clock     : IN     STD_LOGIC;                             --system clock
34     reset_n   : IN     STD_LOGIC;                             --asynchronous
      ↪ reset
35     enable    : IN     STD_LOGIC;                             --initiate
      ↪ transaction
36     cpol      : IN     STD_LOGIC;                             --spi clock
      ↪ polarity
37     cpha      : IN     STD_LOGIC;                             --spi clock phase
38     cont      : IN     STD_LOGIC;                             --continuous mode
      ↪ command
39     clk_div   : IN     INTEGER;                               --system clock
      ↪ cycles per 1/2 period of sclk
40     addr      : IN     INTEGER;                               --address of slave
41     tx_data   : IN     STD_LOGIC_VECTOR(d_width-1 DOWNTO 0);  --data to transmit
42     miso_0    : IN     STD_LOGIC;                             --master in, slave
      ↪ out, channel 0
43     miso_1    : IN     STD_LOGIC;                             --master in, slave
      ↪ out, channel 1
44     sclk      : BUFFER STD_LOGIC;                             --spi clock
45     ss_n      : BUFFER STD_LOGIC_VECTOR(slaves-1 DOWNTO 0);   --slave select
46     mosi      : OUT    STD_LOGIC;                             --master out, slave
      ↪ in
47     busy      : OUT    STD_LOGIC;                             --busy / data ready
      ↪ signal
48     rx_data_0 : OUT    STD_LOGIC_VECTOR(d_width-1 DOWNTO 0);  --data received,
      ↪ channel 0
```

```
49      rx_data_1 : OUT    STD_LOGIC_VECTOR(d_width-1 DOWNTO 0)); --data received,
        ↪  channel 1
50  END spi_master_dual_miso;
51
52  ARCHITECTURE logic OF spi_master_dual_miso IS
53    TYPE machine IS(ready, execute);                            --state machine data
      ↪  type
54    SIGNAL state       : machine;                               --current state
55    SIGNAL slave       : INTEGER;                               --slave selected for
      ↪  current transaction
56    SIGNAL clk_ratio   : INTEGER;                               --current clk_div
57    SIGNAL count       : INTEGER;                               --counter to trigger
      ↪  sclk from system clock
58    SIGNAL clk_toggles : INTEGER RANGE 0 TO d_width*2 + 1;      --count spi clock
      ↪  toggles
59    SIGNAL assert_data : STD_LOGIC;                             --'1' is tx sclk
      ↪  toggle, '0' is rx sclk toggle
60    SIGNAL continue    : STD_LOGIC;                             --flag to continue
      ↪  transaction
61    SIGNAL rx_buffer_0 : STD_LOGIC_VECTOR(d_width-1 DOWNTO 0); --receive data
      ↪  buffer, channel 0
62    SIGNAL rx_buffer_1 : STD_LOGIC_VECTOR(d_width-1 DOWNTO 0); --receive data
      ↪  buffer, channel 1
63    SIGNAL tx_buffer   : STD_LOGIC_VECTOR(d_width-1 DOWNTO 0); --transmit data
      ↪  buffer
64    SIGNAL last_bit_rx : INTEGER RANGE 0 TO d_width*2;          --last rx data bit
      ↪  location
65  BEGIN
66    PROCESS(clock, reset_n)
67    BEGIN
68
69      IF(reset_n = '0') THEN          --reset system
70        busy <= '1';                  --set busy signal
71        ss_n <= (OTHERS => '1');      --deassert all slave select lines
72        mosi <= 'Z';                  --set master out to high impedance
73        rx_data_0 <= (OTHERS => '0'); --clear receive data port
74        rx_data_1 <= (OTHERS => '0'); --clear receive data port
75        state <= ready;               --go to ready state when reset is exited
76
77      ELSIF(clock'EVENT AND clock = '1') THEN
78        CASE state IS                 --state machine
79
80          WHEN ready =>
81            busy <= '0';              --clock out not busy signal
82            ss_n <= (OTHERS => '1'); --set all slave select outputs high
83            mosi <= 'Z';              --set mosi output high impedance
84            continue <= '0';          --clear continue flag
85
86            --user input to initiate transaction
87            IF(enable = '1') THEN
88              busy <= '1';             --set busy signal
89              IF(addr < slaves) THEN   --check for valid slave address
90                slave <= addr;         --clock in current slave selection if valid
91              ELSE
92                slave <= 0;            --set to first slave if not valid
93              END IF;
94              IF(clk_div = 0) THEN     --check for valid spi speed
95                clk_ratio <= 1;        --set to maximum speed if zero
96                count <= 1;            --initiate system-to-spi clock counter
97              ELSE
98                clk_ratio <= clk_div;  --set to input selection if valid
99                count <= clk_div;      --initiate system-to-spi clock counter
```

```
100             END IF;
101             sclk <= cpol;             --set spi clock polarity
102             assert_data <= NOT cpha; --set spi clock phase
103             tx_buffer <= tx_data;     --clock in data for transmit into buffer
104             clk_toggles <= 0;         --initiate clock toggle counter
105             last_bit_rx <= d_width*2 + conv_integer(cpha) - 1; --set last rx data
                ↪  bit
106             state <= execute;         --proceed to execute state
107           ELSE
108             state <= ready;           --remain in ready state
109           END IF;
110
111         WHEN execute =>
112           busy <= '1';        --set busy signal
113           ss_n(slave) <= '0'; --set proper slave select output
114
115           --system clock to sclk ratio is met
116           IF(count = clk_ratio) THEN
117             count <= 1;                     --reset system-to-spi clock counter
118             assert_data <= NOT assert_data; --switch transmit/receive indicator
119             IF(clk_toggles = d_width*2 + 1) THEN
120               clk_toggles <= 0;               --reset spi clock toggles counter
121             ELSE
122               clk_toggles <= clk_toggles + 1; --increment spi clock toggles
                  ↪  counter
123             END IF;
124
125             --spi clock toggle needed
126             IF(clk_toggles <= d_width*2 AND ss_n(slave) = '0') THEN
127               sclk <= NOT sclk; --toggle spi clock
128             END IF;
129
130             --receive spi clock toggle
131             IF(assert_data = '0' AND clk_toggles < last_bit_rx + 1 AND
                ↪  ss_n(slave) = '0') THEN
132               rx_buffer_0 <= rx_buffer_0(d_width-2 DOWNTO 0) & miso_0; --shift in
                  ↪  received bit, channel 0
133               rx_buffer_1 <= rx_buffer_1(d_width-2 DOWNTO 0) & miso_1; --shift in
                  ↪  received bit, channel 1
134             END IF;
135
136             --transmit spi clock toggle
137             IF(assert_data = '1' AND clk_toggles < last_bit_rx) THEN
138               mosi <= tx_buffer(d_width-1);                    --clock out data
                  ↪  bit
139               tx_buffer <= tx_buffer(d_width-2 DOWNTO 0) & '0'; --shift data
                  ↪  transmit buffer
140             END IF;
141
142             --last data receive, but continue
143             IF(clk_toggles = last_bit_rx AND cont = '1') THEN
144               tx_buffer <= tx_data;                       --reload transmit
                  ↪  buffer
145               clk_toggles <= last_bit_rx - d_width*2 + 1; --reset spi clock
                  ↪  toggle counter
146               continue <= '1';                            --set continue flag
147             END IF;
148
149             --normal end of transaction, but continue
150             IF(continue = '1') THEN
151               continue <= '0';          --clear continue flag
152               busy <= '0';              --clock out signal that first receive
                  ↪  data is ready
```

```
153               rx_data_0 <= rx_buffer_0; --clock out received data to output port,
                  ↪  channel 0
154               rx_data_1 <= rx_buffer_1; --clock out received data to output port,
                  ↪  channel 1
155             END IF;
156
157             --end of transaction
158             IF((clk_toggles = d_width*2 + 1) AND cont = '0') THEN
159               busy <= '0';               --clock out not busy signal
160               ss_n <= (OTHERS => '1');   --set all slave selects high
161               mosi <= 'Z';               --set mosi output high impedance
162               rx_data_0 <= rx_buffer_0;  --clock out received data to output
                  ↪  port, channel 0
163               rx_data_1 <= rx_buffer_1;  --clock out received data to output
                  ↪  port, channel 1
164
165               -- Custom logic
166               --rx_buffer_0 <= (others => '0');
167               --rx_buffer_1 <= (others => '0');
168
169               state <= ready;            --return to ready state
170             ELSE                        --not end of transaction
171               state <= execute;          --remain in execute state
172             END IF;
173
174           ELSE                    --system clock to sclk ratio not met
175             count <= count + 1;     --increment counter
176             state <= execute;       --remain in execute state
177           END IF;
178
179       END CASE;
180     END IF;
181   END PROCESS;
182 END logic;
```

Source Code C.21: Top

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.array_pkg.all;
use work.sine_wave_pkg.all;


entity top is
  port(
    clk           :  in std_logic;
    rst           :  in std_logic;

    -- Random input
    --noisy_signal :   in std_logic_vector(vector_length-1 downto 0);

    --Zynq
    --sys_halt_axi  : in std_logic_vector(31 downto 0);
    clk_slow      : in std_logic;
    rst_slow      : in std_logic;
    --ADC
    d0            :  in std_logic;
    d1            :  in std_logic;
    sclk_adc      : out std_logic;
    ss_n_adc      : out std_logic_vector(0 downto 0);

    --DAC
    tx            :  in std_logic;
    o_mosi        : out std_logic;
    o_sclk        : out std_logic;
    o_ss          : out std_logic_vector(0 downto 0);

    --MUX
    mux_s_i       : out std_logic_vector(2 downto 0);
    mux_s_o       : out std_logic_vector(2 downto 0);
    oe_n          : out std_logic;
    --ILA signals
    x_ila           : out unsigned(11 downto 0);

    angle_out       : out std_logic_vector(15 downto 0);
    magnitude_out   : out std_logic_vector(11 downto 0);
    --fcw           : out std_logic_vector(9 downto 0);
    --res           : out std_logic_vector(2*vector_length-1 downto 0);
    --rea           : out std_logic_vector(2*vector_length-1 downto 0);
    sref          : out unsigned(vector_length-1 downto 0);
    --cref          : out std_logic_vector(vector_length-1 downto 0);
    -- DMA signals
    ref_cs          : out std_logic_vector(31 downto 0);--63 downto 0);
    xin             : out std_logic_vector(31 downto 0);--63 downto 0);
    --impedance       : out std_logic_vector(63 downto 0);
    LIA             : out std_logic_vector(31 downto 0);
    CORDIC          : out std_logic_vector(31 downto 0);
    --phase_uut       : out std_logic_vector(vector_length-1 downto 0);
    --cordic          : out std_logic_vector(31 downto 0);
    --magnitude       : out std_logic_vector(31 downto 0);

    --x_out            : out signed(vector_length+1 downto 0);
    --s_out            : out signed(vector_length+1 downto 0);
    --c_out            : out signed(vector_length+1 downto 0);
    --inph_out         : out signed(2*vector_length+3 downto 0);
```

```
60      --quad_out         : out signed(2*vector_length+3 downto 0);
61
62      btn             :  in std_logic_vector(3 downto 0);
63      led             : out std_logic_vector(3 downto 0)
64      --dma_out       : out std_logic_vector(31 downto 0)
65      --array_out     : out std_logic_vector(2*vector_length-1 downto 0);
66      --sin_out       : out std_logic_vector(vector_length-1 downto 0)
67    );
68  end entity top;
69
70  architecture rtl of top is
71
72    signal btn_i : std_logic_vector(3 downto 0);
73
74    signal sin_ref_i, cos_ref_i, sin_i : std_logic_vector(vector_length-1 downto
      ↪  0);
75    signal res_i, rea_i, mag_i, ang_i  : std_logic_vector(11 downto 0);
76    type   cordic_array is array (0 to electrodes-1) of std_logic_vector(11 downto
      ↪  0);
77    type   angle_array is array (0 to electrodes-1) of std_logic_vector(11+4 downto
      ↪  0);
78    signal rea_array_i, res_array_i, mag_array_i  : cordic_array;
79    signal  ang_array_i : angle_array;
80
81    signal x_mean : std_logic_vector(11 downto 0);
82    signal inph_mean, quad_mean : std_logic_vector(12 downto 0);
83    signal period_str_i : std_logic;
84
85    signal sys_clk    : std_logic;
86    signal start_btn_i, freq_up, freq_down  : std_logic;
87    --type t_bit is array (0 to electrodes-1) of std_logic;
88    --signal valid_i, en_wave_i : t_bit;
89    --type t_fcw is array (0 to electrodes-1) of unsigned(4 downto 0);
90    --signal fcw_dsp_i : t_fcw;
91    --signal fcw_i, fcw_ii : unsigned(4 downto 0);
92    -- PS
93    signal sys_halt_i : std_logic;
94
95    --ADC
96    signal xn         : adc_array;
97    signal xin_n      : dsp_array;
98    signal sclk_adc_i : std_logic;
99    signal ss_n_adc_i : std_logic_vector(0 downto 0);
100   signal rd_en_i    : std_logic;
101
102   --DAC
103   signal addr_i      : std_logic_vector(3 downto 0);
104   signal n_electrode : unsigned(5 downto 0);
105   signal curr_in, curr_out : unsigned(2 downto 0);
106   signal fcw         : std_logic_vector(3 downto 0);
107   signal measurement_i : unsigned(4 downto 0);
108
109   --FIFO
110   signal full_i, full_ref, full_xin, full_res, full_rea, empty_i, empty_ref,
      ↪  empty_xin, empty_res, empty_rea  : std_logic;
111   signal ref_cs_slow, xin_slow, resistance_slow, reactance_slow :
      ↪  std_logic_vector(31 downto 0);
112   signal x1_sync         : std_logic_vector(vector_length-1 downto 0);
113
114   signal sin_slow  : std_logic_vector(vector_length-1 downto 0);
115   signal cos_slow  : std_logic_vector(vector_length-1 downto 0);
116   signal str_50MHz : std_logic;
```

```
117
118
119     component dsp is
120       port (
121         clk        :  in std_logic;
122         rst        :  in std_logic;
123
124         -- ADC
125         sin_in     :  in std_logic_vector(vector_length-1 downto 0);
126         o_sclk     :  in std_logic;
127
128         -- DDS
129         fcw             :  in std_logic_vector(3 downto 0);
130         sin_ref         : in std_logic_vector(vector_length-1 downto 0);
131         cos_ref         : in std_logic_vector(vector_length-1 downto 0);
132
133         -- LIA
134             x_out            : out signed(vector_length+1 downto 0);
135                 s_out            : out signed(vector_length+1 downto 0);
136       c_out            : out signed(vector_length+1 downto 0);
137       inph_out         : out signed(2*vector_length+3 downto 0);
138       quad_out         : out signed(2*vector_length+3 downto 0);
139         period_str      :  in std_logic;
140
141         magnitude       : out std_logic_vector(11 downto 0);
142         angle           : out std_logic_vector(15 downto 0);
143         xin_mean        : out std_logic_vector(11 downto 0);
144         inphase_mean    : out std_logic_vector(12 downto 0);
145         quadrature_mean : out std_logic_vector(12 downto 0)
146       );
147     end component dsp;
148
149     component signal_loop_rand is
150         port (
151           clk         :  in std_logic;
152           rst         :  in std_logic;
153           btn         :  in std_logic_vector(3 downto 0);
154           period_str  : out std_logic;
155           done_ld     : out std_logic;
156           fcw_out     : out std_logic_vector(3 downto 0);
157           sin         : out std_logic_vector(vector_length-1 downto 0);
158           cos         : out std_logic_vector(vector_length-1 downto 0);
159           measurement : out unsigned(4 downto 0);
160           n_electrode : out unsigned(5 downto 0)
161         );
162     end component signal_loop_rand;
163
164     component dac_imp is
165       port (
166           clk          : IN      STD_LOGIC;                      --system clock
167           reset_n      : IN      STD_LOGIC;                      --active low
              ↪  asynchronous reset
168           dac_tx_ena   : IN      STD_LOGIC;                      --enable
              ↪  transaction with DAC
169           addr         : in std_logic_vector(3 downto 0);
170           data         : in std_logic_vector(11 downto 0);
171           mosi_o       : OUT     STD_LOGIC;                      --SPI bus to DAC:
              ↪  master out, slave in (DIN)
172           sclk_o       : out  STD_LOGIC;                     --SPI bus to DAC:
              ↪  serial clock (SCLK)
173           ss_n_o       : out  STD_LOGIC_VECTOR(0 DOWNTO 0)     --SPI bus to DAC:
              ↪  slave select (~SYNC)
```

```
174       );
175     end component;
176
177     component adc_imp is
178       port (
179           clk        : IN      STD_LOGIC;                          --system clock
180           reset_n    : IN      STD_LOGIC;                          --active low
              ↪  asynchronous reset
181
182           -- Data in
183           d0    :  IN      STD_LOGIC;                        --channel 0 serial data
              ↪  from ADC
184           d1    :  IN      STD_LOGIC;                        --channel 1 serial data
              ↪  from ADC
185
186           -- SPI
187           sclk_o       : out  STD_LOGIC;                       --SPI bus from ADC:
              ↪  serial clock (SCLK)
188           ss_n_o       : out  STD_LOGIC_VECTOR(0 DOWNTO 0);    --SPI bus from ADC:
              ↪  slave select (~SYNC)
189
190           -- Sampled data
191           electrode0   : out std_logic_vector(vector_length-1 downto 0);
192           electrode1   : out std_logic_vector(vector_length-1 downto 0)
193       );
194     end component;
195
196     component fifo is
197       generic (
198         inout_length : integer := (vector_length-1);
199         new_array_length : integer := 262143--(4*array_length-1)
200       );
201       port (
202         clk      :  in std_logic;
203         rst      :  in std_logic;
204         -- Read/write
205         wr_en    :  in std_logic;
206         rd_en    :  in std_logic;
207         full     : out std_logic;
208         empty    : out std_logic;
209         -- Data
210         sin_wr   :  in std_logic_vector(inout_length downto 0);
211         sin_rd   : out std_logic_vector(inout_length downto 0)
212       );
213     end component fifo;
214
215   begin
216
217     GENERATION: signal_loop_rand
218       port map(
219         clk         => str_50MHz,
220         rst         => rst,
221         btn         => btn_i,
222         period_str  => period_str_i,
223         done_ld     => led(0),
224         fcw_out     => fcw,
225         sin         => sin_ref_i,
226         cos         => cos_ref_i,
227         measurement => measurement_i,
228         n_electrode => n_electrode
229       );
230
```

```
231       ADC: adc_imp
232         port map(
233           clk          => sys_clk,
234           reset_n      => not(rst),
235           d0           => d0,
236           d1           => d1,
237           sclk_o       => sclk_adc_i,
238           ss_n_o       => ss_n_adc_i,
239           electrode0   => xn(0),
240           electrode1   => xn(1)
241         );
242
243
244       --PROCESSING: for I in 0 to electrodes-1 generate
245         DSP_gen: dsp
246           port map(
247             clk        => sys_clk,
248             rst        => rst,
249             sin_in     => xn(0),
250             o_sclk     => sclk_adc_i,
251             fcw        => fcw,
252             sin_ref    => sin_ref_i,
253             cos_ref    => cos_ref_i,
254
255             x_out => open,
256             s_out => open,
257             c_out => open,
258             inph_out => open,
259             quad_out => open,
260             period_str => period_str_i,
261
262             magnitude  => mag_array_i(0),
263             angle      => ang_array_i(0),
264             xin_mean   => x_mean,
265             inphase_mean => inph_mean,
266             quadrature_mean => quad_mean
267             --eidors_rea => rea_i,--rea_array_i(0),
268             --eidors_res => res_i--res_array_i(0)
269           );
270       --end generate PROCESSING;
271
272       DAC: dac_imp
273       port map(
274         clk            => sys_clk,
275         reset_n        => not(rst),
276         dac_tx_ena     => tx,
277         addr           => addr_i,
278         data           => sin_ref_i,
279         mosi_o         => o_mosi,
280         sclk_o         => o_sclk,
281         ss_n_o         => o_ss
282
283       );
284
285
286       process(clk, rst)
287         begin
288           if rst = '1' then
289             str_50MHz <= '0';
290           elsif rising_edge(clk) then
291             str_50MHz <= not(str_50MHz);
292           end if;
```

```
293       end process;
294
295     curr_out  <= n_electrode(5 downto 3);
296     curr_in <= n_electrode(2 downto 0);
297
298     addr_i <= "0000";--std_logic_vector("0" & curr_in);
299
300     --xin_n(to_integer(curr_out)) <= xn(0);
301     sclk_adc <= sclk_adc_i;
302     ss_n_adc <= ss_n_adc_i;
303
304     mux_s_o <= std_logic_vector(curr_out);
305     mux_s_i <= std_logic_vector(curr_in);
306     oe_n  <= '0';
307
308    -- process(clk, rst)
309    -- begin
310    --   if rst = '1' then
311    --     ref_cs     <= (others => '0');
312    --     xin        <= (others => '0');
313    --     res <= (others => '0');
314    --     rea  <= (others => '0');
315    --   elsif rising_edge(clk) then
316         ref_cs     <= "0" & btn_i & std_logic_vector(curr_in) & cos_ref_i &
         ↪  sin_ref_i;
317         xin        <= "0" & fcw & std_logic_vector(curr_out) & xn(1) & xn(0);
318         --res <= start_btn_i & freq_up & freq_down  & "00000000000000000" &
         ↪  inph_mean;-- 1 + 1 + 1 + 12 = 15
319         --rea  <= "00000000000000000000" & quad_mean;-- 2 + 11 = 13
320
321         CORDIC    <= "0000" & std_logic_vector(mag_array_i(0)) &
         ↪  std_logic_vector(ang_array_i(0));
322         LIA       <= "000000" & quad_mean & inph_mean;
323    --   end if;
324    -- end process;
325
326     --sys_halt_i <= '0' when rst = '1' else
327     --              sys_halt_axi(31) when rising_edge(clk) else
328     --              sys_halt_i;
329
330
331     sys_clk    <= clk;
332     --led(3)     <= sys_halt_i;--'0' when rst = '1' else
333                   -- clk and not(sys_halt_i) when rising_edge(clk) else
334                   -- sys_clk;
335     start_btn_i    <= btn(3);
336     btn_i          <= btn;
337     freq_up        <= btn(1);
338     freq_down      <= btn(0);
339     led(3)         <= btn(3);
340     --angle      <= "00000000" & ang_array_i(0);
341     --magnitude  <= "00000000" & mag_array_i(0);
342
343     --impedance <= "0000000000000"& std_logic_vector(i_electrode) &
       ↪  res_array_i(to_integer(i_electrode)) &
       ↪  rea_array_i(to_integer(i_electrode));
344     --cordic    <= "0000000000000" & std_logic_vector(i_electrode) &
       ↪  mag_array_i(to_integer(i_electrode)) &
       ↪  ang_array_i(to_integer(i_electrode));
345
346     x_ila      <= unsigned(xn(0));
347     sref       <= unsigned(sin_ref_i);
```

```
348     angle_out     <= std_logic_vector(ang_array_i(0));
349     magnitude_out <= std_logic_vector(mag_array_i(0));
350 
351   end rtl;
```

Source Code C.22: Randomized testbench

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use work.array_pkg.all;
5  use work.sine_wave_pkg.all;
6  use work.timer_pkg.all;
7  use std.textio.all;
8  use ieee.std_logic_textio.all;
9
10 entity tb_randomized_top is
11 end entity;
12
13 architecture beh of tb_randomized_top is
14
15   -- IO
16   signal clk     : std_logic := '0';
17   signal HALF_PERIOD : time := 5 ns;
18
19   signal o_sclk : std_logic := '0';
20
21   signal rst     : std_logic;
22   signal tx      : std_logic;
23
24   signal btn     : std_logic_vector(3 downto 0);
25
26
27   -- Seed
28   type t_seed_table is array (0 to 7) of std_logic_vector(11 downto 0);
29   constant c_seed_table : t_seed_table := ("000000110011", "000000110100",
     ↪  "000000110101", "000000110110", "000000110111", "000000111000",
     ↪  "000000111001", "000000111010");
30   signal start_seed : std_logic;
31
32   -- Phase
33   signal phase_uut   : std_logic_vector(vector_length-1 downto 0); --Phase from
     ↪  phase_acc
34   signal phase_shift, phase_shift_i : std_logic_vector(vector_length-1 downto 0);
35   signal phase       : unsigned(vector_length-1 downto 0);
36   signal phase_seed  : std_logic_vector(vector_length-1 downto 0);
37   signal phase_load  : std_logic;
38
39   -- Dampening
40   signal dampening      : unsigned(vector_length-1 downto 0);
41   signal dampening_i    : std_logic_vector(vector_length-1 downto 0);
42   signal dampening_load : std_logic;
43   signal dampening_seed : std_logic_vector(vector_length-1 downto 0);
44
45   -- Signal
46   signal sine_clean    : std_logic_vector(vector_length-1 downto 0);
47   signal signal_noise  : unsigned(vector_length-1 downto 0);
48   signal signal_output : std_logic_vector(vector_length-1 downto 0);
49
50   signal sine_double_freq : unsigned(vector_length downto 0);
51
52   -- Noise
53   signal noise : unsigned(7 downto 0);
54   signal noise_pos, noise_neg : std_logic_vector(7 downto 0);
55   signal noise_neg_seed, noise_pos_seed : std_logic_vector(7 downto 0);
56
```

```
57    signal noisy_signal, noisy_signal_d   : unsigned(11 downto 0);
58
59    -- Outputs
60    signal ref_cs, res, rea, mag, ang, xin : std_logic_vector(31 downto 0);
61    alias sin : std_logic_vector(11 downto 0)
62         is ref_cs (11 downto 0);
63
64    signal led : std_logic_vector(3 downto 0);
65
66
67      signal  x_out, s_out, c_out            : signed(vector_length+1 downto 0);
68      signal inph_out         : signed(2*vector_length+3 downto 0);
69      signal quad_out         : signed(2*vector_length+3 downto 0);
70
71    -- Files
72    file ref_file   : text open write_mode is "output/ref.txt";
73    file ref_cos_file : text open write_mode is "output/ref_cs.txt";
74    file sin_file   : text open write_mode is "output/sin.txt";
75    file sig_file   : text open write_mode is "output/signal_output.txt";
76    file res_file   : text open write_mode is "output/res.txt";
77    file rea_file   : text open write_mode is "output/rea.txt";
78    file mag_file   : text open write_mode is "output/mag.txt";
79    file ang_file   : text open write_mode is "output/ang.txt";
80    file phase_file : text open write_mode is "output/phase.txt";
81    file electrode_in_file : text open write_mode is "output/electrode_in.txt";
82    file electrode_out_file : text open write_mode is "output/electrode_out.txt";
83    file freq_file : text open write_mode is "output/frequency.txt";
84    file xmean_file : text open write_mode is "output/xmean.txt";
85
86    file x_file : text open write_mode is "output/x_wave.txt";
87    file s_file : text open write_mode is "output/s_wave.txt";
88    file c_file : text open write_mode is "output/c_wave.txt";
89    file inph_file : text open write_mode is "output/inph_wave.txt";
90    file quad_file : text open write_mode is "output/quad_wave.txt";
91
92    file noise_file : text open write_mode is "output/noise.txt";
93
94    file sin1k : text open write_mode is "output/sin6k_ref.txt";
95    file sin2k : text open write_mode is "output/sin7k_ref.txt";
96    file sin3k : text open write_mode is "output/sin8k_ref.txt";
97    file sin4k : text open write_mode is "output/sin9k_ref.txt";
98    file sin5k : text open write_mode is "output/sin10k_ref.txt";
99
100   signal i       : unsigned(2*vector_length-1 downto 0);
101
102   -- Open signals
103   signal cos_addr_open : unsigned(vector_length-1 downto 0);
104   signal cos_open      : std_logic_vector(vector_length-1 downto 0);
105   signal ss_n_adc_open, o_ss_open : std_logic_vector(0 downto 0);
106   signal sclk_adc_open, clk_open, rst_open, d_open, oe_open, o_mosi_open,
      ↪  o_sclk_open: std_logic;
107   signal halt_open : std_logic_vector(31 downto 0);
108   signal mux_open : std_logic_vector(2 downto 0);
109
110   component lfsr8 is
111     port (
112       clk        :  in std_logic;
113       rst        :  in std_logic;
114       start_seed :  in std_logic;
115       seed       :  in std_logic_vector(7 downto 0);
116       random_out : out std_logic_vector(7 downto 0)
117     );
```

```
118     end component;
119
120     component lfsr12 is
121       port (
122         clk        :  in std_logic;
123         rst        :  in std_logic;
124         start_seed :  in std_logic;
125         seed       :  in std_logic_vector(11 downto 0);
126         random_out : out std_logic_vector(11 downto 0)
127       );
128     end component;
129
130     component top_rand is
131       port(
132         clk           :  in std_logic;
133         rst           :  in std_logic;
134
135         noisy_signal  :  in std_logic_vector(vector_length-1 downto 0);
136         -- Zynq
137         sys_halt_axi  : in std_logic_vector(31 downto 0);
138         clk_slow      : in std_logic;
139         rst_slow      : in std_logic;
140         -- ADC
141         d0            :  in std_logic;
142         d1            :  in std_logic;
143         sclk_adc      : out std_logic;
144         ss_n_adc      : out std_logic_vector(0 downto 0);
145         -- DAC
146         tx            :  in std_logic;
147         o_mosi        : out std_logic;
148         o_sclk        : out std_logic;
149         o_ss          : out std_logic_vector(0 downto 0);
150         -- MUX
151         mux_s         : out std_logic_vector(2 downto 0);
152         oe_n          : out std_logic;
153         -- DMA signals
154         ref_cs          : out std_logic_vector(31 downto 0);--63 downto 0);
155         xin             : out std_logic_vector(31 downto 0);--63 downto 0);
156         -- Impedance
157         res             : out std_logic_vector(31 downto 0);
158         rea             : out std_logic_vector(31 downto 0);
159         cordic          : out std_logic_vector(31 downto 0);
160         magnitude       : out std_logic_vector(31 downto 0);
161         phase_uut       : out std_logic_vector(vector_length-1 downto 0);
162
163             x_out           : out signed(vector_length+1 downto 0);
164                 s_out           : out signed(vector_length+1 downto 0);
165       c_out           : out signed(vector_length+1 downto 0);
166       inph_out        : out signed(2*vector_length+3 downto 0);
167       quad_out        : out signed(2*vector_length+3 downto 0);
168         -- PHY
169         btn           :  in std_logic_vector(3 downto 0);
170         led           : out std_logic_vector(3 downto 0)
171       );
172     end component top_rand;
173
174     component sin_table is
175       port (
176         clk           :  in std_logic;
177         sin_addr      :  in unsigned(vector_length-1 downto 0);
178         cos_addr      :  in unsigned(vector_length-1 downto 0);
179         sin           : out std_logic_vector(vector_length-1 downto 0);
```

```
180         cos           : out std_logic_vector(vector_length-1 downto 0)
181       );
182     end component sin_table;
183 
184     component fifo is
185       generic (
186         inout_length : integer := vector_length-1;
187         new_array_length : integer := 20--0000
188       );
189       port (
190         clk      :  in std_logic;
191         rst      :  in std_logic;
192         -- Read/write
193         wr_en    :  in std_logic;
194         rd_en    :  in std_logic;
195         full     : out std_logic;
196         empty    : out std_logic;
197         -- Data
198         sin_wr   :  in std_logic_vector(inout_length downto 0);
199         sin_rd   : out std_logic_vector(inout_length downto 0)
200       );
201     end component fifo;
202 
203     component dds_rand is
204     port(
205       clk           :  in std_logic;
206       rst           :  in std_logic;
207       fcw           :  in unsigned(4 downto 0);
208       en_wave       :  in std_logic;
209       phase_uut     : out std_logic_vector(vector_length-1 downto 0);
210       sin           : out std_logic_vector(vector_length-1 downto 0);
211       cos           : out std_logic_vector(vector_length-1 downto 0)
212     );
213     end component dds_rand;
214 
215     -- Frequencies
216     type type_freq is array (0 to 5) of unsigned(4 downto 0);
217     constant fcw_table : type_freq := ("00101", "00110", "00111", "01000",
        ↪  "01001","00000");-- "00001", "00010", "00011", "00100", "00101");
218 
219     signal freq_modulated : unsigned(vector_length-1+3 downto 0);
220 
221     type signal_table is array (0 to 4) of std_logic_vector(vector_length-1 downto
        ↪  0);
222     signal sin_t : signal_table;
223     signal cos_t : signal_table;
224 
225     signal str_50MHz : std_logic;
226     signal sine_slow : std_logic_vector(vector_length-1 downto 0);
227 
228   begin
229     --Clocking
230 
231       process(clk, rst)
232       begin
233         if rst = '1' then
234           str_50MHz <= '0';
235           sine_double_freq <= (others => '0');
236         elsif rising_edge(clk) then
237           str_50MHz <= not(str_50MHz);
```

```
238           freq_modulated <= ("000" & unsigned(sin_t(1)) + ("000" &
              ↪  unsigned(sin_t(3))));  --("0" & ("0" & ("0" & unsigned(sine_slow)) +
              ↪  ("0" & unsigned(sin_t(0)))) + ("0" & (("0" & unsigned(sin_t(1))) +
              ↪  ("0" & unsigned(sin_t(2))))))) + ("0" & ("0" & (("0" &
              ↪  unsigned(sin_t(3))) + ("0" & unsigned(sin_t(4))))));
239         end if;
240       end process;
241
242     clk <= not(clk) after HALF_PERIOD;
243     o_sclk <= not(o_sclk) after 2*HALF_PERIOD;
244     rst <= '1', '0' after 100 ns;
245
246     -- Loads
247     phase_load     <= '0', '1' after 200 ns, '0' after 210 ns;--, '1' after 200000
        ↪  ns, '0' after 200010 ns;
248     dampening_load <= '0', '1' after 200 ns, '0' after 210 ns;--, '1' after 200000
        ↪  ns, '0' after 200010 ns;
249     --phase_uut <= << signal .UUT.GENERATION.DDS_out.NCO.nco : std_logic_vector >>;
250
251     -- Buttons
252     btn(3) <= '0', '1' after 100 ns, '0' after 1500 ns;
253     btn(2 downto 0) <= "010";
254
255     -- Seeds
256     start_seed     <= '0', '1' after 100 ns, '0' after 120 ns;
257     noise_pos_seed <= c_seed_table(0)(7 downto 0);
258     noise_neg_seed <= c_seed_table(1)(7 downto 0);
259     phase_seed     <= c_seed_table(2);
260     dampening_seed <= c_seed_table(3);
261
262     DDS_gen: for J in 0 to 4 generate
263       DDS: dds_rand
264         port map(
265           clk           => str_50MHz,
266           rst           => rst,
267           fcw           => fcw_table(J),
268           en_wave       => '1',
269           phase_uut     => open,
270           sin           => sin_t(J),
271           cos           => cos_t(J)
272         );
273     end generate;
274
275     XIN_fifo: fifo
276       port map(
277         clk    => clk,
278         rst    => rst,
279         wr_en  => '1',
280         rd_en  => str_50MHz,
281         full   => open,
282         empty  => open,
283         sin_wr => std_logic_vector(sine_clean),
284         sin_rd => sine_slow
285       );
286
287     PAC: sin_table
288       port map(
289         clk        => clk,
290         sin_addr   => phase,
291         cos_addr   => cos_addr_open,
292         sin        => sine_clean,
293         cos        => cos_open
```

```
294       );
295
296     NOISE_POS_LFSR: lfsr8
297       port map(
298         clk        => o_sclk,
299         rst        => rst,
300         start_seed => start_seed,
301         seed       => noise_pos_seed,
302         random_out => noise_pos
303       );
304
305     NOISE_NEG_LFSR: lfsr8
306       port map(
307         clk        => o_sclk,
308         rst        => rst,
309         start_seed => start_seed,
310         seed       => noise_neg_seed,
311         random_out => noise_neg
312       );
313
314     PHASE_LFSR: lfsr12
315       port map(
316         clk        => clk,
317         rst        => rst,
318         start_seed => start_seed,
319         seed       => phase_seed,
320         random_out => phase_shift
321       );
322
323     DAMPENING_LFSR: lfsr12
324       port map(
325         clk        => o_sclk,
326         rst        => rst,
327         start_seed => start_seed,
328         seed       => dampening_seed,
329         random_out => dampening_i
330       );
331
332     UUT: top_rand
333       port map(
334         clk           => clk,
335         rst           => rst,
336         noisy_signal  => std_logic_vector(freq_modulated(vector_length downto
            ↪  1)),--std_logic_vector(sine_slow(11 downto
            ↪  0)),std_logic_vector(sine_clean),
337         -- Zynq
338         sys_halt_axi  => halt_open,
339         clk_slow      => clk_open,
340         rst_slow      => rst_open,
341         -- ADC
342         d0            => d_open,
343         d1            => d_open,
344         sclk_adc      => sclk_adc_open,
345         ss_n_adc      => ss_n_adc_open,
346         -- DAC
347         tx            => '1',
348         o_mosi        => o_mosi_open,
349         o_sclk        => o_sclk_open,
350         o_ss          => o_ss_open,
351         -- MUX
352         mux_s         => mux_open,
353         oe_n          => oe_open,
```

```
354         -- DMA signals
355         ref_cs        => ref_cs,
356         xin           => xin,
357         -- Impedance
358         res           => res,
359         rea           => rea,
360         cordic        => ang,
361         magnitude     => mag,
362         phase_uut     => phase_uut,
363 
364         x_out => x_out,
365         s_out => s_out,
366         c_out => c_out,
367         inph_out => inph_out,
368         quad_out => quad_out,
369         -- PHY
370         btn           => btn,
371         led           => led
372       );
373 
374     load_PROC: process(clk, rst)
375     begin
376       if rst = '1' then--sine_clean
377         phase         <= (others => '0');
378         phase_shift_i <= (others => '0');
379         dampening     <= (others => '0');
380       elsif rising_edge(clk) then
381         if phase_load = '1' then
382           phase_shift_i <= phase_shift;
383         end if;
384         if dampening_load = '1' then
385           dampening   <= unsigned(dampening_i);
386         end if;
387         phase <= unsigned(phase_uut) + to_unsigned(511,
            ↪  12);--unsigned(phase_shift_i);
388       end if;
389     end process;
390 
391     NOISE_PROC: process(clk, rst)
392     variable noisy_signal_var : unsigned(12 downto 0);
393     variable noise_var        : unsigned(7 downto 0);
394     begin
395       if rst = '1' then
396         noisy_signal_var := (others => '0');
397         noise_var        := (others => '0');
398         noise            <= (others => '0');
399       elsif rising_edge(clk) then
400         if unsigned(noise_neg) < unsigned(noise_pos) then
401           noise_var        := unsigned(noise_pos) - unsigned(noise_neg);
402           noisy_signal_var := unsigned(("0" & unsigned(sine_slow)) + ("00000" &
              ↪  noise_var));
403           if noisy_signal_var(12) = '1' then
404             noisy_signal <= (others => '1');
405           else
406             noisy_signal <= noisy_signal_var(11 downto 0);
407           end if;
408         else
409           noise_var := unsigned(noise_neg) - unsigned(noise_pos);
410           if unsigned(sine_slow) > noise_var then
411             noisy_signal <= unsigned(sine_slow) - ("0000" & noise_var);
412           else
413             noisy_signal <= (others => '0');
```

```
414           end if;
415         end if;
416         if noisy_signal > dampening then
417           noisy_signal_d <= noisy_signal - dampening;
418         else
419           noisy_signal_d <= (others => '0');
420         end if;
421         noise <= noise_var;
422       end if;
423     end process;
424
425     WRITE_2_FILE_PROC: process(clk, rst)
426     variable ref_row   : line;
427     variable ref_cos_row : line;
428     variable sin_row   : line;
429     variable sig_row   : line;
430     variable phase_row : line;
431     variable res_row   : line;
432     variable rea_row   : line;
433     variable mag_row   : line;
434     variable ang_row   : line;
435     variable curr_in_row : line;
436     variable curr_out_row : line;
437     variable freq_row  : line;
438     variable xmean_row : line;
439
440     variable x_wave : line;
441     variable s_wave : line;
442     variable c_wave : line;
443     variable inph_wave : line;
444     variable quad_wave : line;
445
446     variable noise_row : line;
447
448     variable s1_ref : line;
449     variable s2_ref : line;
450     variable s3_ref : line;
451     variable s4_ref : line;
452     variable s5_ref : line;
453
454     begin
455       if rst = '1' then
456         i <= (others => '0');
457       elsif rising_edge(clk) then
458         i <= i + 1;
459         -- noise
460         -- dampening
461         -- fcw
462         -- eletrode-ut og inn
463         write(ref_row, to_integer(unsigned(ref_cs(11 downto 0))));
464         write(ref_cos_row, to_integer(unsigned(ref_cs(23 downto 12))));
465         write(sin_row, to_integer(unsigned(sin_t(0))));
466         write(sig_row, to_integer(unsigned(noisy_signal_d)));
467         write(phase_row, to_integer(unsigned(phase)));
468         write(res_row, to_integer(signed(res(12 downto 0))));
469         write(rea_row, to_integer(signed(rea(12 downto 0))));
470
471         write(noise_row, to_integer(unsigned(noise)));
472         writeline(noise_file, noise_row);
473         --write(xmean_row, to_integer(unsigned(rea(31 downto 16))));
474
475         write(curr_in_row, to_integer(unsigned(ref_cs(26 downto 24))));
```

```
476         write(curr_out_row, to_integer(unsigned(xin(26 downto 24))));
477 
478         write(freq_row, to_integer(unsigned(ref_cs(31 downto 28))));
479 
480         write(ang_row, to_integer(signed(ang)));
481         write(mag_row, to_integer(unsigned(mag)));
482 
483         -- Write refs
484         write(s1_ref,  to_integer(unsigned(sin_t(0))));
485         write(s2_ref,  to_integer(unsigned(sin_t(1))));
486         write(s3_ref,  to_integer(unsigned(sin_t(2))));
487         write(s4_ref,  to_integer(unsigned(sin_t(3))));
488         write(s5_ref,  to_integer(unsigned(sin_t(4))));
489 
490         writeline(sin1k, s1_ref);
491         writeline(sin2k, s2_ref);
492         writeline(sin3k, s3_ref);
493         writeline(sin4k, s4_ref);
494         writeline(sin5k, s5_ref);
495 
496         -- Write waveforms
497         write(x_wave, to_integer(x_out));
498         write(s_wave, to_integer(s_out));
499         write(c_wave, to_integer(c_out));
500         write(inph_wave, to_integer(inph_out));
501         write(quad_wave, to_integer(quad_out));
502 
503         writeline(x_file, x_wave);
504         writeline(s_file, s_wave);
505         writeline(c_file, c_wave);
506         writeline(inph_file, inph_wave);
507         writeline(quad_file, quad_wave);
508         --writeline(xmean_file, xmean_row);
509 
510         writeline(ref_file, ref_row);
511         writeline(ref_cos_file, ref_cos_row);
512         writeline(sin_file, sin_row);
513         writeline(sig_file, sig_row);
514         writeline(phase_file, phase_row);
515         writeline(res_file, res_row);
516         writeline(rea_file, rea_row);
517 
518         writeline(electrode_in_file, curr_in_row);
519         writeline(electrode_out_file, curr_out_row);
520         writeline(freq_file, freq_row);
521 
522         writeline(mag_file, mag_row);
523         writeline(ang_file, ang_row);
524       end if;
525     end process;
526 
527 end beh;
```

Appendix D

MATLAB code

Source Code D.1: Generating sine table values

```
close all;

np = 4096; % samples
A = floor((2^12-1)/2); % max quantized value
t = linspace(0,1-1/np,np);
sin_table = (round(sin(2*pi*t)*A));
sin_table = sin_table + A;
max(sin_table);
cos_table = (round(cos(2*pi*t)*A)) + A;
sin_table_bin = de2bi(sin_table, 'left-msb');


sin2 = sin_table.*sin_table;
cos2 = cos_table.*sin_table;
cosref = cos_table.*cos_table;
figure()
plot(t, sin2/(max(sin2)))
hold on
plot(t, cos2/(max(cos2)))
hold on
plot(t, cosref/(max(cosref)))
hold on
plot(t, sin_table/(max(sin_table)))
legend("sin2", "cos2", "cos", "sin")

theta1 = abs((atan2(cos2, sin2)).*(180/pi));
theta2 = abs((atan2(sin2, sin_table)).*(180/pi));
thetacos = abs((atan2(cos2, cos_table)).*(180/pi));

figure()
plot(t, theta1)
hold on
plot(t, theta2)
hold on
plot(t, thetacos)
legend("theta1", "theta2", "thetacos")
%% lp pi/3

np = 255;
A = 2^14 -1;
t = linspace(0,1-1/np,np);
sin_table = (round(sin(2*pi*t)*A));
```

```
43  plot(t, sin_table)
44  hold on
45  plot(t(round(np/4):end), sin_table(round(np/4):end))
```

Source Code D.2: Generating frequency tuning words

```
%% first
np = 16384; % samples
electrodes = 8;
measurements = 28;
sys_clk = 650000000;
%sys_clk = 100000000;
%ftw = sys_clk./freq;
%timer = sys_clk./freq*measurements;
k = 1;
spi_clk = 50000000;


for i = 12:15
    for j = 1:4
        if j == 1
            bin(k) = 2^i;
        elseif j == 2
            bin(k) = 2^i + 2^(i-1);
        elseif j == 3
            bin(k) = 2^i + 2^(i-1) + 2^(i-2);
        elseif j == 4
            bin(k) = 2^i + 2^(i-1) + 2^(i-2) + 2^(i-3);
        end
        k = k + 1;
    end
end

freqs = sys_clk./bin;

x = linspace(1, length(bin), length(bin));

%plot(x, freqs)
%ylabel('Frequency')

ftw = de2bi(ceil(bin/np),16,'left-msb');

timer = ceil((sys_clk./freqs).*measurements);
timer_bin = de2bi(timer,'left-msb');

%% second
n = 14;
x = linspace(0, 2^n-1, 2^n);
x1 = linspace(0, 1, 2^n);
a = pi*x1/2;

f(x+1) = sin(a) - a;

plot(x, f)

%% ny
fs = 50000000;
%fs = floor(fs/33);
per = 1/fs;
%freqs = linspace(10000, 100000, 20);
freqs = linspace(1000, 100000, 10); %low freq
incs = floor((freqs*2^(32))/fs);
cycles = ceil(fs./(freqs));

freqs_rev = (fs*incs)/2^(32);
```

```
60    incs_bin = de2bi(incs, 32, 'left-msb');
```

Source Code D.3: Reading and plotting ILA data

```
%% Startup
%close all;

%ila = readtable("ila_electrode.csv");
ila = readtable("x_ref_5khz.csv");

ch0 = ila.EIT_i_top_0_x_ila_11_0_;
ch1 = ila.EIT_i_top_0_sref_11_0_;
%h2 = ila.Sweep_i_top_0_s_out_13_0_;
%% plot
close all;

len = length(ch0);
t = linspace(1, len-1, len);

figure()
plot(t, ch0.*3.3./(2^(12)-1));
hold on
plot(t, ch1.*2.5./(2^(12)-1))
hold on
%plot(t, ch2.*2.5./(2^(12)-1))
title("ILA 5 kHz")
legend("ADC channel 0", "DDS sine");
xlabel("Sample at 100 MHz")
ylabel("Volt")
legend("ADC channel 0", "DDS sine output", "Ref offset");
ch0_max = max(ch0)
ch1_max = max(ch1)

adc_ch0 = ch0;
ref_ch1 = ch1;

dc_adc = max(adc_ch0)/2
dc_ref = max(ref_ch1)/2

adc_cross = [];
j = 1;
for i = 1:len-1
    if adc_ch0(i) >= dc_adc
        if adc_ch0(i+1) < dc_adc
            adc_cross(j) = i;
            j = j+1;
        end
    end
end

ref_cross = [];
j = 1;
for i = 1:len-1
    if ref_ch1(i) >= dc_ref
        if ref_ch1(i+1) < dc_ref
            ref_cross(j) = i;
            j = j+1;
        end
    end
end

diff_adc = [];
diff_ref = [];
```

```
j = 1;
k = 1;
for i = 1:len-1
    diff = abs(adc_ch0(i+1)-adc_ch0(i));
    if diff > 0
        diff_adc(j) = (diff);
        j = j + 1;
    end
    diff = abs(ref_ch1(i+1)-ref_ch1(i));
    if diff > 0
        diff_ref(k) = (diff);
        k = k +1;
    end
end
min(diff_adc)
min(diff_ref)
adc_scale = 3.3/(2^(12)-1);

adc_min = min(adc_ch0)*adc_scale
adc_max = max(adc_ch0)*adc_scale
rms = (adc_max-adc_min)/(2*sqrt(2))
%%
%close all;
%len = length(ch0);
%t = linspace(1, len-1, len);
%for i = 1:10
%
%    ila = readtable(string("x_ref_"+i+"khz.csv"));%

 %   ch0 = ila.Sweep_i_top_0_x_ila_11_0_;
 %   ch1 = ila.Sweep_i_top_0_sref_11_0_;
%    figure()
%    plot(t, ch0.*3.3./(2^(12)-1));
%    hold on
%    plot(t, ch1.*2.5./(2^(12)-1))
 %   freq = 2.5*i;
%    title(string("ILA "+freq+" kHz"))
%    legend("ADC channel 0", "DDS sine output");
 %   xlabel("Sample at 10 MHz")
 %   ylabel("Amplitude (V)")
 %   ch0_rms(i) = (max(ch0)-min(ch0))/(2*sqrt(2))*3.3/(2^(12)-1);
%    ch1_rms(i) = (max(ch1)-min(ch1))/(2*sqrt(2))*2.5/(2^(12)-1);
%    angle_arr(i) =
↪  atan(ch0_rms(i)/ch1_rms(i))*180/pi;%atan(mean(ch0)/mean(ch1))*180/pi;%mean(atan(ch0./ch1))*180/pi;
%    adc_min(i) = min(ch0)*3.3/(2^(12)-1);
%    adc_max(i) = max(ch0)*3.3/(2^(12)-1);
%    saveas(gcf, string("ila_measurement_"+freq+"_kHz.svg"));
%end

%adc_range = adc_max-adc_min

%adc_range_bin = adc_range*(2^(12)-1)/3.3
%adc_min_bin = adc_min*(2^(12)-1)/3.3
%adc_max_bin = adc_max*(2^(12)-1)/3.3

%adc_range_diff = max(adc_range_bin) - min(adc_range_bin)
%adc_max_diff = max(adc_max_bin) - min(adc_max_bin)
%adc_min_diff = max(adc_min_bin) - min(adc_min_bin)

%%
%figure()

```

```
%freqs = linspace(2500, 25000, 10);
%plot(freqs, adc_min, 'ro')
%title("LSB at different frequencies")
%ylabel("amplitude (V)")
%xlabel("Frequency")

%figure()
%plot(freqs, adc_max, 'ro')
%title("Highest at different frequencies")
%ylabel("amplitude (V)")
%xlabel("Frequency")

%figure()
%plot(freqs, adc_range, 'ro')
%title("Range at different frequencies")
%ylabel("amplitude (V)")
%xlabel("Frequency")

%figure()
%plot(freqs, ch0_rms)
%hold on
%plot(freqs, ch1_rms)
%title("RMS at different frequencies")
%ylabel("Amplitude (V)")
%xlabel("Frequency")
%legend("x in", "reference")

%% Magnitude
vout = 1368
mag_scaled = ((vout)/(2^(12)-1))*(3.3/2.5)/(1997/1949)

%% Scaling

mag_scale = (1/(2^(12)-1))*(3.3/2.5)/(1997/1949);
ang_fixed = 1/(2^(7));

mag_noload = [2476, 967, 978, 1085, 1120, 1129, 1546, 1581, 1578,
↪  1578].*mag_scale;
mag10 = [938, 967, 978, 1090, 1120, 1128, 1546, 1581, 1578, 1686].*mag_scale;
mag984 = [3177, 3187, 3293, 3303, 3331, 3340, 1365, 1355, 1327, 1368].*mag_scale;
mag3k = [2590, 2599, 2627, 1752, 1755, 2848, 2855, 2884, 2890, 2892].*mag_scale;

ang_noload = [-3549, -3509, -3509, -3541, -3541, -3541, -3549, -3549, -3549,
↪  -3549].*ang_fixed;
ang_10 = [-3395, -3509, -3509, -3541, -3541, -3541, -3549, -3549, -3549,
↪  -3546].*ang_fixed;
ang_984 = [-3509,  -3509, -3541, -3541, -3541, -3541, -3549, -3549, -3549,
↪  -3549].*ang_fixed;
ang_3k = [-3541, -3541, -3541, -3507, -3507, -3549, -3549, -3549, -3549,
↪  -3549].*ang_fixed;

%% Std and variance

noload_var = var(ang_noload)
noload_std = sqrt(noload_var)

ang10_var = var(ang_10)
ang10_std = sqrt(ang10_var)

ang984_var = var(ang_984)
ang984_std = sqrt(ang984_var)

```

```
ang3k_var = var(ang_3k)
ang3k_std = sqrt(ang3k_var)
%% Plot

frequency = linspace(1, 10, 10);
figure()
plot(frequency, mag_noload)
hold on
plot(frequency, mag10);
hold on
plot(frequency, mag984);
hold on
plot(frequency, mag3k);
xlabel("Frequency");
ylabel("Magnitude (A)")
title("Magnitude over frequency")
legend("No load", "10 ohm", "984 ohm", "3 kohm")

figure()
plot(frequency, ang_noload)
hold on
plot(frequency, ang_10);
hold on
plot(frequency, ang_984);
hold on
plot(frequency, ang_3k);
xlabel("Frequency");
ylabel("Angle")
title("Angle over frequency")
legend("No load", "10 ohm", "984 ohm", "3 kohm")

%%
mag_scale = (1/(2^(12)-1))*(3.3/2.5);
ang_fixed = 1/(2^(7));

mag_dac = [2476, 2447, 2435, 1085, 1120, 1129, 1549, 1556, 1590, 1686];
ang_dac = [-3549, -3549, -3549, -3541, -3541, -3541, -3549, -3549, -3549, -3549];

mag_dac_scaled = mag_dac.*mag_scale
ang_dac_scaled = ang_dac.*ang_fixed
```

Source Code D.4: Post-processing and plotting of measured data from TIA

```
%% Startup
close all;
clear all;

path = "june12/";
% Resistors

filename = string(path+"10ohm_1206");
fn1 = filename;

res10_1 = readtable(string(filename + "_1_0.csv"));
res10_2 = readtable(string(filename + "_2_0.csv"));
res10_3 = readtable(string(filename + "_3_0.csv"));
res10_4 = readtable(string(filename + "_4_0.csv"));
res10_5 = readtable(string(filename + "_5_0.csv"));
res10_6 = readtable(string(filename + "_6_0.csv"));
res10_7 = readtable(string(filename + "_7_0.csv"));
res10_8 = readtable(string(filename + "_8_0.csv"));
res10_9 = readtable(string(filename + "_9_0.csv"));
res10_10 = readtable(string(filename + "_10_0.csv"));

filename = string(path+"984ohm_1206");
fn2 = filename;

res984_1 = readtable(string(filename + "_1_0.csv"));
res984_2 = readtable(string(filename + "_2_0.csv"));
res984_3 = readtable(string(filename + "_3_0.csv"));
res984_4 = readtable(string(filename + "_4_0.csv"));
res984_5 = readtable(string(filename + "_5_0.csv"));
res984_6 = readtable(string(filename + "_6_0.csv"));
res984_7 = readtable(string(filename + "_7_0.csv"));
res984_8 = readtable(string(filename + "_8_0.csv"));
res984_9 = readtable(string(filename + "_9_0.csv"));
res984_10 = readtable(string(filename + "_10_0.csv"));

filename = string(path+"3kohm_1206");
fn3 = filename;

res3k_1 = readtable(string(filename + "_1_0.csv"));
res3k_2 = readtable(string(filename + "_2_0.csv"));
res3k_3 = readtable(string(filename + "_3_0.csv"));
res3k_4 = readtable(string(filename + "_4_0.csv"));
res3k_5 = readtable(string(filename + "_5_0.csv"));
res3k_6 = readtable(string(filename + "_6_0.csv"));
res3k_7 = readtable(string(filename + "_7_0.csv"));
res3k_8 = readtable(string(filename + "_8_0.csv"));
res3k_9 = readtable(string(filename + "_9_0.csv"));
res3k_10 = readtable(string(filename + "_10_0.csv"));

%% Plot magnitude
close all;

gain = 1997/1994;
bits = (2^(12)-1);
scale = (3.3)/(2.5);
complete_gain = scale/(gain*bits);

len = length(res3k_1.impedance);
t = linspace(1, len-1, len);
```

```
xlab = "Samples at 100 MHz";
ylab = "Admittance (S)";
for i = 1:10
    res10 = eval(string("res10_"+i));
    res984 = eval(string("res984_"+i));
    res3k = eval(string("res3k_"+i));

    % Plot magnitude
    figure()
    plot(t, res10.impedance.*complete_gain);
    hold on
    plot(t, res984.impedance.*complete_gain);
    hold on
    plot(t, res3k.impedance.*complete_gain);
    xlabel(xlab)
    ylabel(ylab)
    title(string("Admittance at "+i+" kHz"))
    legend("10 Ohm", "984 Ohm", "3 kOhm")
end
```

Source Code D.5: Characterization of CORDIC output

```
close all;

angle_cordic = [0, 0, 0, 180, 180, 90, 45, 63.4765625, 97.140625, 93.53125, 90, 26.5234375, 45, 104.015625, 97.0625, -90, -7.140625, -14.015625, -135, -116.234375, -90, -3.5703125, -7.0625, -153.8046875, -135];
matlab_atan2 = [0, 0, 0, 180, 180, 90, 45, 63.4349, 97.1250, 93.5485, 90, 26.5651, 45, 104.0362, 97.0699, -90, -7.1250, -14.0362, -135, -116.3857, -90, -3.5485, -7.0699, -153.6143, -135];

mat_atan2 = unique(matlab_atan2);
ang_cord = unique(angle_cordic);

diff_ang = mat_atan2 - ang_cord;

mag_cordic = [0, 2048, 1024, 256, 127, 2048, 2590, 2290, 2063, 2051, 1024, 2290, 1295, 1057, 1030, 256, 2065, 1055, 323, 285, 127, 2055, 1032, 287, 160];
mag_matlab = [0, 2048, 1024, 256, 127, 2048, 2896.309376, 2289.733609, 2063.937984, 2051.933966, 1024, 2289.733609, 1448.154688, 1055.51504, 1031.845434, 256, 2063.937984, 1055.51504, 362.038672, 285.7708873, 127, 2051.933966, 1031.845434, 285.7708873, 179.6051224];

% CORDIC sometimes give different magnitude dependent on the sign
mag_cord_u = unique(mag_cordic);
mag_cord1 = [0, 127, 160, 256, 285, 323, 1024, 1032, 1055, 1295, 2048, 2051, 2063, 2290, 2590];
mag_cord2 = [0, 127, 160, 256, 287, 323, 1024, 1030, 1057, 1295, 2048, 2055, 2065, 2290, 2590];
mag_mat_u = unique(mag_matlab);

diff_mag1 = mag_mat_u - mag_cord1;
diff_mag2 = mag_mat_u - mag_cord2;

%%  Plot

figure()
plot(mag_mat_u, diff_mag1, 'ro')
hold on
plot(mag_mat_u, diff_mag2, 'bo-')
yline(0)
title("Difference between sqrt(X^2 + Y^2), and CORDIC magnitude")
legend("Lowest error", "Highest error");
ylabel("Difference")
xlabel("Magnitude")

figure()
plot(mat_atan2, diff_ang, 'ro-')
yline(0)
title("Difference between MATLAB calculated atan2 of angle, and CORDIC angle")
ylabel("Differance (degrees)")
xlabel("Angle (degrees)")
```

Appendix E

Other code

Source Code E.1: Makefile for simulation

```
GHDL=ghdl
FLAGS="--std=08"
IEEE="--ieee=synopsys"

all:
        @$(GHDL) -a $(FLAGS) $(IEEE) pkg/*.vhd
        @$(GHDL) -a $(FLAGS) $(IEEE) synthesis/*.vhd
        @$(GHDL) -a $(FLAGS) $(IEEE) testbench/*.vhd
        @$(GHDL) -e $(FLAGS) $(IEEE) tb_randomized_top
        @$(GHDL) -r $(FLAGS) $(IEEE) tb_randomized_top --vcd=wave.vcd
        ↪  --stop-time=2ms
```

Source Code E.2: Jupyter Notebook

```
{
 "cells": [
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "import matplotlib.pyplot as plt\n",
    "from pynq import Overlay\n",
    "import pynq.lib.dma\n",
    "from pynq import Xlnk\n",
    "import numpy as np\n",
    "import time\n",
    "from pynq import allocate\n",
    "import pandas as pd\n",
    "import serial\n",
    "from pynq import GPIO\n",
    "import time\n",
    "from pynq import Clocks\n",
    "\n",
    "def plot_data(data, name, show):\n",
    "    if show == True:\n",
    "        t = np.linspace(0, len(data)-1, len(data))\n",
    "        plt.plot(t, data, label=f\"{name}\")\n",
    "    \n",
    "        plt.legend()\n",
    "        plt.show()\n",
    "    return\n",
    "\n",
    "def runall(plot=False, filename=\"sweep\", data_len=32, start=0):\n",
    "    #check for start\n",
    "    freqs = 1\n",
    "    start = 0\n",
    "    start = 1\n",
    "    while start == 0:\n",
    "        dma_xin.recvchannel.transfer(out_buffer_xin)\n",
    "        dma_xin.recvchannel.wait()\n",
    "        start_arr, xin = divmod(out_buffer_xin, 2**25)\n",
    "        start = np.count_nonzero(start_arr)\n",
    "    print(\"Sampling start\")\n",
    "    #Recieve\n",
    "    #dma_ref.recvchannel.wait()\n",
    "    #dma_imp.recvchannel.wait()\n",
    "    #dma_cor.recvchannel.wait()\n",
    "    #dma_xin.recvchannel.wait()\n",
    "    for i in range (10):\n",
    "        \n",
    "        print(f\"Frequency {freqs}kHz\")\n",
    "        t_start = time.time()\n",
    "        \n",
    "        dma_xin.recvchannel.transfer(out_buffer_xin)\n",
    "        dma_ref_cs.recvchannel.transfer(out_buffer_res)\n",
    "        dma_lia.recvchannel.transfer(out_buffer_rea)\n",
    "        dma_cordic.recvchannel.transfer(out_buffer_ang)\n",
    "        #Wait\n",
    "        dma_xin.recvchannel.wait()\n",
    "        dma_ref_cs.recvchannel.wait()\n",
    "        dma_lia.recvchannel.wait()\n",
```

```
60        "        dma_cordic.recvchannel.wait()\n",
61        "        t_end = time.time()\n",
62        "        #axi_gpio.write(0x4, 0)\n",
63        "        #Arrays\n",
64        "        \"\"\"\n",
65        "        if data_len == 64:\n",
66        "            fcw, sincoselec = divmod(out_buffer_ref, 2**27)\n",
67        "            o_electrode, sincos = divmod(sincoselec, 2**24)\n",
68        "            sin, cos = divmod(sincos, 4096)\n",
69        "            fcw_d, x1x0elec = divmod(out_buffer_xin, 2**27)\n",
70        "            i_electrode, x1x0 = divmod(x1x0elec, 2**24)\n",
71        "            x1, x0 = divmod(x1x0, 4096)\n",
72        "            o_el_imp, imp = divmod(out_buffer_imp, 2**27)\n",
73        "            o_el_cor, cor = divmod(out_buffer_cor, 2**27)\n",
74        "            res, rea = divmod(imp, 2**24)\n",
75        "            mag, ang = divmod(cor, 2**24)\n",
76        "            d = {'sin': sin, 'cos': cos, 'x0': x0, 'x1': x1, 'res': res,
          ↪  'rea': rea, 'mag': mag, 'ang': ang, 'fcw': fcw, 'fcw_d': fcw_d, 'o_elec':
          ↪  o_electrode, 'i_elec': i_electrode}\n",
77        "            df = pd.DataFrame(data=d)\n",
78        "            df.to_csv(f'{filename}.csv')\n",
79        "        elif data_len == 32:\n",
80        "        \"\"\"\n",
81        "        # XIN sample\n",
82        "        fcw, xin_rest1 = divmod(out_buffer_xin, 2**27)\n",
83        "        o_electrode, xin_rest2 = divmod(xin_rest1, 2**24)\n",
84        "        x1, xin = divmod(xin_rest2, 2**12)\n",
85        "        \n",
86        "        # RES/REF_CS sample\n",
87        "        btn, ref_rest1 = divmod(out_buffer_res, 2**27)\n",
88        "        i_electrode, ref_rest2 = divmod(ref_rest1, 2**24)\n",
89        "        cos, sin = divmod(ref_rest2, 2**12)\n",
90        "        \n",
91        "        # REA/LIA sample\n",
92        "        quad, inph = divmod(out_buffer_rea, 2**13)\n",
93        "               \n",
94        "        # ANG sample\n",
95        "        mag, angle = divmod(out_buffer_ang, 2**16)\n",
96        "        \n",
97        "        \n",
98        "        # Magnitude\n",
99        "        impedance_gain = 0.6\n",
100       "        impedance = mag*impedance_gain\n",
101       "        \n",
102       "        d = {'btn': btn, 'xin': xin, 'impedance': impedance, 'angle': angle,
          ↪  'inphase': inph, 'quadrature': quad, 'fcw': fcw, 'o_elec': o_electrode,
          ↪  'i_elec': i_electrode}\n",
103       "        df = pd.DataFrame(data=d)\n",
104       "        df.to_csv(f'{filename}_{i}.csv')\n",
105       "        \n",
106       "        print(f\"Sample time: {t_end-t_start}\")\n",
107       "        #Plot/\n",
108       "        if plot == True:\n",
109       "            #plot_data(cos, \"cos\", False)\n",
110       "            #plot_data(sin, \"sin\", True)\n",
111       "            plot_data(xin, \"x0\", True)\n",
112       "            plot_data(x1, \"x1\", False)\n",
113       "            plot_data(inph, \"res\", False)\n",
114       "            plot_data(quad, \"rea\", False)\n",
115       "            plot_data(impedance, \"impedance\", False)\n",
116       "            plot_data(angle/(4095), \"angle\", False)\n",
117       "        print(f\"FCW: {fcw}\")\n",
```

```
    "        print(f\"o_elec: {o_electrode} i_elec: {i_electrode}\")\n",
    "        freq_change = 0\n",
    "        while freq_change == 0:\n",
    "            dma_xin.recvchannel.transfer(out_buffer_xin)\n",
    "            dma_xin.recvchannel.wait()\n",
    "            btn, xin_rest1 = divmod(out_buffer_xin, 2**25)\n",
    "            start, freq_btn = divmod(btn, 2**2)\n",
    "            freq_change = np.count_nonzero(freq_btn)\n",
    "        freqs = freqs + 1\n",
    "    \n",
    "    return"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "scrolled": false
   },
   "outputs": [],
   "source": [
    "# Setup\n",
    "overlay = Overlay('/home/xilinx/pynq/overlays/Sweep/EIT.bit')#eit_fir_wrapper/PYNQ2.bit')#\n",
    "\n",
    "dma_xin = overlay.axi_dma_1\n",
    "dma_ref_cs = overlay.axi_dma_0\n",
    "dma_lia = overlay.axi_dma_2\n",
    "dma_cordic = overlay.axi_dma_3\n",
    "#axi_gpio = overlay.axi_gpio_0\n",
    "\n",
    "#dma_ang = overlay.axi_dma_4\n",
    "#dma_mag = overlay.axi_dma_5\n",
    "\n",
    "#xlnk = Xlnk()\n",
    "data_size = 65536\n",
    "\n",
    "sin = np.zeros(data_size)\n",
    "cos = np.zeros(data_size)\n",
    "x0 = np.zeros(data_size)\n",
    "x1 = np.zeros(data_size)\n",
    "\n",
    "out_buffer_xin = allocate(shape=(data_size,), dtype=np.uint32)\n",
    "out_buffer_res = allocate(shape=(data_size,), dtype=np.uint32)\n",
    "out_buffer_rea = allocate(shape=(data_size,), dtype=np.uint32)\n",
    "out_buffer_ang = allocate(shape=(data_size,), dtype=np.uint32)\n",
    "#out_buffer_ang = allocate(shape=(data_size,), dtype=np.uint32)\n",
    "#out_buffer_mag = allocate(shape=(data_size,), dtype=np.uint32)\n",
    "Clocks.fclk0_mhz = 100"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "scrolled": false
   },
   "outputs": [],
   "source": [
    "#while rst = 1:\n",
    "cap = 1\n",
    "gain = \"1_56\"\n",
```

```
      "supply = 5\n",
      "e_freq = \"2ph\"\n",
      "sweep = \"sweep\"\n",
      "data_len = 32\n",
      "\n",
      "runall(True,
      ↪  filename=f\"3kohm_1206_10\",data_len=32)#{sweep}_{gain}gain_cap{cap}_{e_freq}_{supply}V\",
      ↪  data_len=32)"
     ]
    }
   ],
   "metadata": {
    "kernelspec": {
     "display_name": "Python 3",
     "language": "python",
     "name": "python3"
    },
    "language_info": {
     "codemirror_mode": {
      "name": "ipython",
      "version": 3
     },
     "file_extension": ".py",
     "mimetype": "text/x-python",
     "name": "python",
     "nbconvert_exporter": "python",
     "pygments_lexer": "ipython3",
     "version": "3.6.5"
    },
    "widgets": {
     "application/vnd.jupyter.widget-state+json": {
      "state": {},
      "version_major": 2,
      "version_minor": 0
     }
    }
   },
   "nbformat": 4,
   "nbformat_minor": 2
  }
```