Design and Fabrication of a Measurement Interface for Smart IoT Sensors

Bachelor Thesis

Delft University of Technology

Signal Generation and Hardware Control

by

Patrick Geel
Zep Kleijweg

In partial fulfillment of the requirements for the degree of

Bachelor of Science
in Electrical Engineering

at Delft University of Technology

To be defended on Thursday June 27th 2019 at 10:30

Supervisor:	Dr. Ir. S. Vollebregt
PhD Supervisor:	Ir. J. Romijn
Chairman:	Dr. Ir. M.A.P. Pertijs
Jury Member:	Dr. C. García Almudever

Abstract

This project is to design and implement a reconfigurable measurement interface for Internet of Things sensors, for the Microelectronics Department of the Delft University of Technology. This thesis will discuss the functionality and design process taken in designing such a reconfigurable measurement interface, focussing on generating signals and control control signals for the hardware. The important choices will be highlighted and the strengths and weaknesses of each design choice will be weighed in order to produce a fully functional measurement interface. In the span of 10 weeks this group was able to successfully generate sinusoidal, square, and triangular waves as well as DC voltages at +12V and +24V and in the voltage range of -10V to +10V on different pins of the dip-48 socket. Sensor output voltages can also be measured and observed using an external computer.

Preface

This thesis is written in partial fulfillment of the Electrical Engineering Bachelor at the Delft University of Technology. This project originates out of the Microelectronics Department, the goal to deliver a functioning product in a span of 10 weeks.
We would like to express our gratitude to our daily supervisor Dr. Ir. S. Vollebregt, our PhD supervisor Ir. J. Romijn for their support and openness during this project, Ir. L. Pakula for being present at our green light assessment, as well as Dr. Ir. I.E. Lager for overseeing all bachelor graduation projects. We would also like to thank our families, friends, and roommates for their support during this project and taking the time to supply feedback on this thesis. Finally we would like to thank Ze-Sheng Zhuang, Michel Wervers, Daan Verrer, and Maima Postma for all the hard work put in during this project and most importantly for a lot of laughs over the last 10 weeks.

Patrick Geel & Zep Kleijweg
Delft, June 2019

Contents

Chapter 1

Introduction

The internet of things (IoT) is a rapidly growing technology, and sensors are an important part of this field. Some examples of the use for sensors that are connected to the internet are gathering information about a large system or geographical area without the need to physically collect the data at each sensor, place the sensor in a harsh environment without the need to run a cable to it for real-time read out, and they enable easy integration between multiple sensors. All these different sensors need to be tested extensively before they can be deployed into the real world. Because the sensor requirements are different from case to case, a new test setup must be designed for each new sensor. This can be very time consuming and the test setups are rarely used after testing a single chip design, which is a waste of time and money. In order to simplify the testing process a generalized testing interface is desirable.

1.1 Project Objective

The goal of this project is to design and fabricate a measurement interface with a wide range of functionality with the key factor being that generated signals are all reconfigurable, allowing users to test a variety of sensors using the same setup, saving time and money that would otherwise be spent designing and building a sensor specific test setup. The measurement setup should allow a user to generate simple signals and bias voltages, read sensor output, connect external equipment for higher accuracy, and to control the setup using a computer. As seen in Figure 1.1, the overall project is split into three parts: building a graphical user interface (GUI), generating analog signals and control signals, and hardware to create the correct voltage levels. The part of the project focused on in this thesis is the mediator between the user interface and hardware, generating control signals to integrate with the hardware, as well as generating simple signals according to user input.

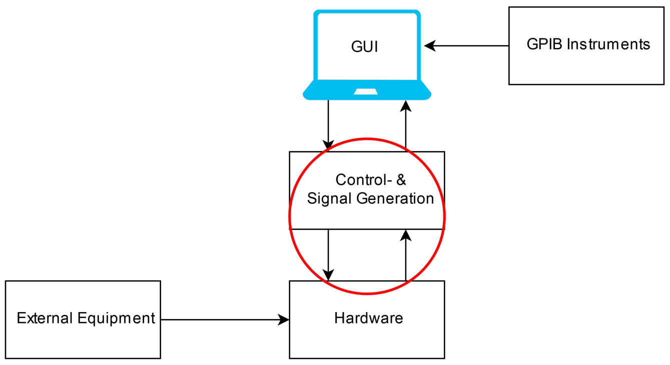

Figure 1.1: System overview

1.2 Signal Generation & Measurements

As highlighted above, generation of basic signals as well as control signals is required. The option should also be available to perform measurements and graphically visualize these measurements performed on a device under test (DUT). The generation of the basic/control signals can be broken down into digital and analog signals. The most common form of analog signal generators found in most research labs is a Direct Digital Synthesis (DDS) signal generator [1]. In Figure 1.2 the block diagram of such a DDS signal generator can be found. The crucial component of a DDS signal generator is the phase accumulator, as this component accumulates the frequency control word, K, at intervals of the clock frequency, f_c. This is then output in binary code to the ROM, which uses a look up table to obtain the correct output amplitudes. As this is still in the digital domain a digital analog converter (DAC) is used to generate the respective signal in the analog domain. A low pass filter is applied to remove distortion of higher frequency components out of the output signal, f_0.

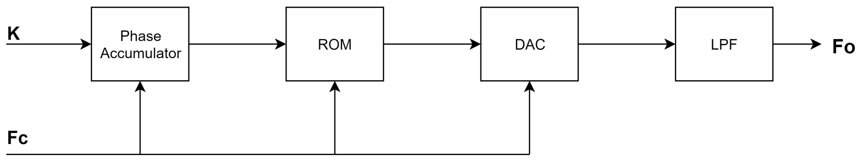

Figure 1.2: DDS block diagram[1]

The measuring and displaying of signals is done using an oscilloscope, this device draws a graph of instantaneous signal voltages. As the signal generator and the oscilloscope are two separate units a desirable result of this project is to have a signal generator and oscilloscope in one easy to use package. Therefore, a method to create both a signal generator and to display measured voltages in one unit is studied. Ways to implement a signal generator that can be controlled via a computer are by the use of FPGA's, microcontrollers, and multifunctional I/O modules.

Field Programmable Gate Array (FPGA)

One method of generating signals is utilizing an FPGA. This contains reprogrammable logic blocks, which allows for flexibility as they are reprogrammable in Hardware Description Languages (HDL) such as VHDL or Verilog. Thus with such reprogrammablity an FPGA can be used as the brains to make a signal generator. Using this method it is possible to make a signal generator capable of generating frequencies from 1 Hz to 20 MHz [2]. A limiting factor of using an FPGA based design is the limitation of the digital to analog converter (DAC), as a DAC with low resolution will create poor signals. The influence of the bit accuracy can be seen in Figure 1.3.
Using this method it is also possible to analyze measured signals. As an FPGA is a digital device, an analog to digital converter [3] is needed to convert the analog signal into a digital signal. This digital signal is then passed along to the FPGA such that it is able to communicate the signal to the user interface (UI). The communication between the UI and the FPGA needs to be able to take place in both directions: the signal generation needs to be controlled from the UI, and the measurement data needs to be sent from the FPGA to the UI. This can be implemented using a communication protocol such as UART [4].

Microcontroller

Another method investigated as possible solution to generate basic/control signals is the approach of using a microcontroller as brains of the signal generation, this method uses the DDS approach to create analog signals. The microcontroller is used to generate a bit pattern at intervals of the clock frequency which represents the amplitude and frequency of the desired output, which is passed to a digital analog converter (DAC) to convert this bit pattern to an analog signal. As for measuring inputs, this is similar to the method explained in the FPGA section. The

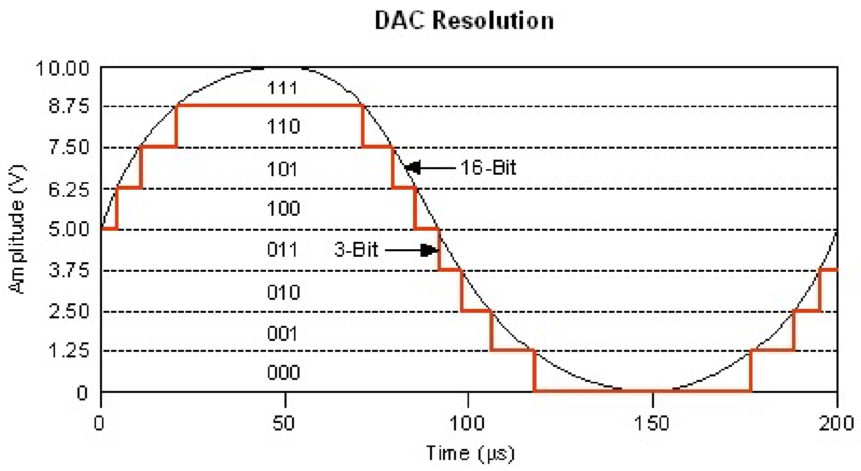

Figure 1.3: Resolution of DAC [5]

microcontroller acts as the brains which do all the computations [6]. Another similarity to the FPGA is that this approach would also require the implementation of a communication protocol to transfer the data back and forth from microcontroller to the UI.

Data Acquisition Device

A final method examined as possible signal generator is using a multifunction I/O module, also called a data acquisition unit (DAQ). Similar to the FPGA the in-/outputs of a DAQ can be reprogrammed, using a graphical programming language (G programming language) like LabVIEW [7]. Other examples of G programming languages are Simulink [8] and NI multisim [9]. Such a unit can also measure input voltages and it is possible to easily display measurement results to a user, through a user interface. LabVIEW makes it easy to program a multifunctional I/O module to output desired signals such as sinusoidal, triangle, and square waves, as well as DC voltages. This approach can shorten the development time as there are a handful of resources available, such as LabVIEW this program has many built in functionalities which are easy to use. An example of using LabVIEW can be found in Figure 1.4, which contains an example of generating a simple sinusoidal wave.

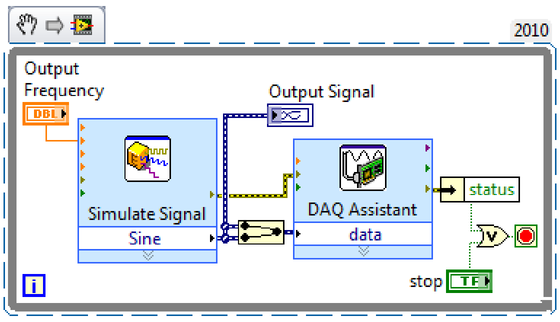

Figure 1.4: Signal generation using LabVIEW

1.3 Signal Quality Specifications

There are a lot of different properties that describe the quality of a signal. Unfortunately, there is not a limited set of properties that fully describes the way a signal behaves. In this section the more commonly used properties will be described [10], [11]. These different measures can be used to indicate to the end user what the accuracy of the generated signals is.

1.3.1 Amplitude Flatness & Bandwidth

The amplitude flatness describes how the output amplitude of the generated signal varies depending on the frequency of the signal. Ideally, the frequencies within the bandwidth are all produced with the same amplitude, but different distortions can occur at different frequencies. A frequency sweep can be performed to measure the output amplitude across a range of frequencies. This involves outputting a signal with a continuously higher frequency, while keeping the intended output power constant. The achieved output power can then be plotted against the frequency, showing the frequency response of the system. An example of a frequency response can be seen in Figure 1.5.
The 3dB-bandwidth of a signal generator is defined as the frequency at which the output signal is attenuated by 3dB with respect to the frequency with the highest power, which should be within the pass-band [12]. The output sample rate is one of the factors that limits the bandwidth, and should at least be twice as high as the highest desired frequency component in the signal, as described by the Nyquist-Shannon sampling theorem [13]. However, a much higher sample rate is desirable to approximate a continuous signal, as shown in Figure 1.6. The method described above not only shows the amplitude flatness, but should also produce a graph from which the -3dB frequency can be read, as long as the sweep continues to high enough frequencies to actually reach the bandwidth frequency.

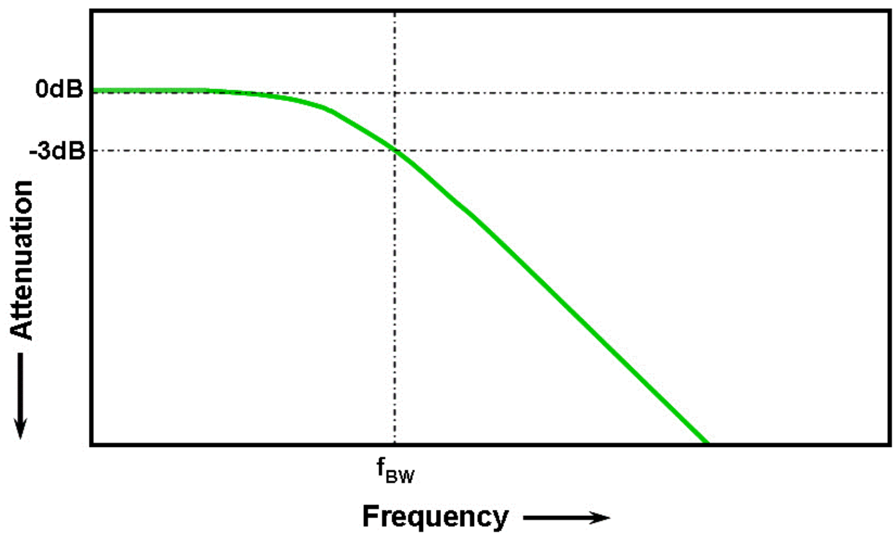

Figure 1.5: Example of a frequency response [14]

1.3.2 Resolution

As described in Section 1.2, the analog output signals will be created using a digital to analog converter (DAC). This means that the output is still quantized in amplitude, and not completely smooth. The distance between the different voltage levels depends on the amount of bits used to create the signal according to Equation (1.1), where step is the step size between the discrete voltage levels, V_R is the maximum input voltage range and N is the number of bits. A graphical representation of this can be seen in Figure 1.3, where the same sine wave is plotted when output using both a 3-bit and a 16-bit DAC. Naturally, using a DAC that uses more bits results in the generation of a more smooth output wave.

$$Step = \frac{V_R}{2^N} \tag{1.1}$$

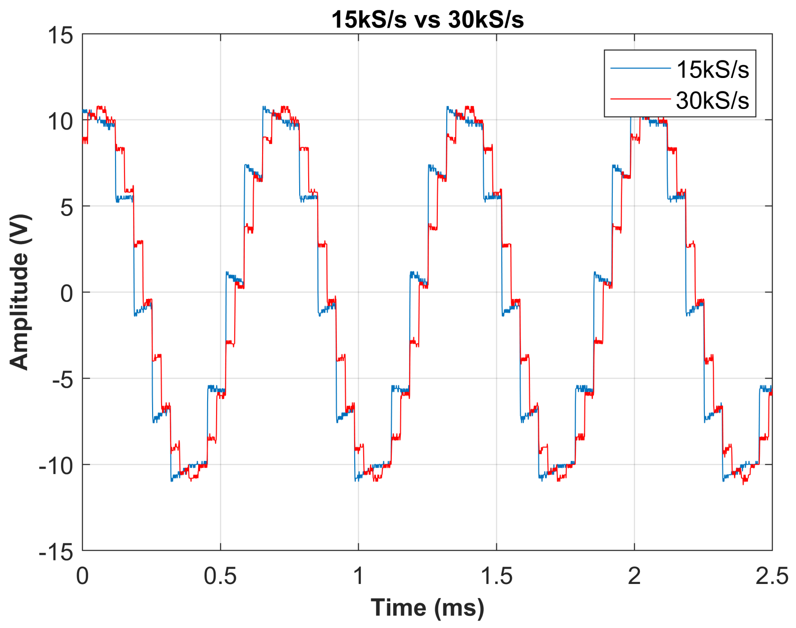

Figure 1.6: Sampling a 1.5 kHz sine with different sample rates

1.3.3 Total Harmonic Distortion

The lowest frequency in a periodic signal is the fundamental frequency, this is the desired frequency. However, the power spectral density of a periodic signal shows that the power is distributed into multiple frequency components, which are at integer multiples of the fundamental frequency [15]. An important measure to quantify this type of distortion is the Total Harmonic Distortion (THD) [16].
The THD can be calculated in two different ways. In one definition the harmonic content is compared to the fundamental frequency, THD_F, in the second it is compared to the signals root mean square value, THD_R. Equations (1.2) and (1.3) can be used to calculate the different values. The different definitions can cause ambiguity, but in this thesis the first definition, Equation (1.2), is meant when discussing the THD.

$$THD_F = \frac{\sqrt{\sum_{n=2}^{\infty} V_n^2}}{V_1^2} \tag{1.2}$$

$$THD_R = \frac{\sqrt{\sum_{n=2}^{\infty} V_n^2}}{\sqrt{\sum_{n=1}^{\infty} V_n^2}} \tag{1.3}$$

1.3.4 Intermodulation Components

Another aspect is that the output of the devices is not continuous, but updated at a set rate: the sample rate. This causes intermodulation between the sample frequency and the fundamental frequency of the generated signal [17]. These frequency components appear at the frequencies described in Equation (1.4), where f_s is the sample frequency, f_0 the fundamental frequency of the signal, n and m are natural numbers, and f is the unwanted frequency component.
The amount of distortion can be determined by looking at the power spectral density of the signal and comparing the power of the unwanted frequency components to the power of the fundamental frequency.

$$f = \pm n f_s \pm m f_0 \tag{1.4}$$

1.3.5 Aliasing

Aliasing is a phenomenon that can occur due to sampling a signal. When sampling a signal, its spectrum will repeat not only around frequency zero, but also every multiple of the sampling rate [18], as shown in Figure 1.7 (b). In the case when the maximum frequency in the signal is higher than half of the sampling frequency, as described by Equation (1.5), the spectra of the original signal and the spectra around the next multiple of the sample frequency will overlap, an extreme case of which is shown in Figure 1.7 (c).

$$f_s \leq 2f_{max} \tag{1.5}$$

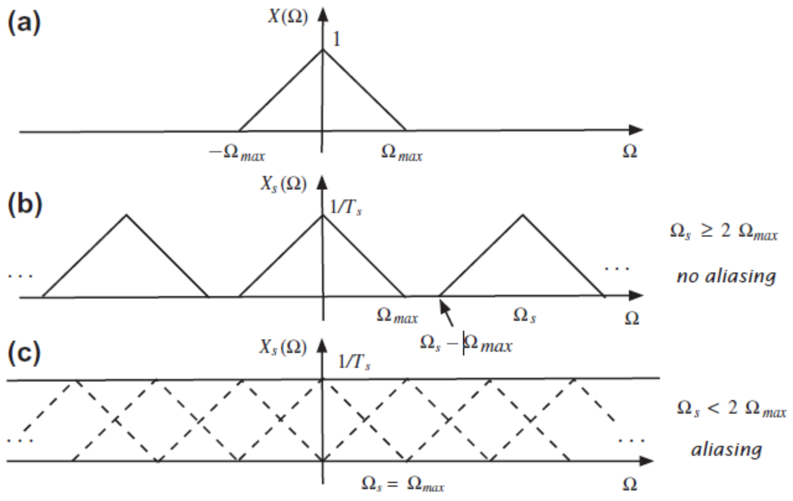

Figure 1.7: (a) Frequency spectrum of a signal, (b) spectrum of sampled signal when satisfying Nyquist condition, (c) spectrum with aliasing: superposition of individual spectra, which are shown as dotted lines [18]

1.3.6 Spurious Free Dynamic Range

The spurious free dynamic range (SFDR) is the ratio between the RMS value of the fundamental frequency component and the next largest RMS frequency component, regardless of its frequency [19]. Frequency components at different frequencies than the fundamental, with an amplitude above the noise floor, are called spurs. The SFDR characterizes the dynamic range of a signals generator [20]. It can be calculated visually from the frequency spectrum of a generated signal, by finding the RMS value of the fundamental frequency as well as the RMS value of the highest spur.

1.3.7 Phase Noise

A perfect signal generator would produce a discrete frequency component at exactly the fundamental. In reality, there is usually a small band of frequencies around the fundamental where power is concentrated, as seen in Figure 1.8. This is often due to clock jitter in the clock signal of the signal generator [21]. Sometimes the frequency of the clock is a bit too high, causing the generated signal to be of a too high frequency, or the clock frequency is slightly too low and the generated frequency is also slightly lower than intended. This phenomenon is called phase noise and the result is that frequency components occur in a small range and not as one peak.

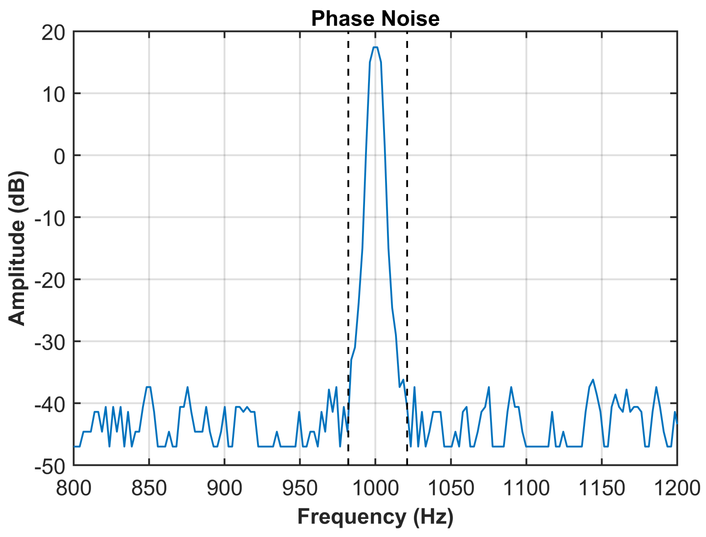

Figure 1.8: Phase noise

1.3.8 Rise and Fall Times

Rise and fall times are relevant when generating a square wave. The rise time is the time it takes to switch from the low voltage level to the high voltage level, and the fall time is the reverse: the time it takes to switch from the high voltage level to the low voltage level. Both are displayed in Figure 1.9. Ideally, these are both zero: switching is instantaneous. In reality, this takes some time. The main cause for this are capacitances which need to be either charged or discharged on a rising or falling edge respectively.

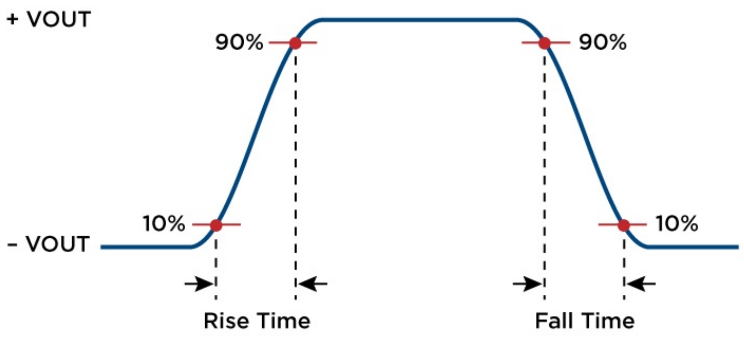

Figure 1.9: Rise and fall times of a signal [22]

1.3.9 Overshoot & Undershoot

Another phenomenon that can occur when generating a square wave is overshoot. This means that the signal rises above the desired voltage on a rising edge, or falls below the desired voltage on a falling edge, and than settles to the desired voltage. During this settling, the voltage can "overshoot" again, but in this case it is called undershoot as it goes in the reverse direction of the edge. This is shown in Figure 1.10.

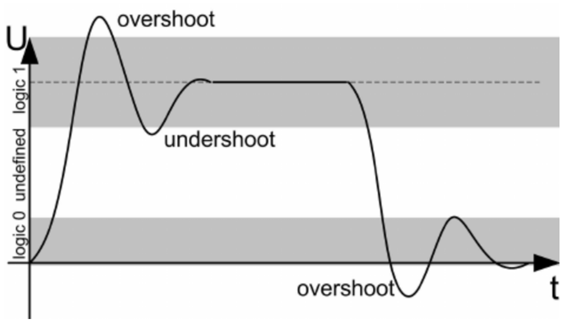

Figure 1.10: Over- and undershoot [23]

1.4 Control Signals

Part of this project focuses on the generation of control signals, these are signals which are needed to translate the data from software to hardware and setting the correct signals on the correct pins. Commonly used methods to transfer data are parallel and serial communication. Parallel data communication is extremely fast as parallel communication allows data to be transferred over multiple lines at the same time, which works well for a short distance. However, at long distances errors occur due to signal attenuation over long distances [24]. Serial communication transmits 1 bit at a time over a single line, which is better for long distance communication. It is, however, slower than parallel communication. Both methods can be seen in Figures 1.11 & 1.12.

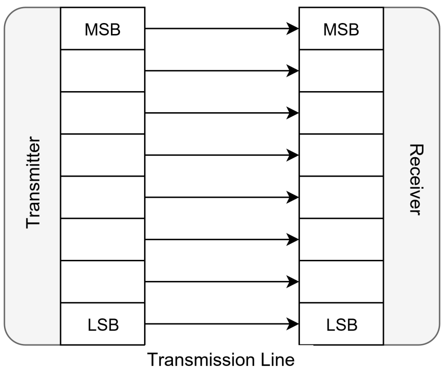

Figure 1.11: Parallel communication

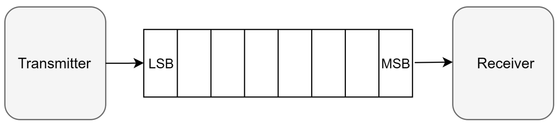

Figure 1.12: Serial communication

The necessary control signals must all arrive simultaneously, as control signals arriving at different times may cause an error to be made due to the incorrect bit pattern arriving at the receiver. The first and most direct way to transfer these control signals is a direct connection from A to B: parallel communication. Using parallel communication each control signal must have its own connection, which can be a problem since the device used to generate these signals will have a limited number of outputs. A second approach would be to use serial communication to transfer the control signals. As the control signals need to arrive simultaneously this replaces the problem of using too many outputs with a new one: how to output serially received data simultaneously? For this, a shift register can be used [24]. A shift register can consecutively read in bits and output all of them simultaneously. This does, however, introduce some latency in the system which does not occur in a parallel system. The choice remains a trade-off between the amount of necessary output channels and latency.
As mentioned with the parallel communication method above, a possible problem that may arise is the limited number of output pins of the selected signal generator. A work around to this problem is to use an encoding and decoding approach. This can reduce the number of

necessary output pins according to Equation (1.6), where e is the number of encoded bits and d is the number of decoded bits. This reduces the amount of necessary output pins, allowing more flexibility when there is a limited number of output pins available on the signal generator.

$$e = \lceil \log_2 d \rceil \tag{1.6}$$

1.5 Thesis Outline

This thesis is structured as follows, Chapter 2 will outline the requirements which are set in accordance with the supervisors needs. Chapter 3 will discuss the thought process taken in decision making of this project, as well as describe the design process in detail. Chapter 4 discusses all the results obtained from testing the design. The results obtained from these measurements will be compared with what is expected to validate that the design choices taken function as expected and any discrepancies from what is expected will be discussed in detail. This chapter is followed by the Discussion, where the most important observations of the project will be elaborated on, finishing with the concluding remarks about this project.

Chapter 2

Programme of Requirements

The functional requirements describe the purpose of the measurement setup, as well as the task that it should be able to perform. The functional requirements are split into primary and secondary requirements, where the primaries are essential in order to produce a properly working product and the secondaries are nice to have to increase functionality or user friendliness. This distinction also takes the available development time into account. The system requirements are more technical and describe the performance of the system.

2.1 Functional Requirements

Primary Requirements

1. 5 Digital outputs:
   a. Constant bias voltages at 3.3 V, 5 V, 10 V, 12 V and 24 V.
2. 4 Analog/Digital outputs:
   a. Sinusoidal, square and triangular waves as well as constant voltages, ranging from -10V to 10V.
   b. Constant (digital) bias voltages at 12 V and 24 V.
3. Instead of using internal signal generation:
   a. The option to connect external equipment to the sensor.
   b. The possibility to measure sensor voltage output and display it in the user interface.
4. The output signals need to be reconfigurable during run time, meaning the signals can be changed and turned on or off.

Secondary Requirements

5. 5 Digital outputs:
   a. Square waves with the same amplitudes as the bias voltages and variable duty cycle and frequency up to 100 kHz.
6. The output current of generated signals can be measured and displayed in the user interface.

2.2 System Requirements

7. The different channels can be controlled independently and simultaneously.
8. Analog signals can be generated with a frequency up to 100 kHz.
9. Distortions in the generated signals need to be documented to allow the user to account for them in their measurements.

Chapter 3

Design Process

This chapter will describe the design process in detail. First the choice of signal generating device will be described, followed by the software design and implementation. Lastly, the trade-offs made in the communication of the control signals will be discussed.

3.1 Generation Device Choice

As highlighted in Section 1.2, a decision needs to be made on which signal generator to use, as this will be the core of this project. In this section a series of comparisons between the FPGA, microcontroller, and DAQ will be assessed to choose the best option for this project.

3.1.1 Comparison DAQ, FPGA and Microcontroller

To better reflect on the differences between each of the three choices a comparison table was formulated in order to better compare each methods strengths and weaknesses, these can be found in Table 3.1.

Table 3.1: Comparison of three options

	DAQ	**Microcontroller**	**FPGA**
Clock Rate	+	-	+
Cost	−	0	+
Development Time	++	-	0
In-/Output	0	+	-
Computing Power	+	0	+
Availability	++	0	0

As outlined the DAQ has many strong points, such as development time and availability. As the choice needs to be narrowed down, either the FPGA or microcontroller should be eliminated as choice to compare the DAQ with either the FPGA or microcontroller side by side. Development time is something that is seriously taken into consideration when choosing which one to remove, as this project should be completed in 10 weeks. A microcontroller is programmed in C/C++ while an FPGA is programmed in Hardware Description Language (HDL). A personal preference is to program in VHDL, as this subgroup has more experience programming in VHDL than in C/C++. Another point to consider is how often each method is used in such a project. Microcontrollers are less commonly used to make a signal generator, this is concluded as there are fewer project which use a microcontroller as signal generator and more commonly they use an FPGA to make signal generators. With these things kept in mind the choice is made to remove a microcontroller as an option, not stating that it is impossible to make a signal generator with a microcontroller, however, the personal preference weighs in favor of the FPGA when comparing it with a microcontroller.

3.1.2 DAQ vs. FPGA

To better evaluate the difference between the final two, the PXI 6229 DAQ [25], which is available from the microelectronics department, and the Nexys 2 Spartan-3E FPGA [26] will be compared. This FPGA was chosen for comparison as it has comparable specifications. In Table 3.2 important criteria for this project are compared of both.

Table 3.2: Comparison between PXI 6229 and Nexys 2

	PXI 6229 (DAQ)	**Nexys 2 Spartan-3E**
Clock Speed	80 MHz	50 MHz
DAC Resolution	16 bit	N/A
Ease to Program	Simple, programmable in LabVIEW	Programming in HDL, more complex
Availability	Direct	Order
In-/outputs ports	32 Analog Inputs, 4 Analog outputs, 48 Digital I/O	60 Digital I/O

As depicted in Table 3.2, the PXI 6229 has more in-/output ports than the Nexys 2. The number of necessary in-/outputs depends on the communication method used, which will be explained in Section 3.3, but both options will have plenty available. An important difference is that the DAQ has a built in DAC, while the FPGA would need an external DAC. Having this integrated in the signal generating device makes controlling it and outputting desired analog signals easier. A key factor that should not be overlooked is the ease to program the device, to keep the development time short. An FPGA is programmable through an HDL, which is quite low-level programming. The DAQ has high-level software and drivers readily available to control it, as well as signal generation built into the software making the generation of sinusoidal waves, square waves, etc. rather simple. Another key factor is the availability of each: the microelectronics department of TU Delft has a DAQ available for use, while an FPGA needs to be bought.
The most important factor in choosing which form of signal generator will be used in this project is development time, as this project only spans a 10 week period and the desired result is to deliver a working product, the complexity of programming weighs heavily in the decision. Easier programming will give less problems thus increasing the chances to deliver a working prototype after 10 weeks. As the DAQ is also readily available for use, has plenty of in-/outputs and a built in DAC with a 16 bit resolution, the choice is made to use this device to generate the signals.
The DAQ will be controlled by a computer that is integrated into the same housing as the DAQ for ease of use as well as mobility: this enables the user to easily move the complete system. This computer runs Windows 10, as well as LabVIEW 2018 to control the DAQ.

3.2 LabVIEW Implementation

The choice of using the DAQ developed by National Instruments also implies the use of their software to control it: LabVIEW. This is a graphical programming language, which aids in visualizing what is being worked on. A LabVIEW program is called a Virtual Instrument (VI) and can contain smaller VI's, called subVI's, to allow encapsulation of functionality. National Instruments also offer an extra driver, NI-DAQmx [27], that is focused on controlling PC-based data acquisition. This offers a lot of VI's to control the in- and output pins of the DAQ. There are also VI's available to generate analog waveforms.
To keep the main VI clear, it needs to be structured properly. A logical split in functionality is to create a subVI for the digital channels and one for the analog/digital channels. Each of these channels is split up into different states, where each state is responsible for one functionality. A channel can not perform two functionalities simultaneously, so a channel should only be able to be in one state at a time.

3.2.1 Digital Channels

The different digital states and the corresponding control signals can be found in Table 3.3, and the way the control signals are built up is shown in Figure 3.1. The control signals are discussed in more detail in Section 3.3. The way the code is implemented is by using a case statement to select between three different output cases: measure, external/digital off and digital on. In the external state the user can use an external connection to the sensor instead of using the DAQ. Since the control signals that are necessary for the external state are all off, this state doubles as the channel off state. The measure voltage state can be used to measure the voltage output of the sensor under test and display it in the user interface. Since there are enough analog inputs available in the DAQ, the choice was made to also use analog inputs for the voltage measurements on the digital channels, to give the user more freedom in how to connect their sensor. If a digital voltage is measured it will be displayed as a DC voltage at the digital voltage level. The aggregated sample rate for the measurements is limited by the DAQ to 250 ks/s, which is split evenly over all the channels the user wants to measure. The digital output case contains another case statement that selects which voltage level is desired, and outputs the corresponding control signals. In this state the user can also select to measure the output current delivered to the sensor, which is then displayed in the user interface.

Table 3.3: Digital channel states and control signals

State	Control Signals
Measure voltage	1000 000
External/off	0000 000
3.3 V	0100 001
5 V	0100 010
10 V	0100 100
12 V	0101 000
24 V	0110 000

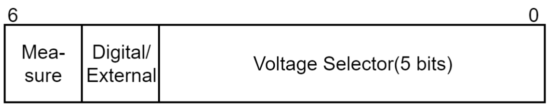

Figure 3.1: Digital control signals

One of the secondary requirements is to generate square waves at the same voltage levels as the digital bias voltages. A possible implementation for this is to use a flat sequence: this first performs the actions in the first frame and continues to the next frame when everything in the first frame is done, and so on. In the first frame the output is made high and the Wait VI is used to wait for a time of $\frac{dutycycle}{frequency}$, which corresponds to the time the signals needs to be high. After this time the program continues to the next frame, in which the output is made low for a time of $\frac{1-dutycycle}{frequency}$, which corresponds to the time the signal needs to remain low.
This approach turned out to work at low frequencies, but problems occurred when raising the frequency: the output would remain either low or high, or change at random times. This is due to the fact that this approach uses software timing: the Wait VI does not use a hardware clock signal to time how long to wait. This approach can not provide accurate wait times for high frequencies. Another problem is that the timing depends on the software: when the user performs an action in the software this unbalances the wait times which translates into jitter in the square wave. These actions can be changing settings in the user interface, or even something as simple as moving the mouse pointer around. This makes that this approach is not reliable, even for the lower frequencies at which it first seemed to work properly.
A solution is to use hardware timing instead of software timing. However, this turned out to be easier said than done: no solution to implement this has been found at this moment in time. Even the active LabVIEW community did not seem to have an answer on how to make this work, unfortunately.

3.2.2 Analog/Digital Channels

The different analog/digital states and the corresponding control signals can be found in Table 3.4, and the way the control signals are built up is shown in Figure 3.2. The way the code is implemented is by using a case statement to select between four different output cases: measure, external/off, analog, and digital. The measurement and off states function the same as in the digital channels, and signal current measurements can be performed in both the analog and the digital states. In the digital case is another selector to select between generating a 12 V or 24 V bias signal. In the analog state different waveforms can be generated: sinusoidal, square and triangular waves, as well as DC bias voltages. For all these signals the voltage can range from -10 V to +10 V. For the waveforms the frequency and offset can also be set up by the user. The square wave's duty cycle can be changed as well. There is a LabVIEW VI available that generates a waveform according to the user input, this waveform is then output to the main VI. The maximum output sample rate for the analog outputs of the DAQ is limited to 625 ks/s. The influence of this will be investigated in Chapter 4.

Table 3.4: Analog/digital channel states and control signals

State	Control Signals
Measure	1000 0
External/off	0000 0
Analog	0001 0
Digital 12 V	0010 1
Digital 24 V	0100 1

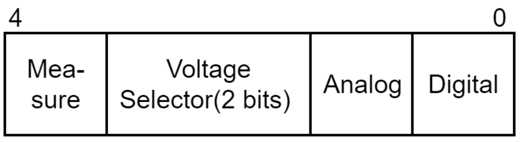

Figure 3.2: Analog/digital control signals

3.2.3 Main Functionalities

A main VI also had to be implemented in order to control the different channels on a high level, as well as receive the settings from the user interface. In this main VI the channel subVI's are used to control the individual channels, and some control over the complete system is also implemented: the stop and update buttons.

Generating Output

In the main VI, 5 of the digital channel subVI's and 4 analog/digital channel subVI's are placed. All the settings are received from the user interface as an array, which are then split into separate elements and connected to the corresponding channel. These channel subVI's only give the necessary output back to the main VI, they do not generate the output of the DAQ. This is necessary because the same type of in-/outputs can only be controlled from one place in LabVIEW: if one digital output is generated in one channel, the other channels can not generate digital output simultaneously. Since all channels need to generate output at the same time, this should all happen in the same place: in the main VI.
The instance that actually controls the in-/outputs of the DAQ is called a task in NI-DAQmx [28]. A task can only contain in-/outputs of the same type, so three tasks are necessary in our case: one for analog output, one for digital output (which includes the signals as well as the hardware control signals), and one for analog input to measure the voltages and currents. The first step to create a task is to use the Create Virtual Channel subVI to add all the necessary pins to the task. This task can than be used as an input for either the Write or the Read subVI, depending on what the task needs to do: generate output signals or measure the voltage, respectively. The Write subVI also takes the data that it needs to output as an input, and generates this data as a physical output on the pins of the DAQ that are configured in its task. The Read subVI outputs the data that it has measured at the pins that are included in the task, which is then given to the user interface so it can be displayed.
The measurement task is the most difficult one to create. The output tasks can just output a zero when a pin in its task is not needed. This also ensures that only the wanted pins generate output at all times. The measurement task, however, is more difficult since the sample rate setting is for the entire task and is limited to a maximum of 250 ks/s, aggregated over all pins. If all the pins are always added to the measurements task, this limits the sample rate of the pins that are actually in use: the sample rate is split evenly over all pins, also the ones not currently

in use to measure. To fix this, only the pins that need to measure according to the user input are added to the task, which is done using a case statement. The maximum sample rate is then divided by the number of pins, to maximize the accuracy of the measured signal. The achieved sample rate is also given to the user interface, so the user can see this and account for it in their measurements. This way to generate the measurement task does introduce a problem when trying to stop the DAQ generating output, which is discussed in the next section.

Stop Button

An important functionality is to stop all channel outputs: the global stop. When testing the signal generation it was discovered that when the LabVIEW program is stopped, each pin continues to generate its last output. This can be hazardous when the user wants to change connections to the sensor, or when testing is finished. To prevent problems, a stop button is implemented that makes all outputs zero. Integrated with this is also an indicator light to show whether the program was terminated correctly, to give a visual aid whether all outputs are actually zero.
The first step when the stop button is pressed by the user is to stop all current tasks. This is done by first using the Stop Task VI, followed by the Clear Task VI. When the tasks are stopped, however, it does not mean that all outputs of the DAQ are zero: the pins continue to output their last value. To make sure all the outputs are zero, a new task is started where the data that will be generated is all zero. This is implemented in LabVIEW using a flat sequence. The tasks are stopped in the first frame, and the all zero output tasks are started in the next frame. This is done to ensure all tasks are stopped before starting a new one, as LabVIEW will give an error when this is not the case. When all outputs are made zero, the light that indicates that output are being generated is also turned off, to show the user that it is safe to handle the hardware components.
As mentioned in the previous section, the measurement task can give some trouble when trying to stop it. This is because when the user does not want to measure any voltages or currents, an empty task is generated: no pins are added to the task. LabVIEW can not stop an empty task and gives an error when this is attempted. To remedy this, the number of measurements wanted by the user is counted. If this is 1 or larger a boolean is set to true, when it is zero the boolean is set to false. The stop button then only stops the task when this boolean variable is true, using a case statement.

Update Button

An update button was also implemented to allow the user to make multiple adjustments in the settings of the signals and apply them all at the same time. A small delay is also implemented: the signals are first turned off and the new signals are turned on after a second. This delay is necessary for the hardware, to prevent damage to the relays.
The update button is implemented by using a flat sequence: the first frame contains the update functionality, the second frame the generation of all the tasks as discussed in Section Generating Outputs. The first part of the update is the same as the stop button: stop the current tasks and start new ones that generate zero output on all pins, again using a flat sequence. Then another frame is added, in which the program waits for a second, this is necessary to give the relays the opportunity to close properly. After this, the new tasks are created according to the new user input.

3.3 Control Signals

All control signals that need to be generated are used to control relays to make sure the correct voltages and signals are connected to the sensors pins. Some of these control signals are correlated: if a digital signal is generated on an analog/digital channel the control signal indicating that it is a digital signal needs to be high, as well as one control signal to select either the 12 V or 24 V option. Since the need for the control signals originates from the hardware group of this project, the exact use and a more in-depth description is not included here, only how they are generated.

3.3.1 Parallel Communication

As discussed in the introduction, Section 1.4, there are two main strategies when data needs to be transferred: parallel or serial communication. For parallel communication no difficult protocols are necessary, each control signal can be sent directly to the hardware component that needs it. This makes that the implementation is more simple when compared to serial communication. However, the 5 digital channels need 7 control signals each and the 4 analog/digital channels each need 5 control signals. Using parallel communication this would use $5*7+4*5 = 55$ output pins for just the control signals, which is too much since the DAQ that will be used only has 48 digital output pins. The amount of necessary output pins can be reduced by encoding the different states, which will be discussed next.

3.3.2 Encoding & Decoding

To reduce the number of needed output pins relative to parallel communication, each state can be encoded and later decoded into the necessary control signals using hardware components. The digital channels have 7 states and the analog/digital channels have 5 states, as discussed in Section 3.2. This means that both could be encoded using 3 bits, since 3 bits can be decoded into $2^3 = 8$ different states. A convenient component to decode the signal into the separate states control signals is a 3 to 8 demultiplexer, which makes 1 of the 8 outputs high depending on the 3 bit input. As can be seen in Table 3.5, more than one control signal can need to be high simultaneously. To solve this, OR gates are used between correlated control signals to make all necessary control signals high. The schematics for this can be seen in Figures 3.3 and 3.4.

Table 3.5: Signal generation states

Analog/Digital	**Digital**
1) Measure	1) Measure
2) External	2) External
3) Analog	3) 3,3 V
4) Digital 12 V	4) 5 V
5) Digital 24 V	5) 10 V
	6) 12 V
	7) 24 V

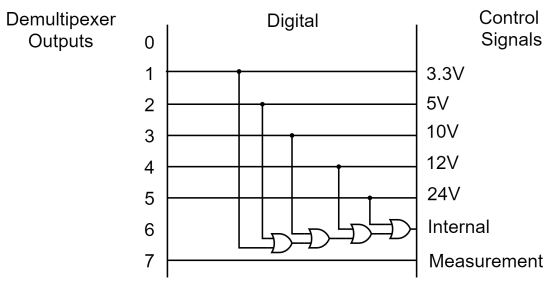

Figure 3.3: Decode circuit for digital channels

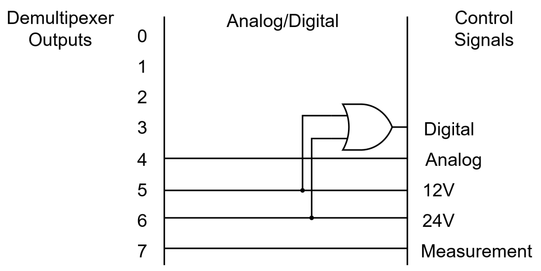

Figure 3.4: Decode circuit for analog/digital channels

3.3.3 Serial Communication

In serial communication the data that belongs to one message is sent consecutively, as seen in Figure 3.5. The data needs to be stored in order to be able to apply the control signals to the hardware continuously, as well as to aggregate the control signals that belong to one state together. For this purpose a shift register can be used, as shown in Figure 3.6. Using the enable signal, new data can be loaded in consecutively, and by making the enable signal low the data is stored for as long as it is necessary.

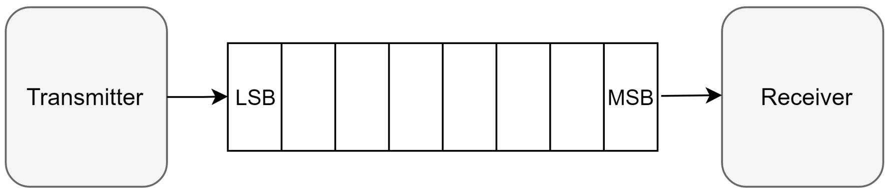

Figure 3.5: Serial data communication

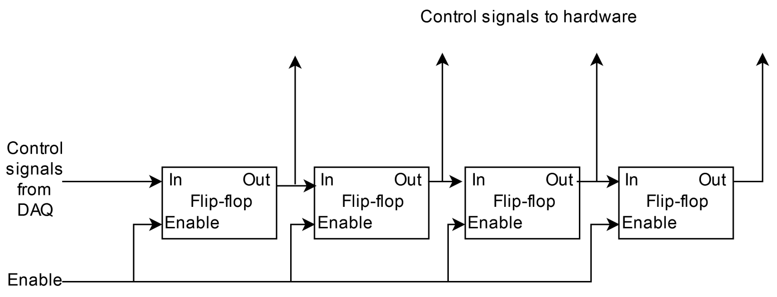

Figure 3.6: Shift register

One problem with this is that the values stored in the shift register are directly connected to the hardware, also when new data is being loaded. This can cause many unwanted phenomena, since part of the old control signals and part of the new control signals are connected to the hardware simultaneously, as shown in Figure 3.7. In the worst case this will cause a short circuit damaging the hardware components. To solve this problem, a buffer can be placed between the output of the shift register and the hardware components. This would enable loading new control signals and only applying them to the hardware when all control bits are received. Using this strategy, a total of three outputs are used: one for data, an enable for the shift register and an enable for the buffer, as is displayed in Figure 3.8. The length of the shift register and buffer determine how many control signals can be stored. The control signals for the different channels can be stored in the same register, as long as the register is long enough and the signals are forwarded to the correct hardware components. This implies that all the channels always need to be reloaded when a single setting is changed.

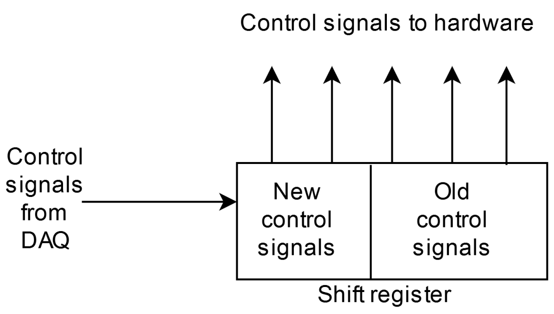

Figure 3.7: The problem with using a shift register

A difficulty with this approach is that the communication of the data needs to be timed correctly: a data bit needs to be available, then the enable of the shift register needs to rise and fall again, after that the next data bit can be sent. After all bits are sent, the enable of the buffer needs

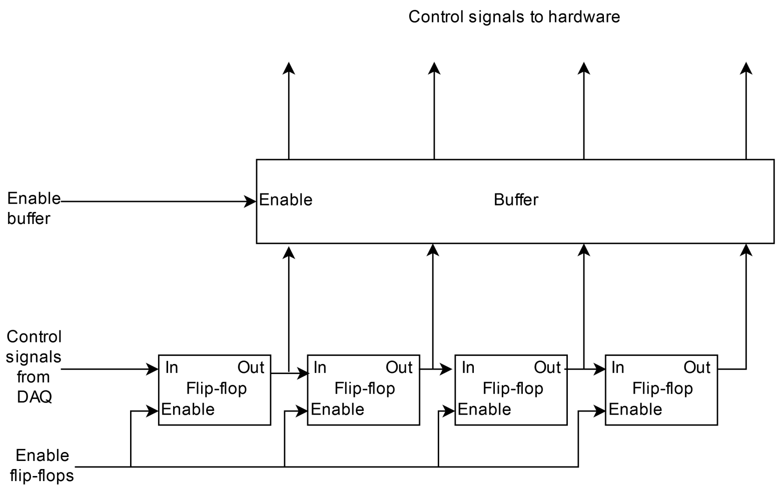

Figure 3.8: Shift register followed by a buffer

to rise and fall to output the new control signals. This all needs to happen at the right time and in the right order, which can be problematic since software timing in LabVIEW is not very accurate, as discussed in Section 3.2.1. Another downside is that it introduces latency in the system: when the output is generated it first needs to be transferred bit by bit before it can be sent through to the hardware, where parallel communication has a direct connection between the control signal generation and the hardware components.

3.3.4 Design Choice

Since the encoding and decoding approach is easier to implement in LabVIEW because it does not require accurate timing, the decision was made to use this approach. Another thing that was taken into account was that the encoding approach would need $3 * 5 + 3 * 4 = 27$ digital output channels. Since the DAQ has 48 available channels, this was not a problem.
The choice for the state encoding is somewhat free, but the output of the demultiplexer needs to be taken into account. As an example the 5 V state for a digital channel will be discussed. The encodings are based on the schematics displayed in Figures 3.3 and 3.4. For the 5 V state, output 2 of the demultiplexer needs to be high. This will directly make the 5 V control signal high, as well as the internal control signal through the OR gates. To make output 2 high, the encoding for the state needs to be 010. The same procedure can be followed for all the encodings, which are all listed in Tables 3.6 and 3.7.

Table 3.6: Digital channel states and control signals

State	**Control Signals**	**Encoding**
Measure voltage	1000 000	111
External/off	0000 000	000
3.3 V	0100 001	001
5 V	0100 010	010
10 V	0100 100	011
12 V	0101 000	100
24 V	0110 000	101

Table 3.7: Analog/digital channel states and control signals

State	**Control Signals**	**Encoding**
Measure	1000 0	111
External/off	0000 0	000
Analog	0001 0	100
Digital 12 V	0010 1	101
Digital 24 V	0100 1	110

Since all the signals generated here are digital signals, the components specifications are not that important as the generated control signals are constant at either 0V or 5V. The choice was made to use the SN74LV4051A demultiplexers [29] and the CD4071BE OR gates [30], as these were easily accessible and inexpensive. The hardware circuit design using the selected hardware can be found in Appendix A. The circuit corresponds to the schematics found in Figures 3.3 & 3.4.

3.4 Connection with User Interface

The user interface made by another subgroup will also be made in LabVIEW. This means that what is discussed here as Main VI can be used as a subVI in the UI VI, to which all the user setting can be connected as input. The user settings will be supplied as arrays, with a separate array for each setting type. These arrays are then split into separate items in the signal generation Main VI, to be connected to the correct channels.

Chapter 4

Results and Validation

This chapter explains the measurements and their results in detail. First describing basic signal generation as well as bias voltages, followed by the frequency response and distortion measurements of the signal generator. The measurements functionality of the system will also be analyzed.

4.1 Signal Generation Measurements

To test whether the DAQ correctly generates both low and high frequency signals, tests were performed to ensure that the desired result is achieved. To perform each of these tests, each type of signal was generated simultaneously to demonstrate that the DAQ is capable of outputting multiple signals simultaneously, as this is a requirement. It demonstrates that the different waveforms can be generated as well.

4.1.1 Low Frequency Signal Generation

During this test the amplitude of each signal was different to verify that the DAQ is capable of achieving different voltages in the range of -10 V to 10 V. In Figure 4.1 the setup of this test can be seen. Two SCB68's (the green boards) are connected to the pins of the DAQ and make it easy and safe to probe the outputs, as well as connect signals to the inputs in order to test the voltage measurement option. The probes are connected to two oscilloscopes to measure the generated signals. The measurements from the oscilloscope can be seen in Figure 4.2. The computer in the image is used to control the DAQ, as well as to display the measurements performed by the DAQ which will be discussed in Section 4.4. Tables 4.1 & 4.2 depict the desired and actual results when generating low frequency signals.

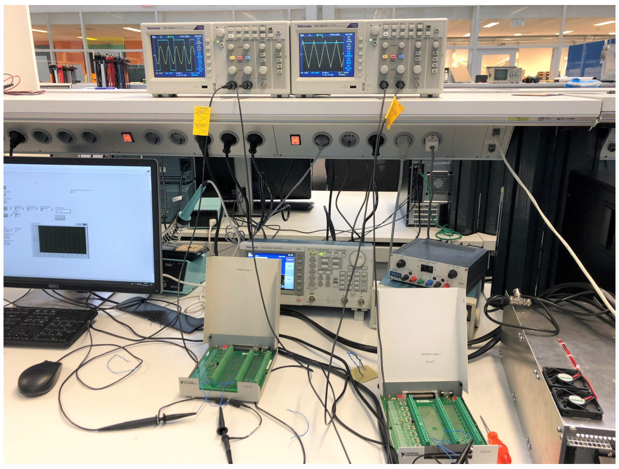

Figure 4.1: Measurement setup

Table 4.1: Voltage measurement of signal generation test

Signal Type	**Desired Voltage** (V_{pk-pk})	**Measured Voltage** (V_{pk-pk})	**Relative Error (%)**
Sine	20	20.8	4
Square (Duty Cycle 65%)	20	21.2	6
Triangle	20	20.8	4
DC (measured in Volt)	5	5.2	4

Table 4.2: Frequency measurement of Signal Generation

Signal Type	**Desired Frequency (Hz)**	**Measured Frequency (Hz)**	**Relative Error(%)**
Sine	1000	1000.04	0.004
Square (Duty Cycle 65%)	500	500	0
Triangle	750	750.8	0.11

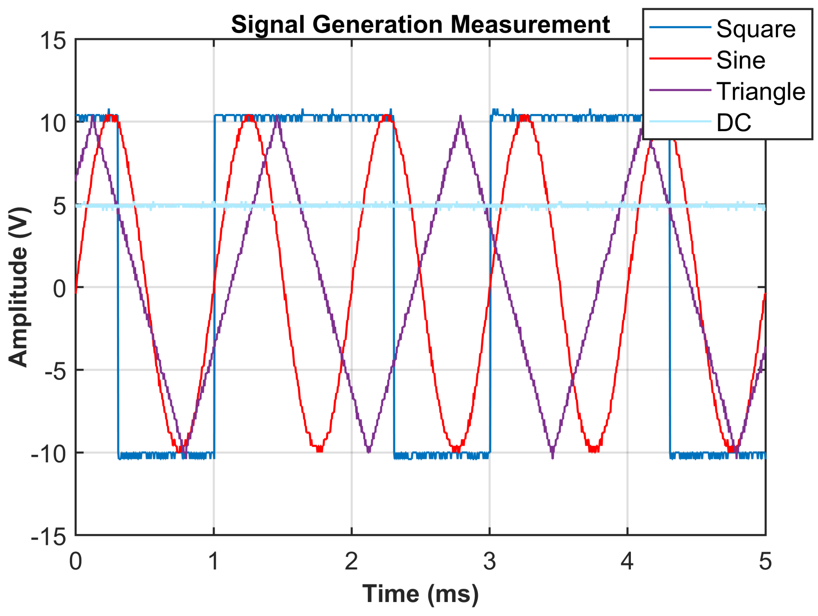

Figure 4.2: Signal generation low frequencies

From these measurements can be concluded that the DAQ performs as expected, with a duty cycle of 65% and a rise and fall time of $4\mu s$ with no overshoot on the generated square wave, as seen in Figure B.1 in the Appendix. The average voltage relative error is 4.5% and the frequency has an average relative error of 0.038%. As there are no hard requirements of the acceptable bounds for the signal generation, this only verifies if the DAQ performs as expected.

4.1.2 High Frequency Signal Generation

As specified in Chapter 2, frequencies up to 100kHz must be generated. To verify that the signal generator is able to output these high frequencies the same measurement setup as in Figure 4.1 was used, with the difference that during this test signals with frequencies above 50 kHz were generated. Tables 4.3 & 4.4 depict the desired and actual results, and Figure 4.3 displays the results graphically.

Table 4.3: Voltage measurement of high frequency measurements

Signal Type	**Desired Voltage** (V_{pk-pk})	**Measured Voltage** (V_{pk-pk})	**Relative Error (%)**
Sine	20	23	15
Square (Duty Cycle 50%)	20	24.2	21
Triangle	20	23	15

Table 4.4: Frequency measurement of high frequency measurements

Signal Type	**Desired Frequency (kHz)**	**Measured Frequency (kHz)**	**Relative Error(%)**
Sine	100	95.97	4.03
Square (Duty Cycle 50%)	50	48.08	3.84
Triangle	60	59.24	1.3

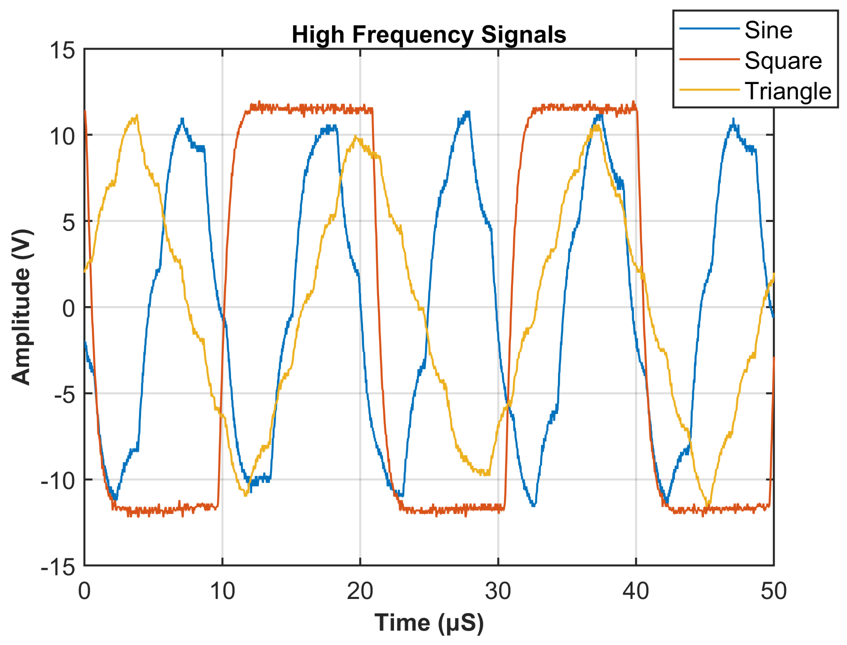

Figure 4.3: Signal generation high frequencies

This test shows that the DAQ has difficulty generating high frequencies, this is not a strange phenomena and was expected due to the limited update rate of the DAQ. The number of samples taken per period of the generated signal is determined by Equation (4.1), with f_s the update rate of the DAQ and f the desired frequency. f_s is set to 625 ks/s.

$$\#\text{samples} = \frac{f_s}{f} \tag{4.1}$$

This can be translated into 6 sample points per period for a desired frequency of 100 kHz, which means that the signal will be constructed out of 6 points resulting in a rough signal construction. The generated square wave has a duty cycle of 51.89% with a rise and fall time of 16 μs with no overshoot, see Figure B.2 in Appendix B. The average relative error for the voltage is 17% and a frequency average relative error of 3.06%.

4.2 Frequency Response and Amplitude Flatness

To better analyze the bandwidth and amplitude flatness of the signal generator a frequency response analysis was performed, the importance of which is explained in Section 1.3.1. To perform this test the signal generator was set up to perform a frequency sweep: setting the voltage amplitude to 10V and varying the frequency of a sine wave from 10Hz to 312.5kHz. This max frequency is half of the update rate of the DAQ and was set to meet the Nyquist criterion. An oscilloscope with a spectrum analyzer was used to visualize and save the data. During this analysis the signal generator output is examined to verify if the response of the signal generator is uniform in the range of 0-100kHz. This analysis will also illustrate the 3dB-bandwidth of the signal generator: the point where the amplitude of the power is 3dB lower than that of the maximum. The results can be found in Figure 4.4.
Out of this analysis it can be concluded that the 3dB-bandwidth of this signal generator is 295kHz, which falls outside the 0-100kHz range the signal generated needs to be able to generate. This means the bandwidth is wide enough to satisfy that requirement. From the same plot can be concluded that the amplitude of the signal strength falls 0.73 dB in the 0 - 100 kHz range, which resembles the amplitude flatness in this range.

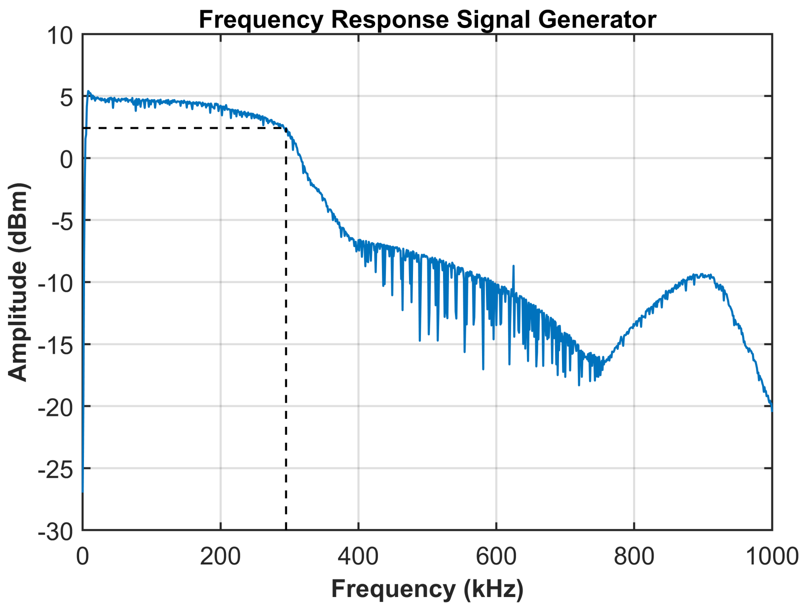

Figure 4.4: Frequency response

4.3 Frequency Spectrum Analysis

In this section the frequency spectra of generated sine waves at different frequencies will be measured and evaluated. The spectra are measured at intervals of 10 kHz. In this section the focus is on 10 kHz and 100 kHz, which are displayed in Figures 4.5 - 4.8, the rest of the measured spectra can be found in Appendix B and the same procedures can be followed to quantify the distortions. The spectra are measured using the Fast Fourier Transform (FFT) functionality of an oscilloscope: the Tektronix TDS2022C [31].

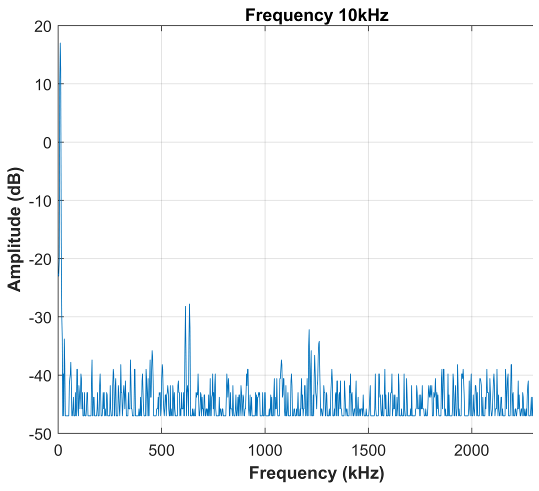

Figure 4.5: Frequency spectrum 10 kHz sine wave

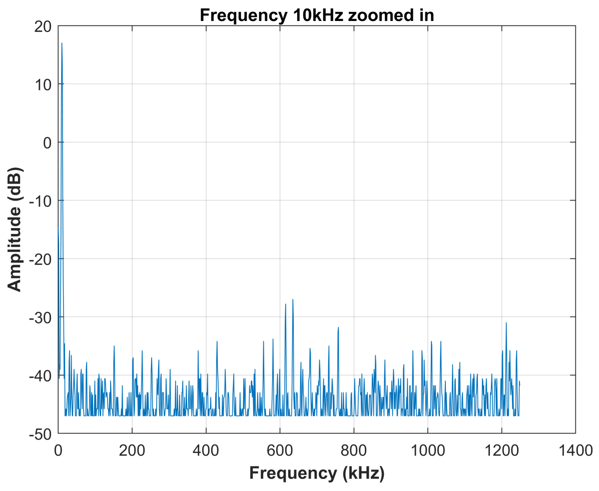

Figure 4.6: Frequency spectrum 10 kHz sine wave zoomed in

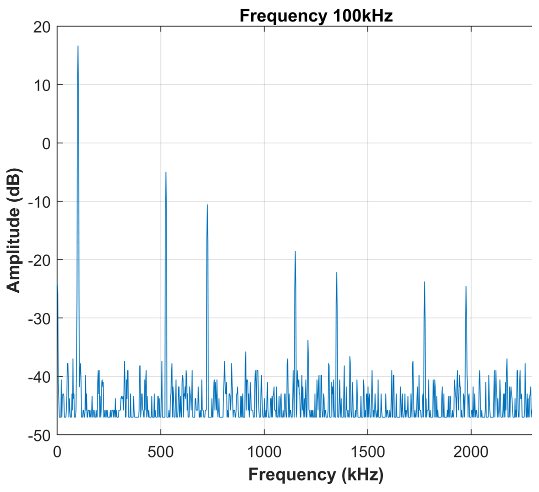

Figure 4.7: Frequency spectrum 100 kHz sine wave

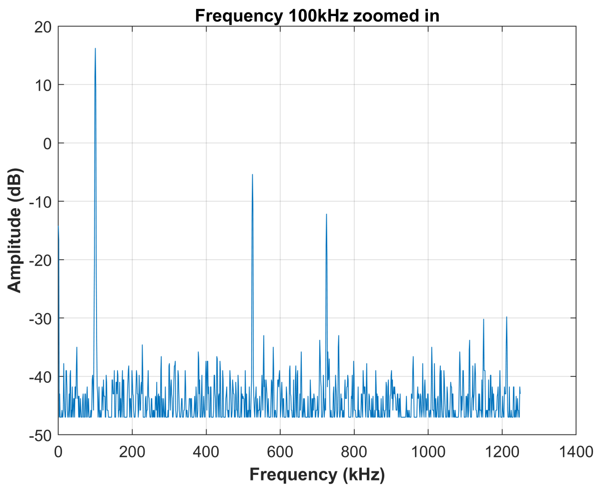

Figure 4.8: Frequency spectrum 100 kHz sine wave zoomed in

4.3.1 Aliasing

In the frequency spectra it can be seen that there are more frequency components present than just the fundamental. The most pronounced peaks lie at frequencies described by Equation (4.3). These occur due to the effects of sampling a signal: its spectrum repeats around every integer multiple of the sampling frequency, f_s. Since the signals here are real, negative frequency components do not exist. Those are mirrored to their absolute frequency value on the positive axis.

$$f = nf_s \pm f_0 \tag{4.2}$$

Let's use the 100 kHz spectrum as an example to make this more clear. The highest peak occurs at 100 kHz, which is the fundamental frequency and thus expected to be there. The peak at 725 kHz is part of the frequency spectrum around $+f_s$: $625kHz + 100kHz = 725kHz$. The peak at 525 kHz is part of the frequency spectrum around $-f_s$, that is mirrored to its absolute value: $|-625kHz + 100kHz| = 525kHz$.
Since the maximum frequency component is always smaller than half of the sampling frequency this phenomenon is not aliasing: it is purely caused by sampling the signal, the spectra around multiples of the sampling frequency do not overlap.

4.3.2 Intermodulation Distortion

Intermodulation can occur at frequencies as described in Equation (4.3), as also described in Section 1.3.4. However, there are no peaks at any of these frequencies other than the frequencies as described in Equation (4.3), which are more likely caused by the sampling. This means that the power of the frequency components due to intermodulation fall below the noise floor.

$$f = \pm nf_s \pm mf_0 \tag{4.3}$$

4.3.3 Spurious Free Dynamic Range

The spurious free dynamic range is different for the different frequencies: the height of the spurs depends on the frequency of the signal. In the 10 kHz spectrum the fundamental frequency has an amplitude of 17.0 dB, and the highest spur occurs at frequency 635 kHz with an amplitude of -27.8 dB. This makes the SFDR $17.0 - (-27.8) = 44.8$ dB. For 100 kHz the highest spur occurs at 525 kHz, and the SFDR is $16.6 - (-5.0) = 21.6$ dB.

4.3.4 Phase Noise

Phase noise occurs due to jitter in the clock frequency the DAQ uses to generate analog output. It means that the fundamental frequency component not a discrete peak, but distributed over a small frequency range. The single sided phase noise is approximately 9.5 kHz for the 10 kHz sine wave, as can be seen in Figure 4.9. For the 100 kHz sine wave it is approximately 7 kHz.

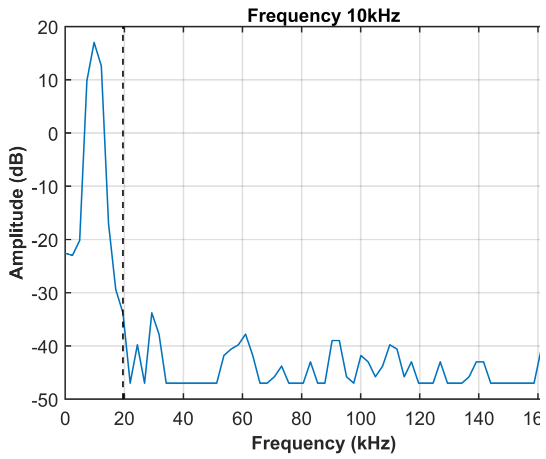

Figure 4.9: Phase noise of 10 kHz sine wave

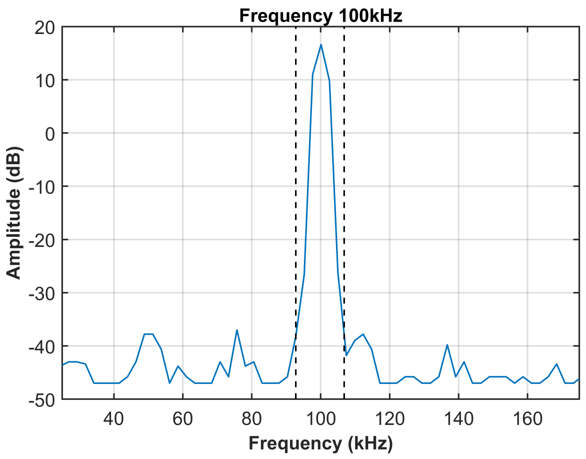

Figure 4.10: Phase noise of 100 kHz sine wave

4.3.5 Total Harmonic Distortion

As can visually be seen from the spectra, the harmonic frequency components in both figures are quite small. Equation (4.4) can be used to calculate the values. The THD is 6.0 % for the 10 kHz sine, when using $n = 6$ as highest value. For the 100 kHz sine wave the THD is 5.0 %.

$$THD_F = \frac{\sqrt{\sum_{n=2}^{\infty} V_n^2}}{V_1^2} \tag{4.4}$$

4.4 Measurement Analysis

The user should be able to obtain voltage measurements out of the device under test (DUT), to verify whether their device is working correctly. An analysis was performed, in which a sinusoidal signal was provided using an external signal generator connected to an input of the DAQ. This signal was generated at 5 V and a frequency of 1 kHz and another at 10 kHz. The 1 kHz signal was sampled with a rate of 250 kHz, while the 10 kHz signal was sample with a rate of 62.5 kHz: the maximum rate when four channels are performing measurements as discussed in Section 3.2.1. The measured signals can be found in Figures 4.11 & 4.12. The desired outcome of this experiment is that the measurement correctly represents the signal with as little distortion as possible. The measured values are shown in Tables 4.5 & 4.6.

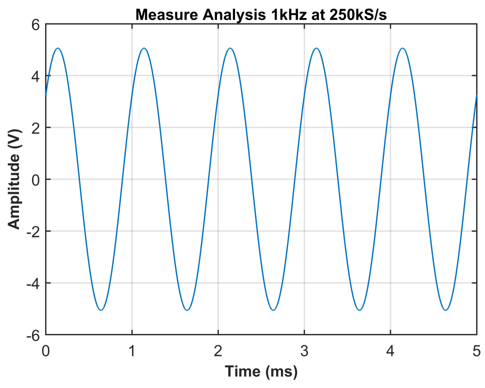

Figure 4.11: Measurement 1 kHz at 250 ks/s

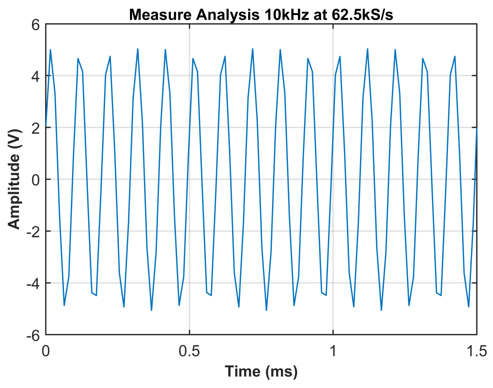

Figure 4.12: Measurement 10 kHz at 62.5 ks/s

Table 4.5: Voltage measurement analysis

Signal Type	**Generated Voltage** (V_{pk-pk})	**Measured Voltage** (V_{pk-pk})	**Relative Error (%)**
Sine 1kHz	5	5.064	1.28
Sine 10kHz	5	5.063	1.26

Table 4.6: Frequency measurement analysis

Signal Type	**Generated Frequency (kHz)**	**Measured Frequency (kHz)**	**Relative Error (%)**
Sine 1kHz	1	1	0
Sine 10kHz	10	10	0

Chapter 5

Discussion and Conclusion

As the field of IoT sensors is rapidly expanding, a method to quickly and efficiently test these sensors was needed. To accomplish this a signal generator was designed that also controls the hardware components. The analog channels can generate DC voltages, sinusoidal, square and triangle waves from -10 V to 10 V and corresponding control signals, as well as control signals for bias voltages at 12 V and 24 V. The digital channels output the correct control signals to put bias voltages of 3.3 V, 5 V, 10 V, 12 V and 24 V on the correct sensor pins. All these signals are reconfigurable during run time, and the system can also measure sensor output voltages instead of generating signals. This means all the primary functional requirements are met. The output current of the bias voltages can also be measured and displayed in the user interface, which was a secondary functional requirement. All the system requirements are also met by the end product. One of the things that was a bit difficult after verifying that the signal generation worked, was that there were no requirements on how accurate the generated signals need to be. This made validation of the generated signals tough, as a different conclusion than "it works" can not really be drawn. An effort was made, however, to quantify the accuracy of the signals using different measures.
Something that is not understood is why the voltage accuracy of a higher frequency signal goes down, while this does not show in the amplitude flatness that was measured. The voltage at 100 kHz should be slightly lower than the voltage at 1 kHz according to the frequency response of the system, while the voltage error when measuring a 100 kHz sine wave shows that the voltage is 15% too high, while at 1 kHz this error is only 4%.
A mistake that was made is the choice of OR gates for the decoding circuit. Due to a lack of attention paid to the current the gates should be able to drive, they were unable to provide enough current to switch the relays. This meant that the OR gates needed to be changed to a different type. Unfortunately the ones that could provide enough current all had a different pin layout, meaning they could not just be used on the PCB that was already made. To solve this problem, a separate board was made with the new OR gates. This is not very neat, but unfortunately it was necessary. In a possible future iteration this problem can be addressed by routing the signals to the OR gates slightly differently to comply with the pin layout of the new gates.
Another issues which presented itself during this project is the capability to measure current of the analog signals. This issue was caused by the choice of current measurement IC by the hardware subgroup, being a high side current measurement device and thus only able to measure positive currents. This functionality was fully implemented however, but can not be used. The option to replace these ICs was investigated. However, the ICs that can measure both positive and negative currents all have a different pin layout so they could not be replaced. Since the ICs are all surface mounted and really small, the same approach as used to replace the OR gates could not be used.

Future Work

One of the secondary requirements was not met unfortunately: the generation of square waves on digital channels. It is however hard to believe that this is completely impossible, so this could be investigated further.
Another thing is that the focus was mainly on achieving signal generation, not so much on generating accurate signals with low distortion. A possibility to improve the frequency spectra would be to use a low pass filter for the generated signals to remove the high frequency components that lie outside the 0 - 100 kHz range.
The communication of control signals between the DAQ and the hardware now takes place using the encoding & decoding approach. While this works, it is a bit sloppy and using a serial communication protocol would be neater. This is something that could be implemented in a next iteration of this project.

Appendix A

Decoding Circuits

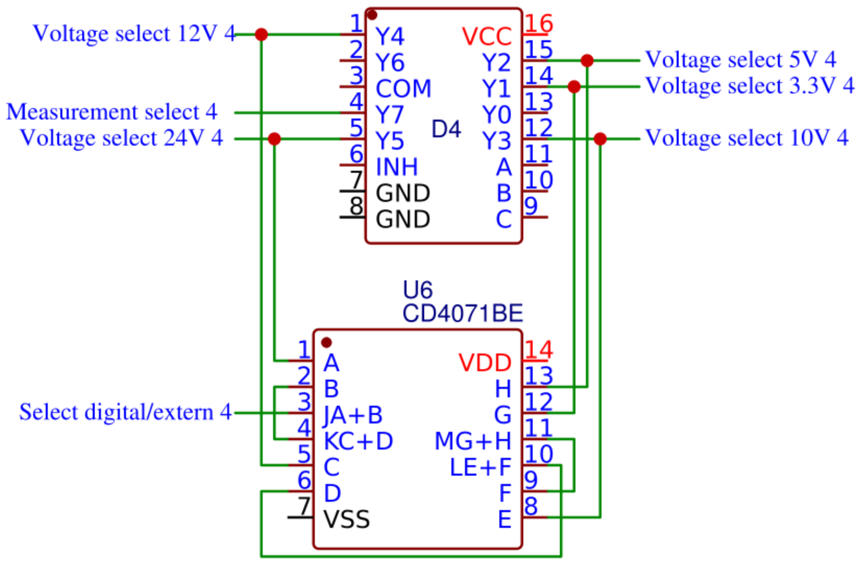

Figure A.1: Digital circuit old

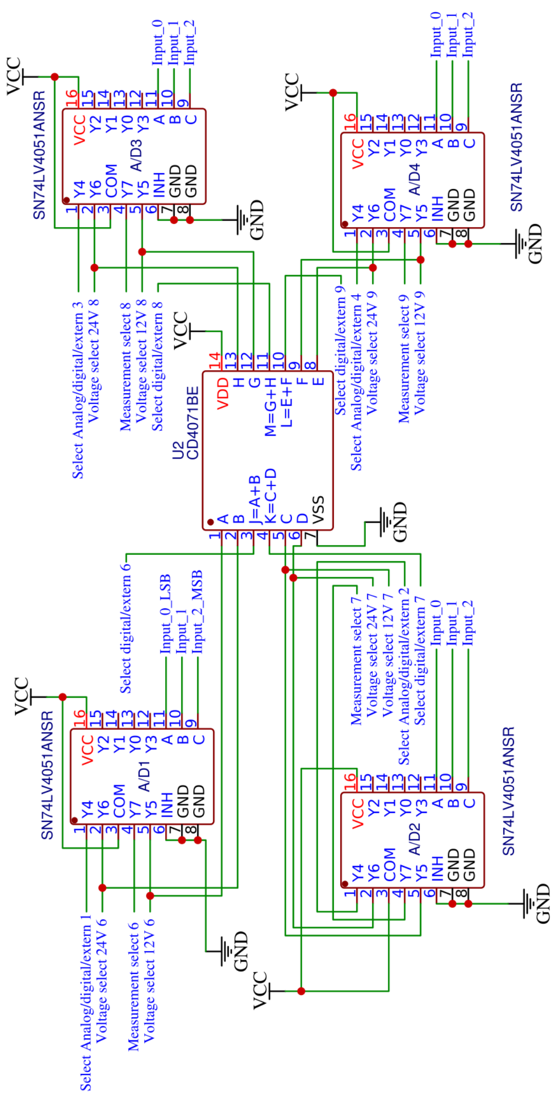

Figure A.2: Analog digital circuit old

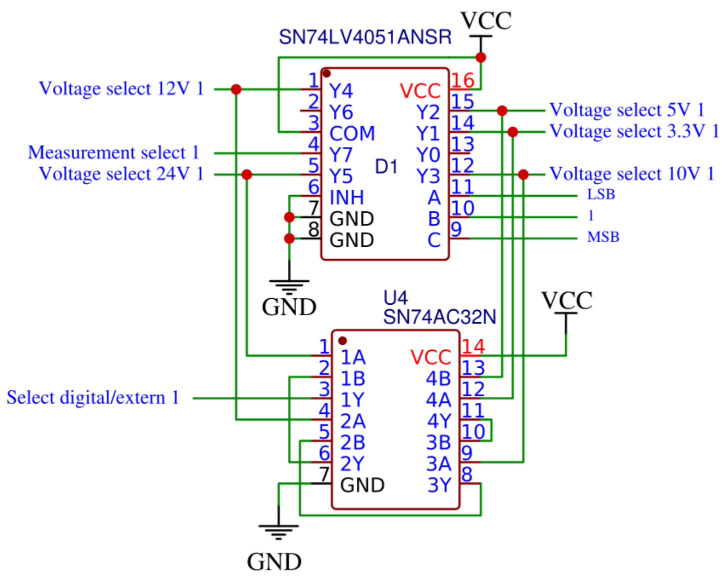

Figure A.3: Digital circuit

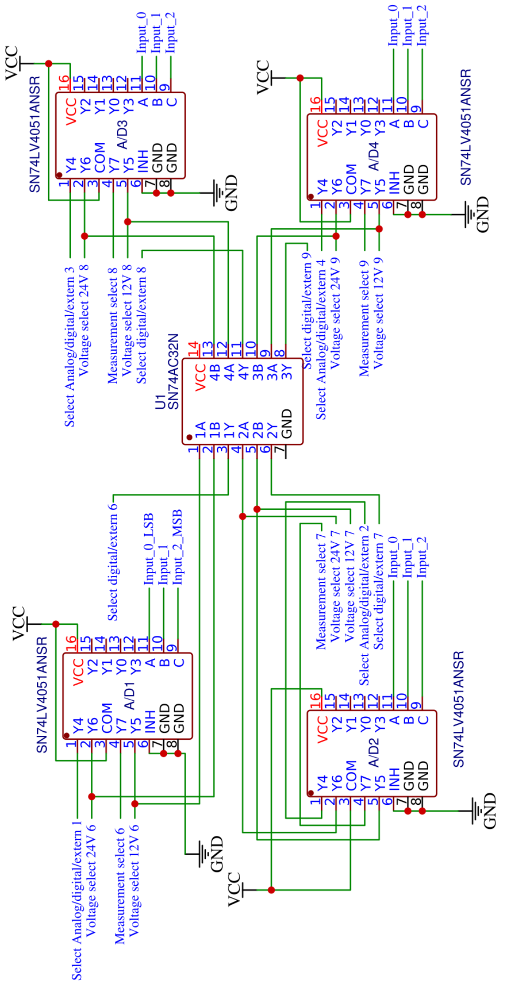

Figure A.4: Analog digital circuit

Appendix B

Frequency Spectra

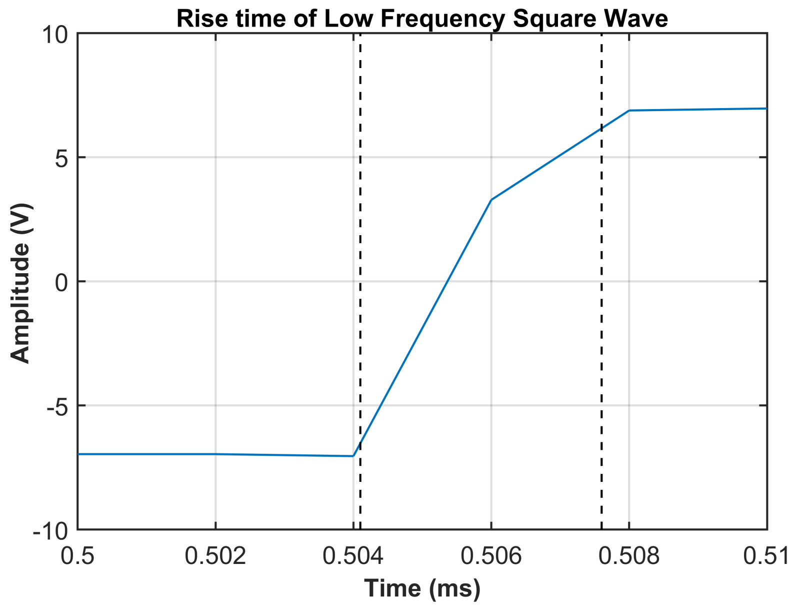

Figure B.1: Rise time for low frequency square wave

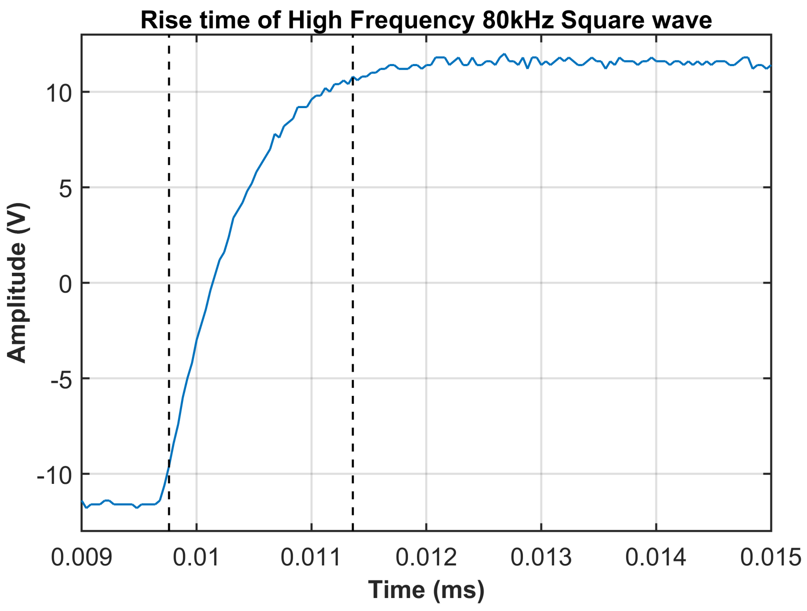

Figure B.2: Rise time for high frequency square wave

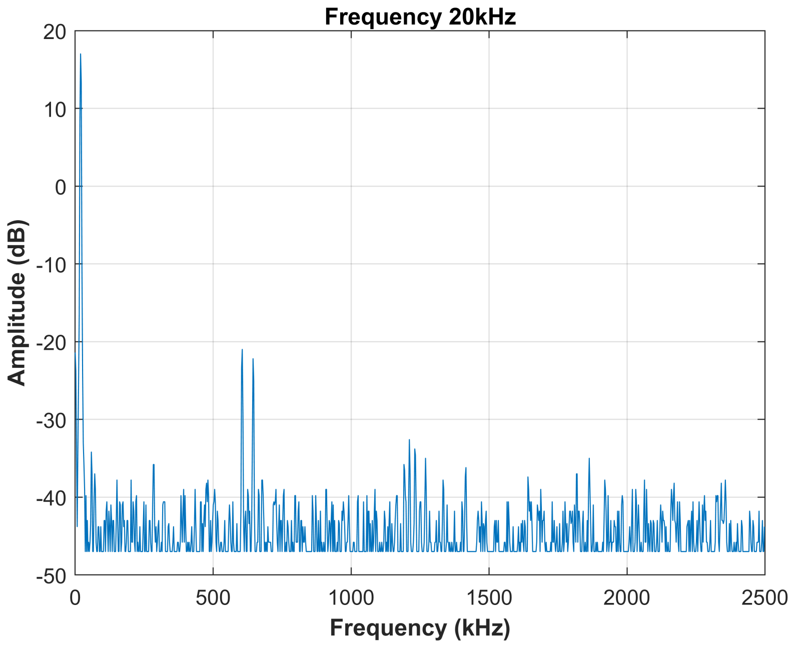

Figure B.3: Spectra 20 kHz

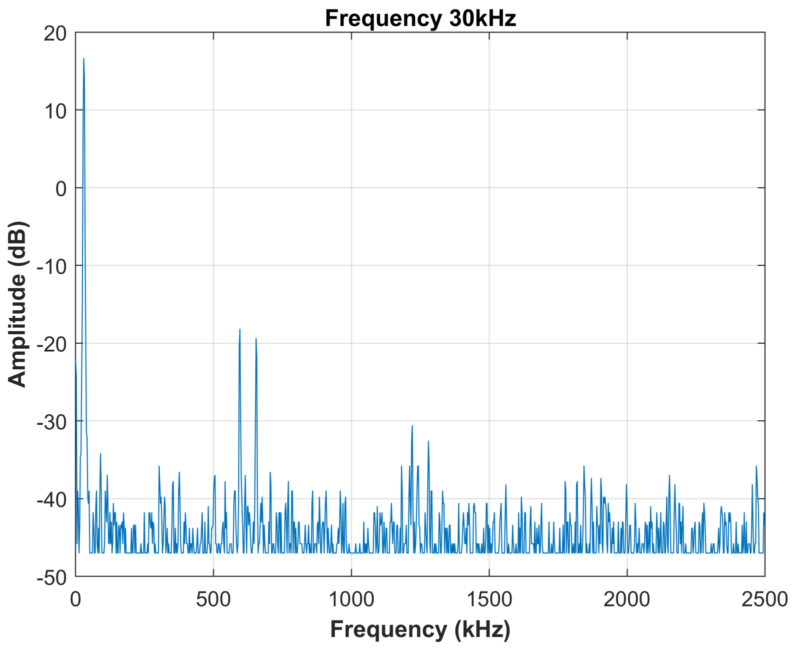

Figure B.4: Spectra 30 kHz

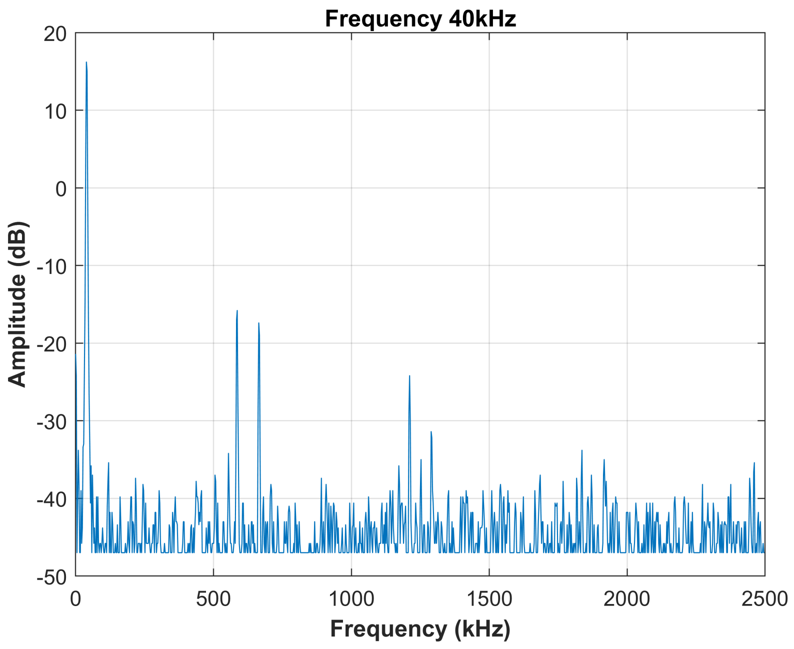

Figure B.5: Spectra 40 kHz

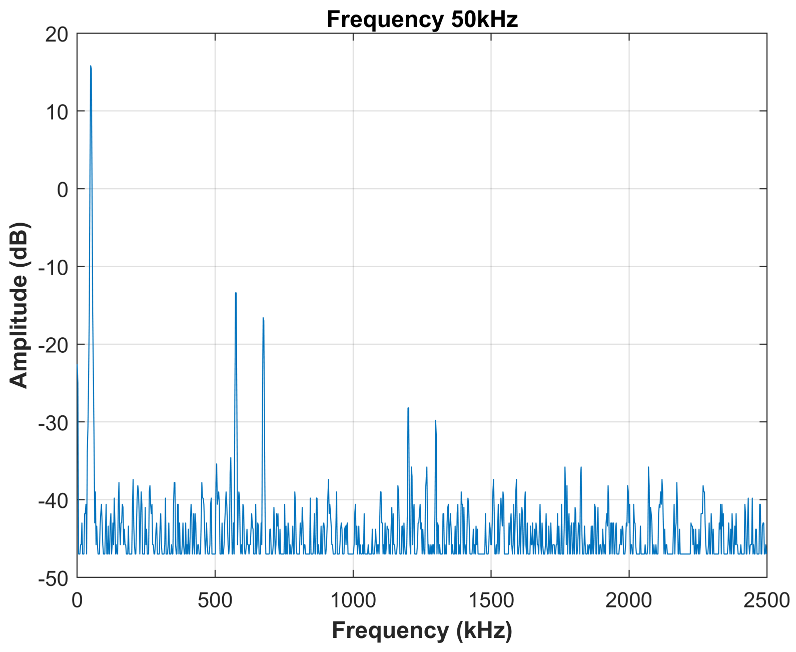

Figure B.6: Spectra 50 kHz

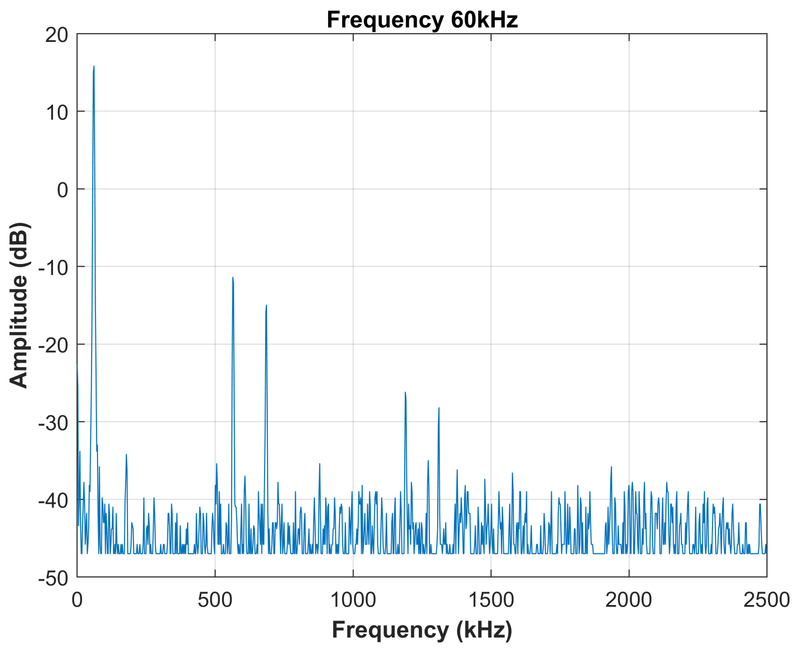

Figure B.7: Spectra 60 kHz

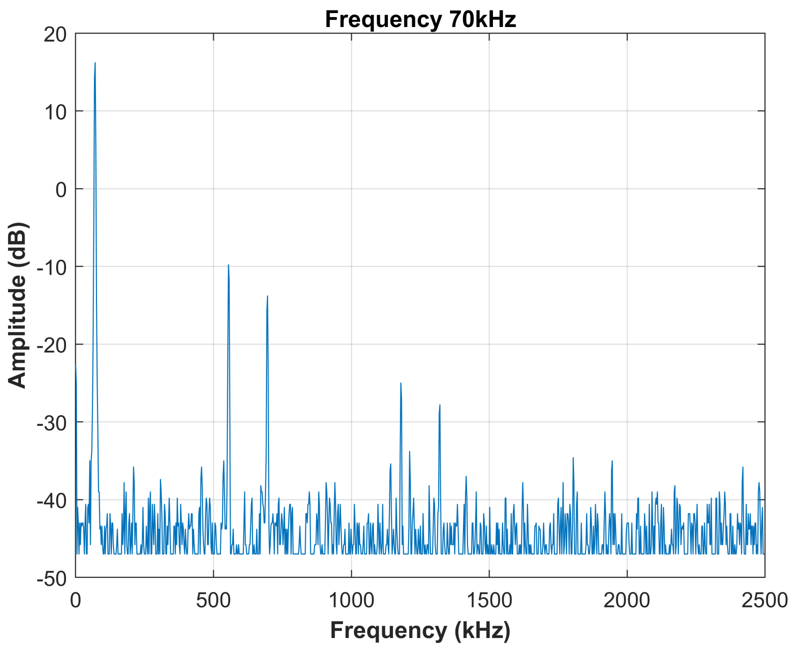

Figure B.8: Spectra 70 kHz

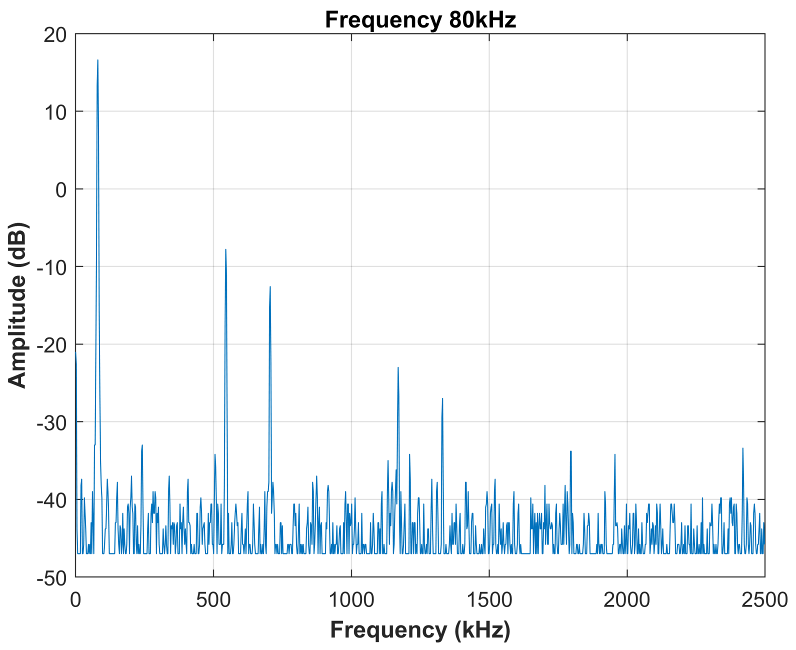

Figure B.9: Spectra 80 kHz

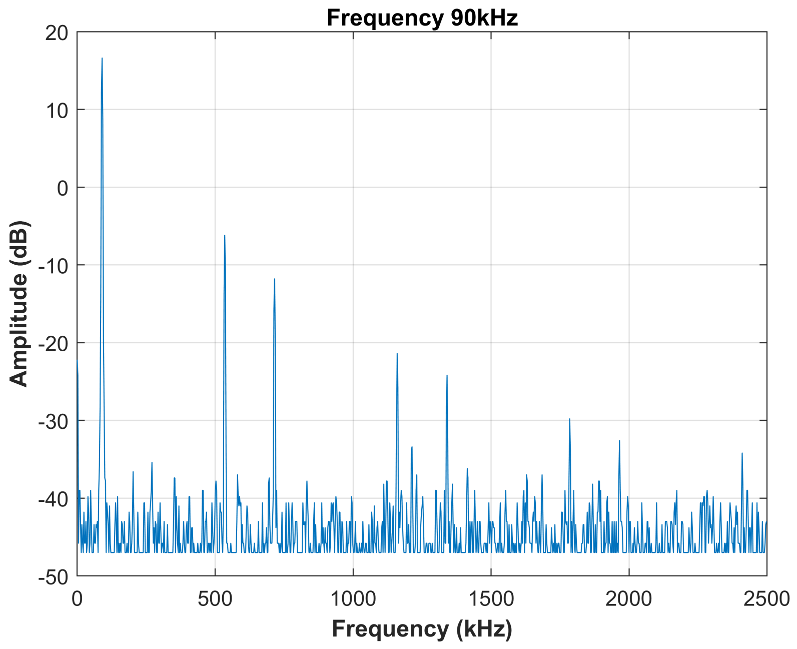

Figure B.10: Spectra 90 kHz

Bibliography

[1] Z. Zhao, L. Wang, J. Chen, Z. Cai, Y. Lv, and Y. Feng, "The design and implementation of signal generator based on dds," in *2017 IEEE 9th International Conference on Communication Software and Networks (ICCSN)*, pp. 920–923, May 2017.

[2] R. Yue, Y. Wen-Ji, and W. Jinming, "An fpga based multi-functional signal generator using sopc design methodology," in *2016 3rd International Conference on Information Science and Control Engineering (ICISCE)*, pp. 1257–1261, July 2016.

[3] S. Ball, "Analog-to-digital converters," May 2001.

[4] , "Basics of UART Communication, Block Diagram, Applications." `https://www.elprocus.com/basics-of-uart-communication-block-diagram-applications/`. [Online; accessed 17 June 2019].

[5] National Instruments, "Signal generator terminology and specifications." `http://www.ni.com/product-documentation/4091/en/#toc2`, Nov 2016. [Online; accessed 12 June 2019].

[6] A.-H. Riyadh, M. A. Ismail, and A. I. Ammar, "Microcontroller-based function generator," *Al-Khwarizmi Engineering Journal*, vol. 4, p. 48–57, 2008.

[7] National Instruments, *LabVIEW User Manual*, Apr 2003.

[8] MathWorks, *Simulink User Guide*, Mar 2015.

[9] National Instruments, *NIMultisim User Manual*, Jan 2009.

[10] National Instruments, "Signal Generator Terminology and Specifications." `http://www.ni.com/product-documentation/4091/en/`, 2016. [Online; accessed 9 June 2019].

[11] Tektronix, *Arbitrary/Function Generators*, Sept. 2016.

[12] L. W. Couch, *Digital and Analog Communication Systems*, p. 105. Pearson Education, 2013.

[13] C. E. Shannon, "Communication in the presence of noise," *PROCEEDINGS OF THE IRE*, vol. 37, pp. 10–21, Jan 1949.

[14] Owon, "Why is the measured amplitude less than the real value?." `http://www.owon.com.hk/newsinfo_248`, June 2017. [Online; accessed 15 June 2019].

[15] J. G. Proakis and D. G. Manolakis, *Digital signal processing: principles, algorithms, and applications*, pp. 235–237. Prentice-Hall International, 1996.

[16] D. Shmilovitz, "On the definition of total harmonic distortion and its effect on measurement interpretation," *IEEE Transactions on Power Delivery*, vol. 20, pp. 526–528, Jan 2005.

[17] J. C. Pedro and N. B. Carvalho, *Intermodulation distortion in microwave and wireless circuits*, pp. 19–20. Artech House, 2003.

[18] L. F. Chaparro, *Signals and Systems Using MATLAB*, pp. 498–499. 2015.

[19] Walt Kester, "Understand SINAD, ENOB, SNR, THD, THD + N, and SFDR so You Don't Get Lost in the Noise Floor." https://www.analog.com/media/en/training-seminars/tutorials/MT-003.pdf, Oct 2008. [Online; accessed 18 June 2019].

[20] National Instruments, "Specifications Explained: Spurious-Free Dynamic Range (SFDR)." http://www.ni.com/product-documentation/54467/en/, Feb 2018. [Online; accessed 18 June 2019].

[21] A. Demir, A. Mehrotra, and J. Roychowdhury, "Phase noise in oscillators: A unifying theory and numerical methods for characterization," *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: FUNDAMENTAL THEORY AND APPLICATIONS*, vol. 47, pp. 655–674, May 2000.

[22] Matt Newton, "To Terminate, Bias, or Both?." https://blog.opto22.com/optoblog/rs-485-to-terminate-bias-or-both, Jan 2017. [Online; accessed 17 June 2019].

[23] V. H. Meyer, A. K. Palit, W. Anheier, A. Sticht, and J. Schloeffel, "Can signal integrity faults be detected by delay tests?,"

[24] L. E. Frenzel, *Principles of electronic communication systems*, pp. 194–197. McGraw-Hill Education, 2016.

[25] National Instruments, "Device Specifications NI-PXI 6229." http://www.ni.com/pdf/manuals/375204c.pdf, Jun 2016. [Online; accessed 9 June 2019].

[26] Digilent, *Digilent Nexys2 Board Reference Manual*, Jul 2011.

[27] National Instruments, "NI-DAQmx Software." https://www.ni.com/dataacquisition/nidaqmx.htm. [Online; accessed 12 June 2019].

[28] National Instruments, "Tasks in NI-DAQmx." http://zone.ni.com/reference/en-XX/help/370466AH-01/mxcncpts/tasksnidaqmx/, Jan 2019. [Online; accessed 14 June 2019].

[29] Texas Instruments, *SN74LV4051A 8-Channel Analog Multiplexers and Demultiplexers*, Sep 2015.

[30] Texas Instruments, *CMOS OR gates*, Aug 2003.

[31] Tektronix, *Digital Storage Oscilloscopes TDS2000C Series Data Sheet*, Mar. 2012.