korean] 혈류 속도 예측을 위한 휴대용 초음파 시스템 english]Portable ultrasound system for blood velocity estimation

korean]힌릭스예페 korean2]힌릭스예페 chinese] english]HinrichsJeppe

[major]이 현 주Hyunjoo Leesigned [major2]Fafoutis Xenofonsigned Professor of Electrical Engineering

EEengineeringa

[1]이 현 주 [2]Fafoutis Xenofon [3]제 민 규

2023615

2023615

2023

[

[

July 4, 2023

초음파 이미징은 혈류 측정을 수행하는 중요한 방법이다. 이 보고서는 그러한 초음파 혈 류 측정 시스템의 설계, 구현과 분석을 담고 있다. 초음파 혈류 측정 시스템은 Xilinx Zynq 7000 SoC 라는 중앙 제어 시스템을 기반으로 자체 제작된 아날로그 프론트 엔드를 구현하 기 위한 다양한 초음파 펄스 및 타이밍 신호를 발생시킨다. 또한, 이 시스템은 직교 복조를 이용하여 복조된 도플러 주파수를 얻는다. 모듈 단위로 시스템을 테스트하였으나 시간 제 약으로 인해 전체를 테스트하지는 못하였다.

초음파, 바이오메디컬, 이미징, 혈류 속도, 속도 추정, 혈류, RF, 아날로그 전자 장치, VHDL, FPGA, 임베디드 시스템, 도플러

**Abstract**

For blood flow estimation, ultrasound imaging is an important tool. This project report outlines the design, implementation, and analysis of such ultrasound blood velocity estimator. The design is based off a central control system on a Xilinx Zynq 7000 SoC for generating ultrasound pulses and the various timing signals needed for operating the custom designed analogue front-end. The chosen design topology uses a pulsed-wave design with quadrature demodulation to obtain the Doppler frequency. The system was module tested, but was not fully end-to-end tested on a physiological simulator due to time constraints.

Ultrasound, biomedical, imaging, blood velocity, velocity estimation, cvd, blood flow, RF, analogue electronics, VHDL, FPGA, embedded systems, Doppler

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

The progress of diagnostic imaging has advanced significantly during the 20th century. As the cost of high-speed computational systems has grown increasingly accessible, so has the use of medical imaging become prominent. Millions of people have potentially been spared painful exploratory surgery through non-invasive diagnostic imaging. Thus, lives can be saved by early diagnosis and intervention through medical imaging. Advancements in scientific visualisation have in turn generated more complex data-sets of increased size and quality. The four major technologies used are ultrasound (*US*), X-ray, computed tomography (*CT*), and magnetic resonance imaging (*MRI*). Each technology has distinct advantages and disadvantages in biomedical imaging, and thus each is still relevant for modern medicine. Table 1.1 contains a comparison and summary of the various fundamental diagnostic imaging modalities.

Between 2004 and 2016, medical imaging has been reported to have been performed more than 5 billion times [16]. Later numbers from 2011 show a general doubling and in particular, a tenfold increase in ultrasound examinations between 2000 and 2011 [38]. Recent data reveal that this trend of doubling has continued throughout the years 2010 to 2020 [57], and reveal that even though patient processes were disrupted during the global SARS-CoV-2 pandemic, the number of medical imaging examinations per 1000 patients still increased. The reasons for this and, particularly, why ultrasound has seen a significant increase in use, can be attributed to its high resolution, cost-effectiveness, portability, and real-time interventional imaging. The downside of ultrasound is its limited penetration, restrictions for use in certain body parts, and inconsistent resolution. When comparing soft tissue examinations, which ultrasound is limited to, both *CT* and *MRI* can image the entire body with consistent resolution and contrast, but are more expensive and have poor portability due to the immense size of their hardware.

The cardiovascular system, which transports oxygen and nutrients to tissue, produces a complex flow pattern that causes velocity fluctuations. Several cardiovascular diseases (*CVD*) are also known to cause abnormal blood flow. In studies published by the Centers for Disease Control, a person dies from CVD every 34 seconds in the United States and complications from CVD cost 229 billion USD between 2017 and 2018 [58]. As mentioned above, ultrasound is a powerful tool for performing non-invasive imaging of the cardiovascular system [26], [32], and has no adverse risk to patients. Determining power spectral density (*PSD*) of a received signal is a common way to estimate blood velocity. A processed image of *PSD* is commonly known as a sonogram, where changes in blood velocity over time can be seen

Table 1.1: Comparison of medical imaging modalities [38]

| **Modality** | **Ultrasound** | **X-ray** | **CT** | **MRI** |
|---|---|---|---|---|
| Topic | Longitudinal, shear, mechanical properties | Mean X-ray tissue absorption | Local tissue X-ray absorbtion | Biochemistry (*T1* and *T2*) |
| Access | Small windows adequate | 2 sides needed | Circumferential around body | Circumferential around body |
| Spatial resolution | 0.2 mm to 3 mm[a] | $\sim 1$ mm | $\sim 1$ mm | $\sim 1$ mm |
| Penetration | 3 cm to 25 cm[b] | Excellent | Excellent | Excellent |
| Safety | Excellent | Ionizing radiation | Ionizing radiation | Very good |
| Speed | Real-time | Minutes | 20 minutes | Varies[†] |
| Cost | $ | $ | $$ | $$$ |
| Portability | Excellent | Good | Poor | Poor |
| Volume coverage | Real-time 3D volumes, improving | 2D | Large 3D volume | Large 3D volume |
| Contrast | Increasing (shear) | Limited | Limited | Slightly flexible |
| Intervention | Real-time 3D increasing | No[c] | No | Yes, limited |
| Functional | Functional ultrasound | No | No | fMRI |

[a] Frequency and axially dependent.

[b] Frequency dependent.

[c] Fluoroscopy limited.

[†] Typical: 45 minutes, fastest: Real-time (*low-res*).

Table 1.2: Comparison of papers in literature study

| | **Huang *et al.* [30]** | **Jana *et al.* [52]** | **Ding *et al.* [55]** | **Song *et al.* [49]** |
|---|---|---|---|---|
| Type | PW 10 MHz | CW 8 MHz | PW 3.7 MHz | PW 2.5 MHz |
| Display | Smartphone | Smartphone | Computer | Smartphone |
| Power | 12 V | 12 V | Details not available | Batteries 2S 3.7 V |
| Components | Timing controller, bipolar pulser, quadrature demodulation, SHA | RF amplifier, envelope detection, LP filter, FPGA, preamplifier, ADC | AFG, RF amplifier, quadrature demodulation, SHA, BP filter | LNA, ADC, FPGA demodulator and filter |
| DSP | 512 pt FFT | 512 pt FFT | FFT size not mentioned | 128 pt FFT |
| Metrics | Doppler spectrogram | Haemodynamic parameters | Doppler spectrogram | Doppler spectrogram |
| Output | Aux microphone signal | Bluetooth wireless | DAQ interface (LabVIEW) | Bluetooth wireless |
| Validation | Animal experiment | ML evaluation on humans | Physiological simulator | Physiological simulator |

## 1.1 Literature review

A systematic review was conducted using PubMed, Google Scholar, Elsevier, DTU FindIt and IEEE Xplore with the search terms "pulsed-wave Doppler ultrasound", "blood velocity estimation", and "ultrasound flow-meter". The search was limited to English-language articles. The literature search yielded more than 50 papers, of which 37 were studied for the purpose of learning from the contents [1]–[4], [6]–[14], [17], [19], [21]–[24], [27]–[31], [35], [37], [40], [43], [44], [47], [50]–[52], [55], [57], [59], [65]. In addition, textbooks [32], [38], [39] were used in the preparation and study of the theoretical principles of biomedical imaging and ultrasound.

Among these works are some of the earliest papers that outlined the field as it was emerging. Other articles study the possibilities of improvements in algorithms and experimental parameters. Overall, the results indicate that the Doppler flow meter is a reliable method for estimating blood flow velocity in various parts of the body. Studies include experiments using physiological simulators and in-vivo on humans and animals alike. Some of the review articles have compared Doppler flow-meters to other imaging techniques for this application, such as magnetic resonance imaging and computed tomography angiography, and have shown that the Doppler flow-meter is a cost-effective, portable and non-invasive choice. Of the selected papers studied in this project, four papers are distinctly relevant for the design and implementation of a blood velocity estimation system. A comparison between these three papers can be seen in table 1.2. Based on the literature review, a gap is identified in the acquisition method of the signal chain. A number of articles studied and developed the algorithms for blood flow estimation and imaging, but do did have an analogue front end (*AFE*) and used offline data acquisition methods which are not usable for clinicians. The selected three articles all feature an online data acquisition method using various methods of data capture. There is a potential to using selected features from all four articles in a combination to achieve a positive result. For instance, the *AFE* in Huang *et al.* is better documented than both other papers, and feature a pulsed-wave (*PW*) design that could be useful. On the other hand, their solution for the pulser is dated and use inflexible discrete timer integrated circuit (*IC*)s. Jana *et al.* use a continuous-wave (*CW*) based design and thus most details are on the receiver. However, it features

an field-programmable gate array (*FPGA*) soft microprocessor design to the fast Fourier transform (*FFT*) engine. Ding *et al.* feature a *PW* design, and use an arbitrary function generator (*AFG*) as the primary signal generator for the pulser as well as the demodulation clock. In that paper, there are some good figures for studying the pulse-echo waveforms of the *PW* type system. In Song *et al.*, a battery powered solution was implemented, and the system is untethered, which makes the system portability quite high. One drawback, however, is that the system uses a custom transducer design in the form of a neckband and is only able to measure the carotid artery.

Some of the project decisions resulting out of the study of these four papers include the desire to implement an *AFE* for a pulsed-wave system using the inspiration from all three papers, but also implement a novel pulse generator not using discrete *IC*s or with a lab instrument *AFG*, since it is not portable. Instead, with a flexible and configurable design that an embedded system enables.

## 1.2 Project scope

Table 1.3: Project specification

| **Project specification** |
|---|
| Study and research ultrasound and its principles and applications |
| Design and implement a device for ultrasound blood velocity estimation |
| Investigate and test the device in an experimental setting |
| Validate results with commercial equipment |
| Make quantifiable performance measurements on the system |
| Write a technical report documenting the project work |

A list of project goals is provided in table 1.3. The project is conducted under the guidance of advisors from the Electronics Laboratory and Embedded Systems Engineering Laboratory at Danmarks Tekniske Universitet (Technical University of Denmark) (*DTU*), and at the Brain/Bio Medical Microsystems Laboratory at Korea Advanced Institute of Science and Technology (*KAIST*). The report is divided into five chapters, and the first part is an introduction to the project. The second chapter will focus on explaining the theory of the topic of the project. The third chapter focuses on the synthesis of a system model for experimental testing. The fourth chapter explains the method of implementation during the assembly of the system. The fifth chapter will explain the testing methodology performed on the hardware. Finally, additional documentation of testing, code, circuit diagrams, and laboratory setups can be found in the appendix.

# Chapter 2

# Theory

This chapter explains the overall theory that forms the fundamental principles of this project. Initially, the characteristics of ultrasound will be explained from an acoustics standpoint. Then, a brief overview of systemic circulation is explained in vivo. Lastly, the various types of flow meters are outlined with their strengths and weaknesses.

## 2.1 Ultrasound

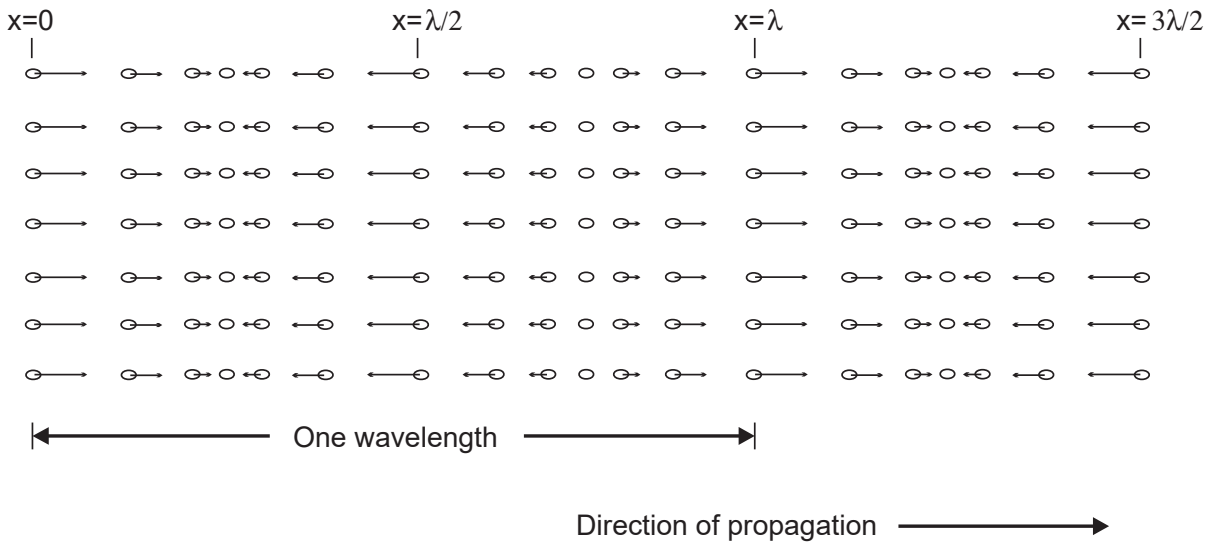



Figure 2.1: Particle displacement for a propagating ultrasound wave [32]

Ultrasound is a technology that transmits sound wave with frequencies above the audible range (20 to $20 \times 10^3$ Hz) to mechanically vibrate matter. The particles in the medium would be at rest and distributed uniformly before any disturbance. The wave propagates as a disturbance and the particles oscillate around their mean position due to the presence of the ultrasonic wave. Typically, the ultrasound (*US*) frequency-band used in clinical settings are from 1 to 15 MHz [38]. Certain specialist applications, such as eye, skin, and small animal imaging may use ultrasound frequencies at or above 20 MHz [31]. The trade-off with very high-frequency ultrasound balances increased resolution with reduction in penetration

depth. For blood velocity estimation or echocardiography, moderate penetration is more important than resolution. Thus, the frequency used is most often at 10 MHz or less. Figure 2.1 visualizes the propagation of a plane wave in matter. The oscillation occurs parallel to the wave's direction, making it longitudinal, and the disturbance will propagate with the variable $c$, which is determined by the medium and is given by eq. (2.1).

$$c = \sqrt{\frac{1}{\rho_0 \kappa_S}} \tag{2.1}$$

where $\rho_0$ is the mean density ($\mathrm{kg\,m^{-3}}$) and $\kappa_S$ is the adiabatic compressibility ($\mathrm{m^2\,N^{-1}}$). Since in the majority of cases, the propagation of ultrasound is linear, the linear propogation is assumed in this work. The acoustic pressure of the harmonic plane wave is expressed by eq. (2.2) and propagates along the $z$-axis.

$$p(t, z) = p_0 e^{j(\omega t - kz)} \tag{2.2}$$

where $\omega$ is the angular frequency, $k$ is the wave number and is expressed by $k = {}^{\omega}/_{c} = {}^{2\pi}/_{\lambda}$, and $p_0$ is the acoustic pressure amplitude. A spherical wave is expressed by eq. (2.3).

$$p(t, r) = p_0 e^{j(\omega t - kr)} \tag{2.3}$$

where $r$ is radial distance and is defined in a polar coordinate system. For each time instance, the acoustic pressure $p(t, r)$ is constant over a fixed radial position. In this scenario, the pressure amplitude is given by $p_0(r) = {}^{k_p}/_{r}$, where $k_p$ is a constant since the energy of the outgoing wave must be constant. Particle speed $u$ is dependent on the pressure caused by a wave expressed by eq. (2.4).

$$u = \frac{p}{Z} \tag{2.4}$$

where $Z$ is the characteristic acoustic impedance, defined as the ratio of acoustic pressure to particle speed at a given position in the medium and is expressed by eq. (2.5).

$$Z = \rho_0 c \tag{2.5}$$

Characteristic acoustic impedance $Z$ is one of the most significant variables in the characterization of propagating plane waves. Reference values for density, speed of sound, and characteristic acoustic impedance can be seen in table 2.1.

In the following sections, various acoustic wave phenomena will be briefly described.

### 2.1.1 Scattering

A wave propagating through a medium continues in the same direction until it encounters a new medium. When this occurs, a portion of the wave is transmitted into the new medium with a change in direction. Because the scattered wave is the result of several contributors, it is necessary to define it statistically. The amplitude distribution is Gaussian [32] and can thus be fully described by its mean and variance. The mean value is zero because the dispersed signal is caused by variances in the acoustic characteristics in the tissue. The correlation between multiple data is what allows ultrasound to determine blood velocities. Because minor movements have a significant correlation, it is feasible to discover alterations in location by comparing sequential measurements of moving structures, such as blood cells. In medical ultrasound, only one transducer is used to transmit and receive, and only the backscattered signal is analysed. The

Table 2.1: Approximate density, sound speed, and acoustic impedance of human tissue types [32]

| **Medium** | **Density** ($\rho_0$) kg/m$^3$ | **Speed of sound** ($c$) m/s | **Acoustic impedance** ($Z$) kg/(m$^2$ s) |
|---|---|---|---|
| Air | 1.2 | 333 | $0.4 \times 10^3$ |
| Blood | $1.06 \times 10^3$ | 1566 | $1.66 \times 10^6$ |
| Bone | $1.38$–$1.81 \times 10^3$ | 2070–5350 | $3.75$–$7.38 \times 10^6$ |
| Brain | $1.03 \times 10^3$ | 1505–1612 | $1.55$–$1.66 \times 10^6$ |
| Fat | $0.92 \times 10^3$ | 1446 | $1.33 \times 10^6$ |
| Kidney | $1.04 \times 10^3$ | 1567 | $1.62 \times 10^6$ |
| Lung | $0.4 \times 10^3$ | 650 | $0.26 \times 10^6$ |
| Liver | $1.06 \times 10^3$ | 1566 | $1.66 \times 10^6$ |
| Muscle | $1.07 \times 10^3$ | 1542–1626 | $1.65$–$1.74 \times 10^6$ |
| Spleen | $1.06 \times 10^3$ | 1566 | $1.66 \times 10^6$ |
| DI | $1 \times 10^3$ | 1480 | $1.48 \times 10^6$ |

power of the scattered signal is defined by the scattering cross-section, which in small cases means a uniform intensity $I_i$, and is expressed by eq. (2.6).

$$P_s = I_i \sigma_{sc} \tag{2.6}$$

where $\sigma_{sc}$ is the scattering cross-section in square meters. The backscattering cross section is material dependant and determines the intensity of the scattering. If the dispersed energy is evenly emitted in all directions, the scattered intensity is given by eq. (2.7).

$$I_s = \frac{P_s}{4\pi R^2} = \frac{\sigma_{sc}}{4\pi R^2} \cdot I_i \tag{2.7}$$

where $R$ is the distance to the scattering region [32]. This results in a spherical wave. A transducer with radius $r$ gives the power $P_r$, presuming the attenuation and focus are neglected, and is expressed by eq. (2.8).

$$P_r = I_s \pi r^2 = \sigma_{sc} \frac{r^2}{4R^2} \cdot I_i \tag{2.8}$$

The backscattering coefficient, which defines the characteristics from its volume, is another measure of scattering strength. It is defined as the average received power per steradian volume of scatterers when flooded with plane waves of unit amplitude and the unit is 1/cmsr. Back-scattering coefficients in the blood are significantly lower than the backscattering coefficients from various tissue types. This poses a challenge when estimating blood flow close to tissue vessel walls [5], [32].

### 2.1.2 Attenuation

The ultrasonic wave will be reduced as it propagates through the tissue due to absorption and scattering. The attenuation in tissue is frequency dependent, with greater attenuation with increasing frequency. Because of absorption and dispersion, the ultrasonic wave will be attenuated as it travels through the

tissue. The relationship between attenuation, distance travelled, and frequency is often linear. Attenuation in the tissue occurs as a result of both dispersion, which spreads energy in all directions, and absorption, which turns it into thermal energy. A table of approximate attenuation values can be seen in table 2.2.

Table 2.2: Approximate attenuation values for human tissue [32]

| **Tissue** | **Attenuation** dB/(MHz · cm) |
|---|---|
| Liver | 0.6–0.9 |
| Kidney | 0.8–1 |
| Spleen | 0.5–1 |
| Fat | 1–2 |
| Blood | 0.17–0.24 |
| Plasma | 0.01 |
| Bone | 16–23 |

Equation (2.9) expresses the the exponential decrease of pressure of a wave propagating in the $z$-direction.

$$p(z) = p(z = 0)e^{-\alpha z} \tag{2.9}$$

In eq. (2.9), $p(z = 0)$ is the pressure in the point of origin and $\alpha$ is the attenuation coefficient. The attenuation coefficient unit is $\mathrm{Np\,cm^{-1}}$ and, alternatively, $\mathrm{dB\,cm^{-1}}$ with the relationship described in eq. (2.10).

$$\alpha = \frac{1}{z} \ln \frac{p(z = 0)}{p(z)} \tag{2.10a}$$

$$\alpha(\mathrm{dB\,cm^{-1}}) = 20(\log_{10} e)\alpha(\mathrm{Np\,cm^{-1}}) = 8.68\alpha(\mathrm{Np\,cm^{-1}}) \tag{2.10b}$$

The significance of absorption and scattering in ultrasonic attenuation in biological tissues is a point of contention. Scattering adds just a few per cent to attenuation in most soft tissues. As a result, it is fair to conclude that absorption is the primary mechanism of ultrasonic attenuation in biological tissues [39].

### 2.1.3 Transducer

A typical layperson considers items such as speakers and microphones in the context of PA systems as transducers. In the case of medical *US* it is the device that generates the acoustic pressure field, which is emitted into the tissue. A common *US* transducer is a lead circonate titanate (*PZT*) type, which has a piezoelectric crystal inside the housing. When excited, this crystal emits ultrasound waves toward flowing blood. The red blood cells will reflect a fraction of the emitted waves. These reflected waves are of a different frequency than the transmitted waves. If the red blood cells move away from the transducer, the frequency will be lower. If the red blood cells are moving towards the transducer, the frequency will be higher. This is caused by the Doppler effect. The reflected ultrasonic waves return to the crystal and are converted back into electrical signals. The single-element transducer shown in fig. 2.2 has a minimal imaging window and has to be mechanically manipulated to obtain a wide window, which is unfeasible

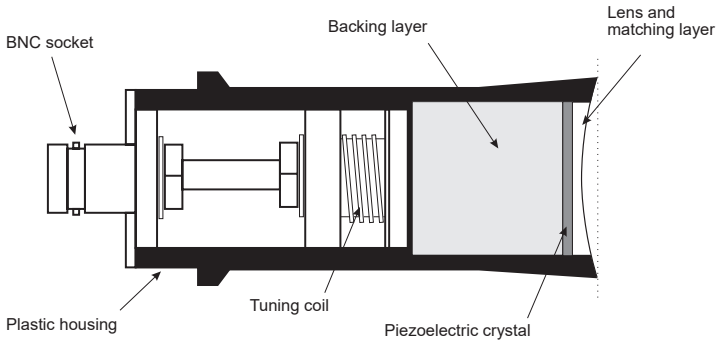



Figure 2.2: Single element ultrasound transducer construction [32]

for responsive high-frequency imaging. Thus, in these cases, a transducer array is used. Various types of *US* transducers exist with different strengths and weaknesses, as shown in fig. 2.3.

### 2.1.4 Doppler effect

The Doppler effect is a phenomenon in which an observer perceives a shift in the frequency of sound emitted from a source when either the source or the observer is moving, or both are moving. The reason for the perceived change in frequency is visualised in fig. 2.4. In diagram (a), the source $S_p$ is stationary and produces a spherical distribution pattern of the wave with the perceived frequency of the observer is given by $f = {}^{c}/_{\lambda}$, where $c$ is the velocity of the wave in the medium and $\lambda$ is the wavelength. In diagram (b), the sound source is moving towards the right with a velocity $v$. The locomotion of the source changes the distribution pattern and causes a longer wavelength on the left, thus resulting a lower perceived frequency, and a shorter wavelength on the right, indicating a higher perceived frequency, both denoted as $\lambda'$ in the diagram. In the case of the observer on the right side, the perceived frequency becomes eq. (2.11).

$$f' = \frac{c}{\lambda} = \frac{c}{\lambda - vT} = \frac{c}{(c-v)T} = \frac{c}{c-v} \cdot f_0 \tag{2.11}$$

And vice versa, on the left side, the perceived frequency becomes eq. (2.12).

$$f' = \frac{c}{c+v} \cdot f_0 \tag{2.12}$$

This perceived difference between the frequency that is transmitted from the source $f_0$, and the perceived frequency $f'$ is also called the Doppler frequency, $f_d$. When these connections are combined, the Doppler frequency for a source moving with velocity $v$ and an observer travelling with velocity $v'$ is given by eq. (2.13).

$$f_d = f' - f = \left( \frac{c+v'}{c-v} - 1 \right) \tag{2.13}$$

If both source and observer are moving with the same velocity, $v$, assuming $c \gg v$, the $v$ cancels out and the expression is reduced to eq. (2.14).

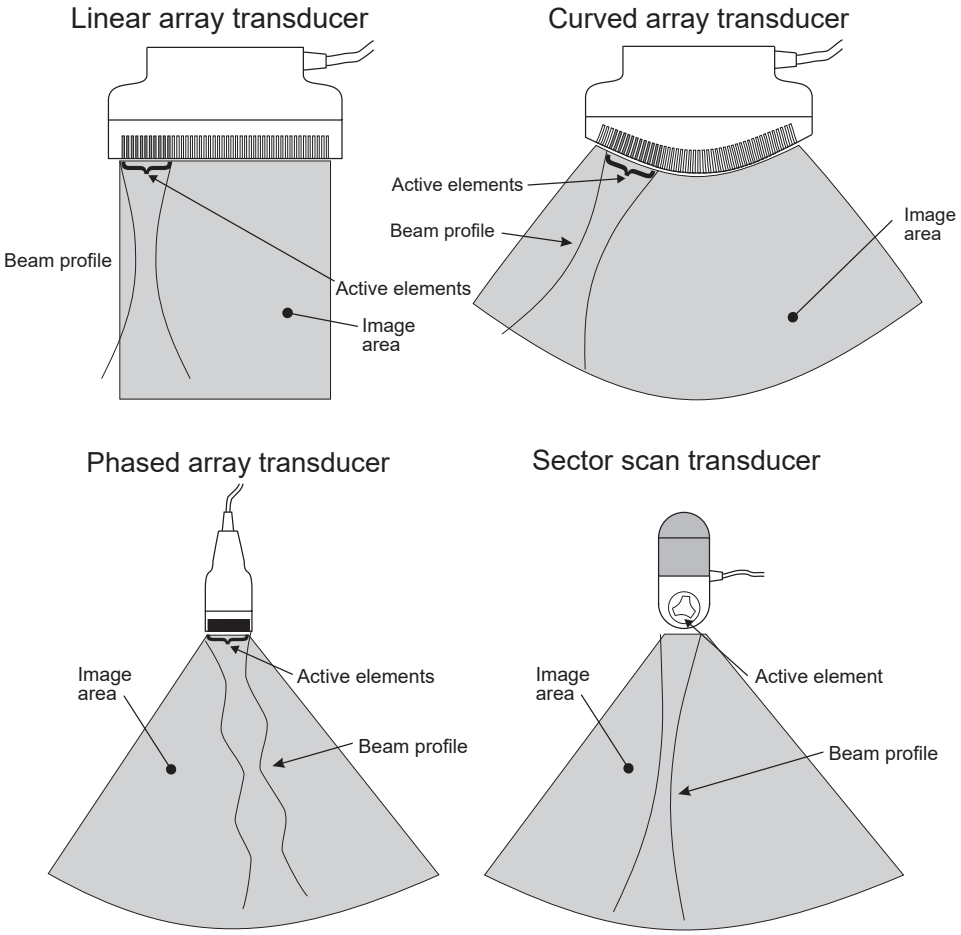



Figure 2.3: Transducer types for acquiring B-mode images [32]

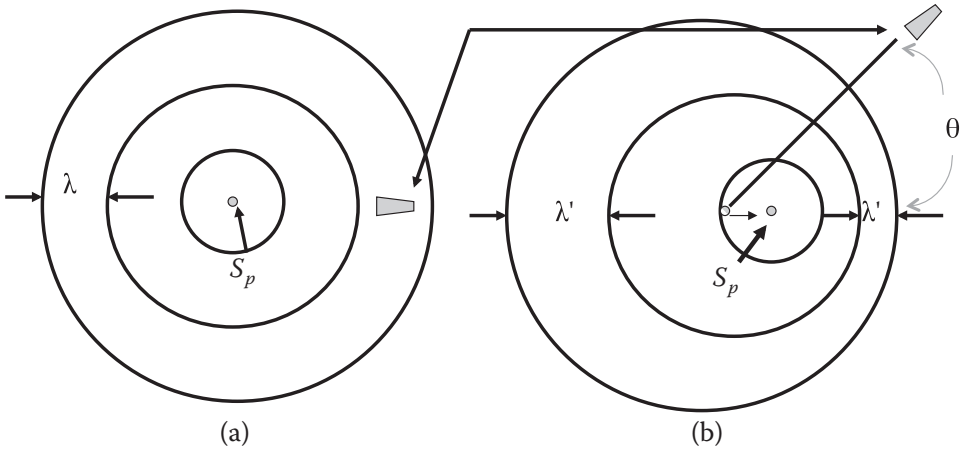



Figure 2.4: Doppler effect diagram. A stationary observer perceives a change in the frequency of a wave generated by a moving source toward the observer as a result of a wavelength shift from $\lambda$ to $\lambda'$. In (a), the source is still. In (b), the source is moving at a velocity $v$. [39]

$$f_d = \frac{2vf}{c} \tag{2.14}$$

If the velocity of the moving source is traveling with an incident angle $\theta$, the $v$ in eq. (2.14) is replaced with $v(\cos\theta)$. This results in the expression found in eq. (2.15) and forms the basis for applied Doppler effect measurements.

$$\boxed{f_d = \frac{2v(\cos\theta)f}{c}} \tag{2.15}$$

The Doppler effect is used in ultrasonic Doppler devices used to image blood flow transcutaneously. An ultrasonic transducer in these devices sends ultrasonic waves into a blood artery, and the scattered radiation from moving red cells is measured by either the same transducer or a second transducer. The Doppler frequency, which is determined by the velocity of red blood cells, is extracted using modern electronic demodulation techniques which will be explored later.

## 2.2 Flow Physics

The flow physics of the human circulatory system is sophisticated, and numerous non-stationary flow patterns are observed. The human circulatory system takes care of transporting oxygen and nutrients to organs, as well as disposing of waste products produced by metabolism. It is possible because the blood within the circulatory system contains several smaller sub-components, such as plasma and formed cellular elements that perform these vital functions. Initially, blood is discharged from the left ventricle of the heart through the aorta and travels to all areas of the body through multiple branches of the arterial tree. When blood flows through the arteries, it enters smaller channels known as arterioles. These arterioles lead to a network of tiny capillaries through which nutrients and waste materials are exchanged between the blood and the organs. The capillaries connect to form a network of venulae, which supply the veins and deliver blood back to the heart. This system, in its entirety, is called systemic circulation. A diagram of the circulatory system as described above can be seen in fig. 2.5. In summary, when examining the elements that comprise the circulatory system, it consists of several components:

- The heart, the primary organ of the circulatory system that maintains blood pressure and controls blood velocity.
- The blood, and its sub-components
  - Plasma, which forms the primary volume, contains nutrients and formed cellular elements.
  - Red and white blood cells, which carry oxygen and fight off infections, respectively.
  - Platelets, which are also known as thrombocytes, have the function of clotting during blood vessel injury.
- The blood vessels
  - Arteries (and arterioles), transport oxygenated blood to organs and tissues at high pressure and velocity.
  - Capillaries are thin but wide-ranging blood vessels that perform the exchange of matter between the circulatory system and tissue.

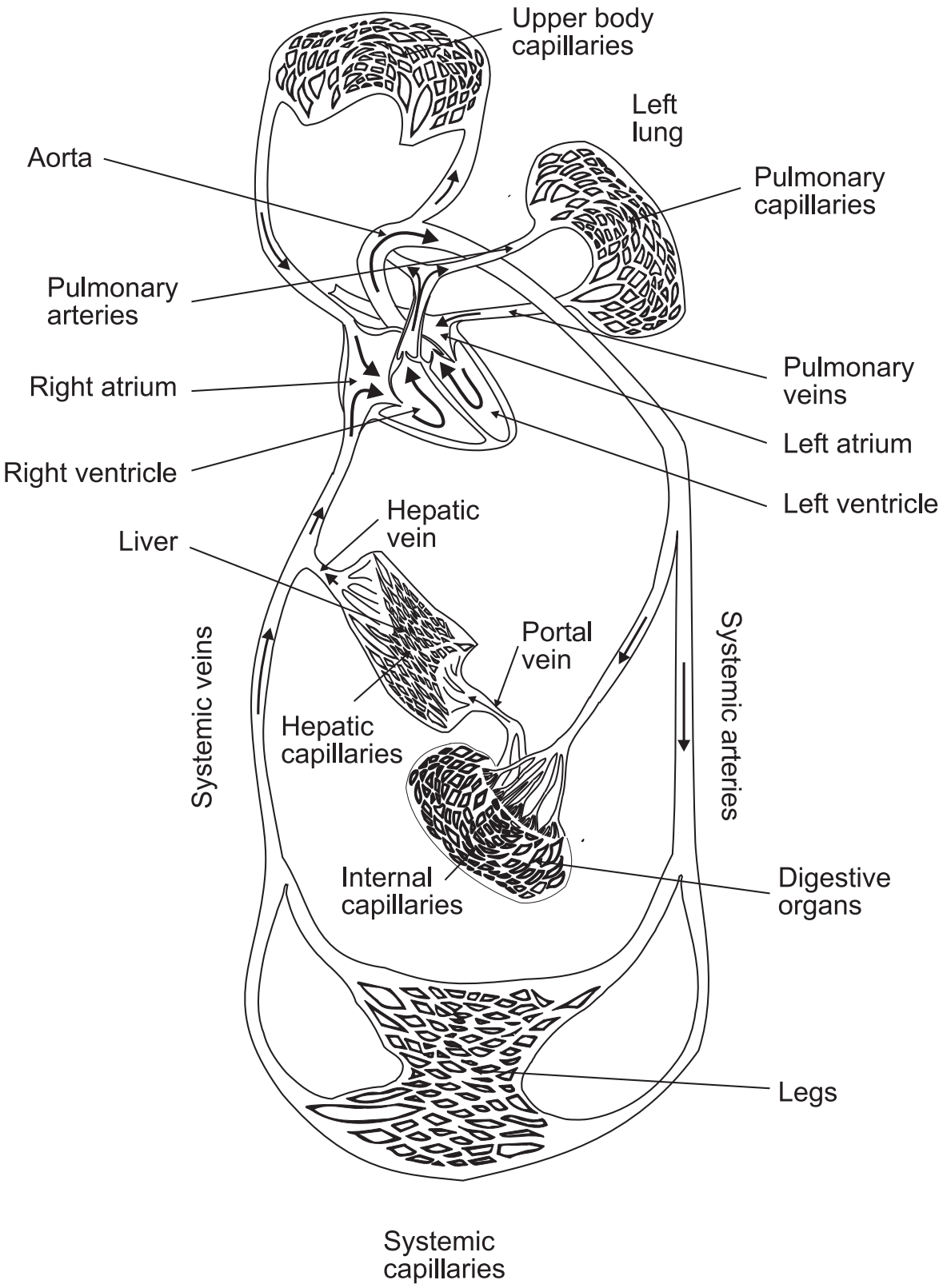



Figure 2.5: Circulatory system of the human body [32]

– Veins (and venulae) carry blood back to the heart at low pressure and velocity.

### 2.2.1 Blood flow

Blood flow is the amount of blood that goes through a blood vessel in a particular period of time, and has a complicated flow pattern due to its pulsing flow. Advanced analysis of haemodynamics is not within the scope of this report, so the explanation will be brief. The primary forces that determine the blood flow $F$ are the pressure difference across a blood vessel and vascular resistance. It is determined by Ohm's law as in eq. (2.16).

$$F = \frac{\Delta P}{R} \tag{2.16}$$

where $\Delta P$ is the pressure difference across the blood vessel and $R$ is the vascular resistance. The pressure difference $\Delta P$ is calculated with eq. (2.17).

$$\Delta P = P_1 - P_2 \tag{2.17}$$

Where $P_1$ and $P_2$ are the blood pressures measured at each end of the blood vessel. Pressure has significant influence on blood flow because an increase in arterial pressure not only increases the force that pushes blood through the capillaries but also expands the vessels, lowering vascular resistance. Selected dimensions and flow characteristics can be seen in table 2.3.

Table 2.3: Typical dimensions and flow in human circulatory system [32]

| **Vessel** | **Internal diameter** (cm) | **Wall thickness** (cm) | **Length** (cm) | **Young's modulus** ($N/m^2 \cdot 10^5$) | **Peak velocity** ($cm\,s^{-1}$) | **Mean velocity** ($cm\,s^{-1}$) | **Reynolds number** (peak) | **Pulse propagation velocity** ($cm\,s^{-1}$) |
|---|---|---|---|---|---|---|---|---|
| Ascending aorta | 1–2.4 | 0.05–0.08 | 5 | 3–6 | 20–290 | 10–40 | 4500 | 400–600 |
| Descending aorta | 0.8–1.8 | 0.05–0.08 | 20 | 3–6 | 25–250 | 10–40 | 3400 | 400–600 |
| Abdominal aorta | 0.5–1.2 | 0.04–0.06 | 15 | 9–11 | 50–60 | 8–20 | 1250 | 600–700 |
| Femoral artery | 0.2–0.8 | 0.02–0.06 | 10 | 9–12 | 100–120 | 10–15 | 1000 | 800–1030 |
| Carotid artery | 0.2–0.8 | 0.02–0.04 | 10–20 | 7–11 | | | | 600–1100 |
| Arteriole | 0.001–0.008 | 0.002 | 0.1–0.2 | | 0.5–1 | | 0.09 | |
| Capillary | 0.0004–0.0008 | 0.0001 | 0.02–0.1 | | 0.02–0.17 | | 0.001 | |
| Inferior vena cava | 0.6–1.5 | 0.01–0.02 | 20–40 | 0.4–1 | 15–40 | | 700 | 100–700 |

## 2.3 Devices

A device that measures the flowing of blood is called a flowmeter. Flowmeters may be used both inside and outside of vessels. One of the flowmeters that may be used outside the vessel to monitor flow is *US*. Figure 2.6 depicts an ultrasonic wave of frequency $f$ insonifying a blood artery, resulting in an angle of $\theta$ relative to velocity $v$. For simplicity, it is assumed that blood flows in a vessel at a constant velocity $v$. The echoes returned are shifted in frequency as described in eq. (2.15) earlier in the chapter. The echoes scattered by blood after being insonified by an ultrasonic wave convey information about the velocity of blood flow. Blood flow measurements are often used in clinical settings to determine the status of blood vessels and organ functioning. The two commonly used fundamental techniques for ultrasound Doppler flow measurements are continuous-wave (*CW*) and pulsed-wave (*PW*). Both will be explained.

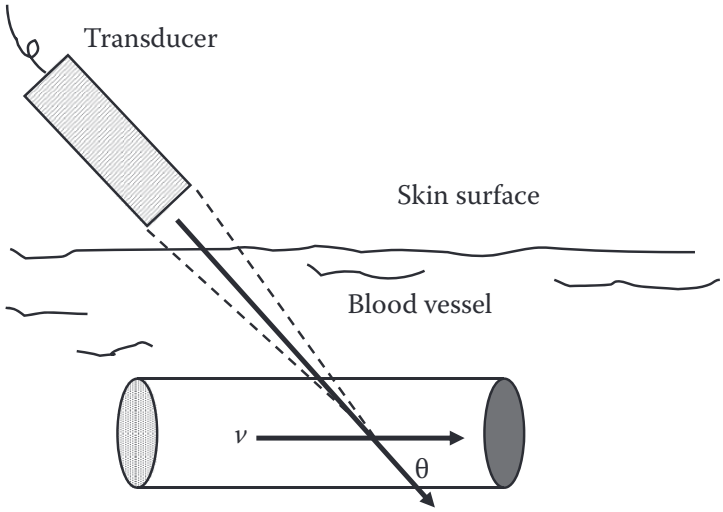



Figure 2.6: Diagram of *US* wave transmitted and reaching blood vessel with incident angle $\theta$ [39]

### 2.3.1 Continuous-wave Flowmeter

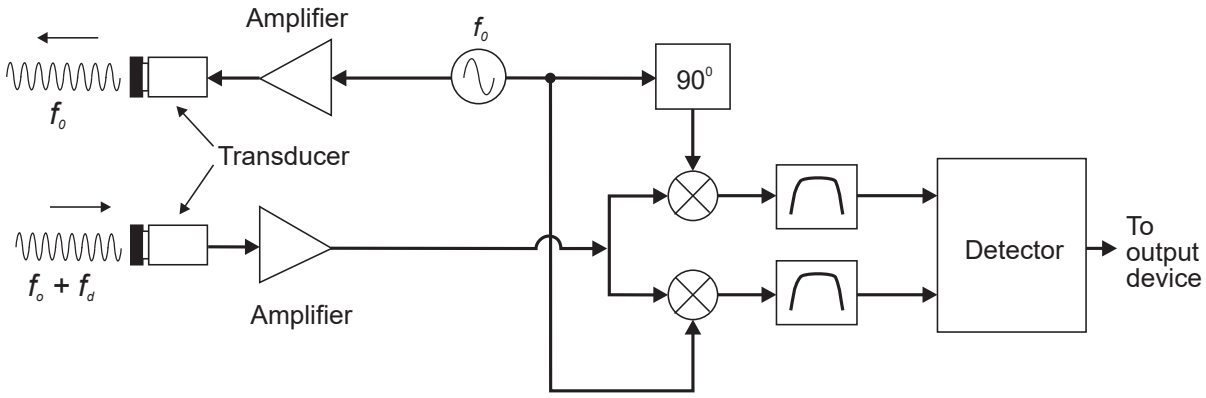



Figure 2.7: Block diagram of *CW* flowmeter [32]

The earliest non-invasive cardiovascular diagnostic technologies relied heavily on *CW* Doppler flowmeters. To continuously transmit waves and receive signals from moving reflectors, the *CW* flowmeter uses two transducers. *CW* flowmeters use less sophisticated electronics than *PW* flowmeters. A drawback to the *CW* flowmeter is the lacking depth discrimination due to the continuous characteristic of this device type. A block diagram of a typical *CW* flowmeter can be seen in fig. 2.7. The basic principles of the device are previously explained in section 2.1.4, and the measurement of the device is described in eq. (2.11). The device continuously emits an ultrasonic wave in the first transducer expressed as a function of time by eq. (2.18) [32].

$$e(t) = \cos(2\pi f_0 t) \tag{2.18}$$

While receiving the backscattered signal on the second transducer expressed by eq. (2.19) [32].

$$r_s(t) = a\cos\left(2\pi f_0 \alpha(t - t_0)\right) \tag{2.19}$$

$$\alpha \approx 1 - \frac{2v_z}{c} \tag{2.20}$$

$$\alpha t_0 \approx \frac{2d_0}{c} \tag{2.21}$$

Where $v_z$ indicates the velocity in the $z$ direction. Applying the Fourier transform, the expression yields eq. (2.22).

$$r_s(t) \cdot e^{j2\pi f_0 t} \Longleftrightarrow R_s(f - f_0) \tag{2.22}$$

Where $R_s(f-f_0)$ is the Fourier transform of $r_s(t)$. The received signal is then multiplied with a quadrature signal of frequency $f_0$ to find the Doppler frequency in eq. (2.23).

$$m(t) = a\left[\cos(2\pi f_0 t) + j\sin(2\pi f_0 t)\right]\cos(2\pi f_0 \alpha(t - t_0)) \tag{2.23}$$

$$\begin{aligned} &= \frac{a}{2}\Big\{\cos(2\pi f_0[(1-\alpha)t - \alpha t_0]) + \cos(2\pi f_0[(1-\alpha)t - \alpha t_0]) \\ &\quad + j\sin(2\pi f_0[(1-\alpha)t - \alpha t_0]) + j\sin(2\pi f_0[(1-\alpha)t - \alpha t_0])\Big\} \end{aligned} \tag{2.24}$$

As is general for quadrature demodulation, the resulting signal contains the frequency components of the sum and difference of the emitted and received signals' frequencies shown in fig. 2.8, where the signals are shown in time and frequency domains.

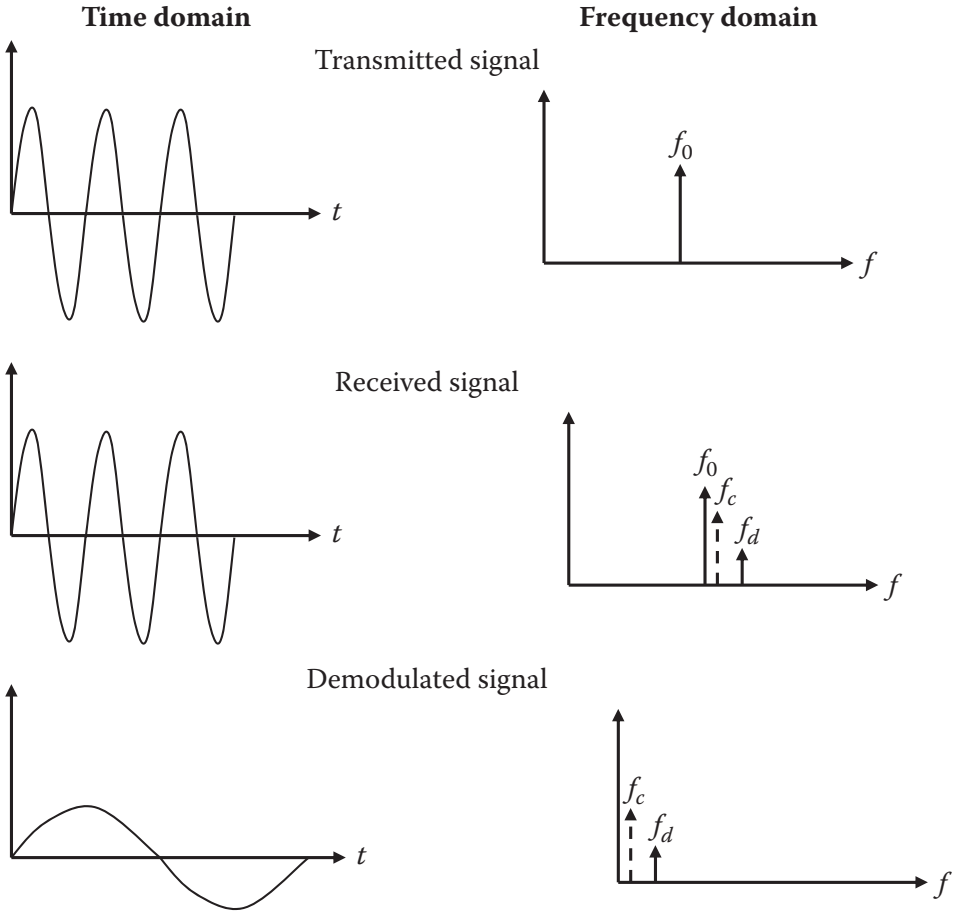



Figure 2.8: Demodulation effects of Doppler signals in time and frequency domain [39]

Generally, a band-pass (*BP*) filter is used on the demodulated signal to remove the high-frequency summed signal at twice the frequency of $f_0$. The filtered signal after the *BP* filter is expressed by eq. (2.25) and contains the Doppler shift of the emitted signal.

$$m_f(t) \approx \frac{a}{2} e^{\left(j2\pi f_0 \frac{2v_z}{c} t\right)} e^{(-j2\pi f_0 \alpha t_0)} \tag{2.25}$$

Where the second exponential term is the delay proportional to the time between transmission and receiving of the signal. The selected cutoff frequency is chosen to be much lower than the carrier frequency to remove the carrier wave. One issue with ultrasonic Doppler blood flow monitoring is that the blood vessels that generate large reflected echoes are also moving with a low velocity. These big, slow-moving echoes are referred to as clutter signals in Doppler nomenclature. The band pass filter's low-end cutoff frequency must be designed to minimize interference from these clutter signals. The design of this band pass filter in the low-frequency region, which serves the function of high pass, also known as a clutter rejection filter, has proven troublesome since the magnitude of clutter signals is many orders greater than that of blood and may obfuscate those from slow-moving blood.

Table 2.4: Measured frequency shifts with a Doppler 3 MHz transducer at various velocities at a 45° incident angle [32]

| **Velocity** ($v$) m/s | **Doppler frequency** ($f_d$) Hz |
|---|---|
| 0.01 | 28 |
| 0.1 | 276 |
| 0.5 | 1377 |
| 1 | 2755 |
| 2 | 5510 |
| 5 | 13 770 |

Seen in table 2.4 is an example of measured Doppler frequencies using a 3 MHz transducer using the method shown in fig. 2.6. Note that the measured frequencies are all within the audible range.

### 2.3.2 Pulsed-wave Flowmeter

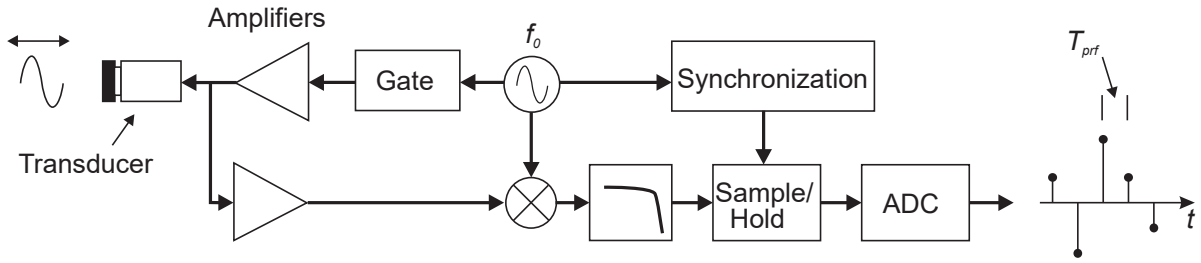



Figure 2.9: Block diagram of *PW* flowmeter [32]

The concept of a pulsed-wave flowmeter was proposed in [2] and other related articles. This type of flowmeter is periodically changing from a transmitter to a receiver. In the transmit mode, the transducer emits a series of pulses. When in the receiving mode, the transducer is listening for the backscattered signal. A simplified block diagram can be seen in fig. 2.9. The movement of particles within the blood causes a displacement in the backscattered signal. These systems are commonly referred to as "Doppler systems" even though it is somewhat misleading. The effects of attenuation are also causing a shift in frequency of a higher magnitude than the velocity of particles in the blood. This is because the conventional Doppler effect is not the straightforward methodology that is applied to the analysis of

the back-scattered signal. It is, in fact, an artefact. It is the shift in the location of the scatters that is observed, not the shift in the transmitted frequency. Figure 2.10 shows the received signal after demodulation and filtering; the depth in tissue is fixed here, and the signals displayed on the left side of the figure are the result of a pulse sequence. Each line represents a single pulse, and each pulse is emitted at a pulse repetition frequency, $f_{\text{prf}}$. Instead, on the right, the dotted line shows the sampled signal formed by taking into account the amplitude of each pulse after a specified time period. To depict the signals on the graph, a single pulse is emitted for each line, and the signals are displaced in amplitude. The sampled signal is displayed on the right. In the pulsed-wave flowmeter, the received signal is described

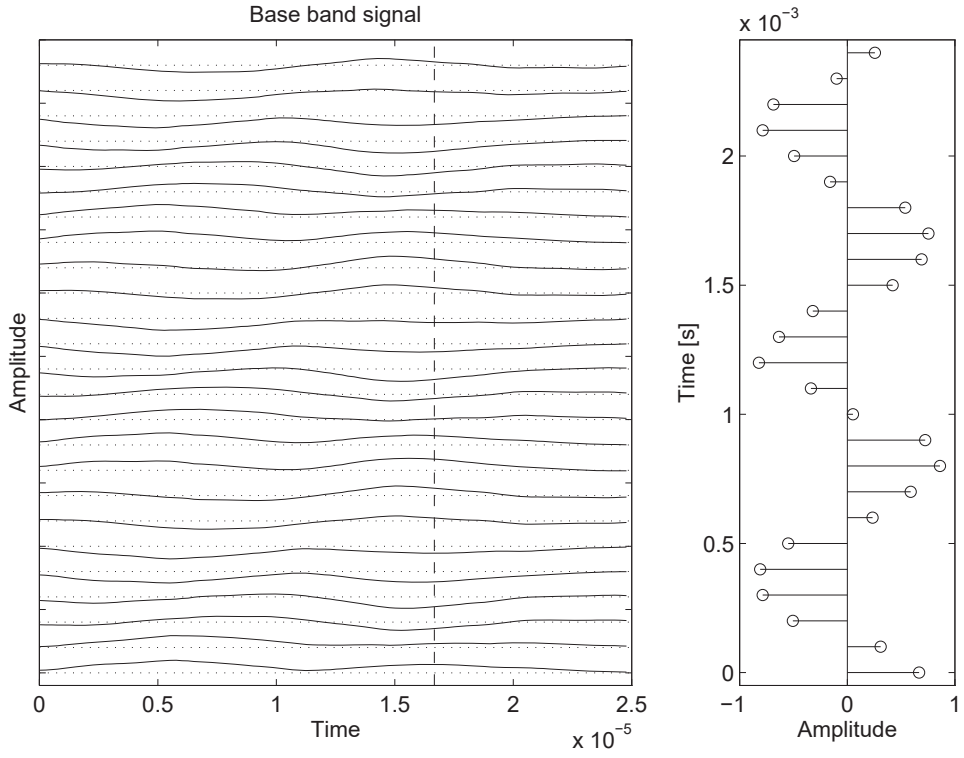



Figure 2.10: Sampling for a gate pulsed wave system with a single range [32]

by eqs. (2.26a) to (2.26d) [32].

$$r_s(t) = a \cdot e^{(\alpha(t-t_0))} \tag{2.26a}$$

$$\alpha = \left(1 - \frac{2v_z}{c}\right) \tag{2.26b}$$

$$\alpha t_0 = \frac{2d_0}{c} \tag{2.26c}$$

$$v_z = |\vec{v}| \cos\theta \tag{2.26d}$$

Equation (2.27) provides the calculation for the time shift $t_s$ of the RF signal between two emissions, which is determined by the distance that the scatterer moves in the direction of the ultrasound beam proportional to the velocity component $v_z$.

$$t_s = \frac{2v_z}{c} T_{\text{prf}} \tag{2.27}$$

Where $c$ is the speed of sound in the medium, $T_{\text{prf}}$ is the interval between each pulse emission. To measure the movement of the scatterer, the signal can be recorded at a certain depth. By selecting one

sample at that specific depth for each line, a sampled signal with a frequency that corresponds to the scatter velocity can be obtained. Hypothetically, if the velocity of stationary scatterers in blood was measured, a constant amplitude would be measured. A change in the sample value is observed when there is movement. Between two pulses, the scatterer movement is proportional to the velocity $v_z$ in the direction of the ultrasound beam. Taking one sample from each line at a certain depth yields a sampled signal with a frequency proportional to the scatter velocity. After the back-scattered signal is received it is multiplied by the centre frequency of the emitted pulse and filtered to remove the sum frequency [32]. A analogue-to-digital converter ($ADC$) quantifies the signal for further signal processing. Referring to displacement fig. 2.10 again, the dashed vertical line represents the sample of each pulse that is taken. If sampling is done $T_s$ after pulse emission, the measurement depth is expressed by eq. (2.28).

$$d_0 = \frac{T_s c}{2} \tag{2.28}$$

Thus, if a sample is taken at the same depth for each line, resulting in a sinusoidal signal proportional in frequency to the scatter velocity [15]. This technique improved the accuracy of the investigations of blood vessels and facilitated the display of velocity profiles. Equation (2.29a) describes the relationship between the received sampled signal for a single scatter and the sinusoidal pulse emitted by the transducer during the $i^{\text{th}}$ pulse emission.

$$r(i) = a(i) \sin\left(2\pi f_p i T_{\text{prf}} + \theta\right) \tag{2.29a}$$

$$f_p = \frac{2v_z}{c} f_0 \tag{2.29b}$$

Where $f_p$ is expressed by eq. (2.29b), $a(i)$ is the amplitude for the $i^{\text{th}}$ pulse, $f_0$ is the emitted frequency, and $\theta$ is the phase factor proportional to the depth of interest.

## 2.4 Blood Velocity Estimation

This section will explain the methodology of velocity estimation, assuming a pulsed-wave system signal is obtained from a back-scattered signal. There are variations of methods that can be applied, but only the implementation that will be used is discussed in this report.

### 2.4.1 Spectral Envelope

Figure 2.11 displays a signal where the scatterers in the sampled depth $d_0$ move away from the transducer, and the shift in the pulsed signal is present. The measurement is created by taking one sample at a specific depth ($d_0$) for each RF line. The sampling is performed at a frequency of $f_{\text{prf}}$, as indicated by the dotted line in the figure. Due to the slow movement of the received signals past the sample volume, there is a gradual change in the sampled signal over time. As a result of the slow movement of the signals, the frequency of the sampled signal is much lower than that of the RF signal. It's worth noting that the received signal not only shifts in time from pulse to pulse, but its shape also changes. This is because the signal is constructed by adding up responses from many scatterers that move at different speeds. A stationary structure will result in an identical sample value for all segmented RF lines. The spectrum of this stationary signal is shown in eq. (2.30).

$$|H_s(f)| = \left| a \frac{\sin(\pi f N T_{\text{prf}})}{\sin(\pi f T_{\text{prf}})} \right| \tag{2.30}$$

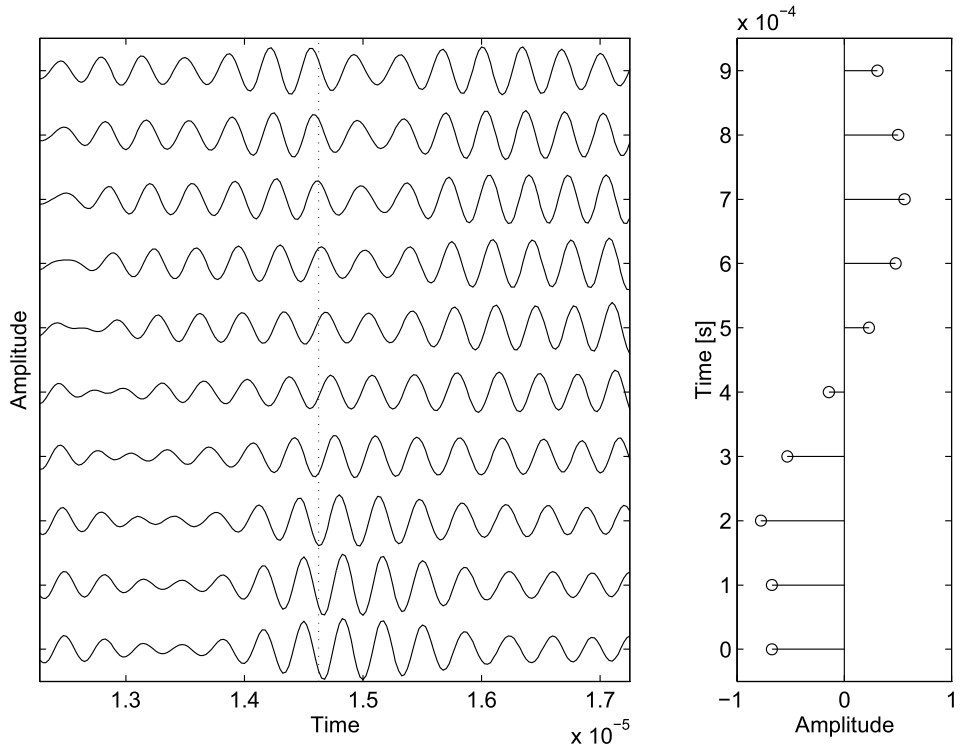



Figure 2.11: Simulated RF sampling of a blood vessel with segmented receiver line with a sample interval denoted by a dotted line (left) and the resulting amplitude of the sample (right) [32]

which for $f = 0$ is equal to $aN$ and has its first zero at ${}^{f_{\text{prf}}}/_{N}$. The signal spectrum based on eq. (2.30) is shown in fig. 2.12.

When dealing with small blood vessels, the stationary echoes can be significantly stronger (up to 40 dB) than the blood signal itself. To properly visualize blood flow details, the stationary signal must be eliminated from the sonogram. This requires setting the cutoff frequency to at least $f_{\text{prf}} = N$, in order to remove the main component. In some cases, a higher cutoff frequency may be necessary, particularly if the stationary echo is strong or the vessel wall is in motion. The speed and pulse repetition frequency of a single moving scatterer affects the signal received. The signal will look like the pulse, but its time scale and frequency will differ from the RF pulse because the pulse waveform slowly moves past the point where it is measured. Figure 2.13 shows this, where a beam from a concave transducer is crossed by a single scatterer. The time shift between each line is determined by the expression in eq. (2.31).

$$t_s = \frac{2v_z}{c} \cdot T_{\text{prf}} \tag{2.31}$$

Which increases linearly by line $i$. Individual sinusoidal components are expressed by eq. (2.32).

$$y(i) = \sin\left(2\pi \frac{2v_z}{c} \cdot f_0 i T_{\text{prf}}\right) \tag{2.32}$$

Where the variable $iT_{\text{prf}}$ corresponds to time and $f_0$ corresponds to the fundamental frequency. The received signal frequency is expressed by ${}^{2v_z}/_{c} \cdot f_0$, where the frequency axis is scaled by factor ${}^{2v_z}/_{c}$ as seen in fig. 2.14. The center frequency transforms to $f_p = \frac{2v_z}{c} f_0$. Every other spectral element is transformed by factor $\frac{2v_z}{c}$ and the received signal has the shape of a scaled frequency pulse. A sufficient number of lines must be captured, or only a part of the pulse is obtained. In practical terms, it corresponds

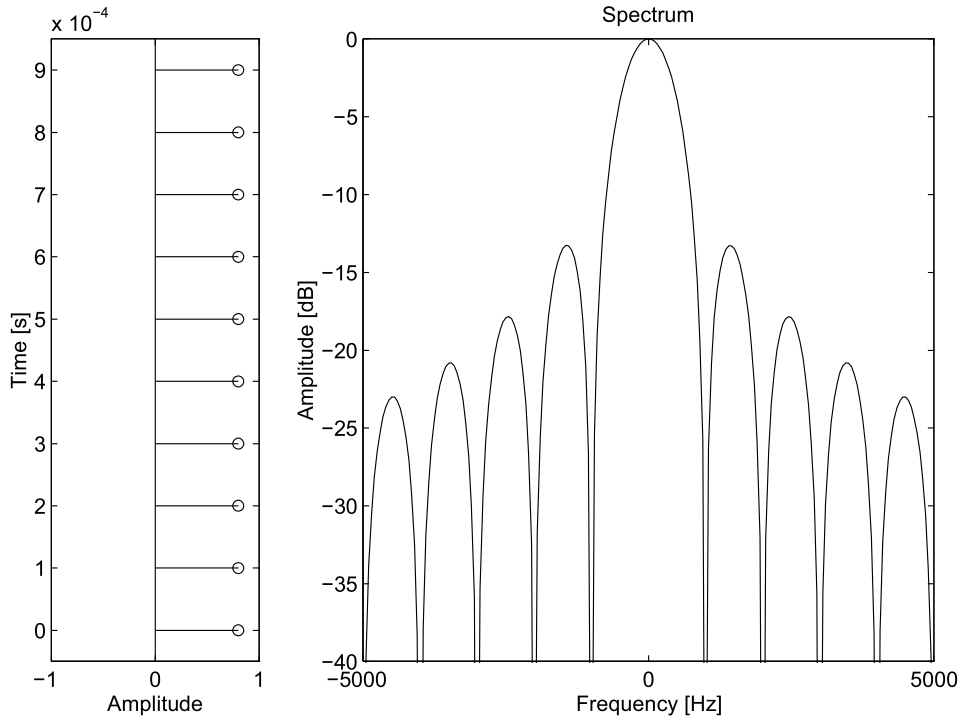



Figure 2.12: Stationary tissue sampled signal (left) and stationary tissue sample spectrum (right) [32]

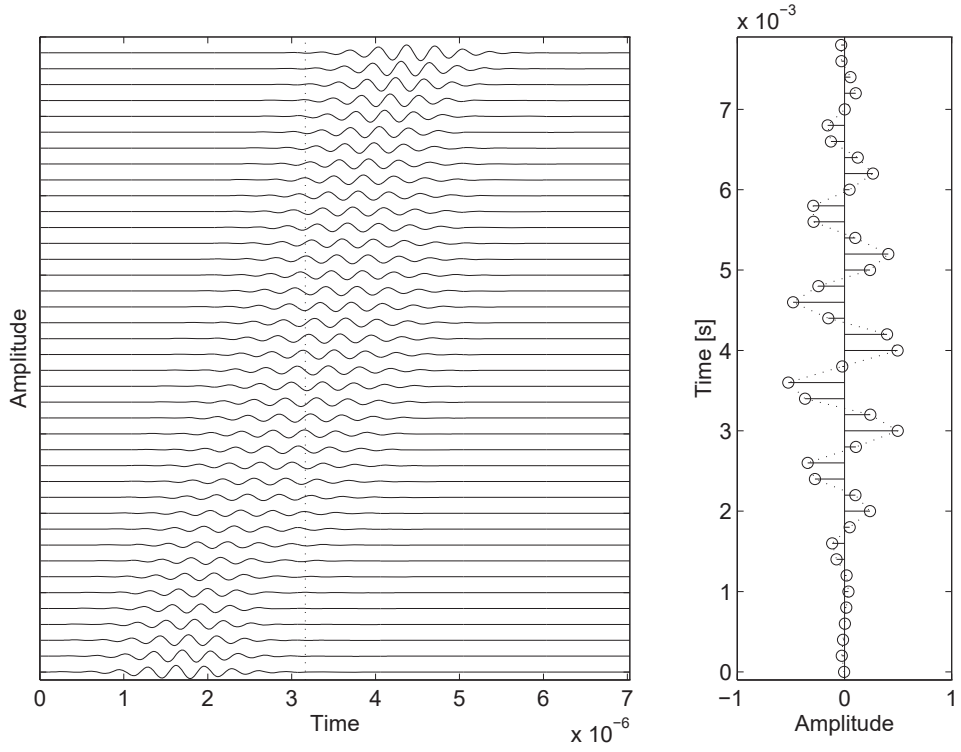



Figure 2.13: Single moving scatterer crossing a concave transducer beam [32]

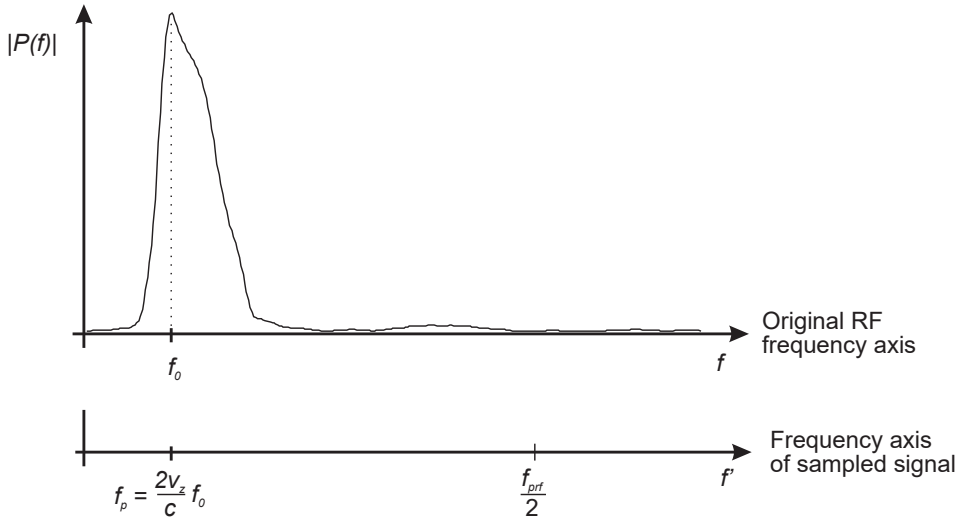



Figure 2.14: Frequency axis scaling for scatterer with velocity $v_z$ [32]

to weighing the pulse with a rectangular window, which broadens the spectrum. The spectrum of the window is expressed by eq. (2.33) and is convolved with the spectrum of the pulse. A narrow spectrum is seen for slow velocities and the resulting spectrum is nearly determined solely by the spectrum of the window.

$$W(f) = \frac{\sin(\pi f N T_{\text{prf}})}{\sin(\pi f T_{\text{prf}})} e^{(-j\pi f N T_{\text{prf}})} \tag{2.33}$$

This window limitation does not apply, as long as the whole pulse is sampled. If $NT_{\text{prf}} = {}^{cM}/_{2v_z f_0}$, the number of lines acquired and the width of the rectangular pulse are matched. A limitation on the lowest possible velocity and frequency is found through eqs. (2.34a) to (2.34c).

$$NT_{\text{prf}} = \frac{1}{\frac{2v_{\min}}{c} f_0} \tag{2.34a}$$

$$v_{\min} = \frac{c}{2} \cdot \frac{f_{\text{prf}}}{N f_0} \tag{2.34b}$$

$$f_{\min} = \frac{f_{\text{prf}}}{N} \tag{2.34c}$$

And conversely, the maximum velocity is determined by the pulse repetition frequency $f_{\text{prf}}$, as aliasing occurs at frequencies ${}^{f_{\text{prf}}}/_2$ or above. The relation of maximum velocity is expressed by eqs. (2.35a) and (2.35b).

$$\frac{f_{\text{prf}}}{2} \leq \frac{2v_{\max}}{c} f_0 \tag{2.35a}$$

$$v_{\max} = \frac{c}{2} \cdot \frac{f_{\text{prf}}}{2 f_0} \tag{2.35b}$$

However, this does not consider the pulse bandwidth. In practice, the maximum velocity $v_{\max}$ will be slightly lower.

### 2.4.2 One-Dimensional Velocity Estimation

If a pulsed sinusoidal signal, i.e. $p(t) = \cos(2\pi f_0 t)$, is emitted into an ultrasound field a number of times, the returned signal is sampled at a depth of interest, $d_0$. The sampled signal of a monochromatic wave

yields eq. (2.36).

$$r\left(k,l\right)=\cos\left(2\pi\left(\frac{f_0}{f_s}k-\frac{2v_z}{c}f_0lT_{\text{prf}}\right)\right) \tag{2.36}$$

Where $c$ is the speed of sound, $v_z$ is the blood velocity component along the axial beam, $f_0$ is the carrier frequency, $l$ is the emission times, $k$ is the sample depth, $T_{\text{prf}}$ is the time between each pulse burst, $f_s$ is the sampling frequency, and $\varphi=2\pi f_0/f_s$ is the phase factor for the depth of interest. The returned signal frequency is given by eq. (2.37) and is proportional to the axial blood velocity component.

$$\psi=-2\pi\frac{2v_zf_0}{f_{\text{prf}}c} \tag{2.37}$$

To obtain positive and negative velocities, a one-sided spectrum should be used in the signal, which can be found with a Hilbert transform of the signal [33]. The sampled signal is expressed by eqs. (2.38a) and (2.38b).

$$r_q(k,l)=\mathrm{e}^{j(\Phi k+\phi l)} \tag{2.38a}$$

$$\Phi=2\pi\frac{f_0}{f_s} \tag{2.38b}$$

### 2.4.3 Sonography

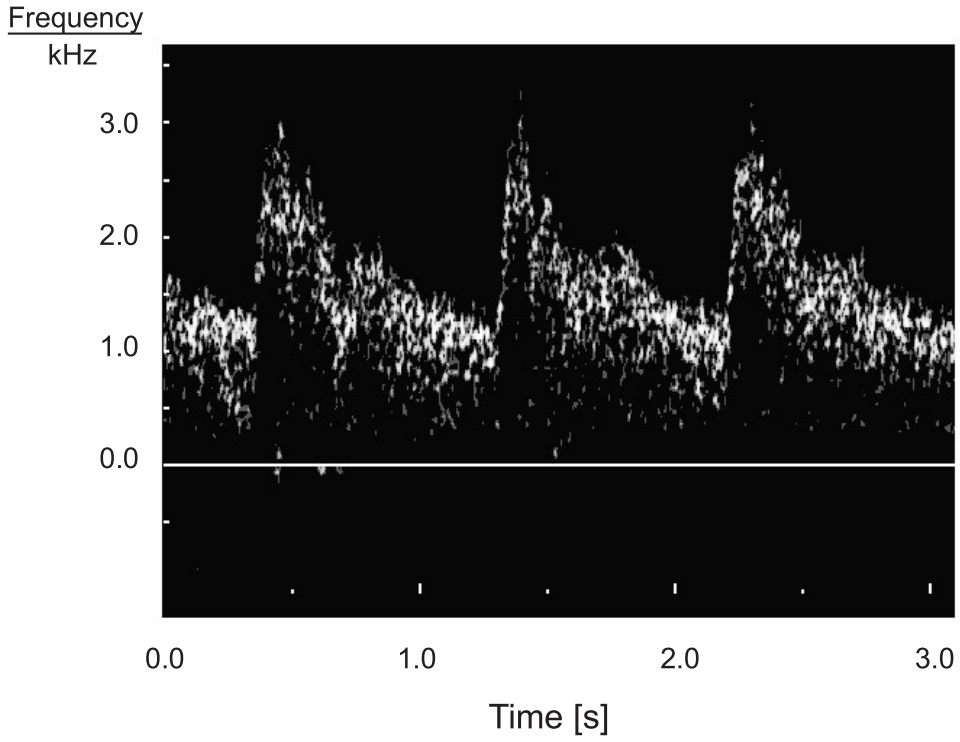



Figure 2.15: Arterial sonogram with time-frequency and Doppler shift [32]

Given that the frequency volume of the received signal is similar to the blood's velocity distribution, the Fourier transform of the received signal can be used to obtain velocity. The spectrogram, usually erroneously known as the Doppler spectrum, can be created by saving the power spectral density (*PSD*) together. The *PSD* is calculated for each of the components that make up the received signal in order to accomplish this. A quadrature-demodulated signal is used to display both positive and negative frequencies. When these spectra are shown side by side, the evolution of the velocity distribution can then be seen. Sonography of an artery is displayed in fig. 2.15.

# Chapter 3

# Synthesis

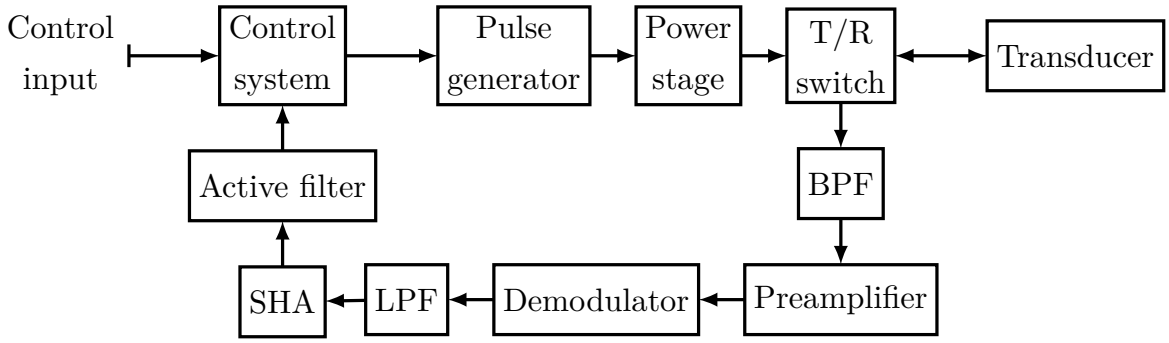



Figure 3.1: Simplified overview of the entire system

A simplified overview of the entire system can be seen in fig. 3.1. Each of the various modules will be explained during this chapter of the report. Initially, the control system will be briefly explained and the reasons for its design choice. Secondly, the signal chain in the transmitter will be outlined and how the transducer is driven by the power stage with the added protective switching circuit. Finally, the analogue front-end will be further explained with its various subcircuits for filtering, amplifying, demodulating, and sampling the signal. Lastly, the design of the digital signal processor (*DSP*) within the control system will be explained.

## 3.1 Control System

The choice of platform for controls and data acquisition is an embedded system. A microcontroller is a small computer that is built into a single integrated circuit (*IC*) chip. It includes a central processing unit (*CPU*), memory, and input/output (*I/O*) peripherals, and it is designed to perform a specific set of tasks. Microcontrollers are used in a wide range of electronic devices, including appliances, automobiles, industrial control systems, and consumer electronics. Microcontrollers are often used in applications where a small, low-power device is needed to perform simple tasks, such as controlling a motor or reading a sensor. They are usually programmed in a high-level language, such as C or C++, and they can be programmed to perform a variety of tasks, depending on the specific application. The chosen microcontroller unit (*MCU*) for this project is XNUCLEO-F411RE, visible in fig. 3.2, because it is sufficient for the application

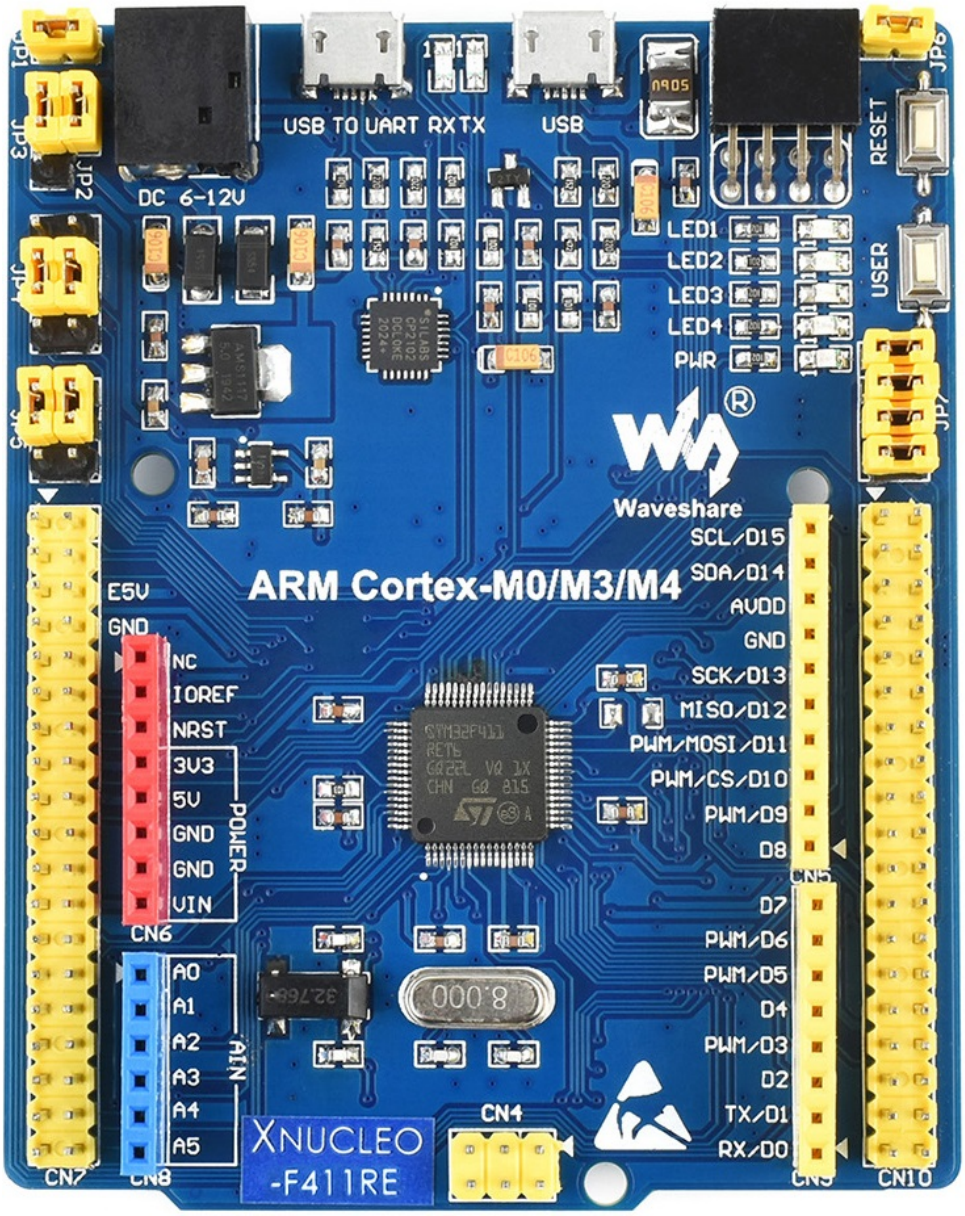



Figure 3.2: XNUCLEO-F411RE development board by Waveshare

and sourcing limitations within the *IC* supply chain. For implementing the control system, a real-time operating system (*RTOS*) can offer multiple benefits for the embedded system development. A RTOS is an operating system that is designed to handle real-time applications. Real-time applications are those that require timely processing of data in order to function correctly. This can include tasks such as controlling industrial machinery, monitoring and controlling processes. Real-time operating systems are designed to prioritize certain tasks and ensure that they are completed within a specific timeframe. They do this by allocating a certain amount of processing resources to each task, and by interrupting the execution of lower-priority tasks as needed to ensure that high-priority tasks are completed on time. RTOSs typically include features such as preemptive scheduling, real-time communication, and support for multiple processors and hardware architectures. Alternatively, a vendor-locked baremetal implementation is an option, in this case, STM32 HAL. Notable differences between the two approaches are, but not limited to:

- Multitasking: RTOS allows for parallel execution that enable more complex applications.
- Portability: Standard modules mean that the same code can be easily ported to other devices and even other platforms without modifications.
- Reduced development time: Especially for rapid prototype development, using pre-existing APIs significantly reduces development time by providing many of the low-level tasks such as scheduling, resource management, and timing by the operating system.

### 3.1.1 Pulse Generator

Initially, a pulse generator was designed by using a programmable synthesizer circuit, but due to constraints within generating complementary PWM with dead-time when driving the half-bridge, a timer based PWM generation in the microcontroller is preferable. In a half-bridge power stage, dead-time refers to the amount of time that elapses between the moment when one of the switches in the half-bridge (either the high-side or the low-side switch) turns off and the moment when the other switch turns on. During the dead-time, both switches in the half-bridge are off, which means that there is no current flowing through either switch in the half-bridge. A scenario where both switches are on, can cause problems if the output of the half-bridge is connected to a load, as it may cause the circuit to behave erratically or even be damaged. To avoid these problems, it is important to carefully consider the amount of dead-time in a half-bridge power stage. In general, a longer dead-time will reduce the risk of damage to the load, but it will also reduce the efficiency of the power stage, as energy will be lost during the dead-time. Therefore, it is important to carefully balance the trade-off between efficiency and safety in order to determine the optimal amount of dead-time for a given half-bridge power stage. Based on discussions during design review, it was decided to change approach and generate the two complementary PWM signals by configuring the PWM module of the controller with the functionality though with a trade-off in resolution. However, for this application there is no need to amplitude modulate the output signal. Seen in table 3.1, the signals being generated by the pulse generator are being described. First off, there is the 5 MHz complementary signal `PWM` and 5 MHz complementary signal `PWMN` with with dead-time for the pulsed burst during transmit mode. 15 kHz signal `PRF` for the timing control of the transmit/receive switch. 20 MHz clock signal `CLK` for the demodulation circuit in the receiver. `PULSE` and `GATE` controlled by `PRF` for S/H control with pulse length $t_{\text{pulse}} = N_{\text{pulse}} \times T_{\text{prd}}$, where $T_{\text{prd}} = {}^{1}/_{f_{\text{pwm}}}$. `GATE` is delayed from pulse by `sample depth`.

Table 3.1: Signals generated by the ultrasound pulse generator

| Name | Frequency Hz | Duty Cycle % | Dead-time |
|---|---|---|---|
| PWM | $5 \times 10^6$ | 50 | Yes |
| PWMN | $5 \times 10^6$ | 50 | Yes |
| PRF | $15 \times 10^3$ | **Pulse length dependent** | No |
| CLK | $20 \times 10^6$ | 50 | No |
| PULSE | **PRF dependent** | **Pulse length dependent** | No |
| GATE | **PRF dependent** | **Pulse length dependent** | No |

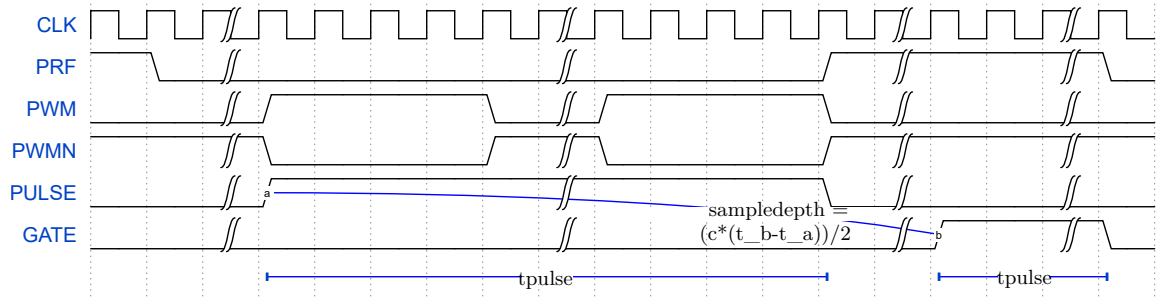



Figure 3.3: Timing diagram of various control signals for an arbitrary $n$ length pulse train expressed by the second diagram gap

By looking at fig. 3.3, we see a timing diagram of the various signals that are generated by the ultrasound pulse generator.

## 3.2 Power Stage

Several MOSFET drivers were considered, e.g. ISL55111[56], EL7104[20], and MD1213[45]. The MD1213 has an advantage over the ISL55111 or EL7104 for ultrasound MOSFET drivers since it is specifically designed for high-voltage P-channel and N-channel MOSFETs in medical ultrasound and other applications needing a high output current for a capacitive load. It has a high-speed input stage with a logic interface that can function from 1.8 V to 5 V and an ideal operating input signal range of 1.8 V to 3.3 V. The DC-coupled adaptive threshold circuit sets the level translator switch threshold to the average of the input logic LOW and logic HIGH levels. Consequentially, the MD1213 is designed primarily for driving MOSFETs in medical ultrasound applications, whereas the ISL55111 and EL7104 are more general-purpose drivers that may not perform as well in ultrasound applications. The MD1213's output stage has a distinguishing feature in that the LOW and HIGH levels of the output signal may be set independently of the rest of the circuit's supply voltages. The input logic levels, for example, might be 0 V and 1.8 V, whereas the control logic is powered by +5 V to −5 V. The output LOW and HIGH values, on the other hand, may be changed between −5 V to +5 V. This gives you greater flexibility in adjusting the output signal levels to meet individual needs. The output stage may also provide peak currents of up to 2 A, depending on the load capacitance and supply voltages employed. Seen in section 3.2 is the circuit diagram of the power stage with the gate driver on the left side and the half-bridge on the right side.

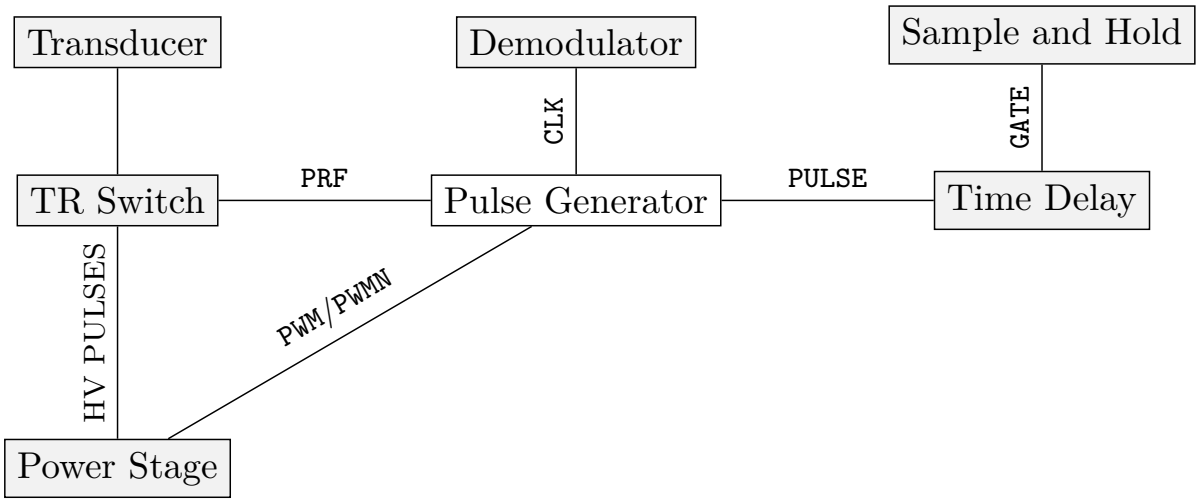



Figure 3.4: Block diagram of pulse generator signals and modules

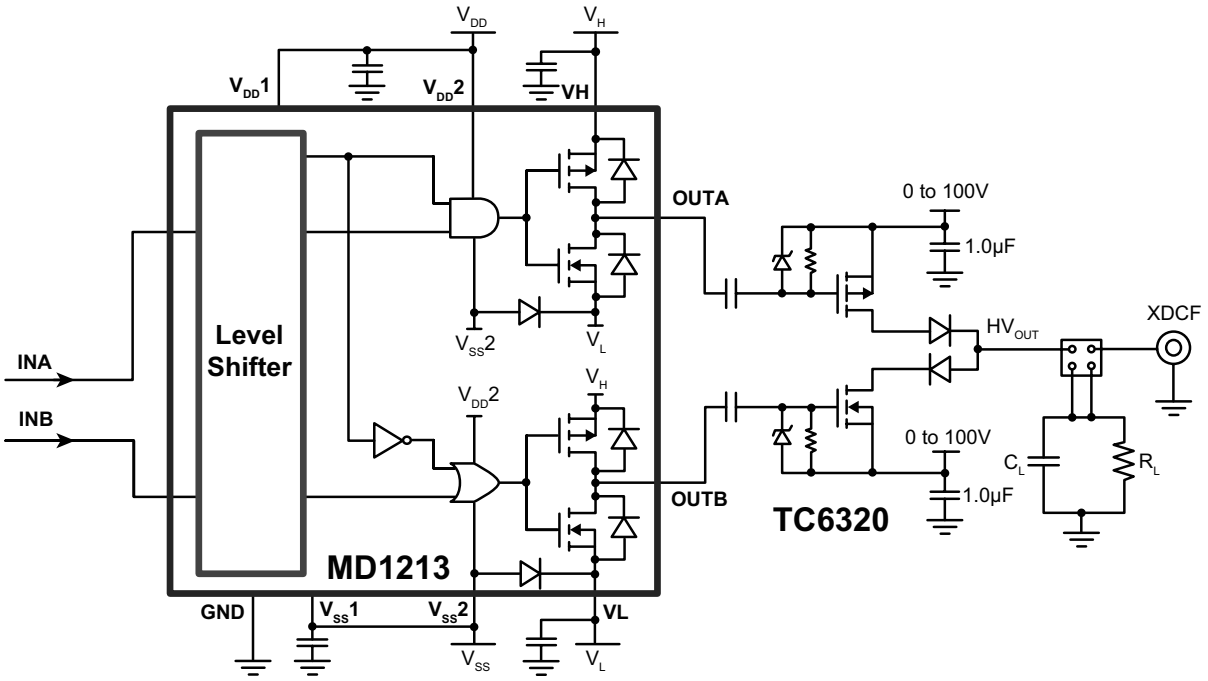



Figure 3.5: Block diagram of power stage [36]

Using a SPICE macro model, an LTspice simulation of the power stage was implemented where the full

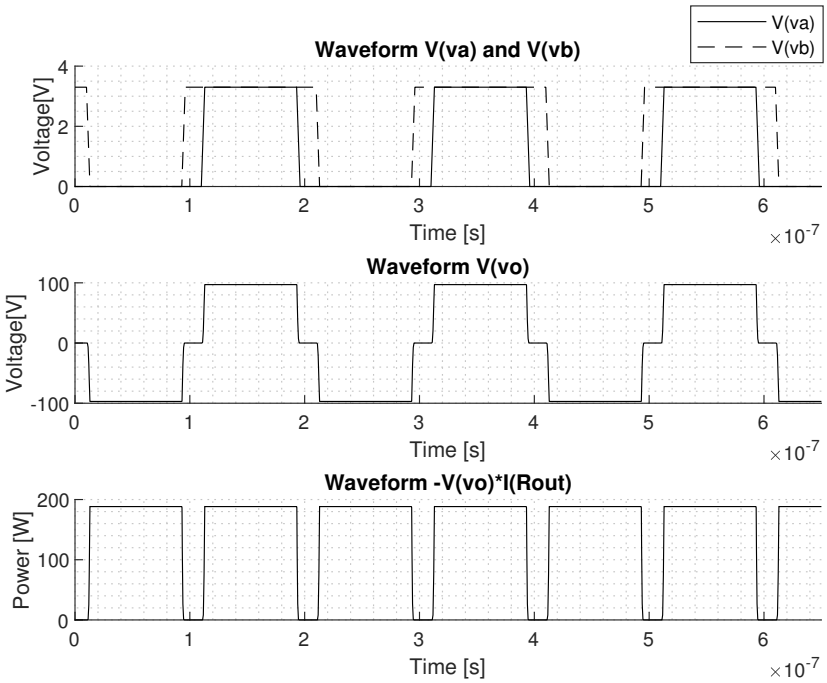



Figure 3.6: LTspice simulation output of transmitter with level shifter and half-bridge power stage from fig. C.1

model can be seen in fig. C.1. The resulting waveforms are seen in fig. 3.6. In the top subplot, the input voltages `INA` and `INB` are seen with their dead-time visible on each overlapped on period. Since `INB` is driving an N-channel metal-oxide-semiconductor field-effect transistor (*MOSFET*), the driving pulse train should be thought of as having the opposite polarity. When looking at the middle subplot, it is noted that dead time is visible as the time when the output voltage is zero. Thus, during that time neither field-effect transistor (*FET*) are allowing a current to pass, and therefore the voltage across the load is equal to zero. The lower subplot shows the maximum ideal power delivery using the peak pulse voltage, assuming the load is equal to $50\,\Omega$. In reality, due to the laboratory instruments available for experiments, the pulse peak voltage will be less than $\pm 100\,\mathrm{V}$.

## 3.3 Transmit/Receive Switch

Among the design considerations for the transmit and receive switch were the TX810[25] and MD0101[53]. Both ICs are acceptable choices, however, the MD0101 is a newer and generally better choice since it has a lower insertion loss, which means that less of the ultrasound signal is lost as it passes through the switch. This results in a higher-quality image with a better signal-to-noise ratio. Additionally, MD0101 has a wider bandwidth, which means that it can transmit and receive ultrasound signals over a broader range of frequencies. However, since the TX810 is in stock and is also acceptable, it was chosen for the design. TX810 is an IC from Texas Instruments that can be used to switch transmit and receive paths of an ultrasound system. The IC fundamentally works by having a 3-bit programmable pin interface that will open and close the switch with a variable bias current. See fig. 3.8 for a visualisation of the switching operation, where `INPUT` is the incoming Doppler waveform being picked up from the transducer,

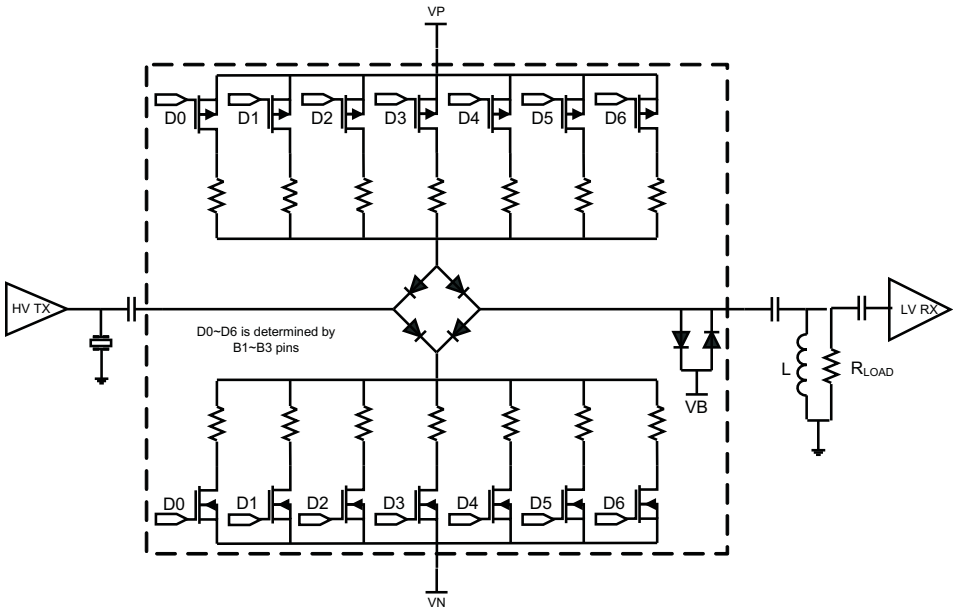



Figure 3.7: Block diagram of TX/RX switching circuit where the inputs D1 through D6 are binary decoded from inputs $B_1$, $B_2$, $B_3$ [25]

`B3/B2/B1` is the switching signal closing the switch and thereby going in receive mode, and `OUTPUT` is the received signal seen in the analogue front end (*AFE*). When high-voltage transmitter signals are applied to the input, the internal diodes limit the output voltage. While in receive mode, the TX810's insertion loss is minimized. The TX810 features a 3-bit interface that may be used to program bias current from 7 mA to 0 mA for varying performance and power requirements, unlike conventional T/R switches. The device is put up in power-down mode when the TX810 bias current is set to 0 mA (high-impedance mode). The TX810 does not put a significant load on high-voltage transmitters when operating in the high-impedance mode.

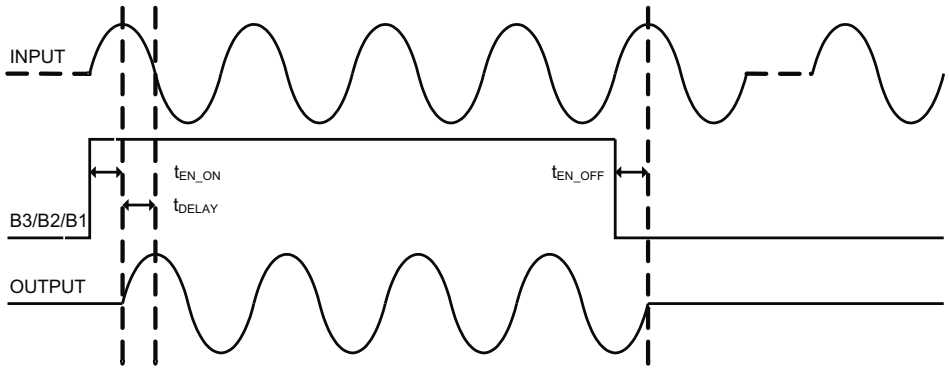



Figure 3.8: Timing diagram of switching interface where $t_{\text{EN\_ON}} = 0.6\,\mu\text{s}$, $t_{\text{EN\_OFF}} = 2.4\,\mu\text{s}$, and $t_{\text{DELAY}} = 1.3\,\text{ns}$ for the condition $B_1 = B_2 = B_3$ [25]

Seen in fig. 3.9 is the recovery time between transient states in the TX810 diode bridge where the red curve is the input and blue is the output signal. According to the specified output curve, it takes approximately 15 µs to recover from the transmitting pulse. Thus, back-calculating to find the minimum distance to

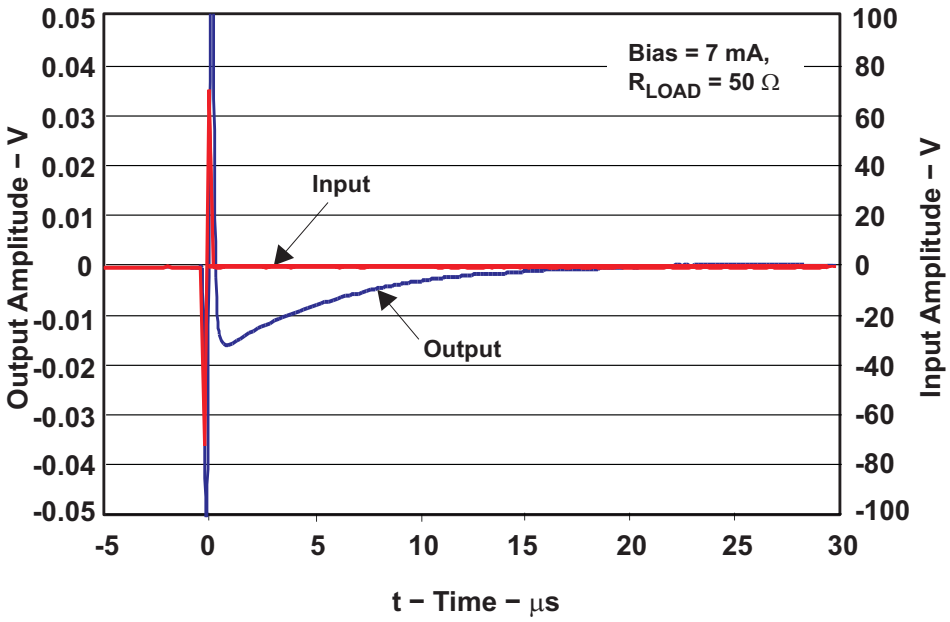



Figure 3.9: Recovery time from transmitting to receiving state with an AC coupled high-voltage pulse and TX810 [25]

reliably receive an undisturbed signal is proportional to eq. (3.1).

$$d = \frac{c \cdot t}{2} \implies \tag{3.1a}$$

$$d = \frac{1480\,\mathrm{m\,s^{-1}} \cdot 15\,\mu\mathrm{s}}{2} = 11.55\,\mathrm{mm} \tag{3.1b}$$

Where $d$ is equal to the travel distance from the transducer to the scatterer, $c$ is the speed of sound in water (assumed to be $1480\,\mathrm{m\,s^{-1}}$), and $t$ is the recovery time as specified in fig. 3.9. The reason for dividing the distance by half is because the travel time is double the distance since the acoustic wave has to travel the distance from the transducer and the reflected wave has to travel back the equal distance. This means that a distance of less than 11.55 mm to a scatterer is likely to produce an unreliable measurement. The ultrasound switch is designed to switch the transmit and receive paths at specific times, as determined by the input signals. A PCB design was implemented in Altium Designer [60] utilising three channels of the maximum eight available channels in the IC. Seen in fig. 3.10 is a 3D render of the designed PCB.

The module is designed with three usable channels, either three separate transducers for multi-angle sonography, or a capacitive micromachined ultrasound transducer (*CMUT*) with three channels in a single angle. However, in the following experiments with the TX/RX switch, only one channel will be used for simplifying the data acquisition experiments.

## 3.4 Band-pass Filter

After the signal is received, it is filtered with a band-pass (*BP*) filter to remove unwanted noise and interference from the received signal. The presence of these unwanted frequency components can distort the received signal and reduce the quality of the resulting imaging. The specs from the datasheet of used modular component BPF-C4R5+ [48] are seen in fig. 3.11. Filtering the received signal through this device means that any signals produced by the transducer at frequencies outside the range of interest will

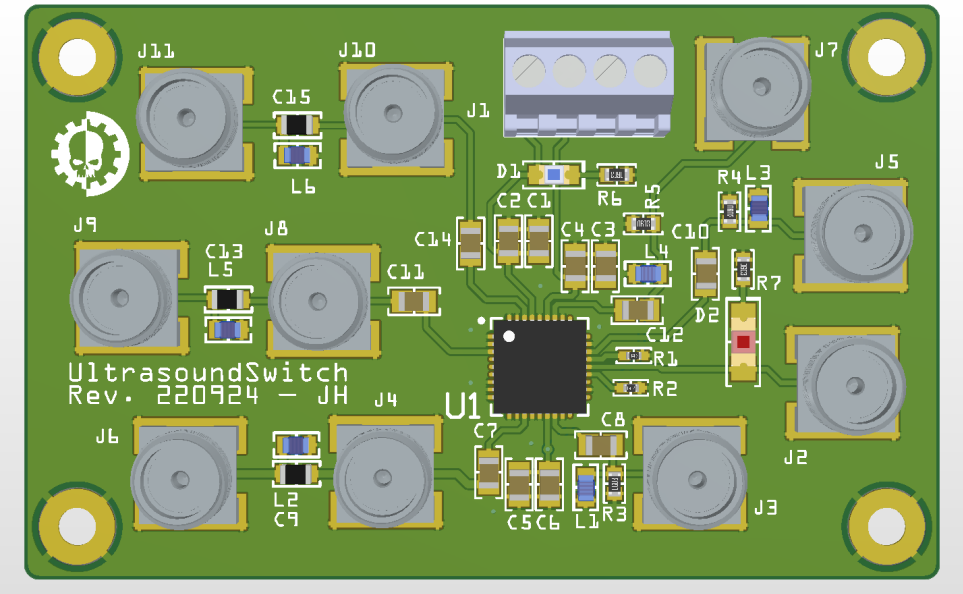
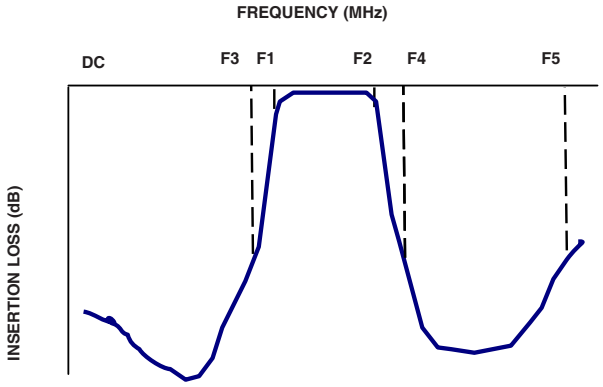



Figure 3.10: 3D Render of PCB in Altium Designer




**Electrical Specifications at 25°C**

| Parameter | | F# | Frequency (MHz) | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| **Pass Band** | Center Frequency | — | — | - | 4.5 | - | MHz |
| | Insertion Loss | F1-F2 | 2-7 | - | 0.5 | 1.5 | dB |
| | VSWR | F1-F2 | 2-7 | - | 1.1 | 1.5 | :1 |
| **Stop Band, Lower** | Insertion Loss | DC-F3 | DC-0.6 | 20 | 35.9 | - | dB |
| | VSWR | DC-F3 | DC-0.6 | - | 20 | - | :1 |
| **Stop Band, Upper** | Insertion Loss | F4-F5 | 17-2100 | 20 | 28.9 | - | dB |
| | VSWR | F4-F5 | 17-2100 | - | 20 | - | :1 |

Figure 3.11: Band-Pass Filter with (above) insertion loss showing a pass band of 2 MHz to 7 MHz of 0.5 dB insertion loss and (below) electrical specifications showing the minimum stopband attenuation of 20 dB [48]

be attenuated. An ultrasound receiver requires a band-pass filter to enable only the frequencies within a predetermined range to pass through while blocking out frequencies outside that range. Using the specs and S21 forward transmission coefficient data, a bode plot can be visualised in fig. 3.12.

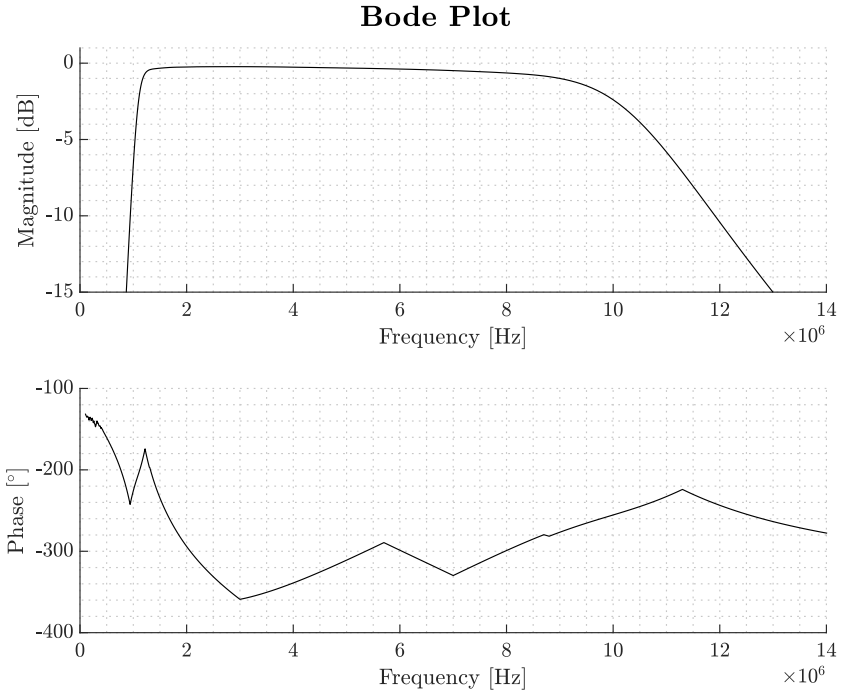



Figure 3.12: Bode plot of BPF-C4R5+ S21 forward transmission coefficient of the active filter as specified by the manufacturer

## 3.5 Preamplifier

The isolated signal is still rather weak to be measured using digital circuits, and therefore the amplitude must be increased with the preamplifier circuit. This circuit is based on the integrated circuit from Analog Devices AD8332 [41], which is a device that combines a dual-channel low noise amplifier (*LNA*) and variable gain amplifier (*VGA*), designed specifically for ultrasound systems. A diagram of its internal functional blocks can be seen in fig. 3.13. The AD8332 functions at frequencies up to 120 MHz. Each channel includes an ultralow noise preamp (*LNA*), a *VGA* with 48 dB of gain range, and a selectable gain post amp with adjustable output limiting. The LNA gain is 19 dB with a single-ended input and differential outputs. To match the signal source without sacrificing noise performance, the LNA input impedance can be adjusted using a single resistor. The VGA has low output-referred noise, which is useful in driving high-speed differential ADCs. The gain of the post amp can be pin-selected to 3.5 dB or 15.5 dB, depending on the converter requirements. The output can be limited to a user-defined clamping level to avoid input overload to a subsequent analogue-to-digital converter (*ADC*), with the clamping level adjusted using an external resistor. A SPICE macro model is provided by the vendor and the preamplification is successfully simulated using LTspice with the full LTspice model found in fig. C.2, and the probed inputs and outputs seen in fig. 3.14.

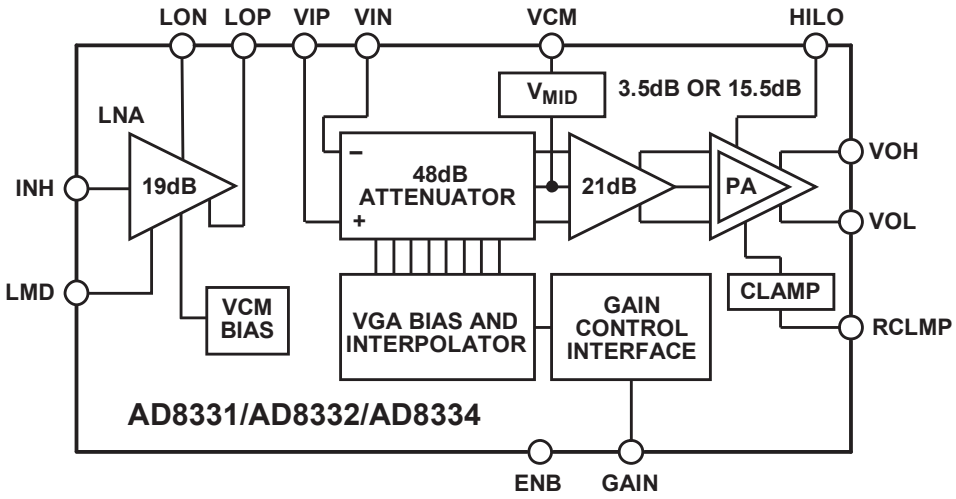



Figure 3.13: Block diagram of preamplifier AD8332 [41]

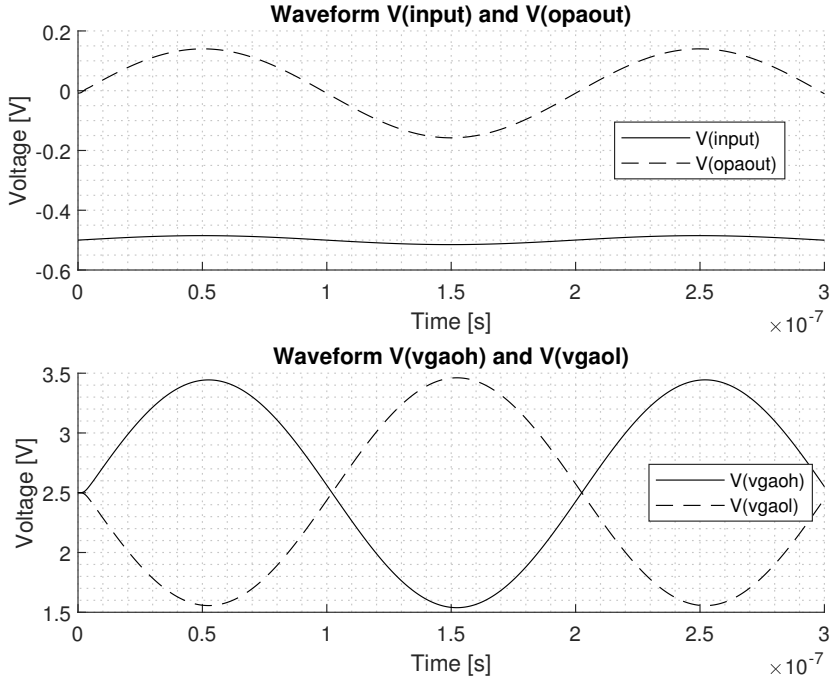



Figure 3.14: LTspice simulation output of preamplifier *LNA* and *VGA* from fig. C.2

## 3.6 Quadrature Demodulator

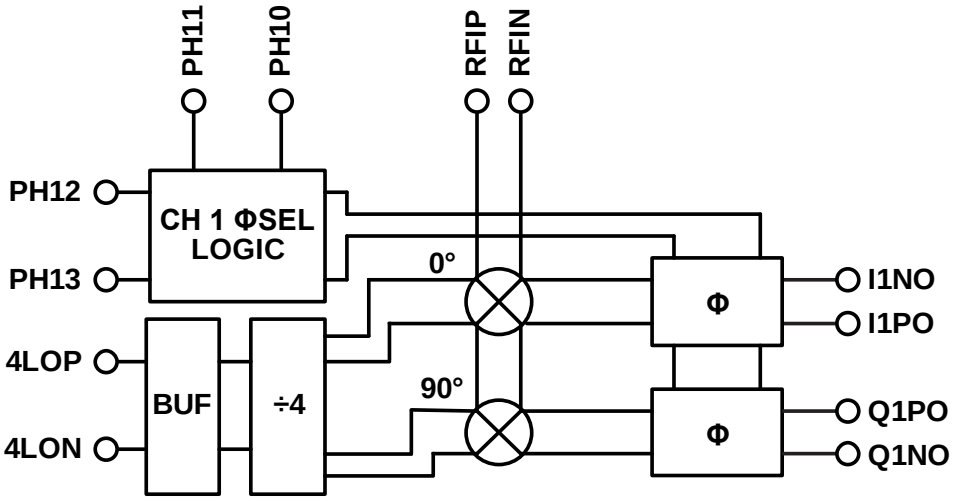



Figure 3.15: Block diagram of demodulator AD8333 [42]

After the preamplifier, the amplified signal must be demodulated to prepare it for sampling. The device used for quadrature demodulation is an integrated circuit from Analog Devices AD8333 [42] I/Q demodulator. A diagram of the internal functional blocks can be seen in fig. 3.15 where the primary inputs are `RFIP` and `RFIN`, which are the two differential RF signals from the preamplifier. The RF inputs connect directly to the outputs of the LNA of the preamplifier. The internal 0° and 90° phases of the local oscillator (LO) are generated by a divide-by-4 circuit that drives the mixers of a matched I/Q demodulator pair. The I and Q outputs are presented as currents, making summation possible. The summed current outputs are then converted to voltages by a high dynamic range, current-to-voltage (I-V) converter, such as the AD8021 [18], which functions as a trans-impedance amplifier. A SPICE macro model is provided by the vendor and the I/Q demodulation is successfully simulated using LTspice using the LTspice model found in appendix fig. C.3, with the probed inputs and outputs seen in figs. 3.16 and 3.17.

## 3.7 Sample and Hold

In this system, the sample-and-hold amplifier is necessary to keep values between each sample line. In chapter 2, it was described how the pulsed-wave flow-meter measures the movement of scatterers by sampling the back-scattered signal at a specific depth. Generally, the intended use for this part is in general data acquisition systems such as an *ADC*. In that application, the sample-and-hold amplifier captures an analogue signal and retains it during certain operations, usually analogue-to-digital conversion. Through a S/H input, two possible modes are selected, *sample* or *hold*. During the sample mode of operation, the output of the sample-and-hold amplifier follows the input. During the hold mode of operation, the output may not change by more than 1 least significant bit (*LSB*). The typical usage of a SHA is to keep the ADC input constant throughout the conversion process. With some types of ADCs, but not all, the input cannot change by more than 1 *LSB* during the conversion, or else the process will be compromised. This can either impose very low-frequency limits on such ADCs or necessitate their use with a SHA to hold the input during each conversion. An internal capacitor forms the key component of

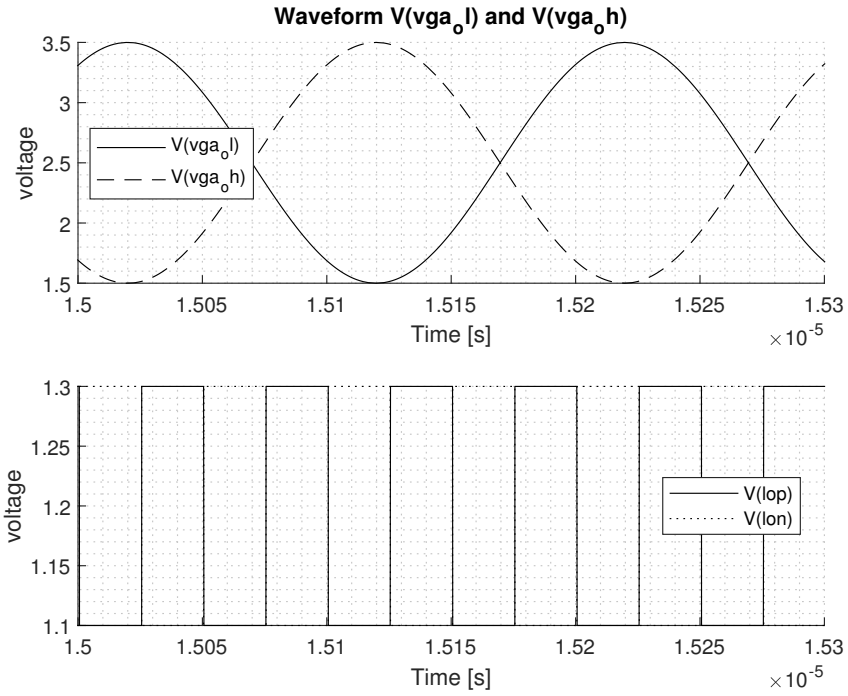



Figure 3.16: LTspice simulation demodulator input variables from fig. C.3

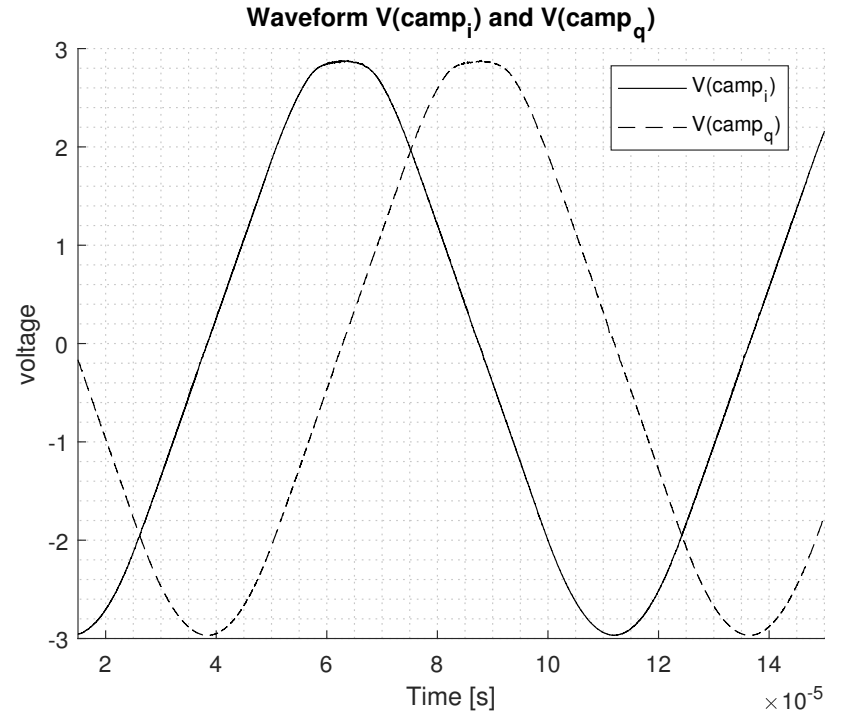



Figure 3.17: LTspice simulation demodulator output variables Q and I voltages from fig. C.3

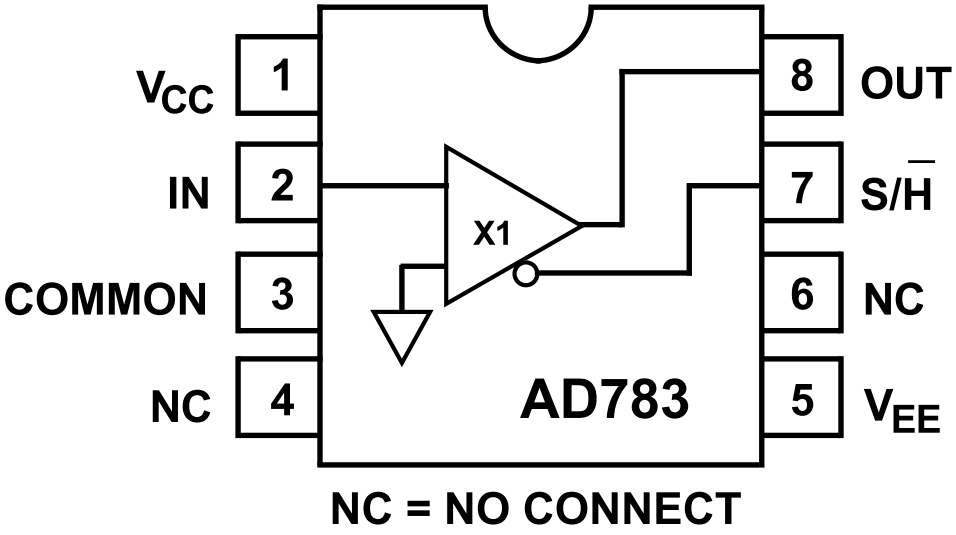



Figure 3.18: AD783 Sample and Hold Amplifier functional block diagram [34]

the sample-and-hold amplifier, which serves as the energy storage device. The input amplifier buffers the input signal by presenting a high impedance to the signal source while providing current gain to charge the hold capacitor. In the sample mode, the voltage on the hold capacitor follows the input signal, albeit with some delay and bandwidth limitations. In the hold mode, the switch is opened, and the capacitor retains the voltage present before being disconnected from the input buffer. The output buffer prevents the held voltage from discharging too soon by offering a high impedance to the hold capacitor. The switching circuit and its driver work together to enable the SHA to alternate between sample and hold modes.

In the pulsed-wave flowmeter, the sample-and-hold amplifier is used to keep each sample value between each gate pulse. This is done for both the I and Q signals in parallel. A diagram of a sample-and-hold operation can be seen in fig. 3.19.

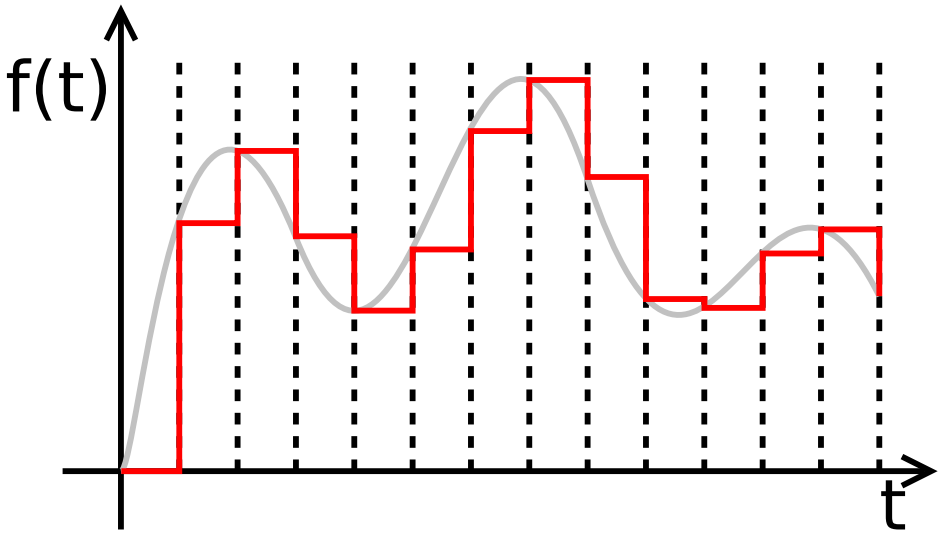
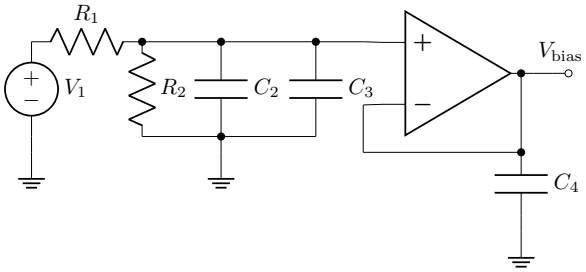
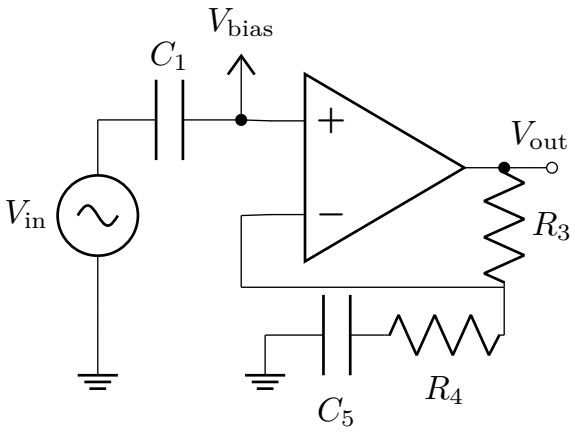



Figure 3.19: Sample and Hold function with input function $f(t)$ over time [66]

## 3.8 Active Filter




(a) Circuit diagram of DC bias circuit




(b) Circuit diagram of high-pass (*HP*), pulse repetition frequency (*PRF*) and Wall filter

Finally, the signal should be DC coupled for sampling and filtered using an active *HP* filter to remove unwanted PRF frequency and wall frequency. For generating a DC bias voltage, the circuit in fig. 3.20a is used. Here, a voltage divider is coupled with some stabilising capacitors and a voltage follower to output a stable output. In the circuit diagram in fig. 3.20b, a *HP* filter is used as a combined *PRF* and Wall-filter in conjunction with a 2 dB gain amplification to maximize the dynamic range of the *ADC* measurements from 0 V to 3.3 V. In the diagram, $V_{bias} = 1.65\,\text{V}$, which is half of the *ADC* dynamic range of 0 V to 3.3 V, and $v_{\text{in}}$ is the input signal to the filter from the output of the sample-and-hold amplifier. To get a 1.65 V DC-bias, a voltage-follower op-amp configuration is used with a voltage divider input on the supply voltage of 5 V, outlined in eq. (3.2).

$$V_{\text{dc}} = V_{\text{cc}} \cdot \frac{R_2}{R_1 + R_2} = 5\,\text{V} \cdot \frac{5\,\text{k}\Omega}{10\,\text{k}\Omega + 5\,\text{k}\Omega} = 1.66\,\text{V} \tag{3.2}$$

For the 2 dB gain, a non-inverting amplifier configuration is used, expressed by eq. (3.3).

$$G_{\text{dB}} = \frac{V_{\text{out}}}{V_{\text{in}}} = 20 \cdot \log_{10}\left(1 + \frac{R_3}{R_4}\right) = 20 \cdot \log_{10}\left(1 + \frac{40\,\text{k}\Omega}{150\,\text{k}\Omega}\right) = 2.05\,\text{dB} \tag{3.3}$$

A SPICE simulation was implemented and can be seen in fig. C.4. From this simulation model, a small signal analysis as well as a transient analysis was conducted. The resulting small signal analysis can be seen in fig. 3.21. The transient analysis can be seen in fig. 3.22 and confirms the expected result of a DC-coupling and $\approx 2\,\text{dB}$ amplification of the input signal.

## 3.9 Digital Signal Processor

In the *DSP* system, the function is to turn a waveform into a humanly readable velocity metric by performing a Fourier analysis of the input signal captured from the output of the active filter of the previous section to determine the dominant frequency components. Since DSP devices are programmable direct current (*DC*) devices of typically 3.3 V to 5 V, it is vital that the input signal is *DC*-coupled. Since the input signal is relatively low frequency, most *ADC* interfaces should be sufficient to capture a frequency window of less than 10 kHz. Several solutions were considered for the design, among them,

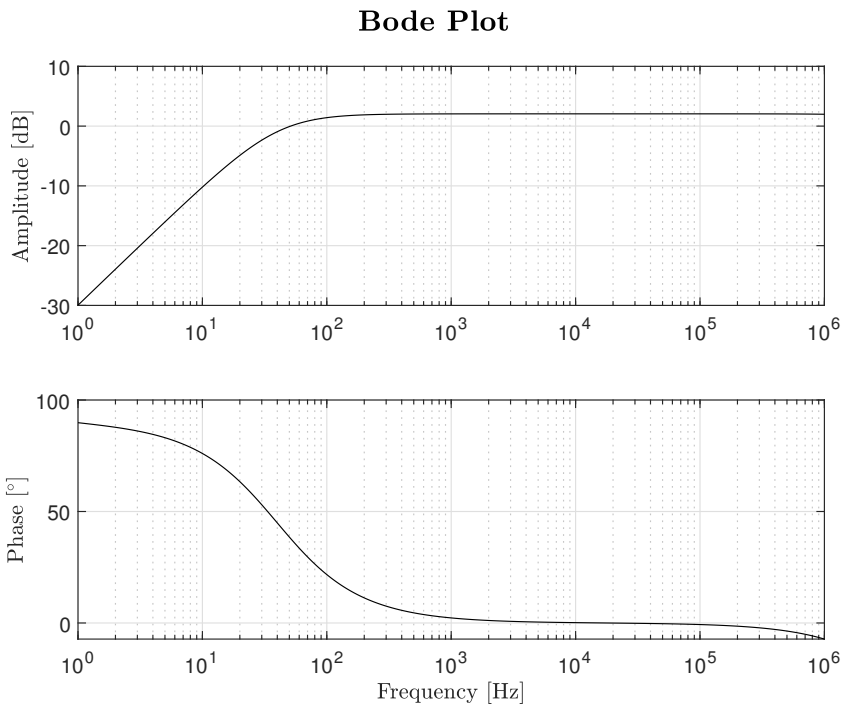



Figure 3.21: Small-signal analysis of DC-Coupling filter circuit

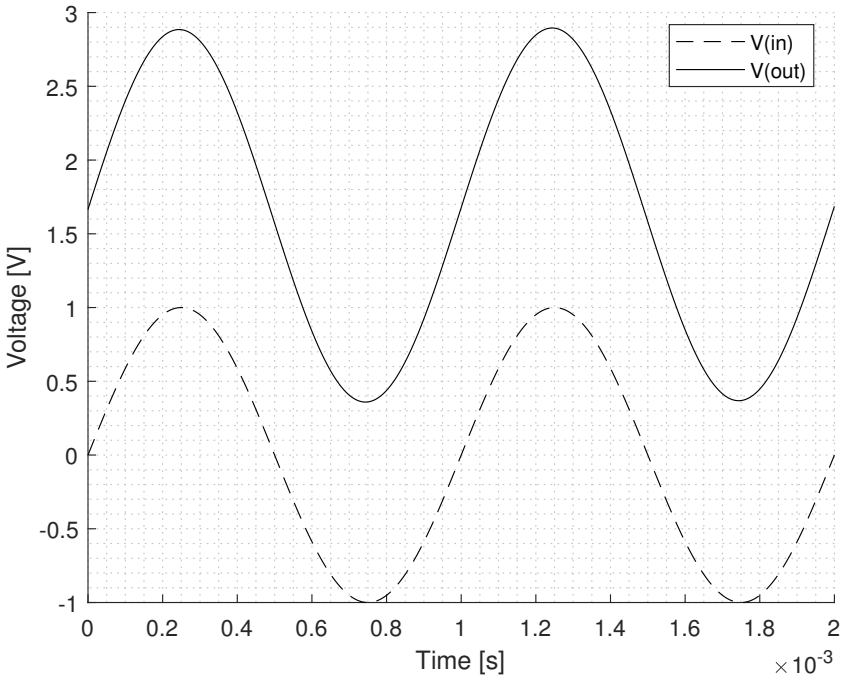



Figure 3.22: Transient analysis of DC-Coupling filter circuit

a manual analysis using a readout on an oscilloscope to calculate the flow velocity. Other than that, online flow velocity analysis options such as an *MCU* is also considered. Principally, what needs to be done in this module is that the peak frequency component must be determined. Manually this can be done by visually inspecting the waveform to determine the frequency, but ideally it is done through automatically determining the frequency by performing Fourier analysis. After identifying the peak frequency component is identified, using the experimental variables such as incident angle $\theta$, speed of sound $c$, transmitted frequency $f_0$, and finally the measured Doppler frequency $f_d$, the velocity $v$ can be obtained using the methods described in the previous chapter. The practical implementation of this will be elaborated in the following chapter.

# Chapter 4

# Implementation

In this chapter, the steps involved in turning a theoretical design into a tangible system will be outlined. Since the synthesis chapter dealt with an explanation of the functions of each module and simulations, with a subsequent evaluation of the outcomes, this chapter will focus on the creation of physical hardware implementations and reproducing the expected results.

## 4.1 Control System

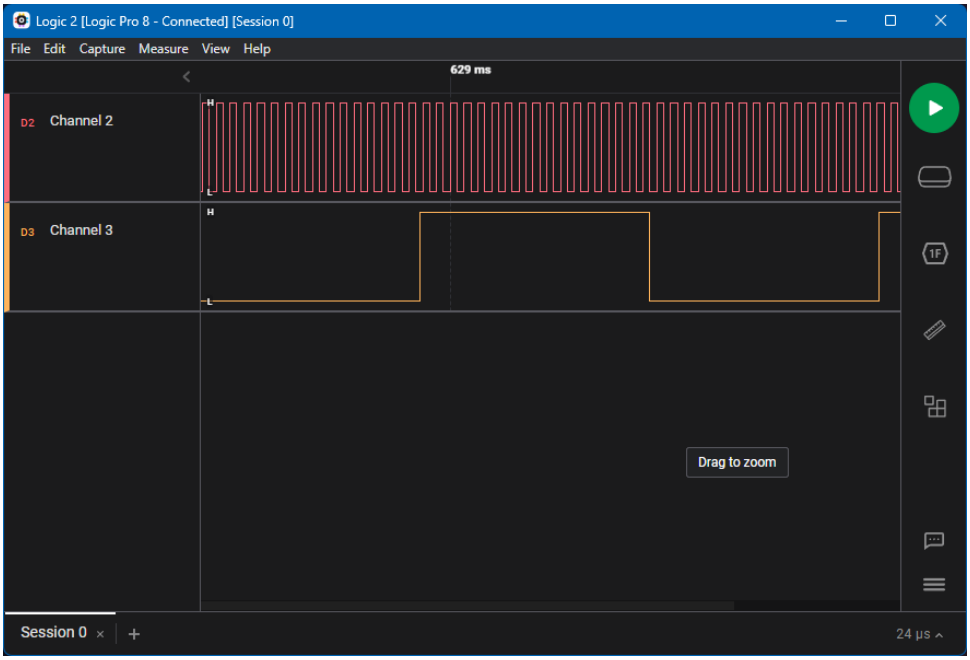



Figure 4.1: STM32 Zephyr RTOS pulser output

During the initial implementation stage of the control system, a bring-up of the development board and several pulse signals were succesfully done. In fig. 4.1, two pulse signals can be seen. The experimental microcontroller PWM generator code can be seen in listing B.1. In Channel 2, 5 MHz ultrasound pulse can be seen. In Channel 3, the 10 kHz PRF signal can be seen. Unfortunately, soon thereafter it was discovered a limitation of the API in Zephyr is not mature enough developed for power systems such

as the half-bridge in the transmitter circuit. In more practical terms, it was not possible to generate two complementary signals with dead-time using the existing Zephyr PWM API. To continue with that solution, a new PWM driver would have to be written from scratch, which is no trivial task. Alternative solutions were investigated. Another option was to use the hardware abstraction layer (*HAL*) provided by the manufacturer of the microcontroller, but this would also mean an increased amount of development time for the control system since the *HAL* is rudimentary in implementation and has little abstraction. However, after finding inspiration [54], it was decided to try the alternative system PYNQ-Z1, which is a development board by Digilent. On the PYNQ Z1 board is a Zynq 7000 system-on-a-chip (*SoC*). Inside the Zynq 7000 SoC there are both an field-programmable gate array (*FPGA*) and Arm based processor. PYNQ is an open-source framework that runs on Xilinx compute platforms where higher levels of abstraction enable faster productivity.

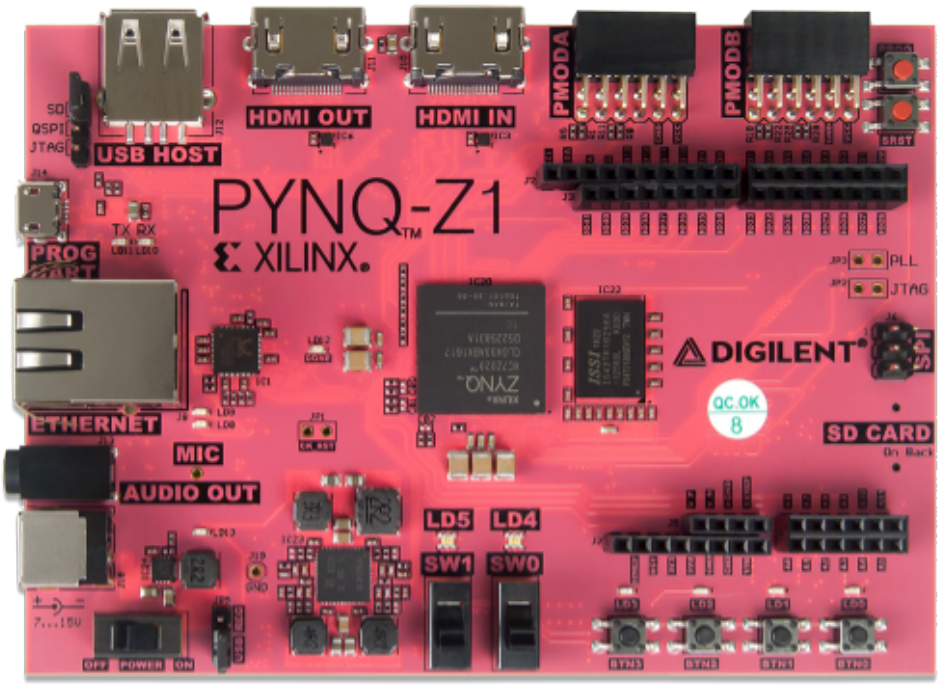



Figure 4.2: PYNQ-Z1 development board

Seen in fig. 4.2 is the development board with its peripherals. The development of a prototype pulser system will be done by implementating an FPGA project in *VHSIC* hardware description language (*VHDL*) using Xilinx Vivado integrated development environment (*IDE*) and then generating the bitstream. After this, the FPGA artifacts are generated as a `.bit` and `.hwh` file. These two files are used in the JupyterLab environment as an overlay to configure the logic of the FPGA and output signals on the PMOD-A connector.

On the PYNQ-Z1 board, the PMOD ports are 12-pin female connectors with 0.1 inch spacing that connect to normal 12-pin headers. As illustrated in fig. 4.3, each 12-pin PMOD port offers two 3.3V VCC signals (pins 6 and 12), two GND signals (pins 5 and 11), and eight logic signals. Each PMOD port on a PYNQ board is classified as normal, MIO linked, XADC, or high-speed. The PYNQ-Z1 features two PMOD ports, both of which are high-speed. For maximal switching rates, the High-speed PMOD ports route their data signals as impedance matched differential pairs. For further protection, they feature pads for loading resistors, however the PYNQ-Z1 is assembled with these loaded as 0 Ω shunts. With the

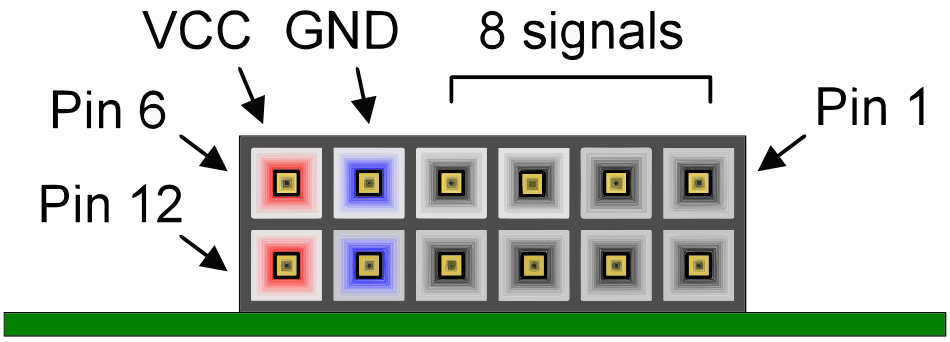



Figure 4.3: PYNQ Z1 PMOD port diagram [46]

series resistors shunted, these PMOD ports provide no short-circuit protection but allow for substantially quicker switching rates. Pins 1 and 2, pins 3 and 4, pins 7 and 8, and pins 9 and 10 are coupled to neighboring signals in the same row. Traces are routed in a 100 Ω (±10 %) differential configuration. In this application however, the pins will be used in a single-ended configuration.

```
1  set_property IOSTANDARD LVCMOS33 [get_ports PULSE]
2  set_property IOSTANDARD LVCMOS33 [get_ports PWM]
3  set_property IOSTANDARD LVCMOS33 [get_ports PWMN]
4  set_property IOSTANDARD LVCMOS33 [get_ports GATE]
5  set_property IOSTANDARD LVCMOS33 [get_ports PRF]
6  set_property IOSTANDARD LVCMOS33 [get_ports CLK]
7  set_property PACKAGE_PIN Y16 [get_ports PULSE]
8  set_property PACKAGE_PIN Y18 [get_ports PWM]
9  set_property PACKAGE_PIN Y19 [get_ports PWMN]
10 set_property PACKAGE_PIN Y17 [get_ports GATE]
11 set_property PACKAGE_PIN U18 [get_ports PRF]
12 set_property PACKAGE_PIN U19 [get_ports CLK]
13
14 set_property IOSTANDARD LVCMOS33 [get_ports {LEDs[3]}]
15 set_property IOSTANDARD LVCMOS33 [get_ports {LEDs[2]}]
16 set_property IOSTANDARD LVCMOS33 [get_ports {LEDs[1]}]
17 set_property IOSTANDARD LVCMOS33 [get_ports {LEDs[0]}]
18 set_property PACKAGE_PIN R14 [get_ports {LEDs[0]}]
19 set_property PACKAGE_PIN P14 [get_ports {LEDs[1]}]
20 set_property PACKAGE_PIN N16 [get_ports {LEDs[2]}]
21 set_property PACKAGE_PIN M14 [get_ports {LEDs[3]}]
```

Listing 4.1: Constraints on Pulse Generator and Control System

When programming an FPGA with software like as Xilinx's Vivado, it becomes necessary to inform the system which physical pins on the FPGA correspond to the FPGA ports defined in the VHDL code. This is quite similar to putting a register high or low on a microcontroller to turn an LED on or off, operate a clock, or function as a data line. However, with a microcontroller, many of these pins are "hard-wired" in the sense that they cannot be relocated to a physically different pin on the microcontroller. In general, it is not an option. This is not the case with an FPGA; instead, the hardware interface is established in VHDL and then the appropriate inputs and outputs on that interface are constrained to whichever pins

on the FPGA are required, making FPGAs incredibly versatile for complicated and bespoke designs. In listing 4.1, the constraints can be seen setting the port names to a certain `IOSTANDARD` and voltage level and the pin name `PACKAGE_PIN`. All the described pins are input/output (*I/O*) available on the PMOD port of the PYNQ-Z1.

Based on inspiration found on tutorials for PWM generators on PYNQ platforms [54], [61], a prototype of an ultrasound pulser is developed by implementing a pulse-width modulation (*PWM*) generator for the complementary PWM output and a signal controller to control pulse timings. A block diagram of the ultrasound pulser system can be seen in fig. 4.5. After that, an interface is developed to enable the control system to take input from a programmable Arm processor.

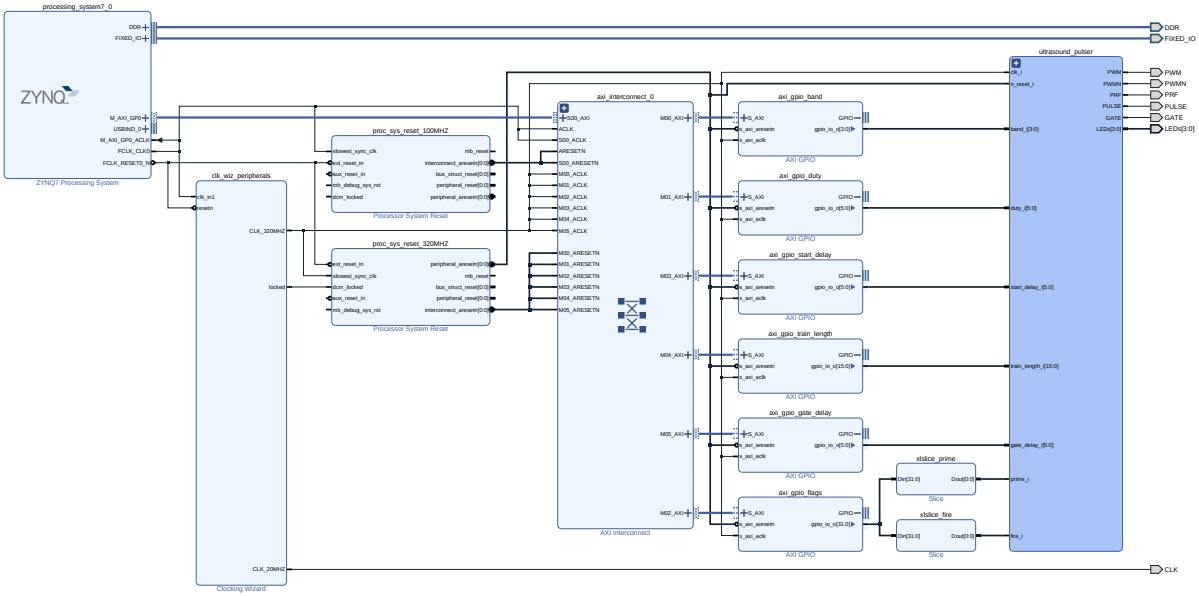



Figure 4.4: Top level block diagram of the FPGA overlay with AXI interconnects and registers

In fig. 4.4, the block diagram of the entire FPGA system is seen with its AXI interconnects that make up the interface between the ultrasound pulser and the Arm microprocessor taking inputs from the user. Also visible is the clock generator, setting the 20 MHz demodulation clock in the analogue front-end and the 320 MHz pulser clock responsible for the resolution of the dead-time and length of pulses.

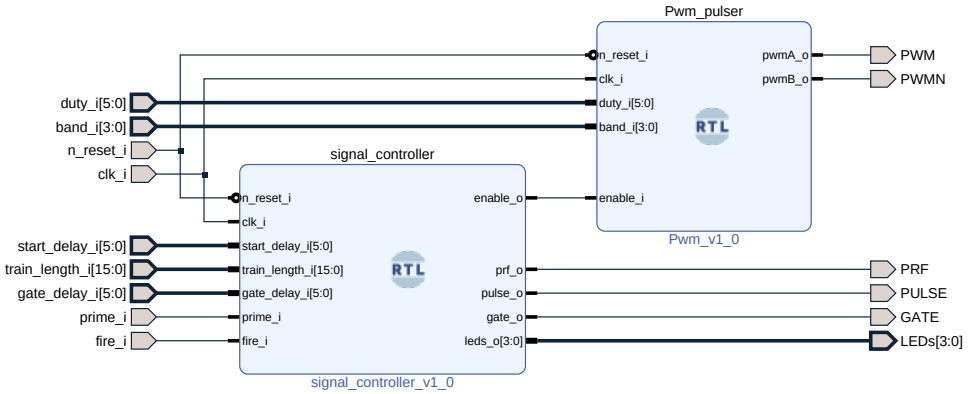



Figure 4.5: PWM pulser and signal controller as a block diagram with inputs and outputs

In fig. 4.5, the block diagram of the ultrasound pulse generator can be seen with its two sub-components, the signal controller and the PWM generator. The primary purpose of the PWM generator is the control of the two complementary PWM outputs and setting the dead-time between them. Its operation is inherently controlled through the enable input set from the signal controller. The signal controller is responsible for setting the outputs depending on the inputs received and its current state.

```
architecture arch of Pwm is
        signal timer_r       : natural range 0 to 2**duty_i'length-1;
        begin
                clocked: process(clk_i)
                begin
                        if rising_edge(clk_i) then
                        -- sync reset
                        if n_reset_i = '0' then
                                pwmA_o   <= LO;
                                pwmB_o   <= LO;
                                timer_r <= 0;
                        else
                        -- timer
                                timer_r <= timer_r + 1;
                                pwmA_o   <= LO;
                                pwmB_o   <= LO;
                                if enable_i = '0' then
                                        pwmB_o   <= HI;
                                        timer_r <= 0;
                                else
                                        -- output a
                                        if timer_r <= unsigned(duty_i) and timer_r >=
                                        ↪  unsigned(band_i)  then
                                                pwmA_o <= HI;
                                        end if;
                                        -- output b
                                        if timer_r > to_integer(unsigned(band_i)) +
                                        ↪  to_integer(unsigned(duty_i)) then
                                                pwmB_o <= HI;
                                        end if;
                                end if; -- enable
                        end if; -- sync reset
                end if; -- rising_edge
        end process clocked;
end architecture;
```

Listing 4.2: Snippet of VHDL code from pulse generator

Looking at listing 4.2, the code is run as a clocked process on the 320 MHz clock. As long as `enable` is HIGH, the pulser will output pulses `PWM` and `PWMN`. If the `enable` input is LOW, the pulse generator makes sure that `PWMN` is HIGH during the time pulses are not transmitted. If the `enable` input is HIGH, a timer controls the output level of `PWM` and `PWMN` and creates the pulses.

In listing 4.3, the state machine code of the signal controller is seen. The signal controller incorporates a few flags that are set when triggering the ultrasound pulser. The signal controller begins in the *ready*

```
case State is
        when ready =>
        if prime_i = '0' then -- check if caller prime bit reset
                if fire_i = '1' then -- then check fire bit to start
                        counter <= 0;
                        state <= delay;
                end if;
        end if;
when delay =>
        prf_o <= '0';
        if counter >= to_integer(unsigned(start_delay_i)) - 1 then
                counter <= 0;
                enable_o <= '1';
                state <= pulse;
        end if;
when pulse =>
        enable_o <= '1';
        prf_o <= '0'; -- sync with last edge of PWM
        if counter < to_integer(unsigned(train_length_i)) - 1 then
                pulse_o <= '1';
        elsif counter = to_integer(unsigned(train_length_i)) - 1 then
                pulse_o <= '1';
                counter <= 0;
                state <= gate_delay;
        end if;
when gate_delay =>
        if counter >= to_integer(unsigned(gate_delay_i)) - 1 then
                counter <= 0;
                state <= gate;
        end if;
when gate =>
        gate_o <= '1';
        if counter = to_integer(unsigned(train_length_i)) - 1 then
                counter <= 0;
                state <= done;
        end if;
when done =>
        -- check if fire bit reset
        if fire_i = '0' then
                -- prime if prime flag set
                if prime_i = '1' then
                        counter <= 0;
                        state <= ready;
                end if;
        end if;
end case;
```

Listing 4.3: Snippet of code from state machine in signal controller

state. Each of the parameters, `delay`, `pulselen`, and `gatedelay`, have been preset by the control system in the front-end interface. Upon getting the `fire` flag, the state changes to *delay* and starts counting clock cycles until the counter is equal to the `delay` parameter. When `counter` $\geq$ `delay`, the state changes to *pulse*, `counter` is reset, and begins pulsing. Next, the counter keeps counting cycles. When `counter` $\geq$ `pulselen` the pulsing stops, and the state changes to *gatedelay*. The `counter` is reset again and keeps counting until `counter` $\geq$ `gatedelay`, when the state changes to *gate* and `counter` is reset. Next, the `GATE` pulse starts and continues until `counter` $\geq$ `pulselen`, where the `GATE` pulse ends. Finally, the state changes to *done* and remains there until the `prime` flag is set. The state flow is a loop that runs continuously until system operation ends. In fig. 4.6 is a diagram of the states in the finite state machine of the signal controller.

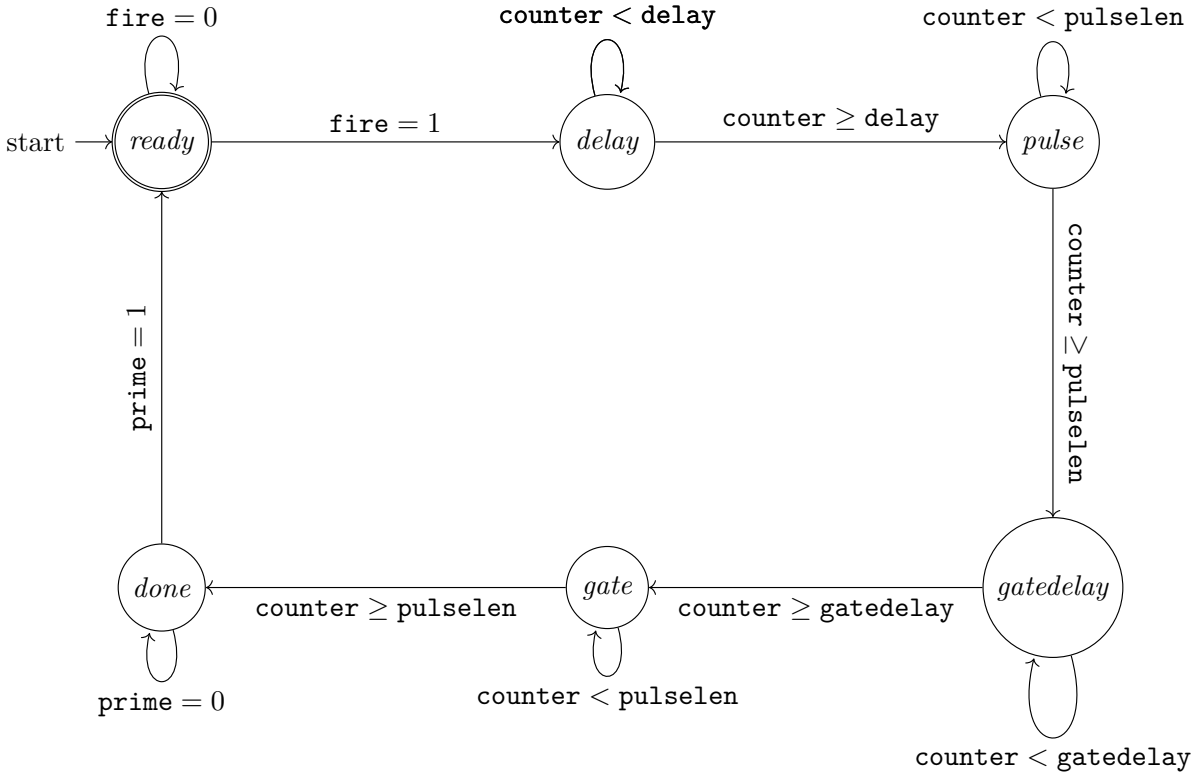



Figure 4.6: Finite state machine (*FSM*) diagram of ultrasound pulser signal controller

In listing 4.4, the code for setting up the registers in JupyterLab can be seen. The code defines the lookup addresses for each register and then defines a Python function to write these registers. For `duty`, `band`, `dutypct`, `startdelay`, `trainlength`, and `gatedelay`, the JupyterLab notebook writes numeric values to each corresponding register. For `prime` and `fire`, these are flags that are either set to *true* or *false*.

In listing 4.5, the commands to write to the pulse registers. The dead-time is configured for $^{10}/_{320\,\mathrm{MHz}} = 31.16\,\mathrm{ns}$. For the start delay, meaning the time from the `PRF` goes LOW to the beginning of the first PWM pulse, is set to 625 ns. The `trainlength` is set to 919 ns, or $\approx$ 4 pulses. The `gatedelay`, also called sample depth, is set to 3.125 µs, or in terms of depth, for speed of sound in water $c = 1480\,\mathrm{m\,s^{-1}}$, $d_{\mathrm{sample}} = \frac{1480\,\mathrm{m\,s^{-1}}/320\,\mathrm{MHz}\cdot 1000}{2} = 2.31\,\mathrm{mm}$. Lastly, `prime`, and `fire` starts the pulsing.

```
from pynq import Overlay
ol=Overlay("pwm_ultrasound_pulser.bit")
from pynq import MMIO
RANGE = 8 # Number of bytes; 8/4 = 2x 32-bit locations which is all we need for this example

duty_address = ol.ip_dict['axi_gpio_duty']['phys_addr']
duty_register = MMIO(duty_address, RANGE)
# Write 0x00 to the tri-state register at offset 0x4 to configure the IO as outputs.
duty_register.write(0x4, 0x0) # Write 0x0 to location 0x4; Set tri-state to output
band_address = ol.ip_dict['axi_gpio_band']['phys_addr']
band_register = MMIO(band_address, RANGE)
# Write 0x00 to the tri-state register at offset 0x4 to configure the IO as outputs.
band_register.write(0x4, 0x0) # Write 0x0 to location 0x4; Set tri-state to output
flags_address = ol.ip_dict['axi_gpio_flags']['phys_addr']
flags_register = MMIO(flags_address, RANGE)
# Write 0x00 to the tri-state register at offset 0x4 to configure the IO as outputs.
flags_register.write(0x4, 0x0) # Write 0x0 to location 0x4; Set tri-state to output
start_delay_address = ol.ip_dict['axi_gpio_start_delay']['phys_addr']
start_delay_register = MMIO(start_delay_address, RANGE)
# Write 0x00 to the tri-state register at offset 0x4 to configure the IO as outputs.
start_delay_register.write(0x4, 0x0) # Write 0x0 to location 0x4; Set tri-state to output
train_length_address = ol.ip_dict['axi_gpio_train_length']['phys_addr']
train_length_register = MMIO(train_length_address, RANGE)
# Write 0x00 to the tri-state register at offset 0x4 to configure the IO as outputs.
train_length_register.write(0x4, 0x0) # Write 0x0 to location 0x4; Set tri-state to output
gate_delay_address = ol.ip_dict['axi_gpio_gate_delay']['phys_addr']
gate_delay_register = MMIO(gate_delay_address, RANGE)
# Write 0x00 to the tri-state register at offset 0x4 to configure the IO as outputs.
gate_delay_register.write(0x4, 0x0) # Write 0x0 to location 0x4; Set tri-state to output

def duty(duty):
        duty_register.write(0x00, duty)
def band(band):
        band_register.write(0x00, band)
def dutypct(duty):
        duty_register.write(0x00, round((0x1F*2)/(100/duty)))
def fire():
        flags_register.write(0x00, 1) # bit 0
        flags_register.write(0x00, 0)
def prime():
        flags_register.write(0x00, 2) # bit 1
        flags_register.write(0x00, 0)
def startdelay(startdelay):
        start_delay_register.write(0x0, startdelay);
def trainlength(trainlength):
        train_length_register.write(0x0, trainlength);
def gatedelay(gatedelay):
        gate_delay_register.write(0x0, gatedelay);
```

Listing 4.4: Snippet from JupyterLab setting up the registers

```
band(10)
startdelay(200)
trainlength(294)
gatedelay(1000)
prime()
fire()
```

Listing 4.5: Snippet from JupyterLab showing the commands to set up a single pulse transmission

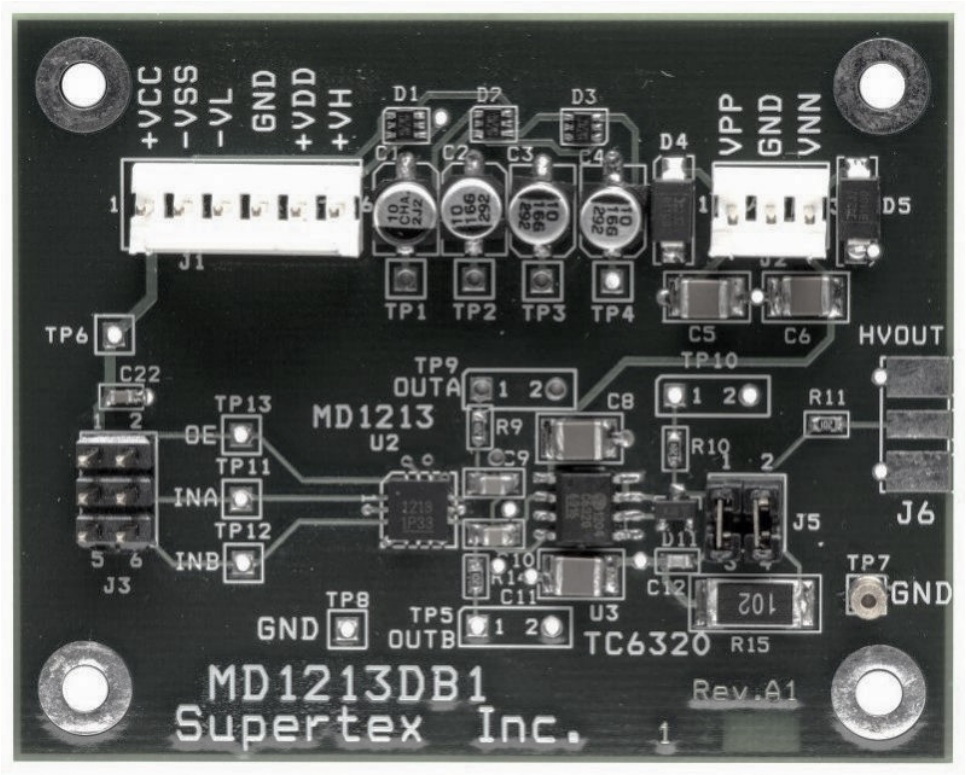



Figure 4.7: MD1213DB1 High Speed Pulser

## 4.2 Power Stage

A picture of the power stage PCB can be seen in fig. 4.7. For purpose of evaluating the power stage, a development board *MD1213DB1* is used. Seen in fig. 4.7 is a picture of the development board. However, as the power connector use MOLEX headers for power inputs, and the power delivery comes from a laboratory *DC* power supply (*DCPS*), two breakout boards are needed to attach 4 mm test leads to J1 and J2. An SMA connector is mounted to J6. On the pulse input connector J3, a shunt short is mounted on the operation enable (*OE*) input. INA and INB are connected using jumper wires to the pulser.

## 4.3 Transmit/Receive Switch

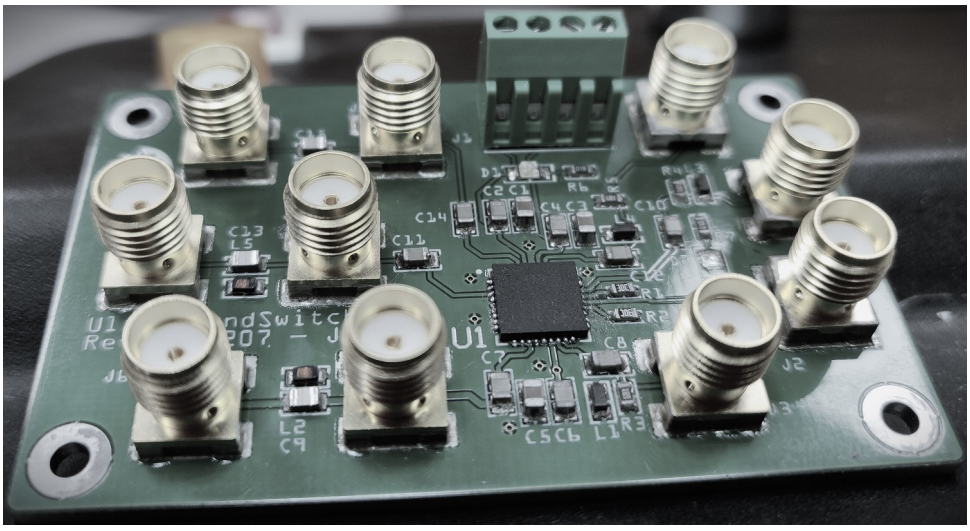


Figure 4.8: Transmit/Receive Switch after assembly

The entire schematic of the transmit/receive switch can be found in the appendix in figs. D.8 and D.9. As mentioned in the previous chapter, a PCB layout was made and a batch of 5 was ordered with an accompanying stencil for fast assembly. After the PCBs arrived, the stencil was mounted in the stencil frame and the PCB was aligned for solder paste application. After the solder paste application is completed, all the components are placed on their corresponding footprints and the PCB is placed in the reflow oven. The equipment used in this process is listed in table F.1. The finished assembly can be seen in fig. 4.8.

## 4.4 Band-pass Filter

Since the band-pass (*BP*) filter is comprised of a module component in a bespoke form factor, a circuit is implemented on a prototyping board. With the BPF-C4R5+ mounted in the center, input and output connectors are placed on either side with SMA connectors. Since the filter is passive, no power connectors are needed. The prototype board of the *BP* filter can be seen in fig. 4.9.

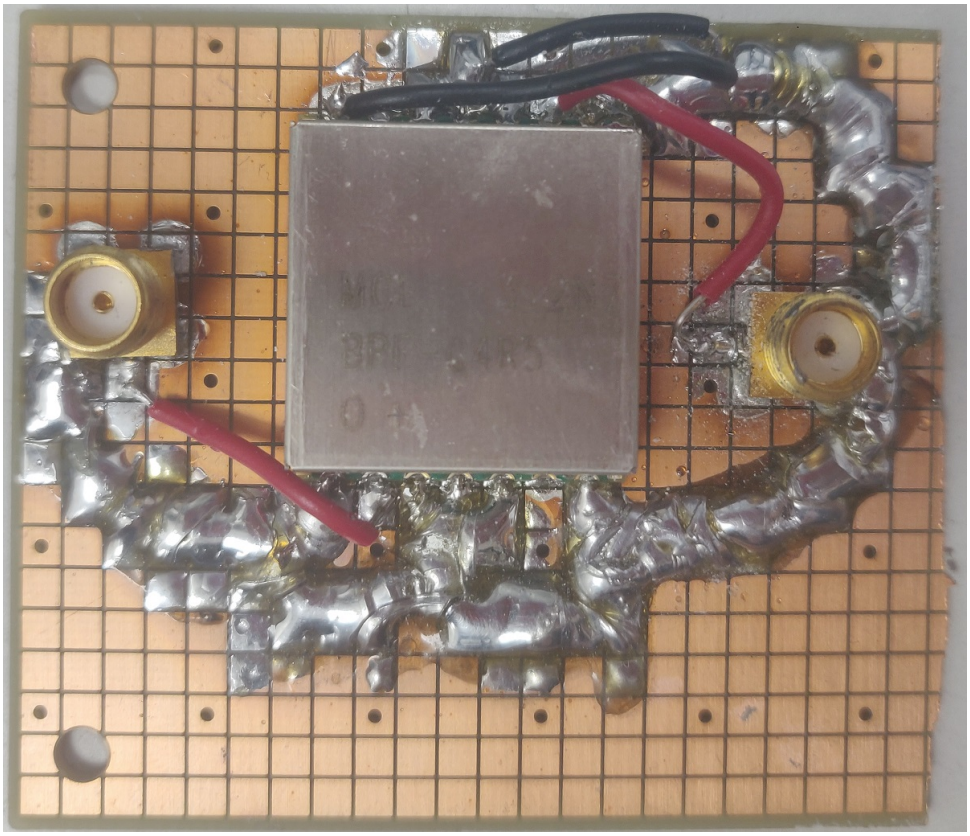


Figure 4.9: Prototype board of the Band-Pass filter

## 4.5 Preamplifier

Before the signal can be demodulated, it must be DC-biased and amplified. This is what the preamplifier is for. For the preamplifier, the circuit is validated using an experiment where a function generator is transmitting a low amplitude sine with ac-coupling and measure the amplified dc-coupled output. Using the AD8332 built-in low noise amplifier (*LNA*) and bypassing the variable gain amplifier (*VGA*) in the circuit, it feeds the amplified output to the next subcircuit Quadrature Demodulator. The preamplifier is part of the same board as the quadrature demodulator in section 4.6.

## 4.6 Quadrature Demodulator

As described in the previous chapter, the demodulator use an I/Q quadrature demodulation scheme to take two differential RF signals and a quadruple frequency signal, in this case, 5 MHz and 20 MHz, respectively, and determines the frequency difference between the fundamental frequency and the Doppler frequency on the output. After the differential signal from the preamplifier is demodulated, it is output as a current. Next, it is converted using the AD8021 current-to-voltage amplifier coupled with an active low-pass (*LP*) filter to remove the summed demodulated frequency component. What remains are the low-frequency $I$ and $Q$ signals in the kHz range. The entire schematic of the demodulator can be found in the appendix in fig. D.11.

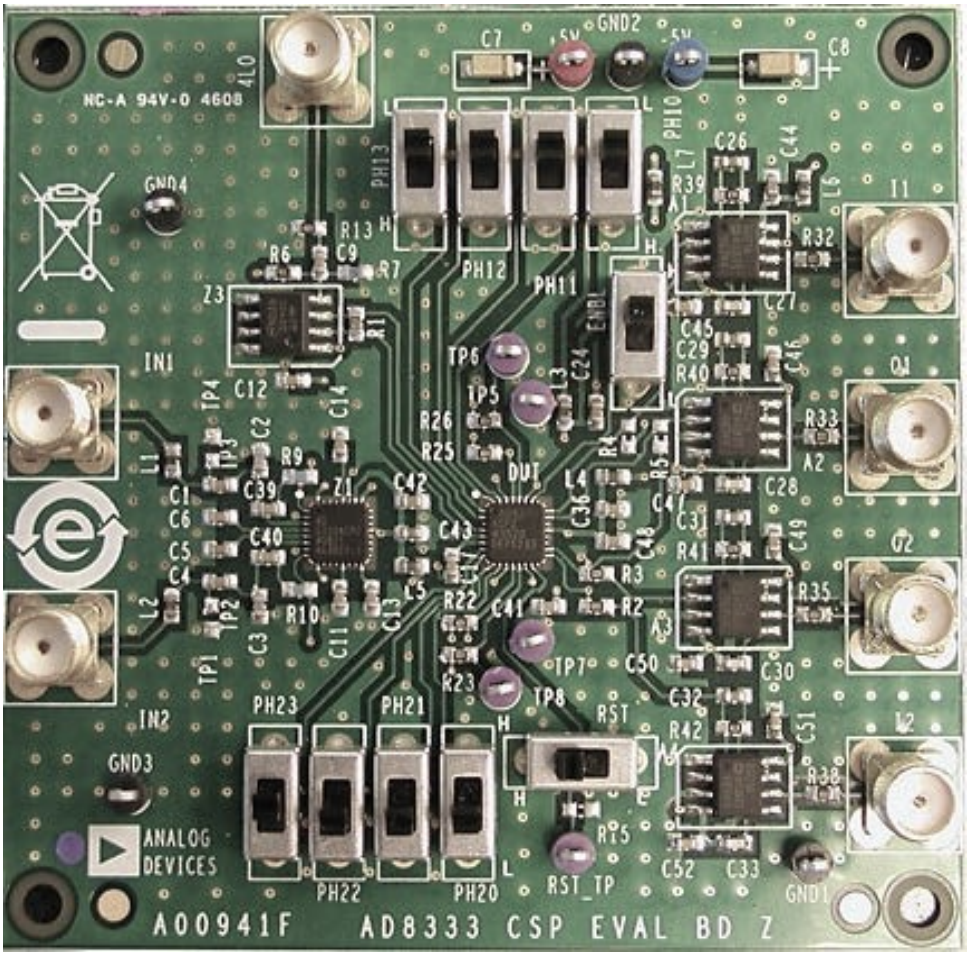



Figure 4.10: Demodulator PCB AD8333-EVALZ

## 4.7 Sample and Hold

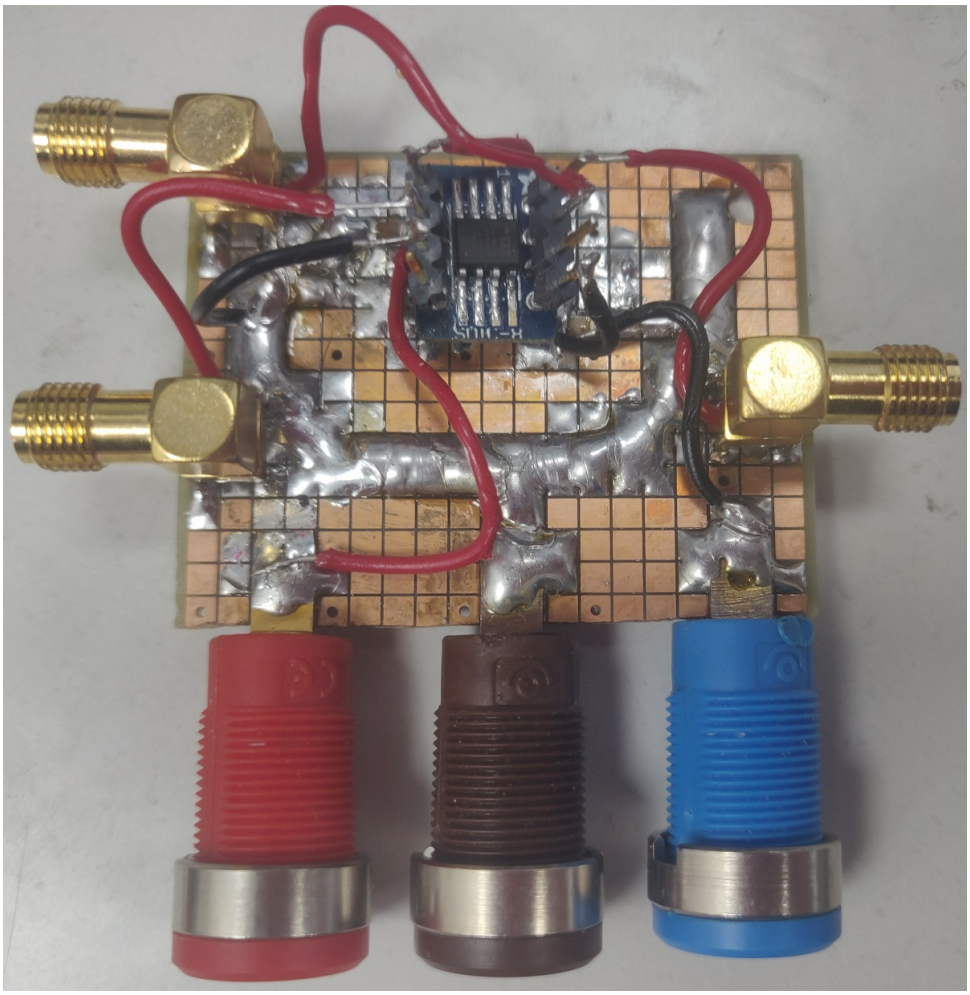


Figure 4.11: Prototype board of the Sample and Hold amplifier

After each demodulated burst is sampled, between each sample line pulse repetition it is desired to hold the voltage, so the analogue-to-digital conversion that may be running asynchronously does not sample zero-values between the bursts. After the low-frequency signals are output from the quadrature demodulator, they are sampled using the `GATE` signal output from the control system. This `GATE` signal is a delayed pulse equal to the length of the pulse train and determines the sampling depth of the analogue front end (*AFE*). The prototype board of the sample and hold amplifier can be seen in fig. 4.11.

## 4.8 Active Filter

Before the signal is digitally quantized by the analogue-to-digital converter (*ADC*), it has to pass through a high-pass (*HP*) filter to remove undesired pulse repetition frequency (*PRF*) or Wall frequency components. The implementation is done on a prototyping board and can be seen in fig. 4.12.

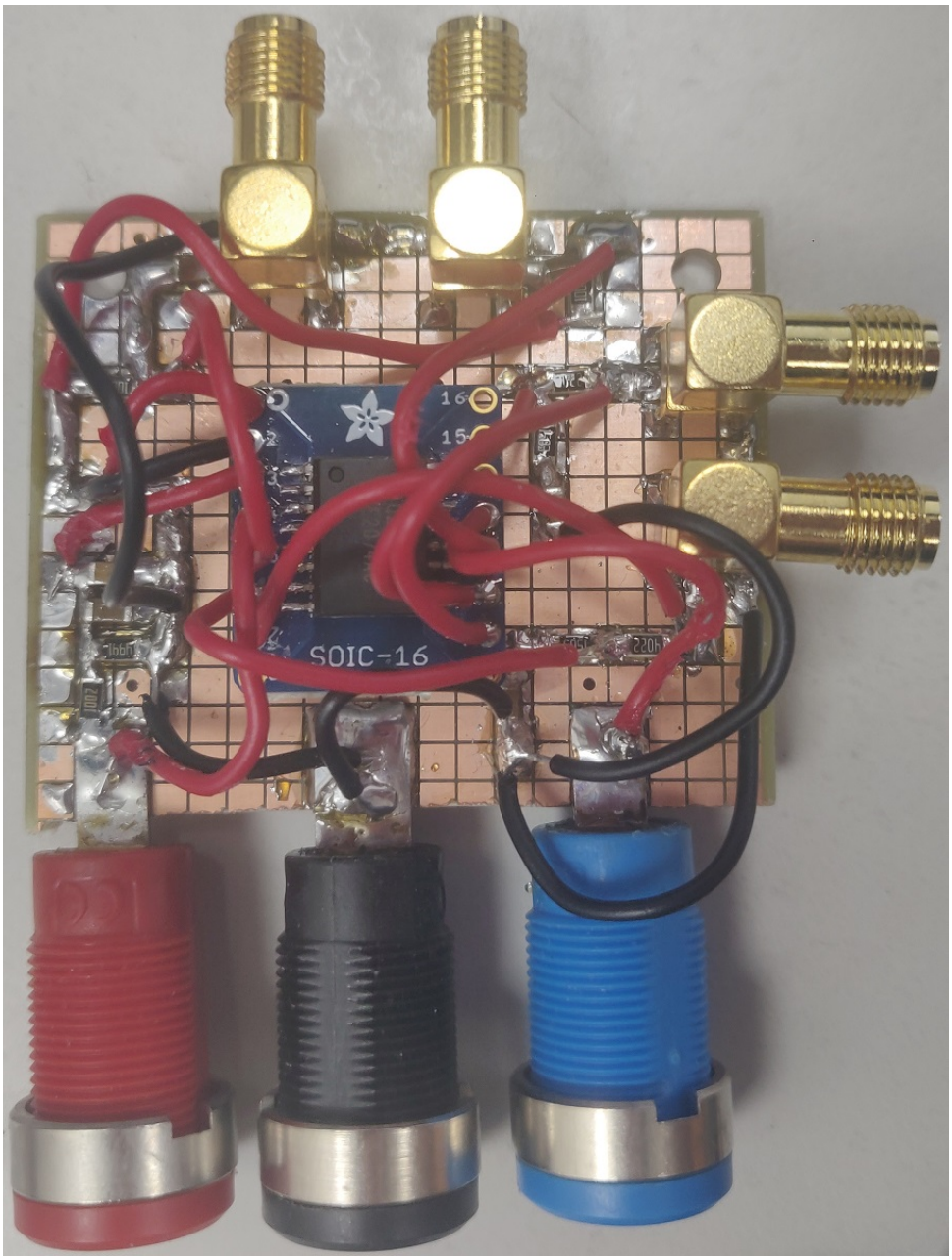



Figure 4.12: Prototype board of the active filter and direct current ($DC$)-coupler

## 4.9 Digital Signal Processor

Finally, the signal can be measured with the control system running the base overlay on the XADC interface. In listing 4.6, the code that samples the input signal on the XADC interface can be seen.

```
import time
from pynq.overlays.base import BaseOverlay
base = BaseOverlay("base.bit")
from pynq.lib.arduino import Arduino_Analog
from pynq.lib.arduino import ARDUINO_GROVE_A1
analog0 = Arduino_Analog(base.ARDUINO,ARDUINO_GROVE_A1)
a = []
t_start = time.time()
t_end = time.time() + 1
while time.time() < t_end:
a.append(analog0.read()[0])
```

Listing 4.6: Snippet of code running the ADC for 1 s and saving the samples into a 1-dimensional table

In listing 4.7, the code that performs the discrete Fourier transform of the input signal can be seen. In essence, Fourier analysis is a technique used to represent a function as a combination of periodic elements. When the function and its Fourier transform are replaced with discrete versions (as is the case of sampled datapoints), it is referred to as the discrete Fourier transform (DFT). The DFT has gained popularity in numerical computing primarily due to an efficient computational methodology called the Fast Fourier Transform (FFT). It effectively divides the input into discrete frequency components. In this context, the input that undergoes discretization and transformation is commonly referred to as a signal, existing in the time domain. On the other hand, the resulting output, known as a spectrum exists in the frequency domain. In the code, the output of the FFT is normalised, so the maximum frequency component is scaled to 1. In Numpy, the implementation of discrete Fourier transform is expressed by eq. (4.1).

$$A_k = \sum_{m=0}^{n-1} a_m e^{\left(-2\pi i \frac{mk}{n}\right)}, \quad k = 0, ..., n-1 \tag{4.1}$$

This is defined for complex inputs and outputs, and a frequency element at frequency $f$ is expressed by a complex exponential $a_m = \exp(2\pi i f m \Delta t)$, where $\Delta t = \frac{1}{f_{\text{sample}}}$, or the sample interval. After the Doppler frequency is succesfully obtained, it is possible to determine the velocity using the expression described in eq. (2.15). In practical terms, the implementation is shown in listing 4.8.

```python
import numpy as np
import matplotlib.pyplot as plt
fs = round(1/((t_end-t_start)/len(a)))
print(f"Calculated samplerate fs = {fs} kS/s")
t = np.linspace(0, t_end-t_start, len(a))

fig, ax = plt.subplots(3, 1)
ax = ax.flatten()
ax[0].plot(t, a)
ax[0].set_title("Original signal")
ax[0].set_xlabel("Time [s]")
ax[0].set_ylabel("Voltage [V]")
plt.tight_layout()
plt.grid()
plt.minorticks_on()

a_fft = np.fft.fft(a)           # Original FFT
a_fft = a_fft[1:]                 # Delete DC component
a_fft = a_fft[:round(len(t)/2)] # First half ( pos freqs )
a_fft = np.abs(a_fft)           # Absolute value of magnitudes
a_fft = a_fft/max(a_fft)        # Normalised so max = 1

freq_x_axis = np.linspace(1, fs/2, len(a_fft))
ax[1].plot(freq_x_axis, a_fft, "o-")
ax[1].set_title("Frequency magnitudes")
ax[1].set_xlabel("Frequency [Hz]")
ax[1].set_ylabel("Magnitude")
f_loc = np.argmax(a_fft) # Finds the index of the max
f_val = freq_x_axis[f_loc] # The strongest frequency value
samplenums = round(2*(1/f_val)*fs)
ax[2].plot(t[0:samplenums],a[0:samplenums])
ax[2].set_title("Two periods")
ax[2].set_xlabel("Time [s]")
ax[2].set_ylabel("Voltage [V]")
plt.show()

print(f"The largest frequency component is f = {f_val}")
```

Listing 4.7: Snippet of code that plots the recorded input signal, plots all frequency components, and two periods of the signal

```
f0=5e6
f_d=200
theta=60*np.pi/180
c=1480
v=((f_val)*c)/(2*np.cos(theta)*f0)
print(f"The estimated velocity is v = {v}")
```

Listing 4.8: Snippet of code that estimates the velocity based on the largest frequency component of the input signal

# Chapter 5

# Analysis

In this chapter every subcomponent will be part of an experiment in isolation to validate their function. Also, various test setups will be discussed for the purpose of making multi-module experiments in the AFE and control system to evaluate their performance and compatibility. As such, each module will be tested independently to validate its function before a larger, more comprehensive experiment is conducted. All figures in this chapter show measurements performed on physical hardware.

## 5.1 Module Testing

### 5.1.1 Control System

First, a bitstream was generated with a Pwm generator and AXI interconnect registers. An AXI interconnect register is a method to create GPIO interfaces between the logical processor and the Arm processor on the Zynq 7000 SoC using registers that temporarily store GPIO data in the bitstream overlay. A block diagram of the overlay can be seen in fig. B.8 and the JupyterLab notebook can be seen in **??**.

A preliminary implementation was implemented in Vivado and JupyterLab for a continuous complementary PWM controller. The resulting measurement can be seen in fig. 5.1 before a clock configuration, which means the frequency corresponded to an arbitrary temporary pulse frequency of 122 kHz. As the initial test proved successful, the continued development and maturation of the pulser enabled a complete signal generator for the ultrasound pulse generator.

Seen in fig. 5.2 are the measured signals from the pulse generator showing the various timings of each signal captured with a Salae Logic Analyzer Pro 8. In the figure, there is observed a continuous 20 MHz `CLK` signal for the demodulator. For `PRF`, the signal sets the ultrasound switch to transmit with a corresponding startdelay to the complementary `PWM` and `PWMN` pulses. To see a further explanation of each signal and its function, refer to fig. 3.4.

### 5.1.2 Power Stage

An experiment is conducted to validate the function of the power stage. Using the jumpers, the PCB is configured without its onboard load, and a lead circonate titanate ($PZT$) transducer is attached with a splitter adapter to connect the other side to an oscilloscope for data acquisition. Seen in fig. 5.3 are actual

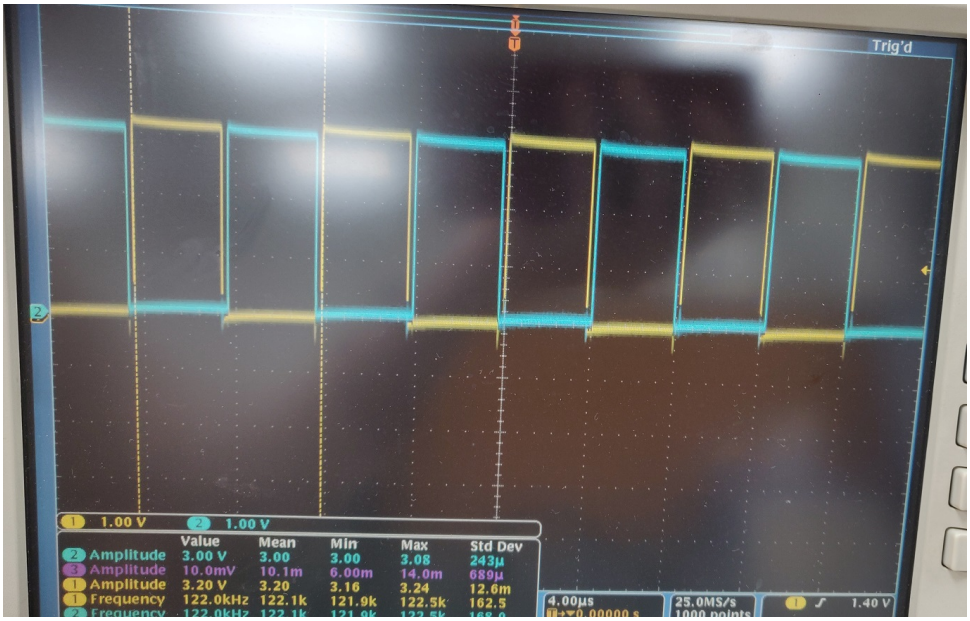



Figure 5.1: Complementary PWM output with Pynq Z1 FPGA and JupyterLab notebook

measured inputs and outputs of the power stage. On the input, there are two complementary 5 MHz signals with varying duty cycle to generate the desired dead time. On the output, we see the rail-to-rail push-pull operation of the metal-oxide-semiconductor field-effect transistor (*MOSFET*) half-bridge. The schematic of the transmitter can be found in the appendix in fig. D.10. Noticeable noise is observed in the input signal top and base but is negligible for successful operation. Possibly, the noise is due to a cable and adapter setup.

### 5.1.3 Transmit/Receive Switch

For validating the TX/RX switch, an experiment is conducted with a *PZT* transducer, water tank, function generator and an oscilloscope. Using two input signals, $f_{\text{prf}} = 10\,\text{kHz}$ switch signal, and $f_0 = 5\,\text{MHz}$ burst mode transmit signal, the switch is configured to transmit and receive. A picture of the submerged transducer with a reflector can be seen in fig. 5.4. After submerging the transducer in distilled water and measuring on the receiver side of the TX/RX switch, a reflected signal from the tank can be observed in fig. 5.5.

### 5.1.4 Band-pass Filter

it is desired to validate its frequency response to determine if it functions as desired. To obtain the frequency response, a bode plot of the magnitude and phase is measured from 300 kHz until 20 MHz using a vector network analyzer (*VNA*) in a S21 configuration, meaning a measurement of the output in respect to the input. This measurement determines the difference in magnitude and phase of the output in comparison with the input signal. Observed in fig. 5.6 is the frequency response of the band-pass filter measured on a *VNA*. It is noted that the pass band frequencies are mostly as expected with $-0.5\,\text{dB}$ frequencies at 1.5 MHz and 7 MHz. Though, the roll-off in the higher stop band appears somewhat lower than in the lower stop band. That would mean that it is plausible that higher frequency noise components are retained in the output than in the lower stop band. For the phase, it seems to have a significant phase

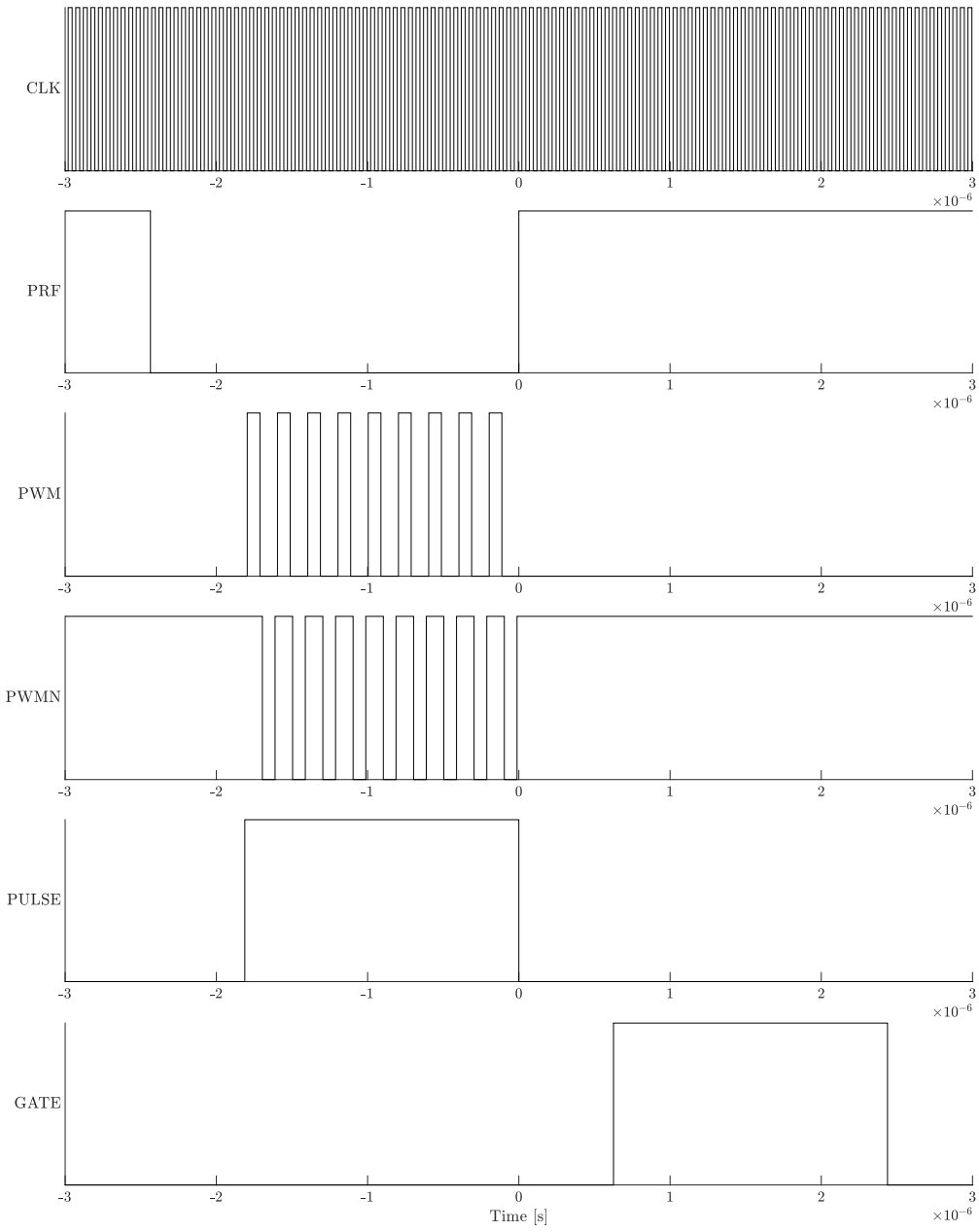



Figure 5.2: Captured timing diagram of the control system pulser

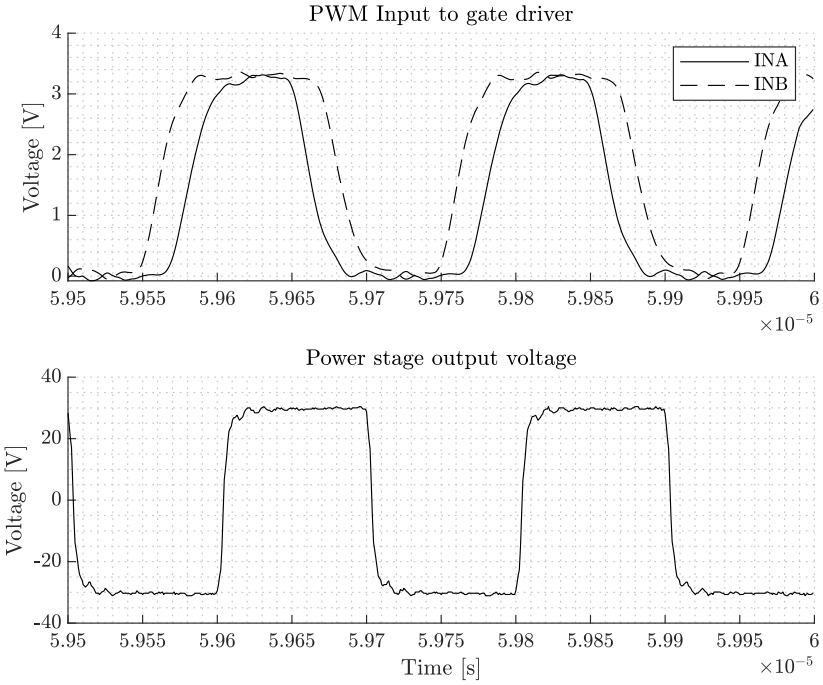



Figure 5.3: Measured input and output of power stage PCB. (Above) Input to gate driver with dead-time (Below) Output of MOSFET half-bridge and the voltage across the load

delay, going from around 100° to 250° from the start to the end of the pass band.

### 5.1.5 Preamplifier

Seen in fig. 5.7 are measurements of the preamplifier circuits showing a 70 mV input signal and a 300 mV output signal with a 2.5 V DC bias. In this application, however, only the low noise amplifier (*LNA*) is used, and the variable gain amplifier (*VGA*) is bypassed in the hardware preamplifier configuration. The schematic of the preamplifier circuit is part of the demodulation schematic and can be found in the appendix in fig. D.11.

### 5.1.6 Demodulator

An experiment is conducted with the sample-and-hold amplifier to verify the functionality. A low-frequency I-Q simulated signal is created from the function generator with a sample gating pulse train to control the sample-and-hold function. Seen in fig. 5.8 are the input signals, differential signals of 5.001 MHz and 20 MHz local oscillator signal. Seen in fig. 5.9 are the differential input signals $A$ and $B$ and the demodulated output signals $I$ and $Q$, where the phase between $I$ and $Q$ denotes the Doppler shift direction, or rather, the direction of flow of the scatterer. It is noted that the differential signal is so high frequency compared to the timescale so there has to be a zoomed in subplot where the waveform is visible to show the waveform. This highlights the observation of the Doppler shift being pushed down in frequency, which is one of the key functions of the demodulator in this system.

Figure 5.4: TX/RX Switch reflection experiment with water tank

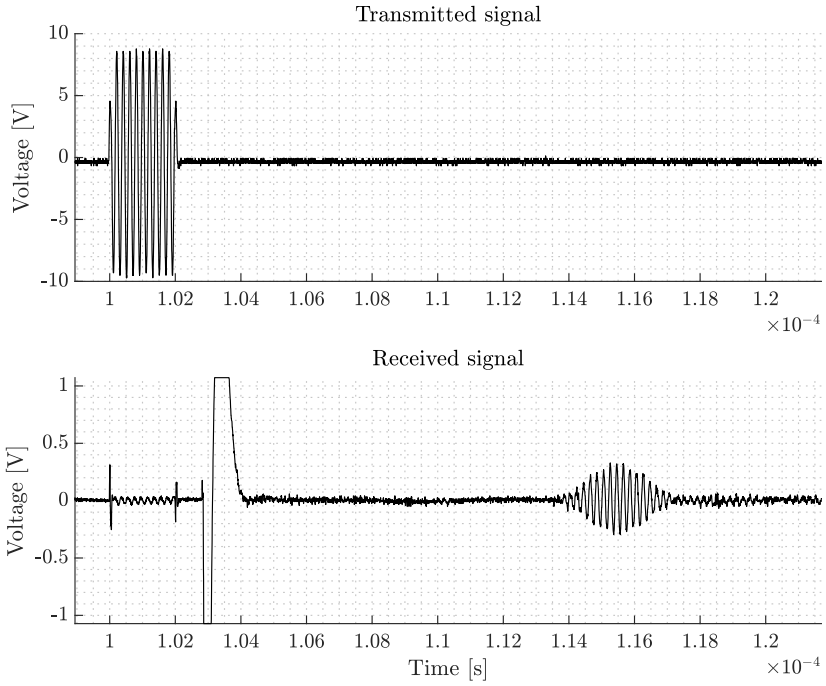



Figure 5.5: Measured transmit and receive on Transmit/Receive Switch PCB (Above) Measured transmit voltage (Below) Received reflected signal off water tank

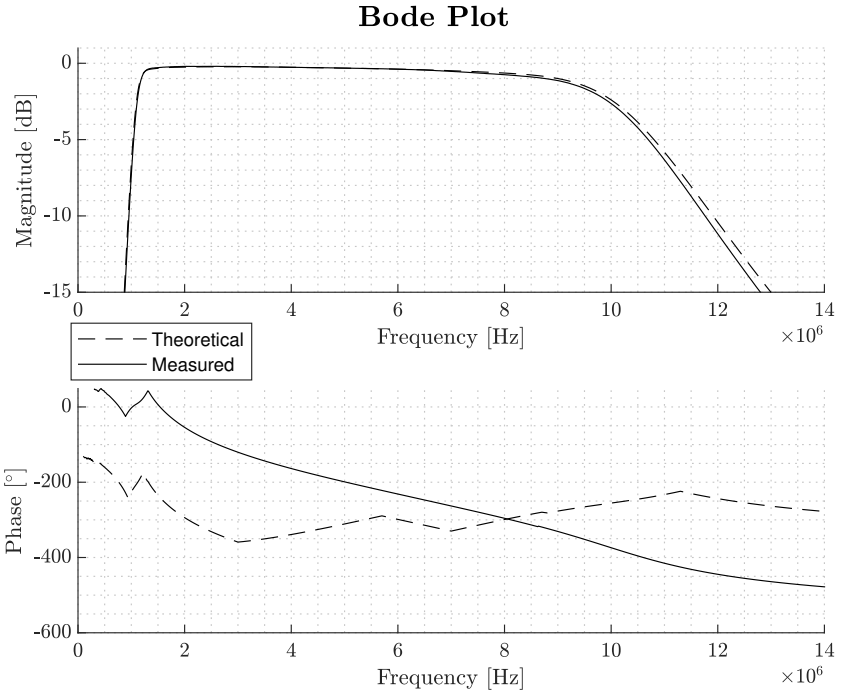



Figure 5.6: Band-pass Filter bode plot from 0.3 MHz to 14 MHz with (above) magnitude and (below) phase

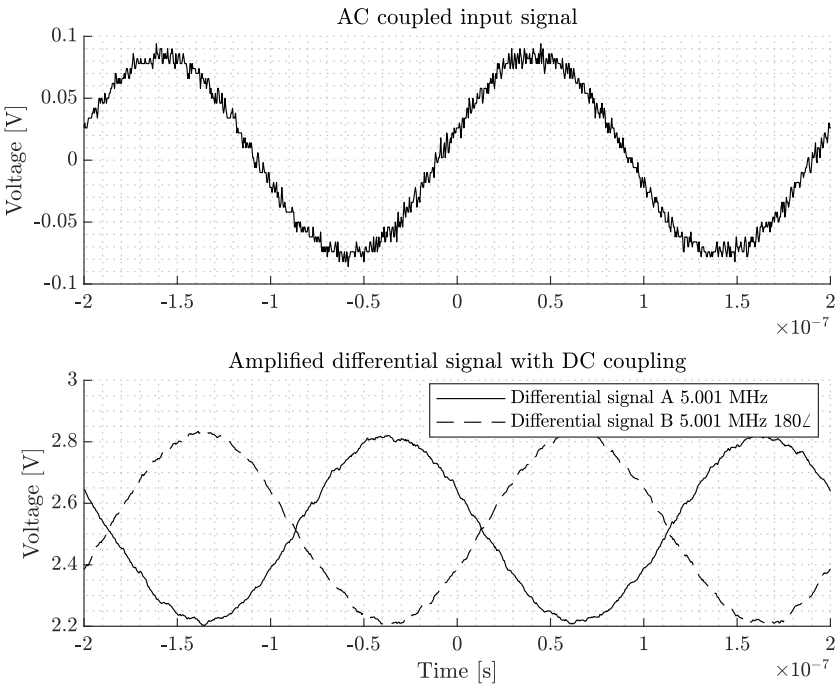



Figure 5.7: Measured input of preamplifier PCB, (Above) AC coupled input signal with amplitude 1 V (Below Measured output of preamplifier PCB, Differential signal with DC coupling and ×19 dB amplification)

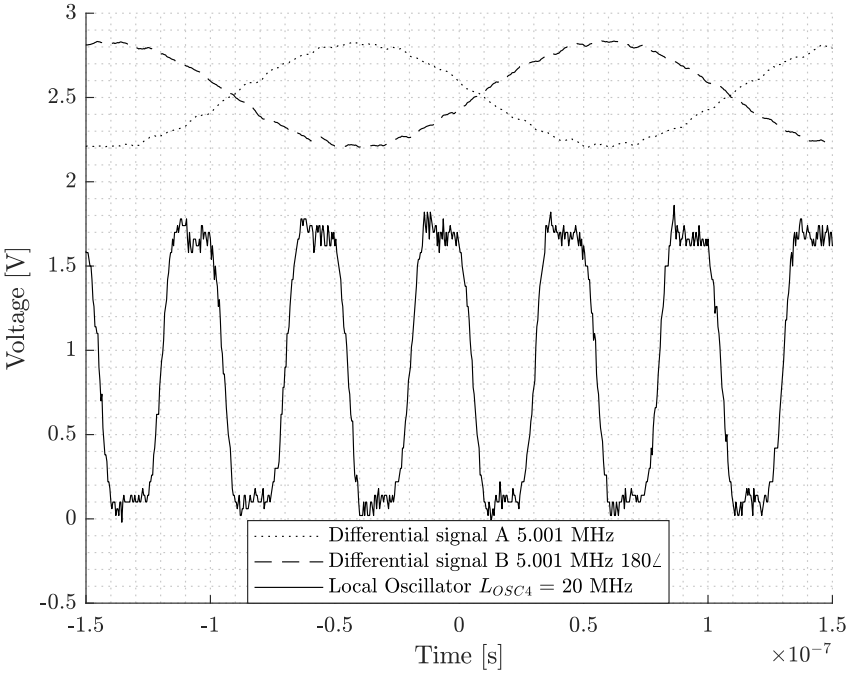



Figure 5.8: Measured input of demodulator PCB (Above) Input from received signal (Below) Input from local oscillator ($f_0 \cdot 4$)

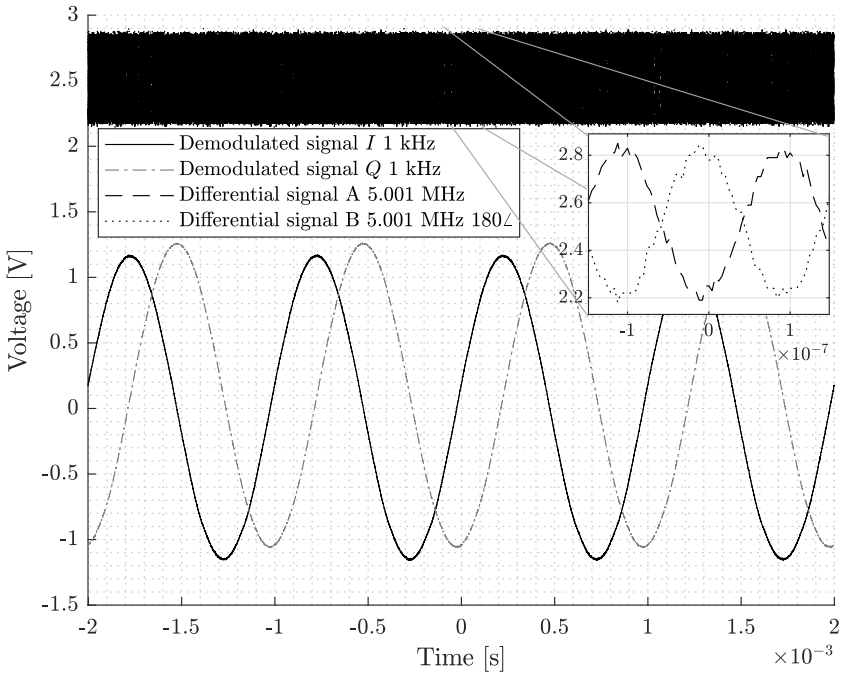



Figure 5.9: Measured output of demodulator PCB

### 5.1.7 Sample and Hold Amplifier

An arbitrary function generator (*AFG*) is configured to simulate the output signal of the quadrature demodulator. Next, the ultrasound pulser `GATE` signal is connected to the sample and hold amplifier. Seen in fig. 5.10 is the measured inputs and outputs of the circuit during the experiment. Above is the I-Q input and in the middle is the sample gating, and below is the output signal. On the output signal, it is noted the corresponding voltage transients for every pulse in the gate input.

### 5.1.8 Active Filter

A 1 kHz input signal is received on the input SMA connector from a function generator and probed with an oscilloscope. ±5 V is supplied to the power terminals. Next, the output SMA connector is probed with an oscilloscope and the data is recorded. The dashed line is the alternating current (*AC*) coupled input signal with 1 V amplitude and the solid line is the amplified direct current (*DC*) coupled output signal. Seen in fig. 5.11 is the measured input and output of the DC coupler and active filter. This measurement confirms its operation as expected when comparing to the simulation shown earlier in the report.

### 5.1.9 Digital Signal Processor

A function generator is configured to transmit a 100 Hz sinusoidal signal with a 0.5 V offset and a 1 V peak-peak amplitude. This signal is then measured by the DSP XADC and saved for processing. A peak samplerate is determined by recording the maximum amount of samples possible but limiting the collection time to one second. Then, the samplerate is identical to the number of samples collected. After one second of sampling, a record of 4422 samples were stored in the buffer. After processing the data, a total of three plots are generated and can be seen in fig. 5.12. In the top plot, the whole buffer is plotted

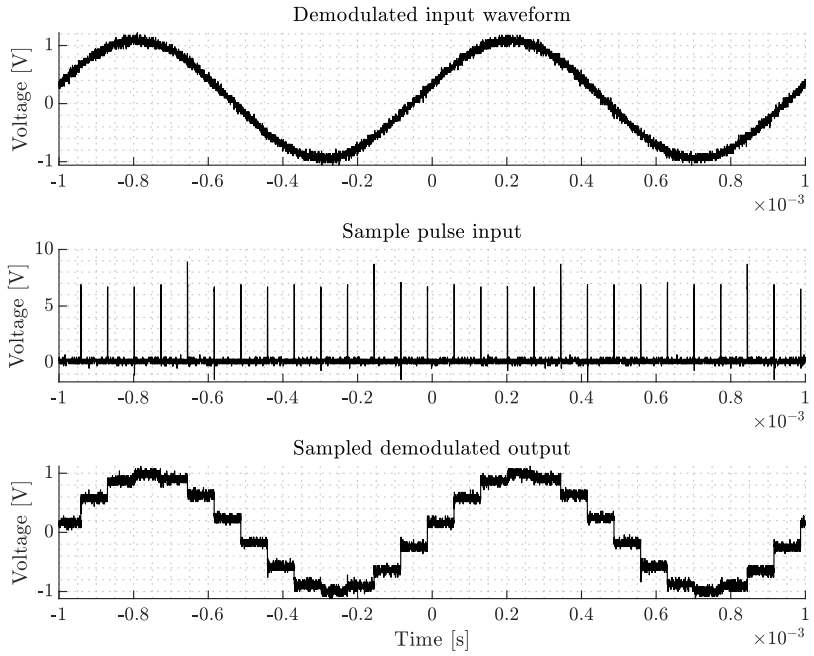



Figure 5.10: Measured input and output of Sample and Hold amplifier

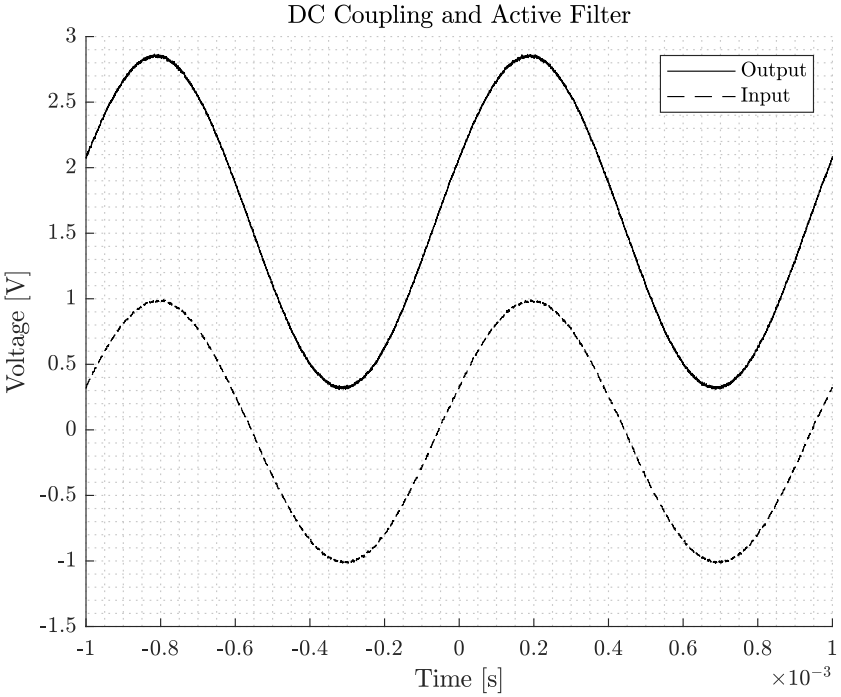
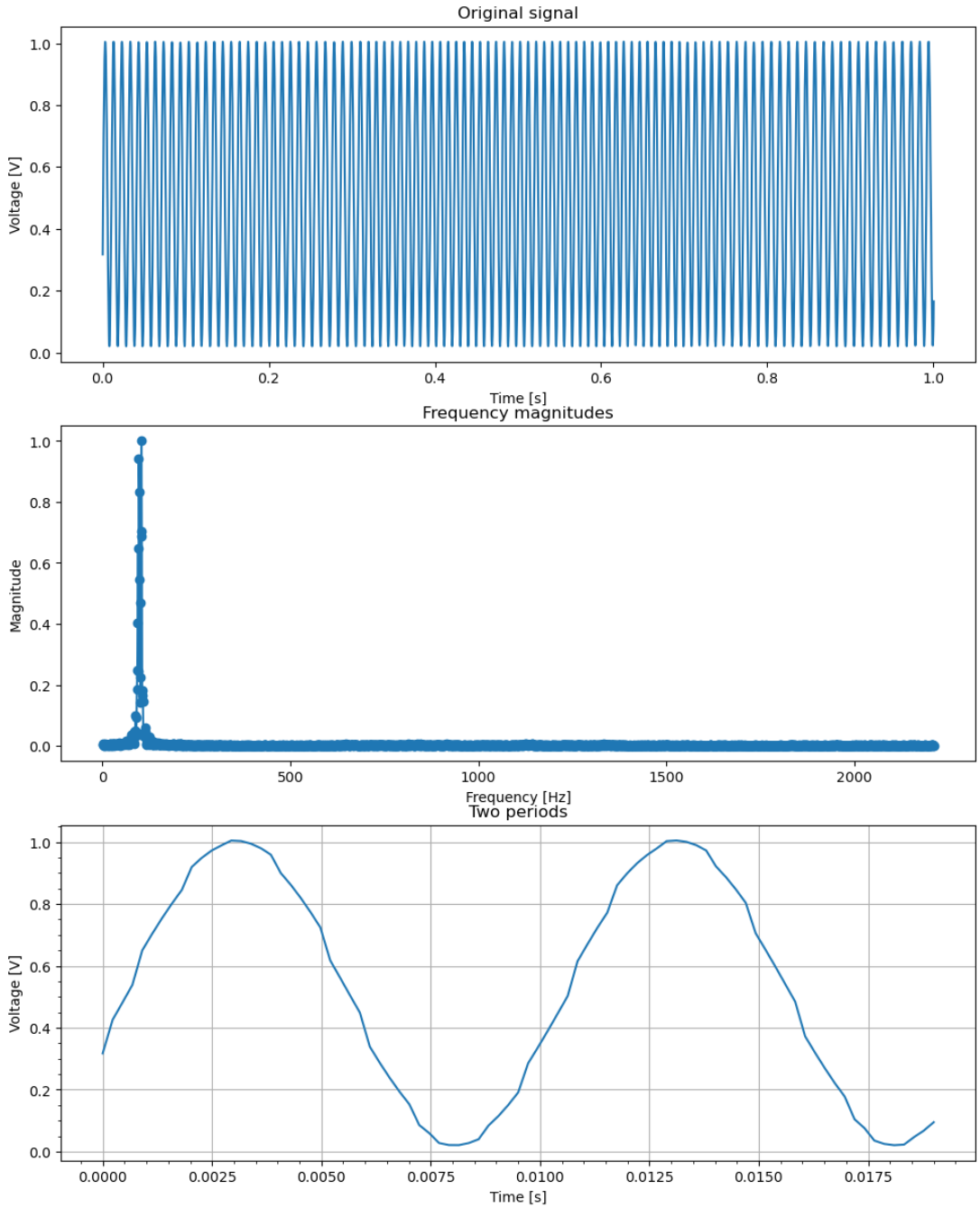



Figure 5.11: Measured input and output of Active filter and DC Coupler




```
The largest frequency component is f = 103.95341474445952
```

In [6]:
```
f0=5e6
f_d=200
theta=60*np.pi/180
c=1480
v=((f_val)*c)/(2*np.cos(theta)*f0)
print(f"The estimated velocity is v = {v}")
```

```
The estimated velocity is v = 0.030770210764360015
```

Figure 5.12: Velocity estimation with XADC and Jupyter

over time. In the middle plot, the spectrum of the recorded measurements are plotted. In the lower plot, two periods of the largest frequency components are plotted. In the measurement, with a 103 Hz signal being measured on the input of the XADC system, a velocity is estimated to be 3.1 $\mathrm{cm\,s^{-1}}$. Since

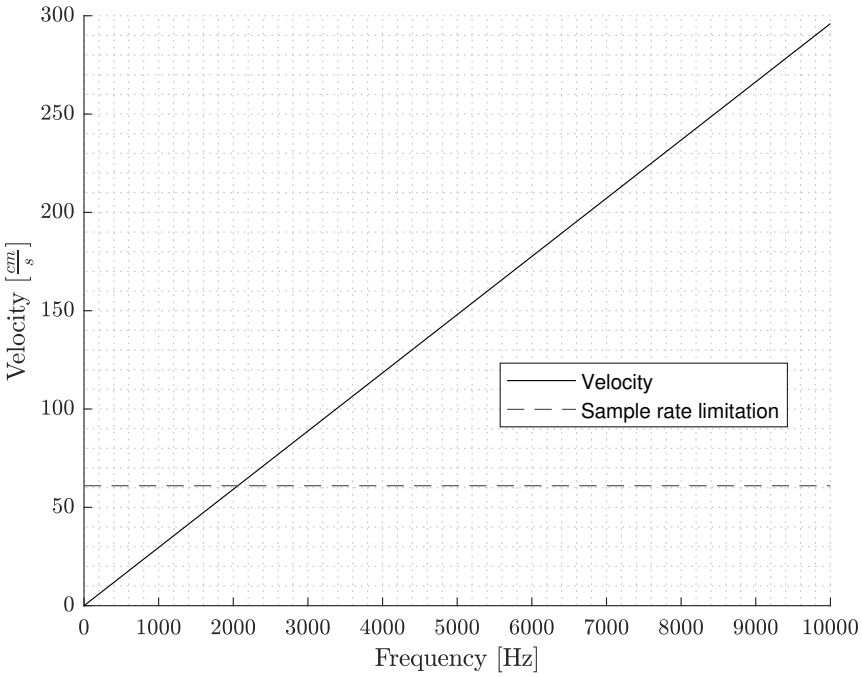



Figure 5.13: Velocity estimator with sample rate limitation

the maximum sample rate with the XADC implementation used was determined, it is desired to know the limitations of the Doppler velocity estimation. Based on the Nyquist limit, a with 0.5 cycle/sample results in distortion known as aliasing. In principle, it means that when the same rate was determined to be 4.422 kS/s that means the Nyquist frequency for this system is equal to 2.211 kHz. In the application of a velocity estimator, that means that a velocity of over 60 $\mathrm{cm\,s^{-1}}$ will result in high distortion and unreliable measurements. This is visualised in fig. 5.13 where the velocity curve intersects the sample rate limitation.

## 5.2 Pulse Generator and Power Stage

Using the pulse generator and the power stage, an experiment is performed where the complementary PWM signals are generated and output to the gate driver of the power stage. In turn, the gate driver will drive the MOSFET pair and output a high-power output than the pulse generator can supply on its own through its general purpose *I/O* (*GPIO*).

Seen in fig. 5.14 are the measurements obtained from the pulse generator and power stage experiment. The measurements show the ability of the Pynq Z1 as a pulse generator is functioning as expected. As the power stage half-bridge is rail-to-rail, the output pulse voltages depend on the power supply voltage. In this case, the experiment is using a 30 V maximum *DC* power supply (*DCPS*), and the output peak pulse voltages are ±30 V. However, the power stage itself is specified for bipolar operating voltages up to ±100 V, or unipolar voltages ±200 V.

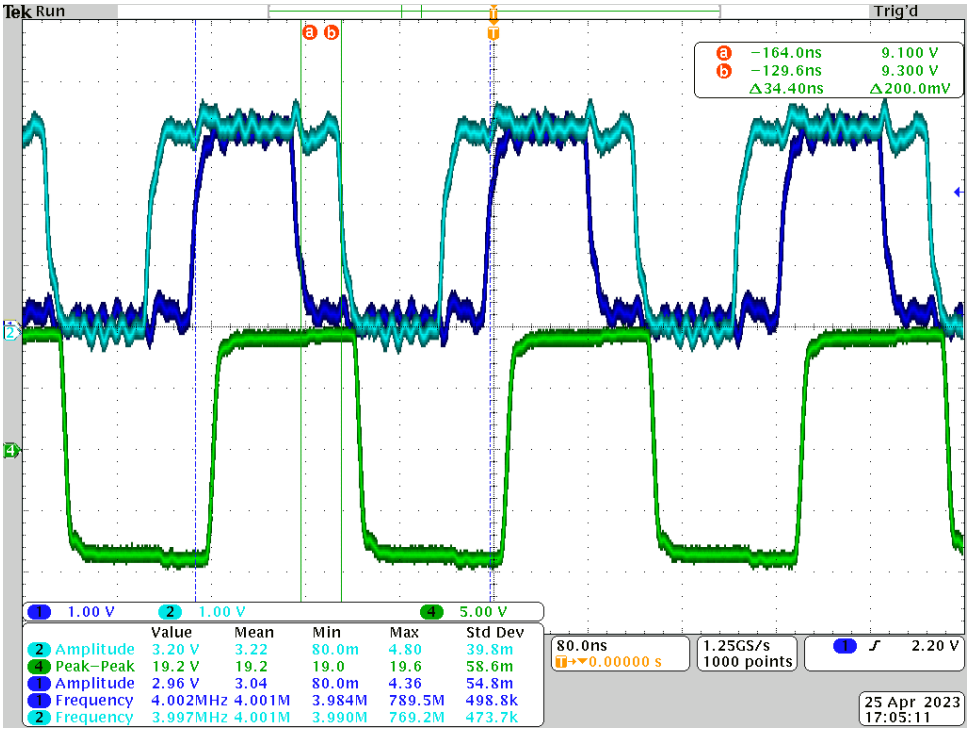



Figure 5.14: Complementary PWM output from the pulse generator and bipolar high power pulses

## 5.3 Doppler String Phantom Experiment

Seen in fig. 5.15 is the CIRS A043 Doppler String Phantom is a device used to evaluate the performance of Doppler ultrasound imaging systems. The phantom consists of a string that is driven by a motor. The motor can be programmed to move at different speeds and directions, and can also be configured to drive the string with physiological waveforms, such as a carotid artery for validation purposes. A Doppler string phantom validation technique is widely used in research, development, and quality assurance of Doppler ultrasound equipment. In fig. 5.16 is a diagram of the experiment to be performed with the analogue front-end to evaluate the performance of the system. The ultrasound system is being controlled by the ultrasound pulser running on the PYNQ-Z1 board. The transducer used in the experiment is a piezoelectric 5 MHz commercial transducer. The entire system operates at a `PRF` frequency of 15 kHz. First, the string is mounted in the string phantom loop. Next, the transducer is mounted on a movable arm and submerged in the water. When the ultrasound waves strike the moving string, the frequency of the reflected waves is shifted due to the Doppler effect. This frequency shift provides information about the velocity and direction of the simulated blood flow. Here, the ultrasound system measures the frequency shift and this can in turn be calculated into a velocity. This data can be analyzed to assess the accuracy and performance of the Doppler ultrasound system. The phantom allows for calibration of the system and evaluation of factors such as velocity sensitivity, and spectral Doppler analysis when looking at the power spectral density (*PSD*) over time. Unfortunately, when conducting the experiment, a design review indicated that the pulser required a modification to the output of the power stage to ensure the potential across the transducer is bled off during rest. However, this must be carefully balanced to ensure that an increased load is not attenuating the received signal. Two solutions were proposed:

1. A resistor to reference voltage in parallel with the transducer


Figure 5.15: CIRS Model 043A Doppler String Phantom with (left) motor controller and (right) tank and string loop

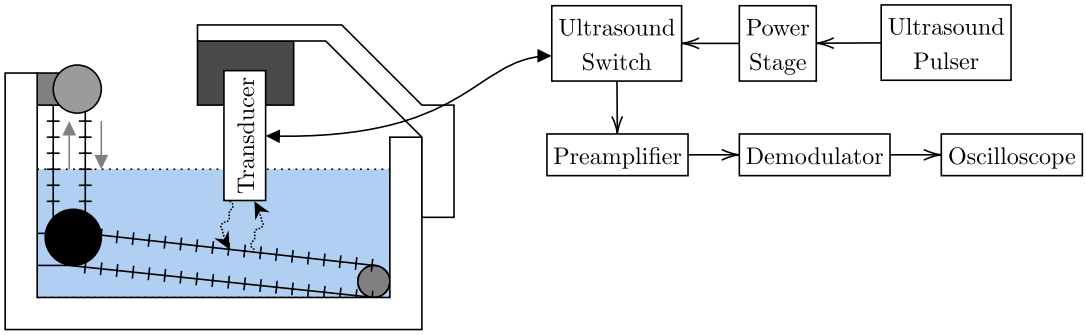



Figure 5.16: Doppler String Phantom experiment diagram (not to scale)

2. A second ultrasound switch, functionally inverse of the primary ultrasound switch to ensure received signal does not get attenuated by pulser circuit

Unfortunately, the deadline for submitting the project had been reached at this point, and ongoing experimentation with the transmitter and String Phantom had to be discontinued to finalise the report.

# Chapter 6

# Conclusion

In conclusion, this project has successfully gone through thorough research, data collection, and analysis, and have obtained a comprehensive understanding of the subject matter. Through a literature review, key gaps in the research have been identified. This formed the basis of a design approach to the ultrasound system that was followed throughout the project. This project aimed to design, build, and test an ultrasound system for blood velocity estimation that can be miniaturised. The implementation of all the sub-components in the system was simulated and/or tested thoroughly to verify the function. Moreover, some of the combined testing revealed some implementation changes necessary in the power stage output. Despite the numerous achievements and contributions made throughout this project, it is important to acknowledge its limitations. Factors such as time constraints, limited resources, and external factors beyond our control may have impacted the scope and depth of our study. However, the findings and insights presented in this report should provide a solid foundation for future research and further exploration of the topic. Due to time limitations, experiments were focused on module testing each part of the system independently. It is therefore difficult to compare it with prior research directly. The system was designed around a control system based on the Zynq 7000 SoC, which contains both the configured ultrasound pulser logic controller and the Python-based programmable interface. The transmitter, which includes the ultrasound pulser and the power stage, was tested with the switch, and while the transmitting was successful, a received reflected signal could not be found. It is believed that the received switch was sinking into the on-board load of the power stage. On the receiver, which includes the bandpass filter, preamplifier, quadrature demodulator, sample and hold circuit, and active filter, were also verified functioning as intended in isolated experiments. The implementation of the analogue frontend can result in a more compact commercial product. However, more work is required before a working single-device prototype can be build and evaluated.

## 6.1 Future Work

The system was fully validated, although in module testing. The absence of reflected signals, when using the power stage, should be further investigated. In the future, it is desired to perform a complete system test and evaluate the performance using the physiological simulator. After a successful experiment, a single PCB layout can be completed from the CAD work already done for the documentation in Altium Designer. Another future work would be further implementing a continuous data acquisition with a

spectral image, thus fully displaying the sonographic data in the DSP.

# Bibliography


[1] S. Satomura, T. Yoshida, M. Mori, Y. Nimura, G.-i. Hikita, S. Takagishi, and K. Nakanishi, "Analysis of heart motion with ultrasonic Doppler method and its clinical application," *American Heart Journal*, vol. 61, pp. 61–75, 1 January 1961, ISSN: 00028703. DOI: 10.1016/0002-8703(61)90517-8. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/0002870361905178.

[2] D. Baker, "Pulsed Ultrasonic Doppler Blood-Flow Sensing," *IEEE Transactions on Sonics and Ultrasonics*, vol. 17, pp. 170–184, 3 Jul. 1970, ISSN: 0018-9537. DOI: 10.1109/t-su.1970.29558. [Online]. Available: http://ieeexplore.ieee.org/document/1538563/.

[3] K. K. Shung, R. A. Sigelmann, and J. M. Reid, "Scattering of Ultrasound by Blood," *IEEE Transactions on Biomedical Engineering*, vol. Bme-23, pp. 460–467, 6 November 1976, ISSN: 0018-9294. DOI: 10.1109/tbme.1976.324604. [Online]. Available: http://ieeexplore.ieee.org/document/4121084/.

[4] F. S. Schlindwein, M. J. Smith, and D. H. Evans, "Spectral analysis of Doppler signals and computation of the normalised first moment in real time using a digital signal processor," *Medical & Biological Engineering & Computing*, vol. 26, pp. 228–232, 2 March 1988, ISSN: 0140-0118. DOI: 10.1007/bf02442271. [Online]. Available: http://link.springer.com/10.1007/BF02442271.

[5] K. K. Shung and G. A. Thieme, *Ultrasonic scattering in biological tissues*. CRC Press, 1993, p. 499, ISBN: 9780849365683. [Online]. Available: https://www.routledge.com/Ultrasonic-Scattering-in-Biological-Tissues/Shung-Thieme/p/book/9780849365683.

[6] T. Hall, "An improved wall filter for flow imaging of low velocity flow," vol. 3, IEEE, October 1994, 1701–1704 vol.3, ISBN: 0-7803-2012-3. DOI: 10.1109/ultsym.1994.401918. [Online]. Available: http://ieeexplore.ieee.org/document/401918/.

[7] L. Bessi, F. Guidi, F. Gucci, C. Atzeni, and P. Tortoli, "Single-channel digital demodulation of ultrasound bandpass signals for flow-imaging applications," in Springer, Boston, MA, 1995, pp. 331–335. DOI: 10.1007/978-1-4615-1943-0_34. [Online]. Available: http://link.springer.com/10.1007/978-1-4615-1943-0_34.

[8] J. A. Jensen, "An Analysis of Pulsed Wave Ultrasound Systems for Blood Velocity Estimation," *Acoustical Imaging*, L. T. Piero and Masotti, Eds., pp. 377–384, 1996. DOI: 10.1007/978-1-4419-8772-3_61. [Online]. Available: http://link.springer.com/10.1007/978-1-4419-8772-3%5C%5F61.

[9] I. Güler and Y. Savaş, "Design parameters of pulsed wave ultrasonic Doppler blood flowmeter," *Journal of Medical Systems*, vol. 22, pp. 273–278, 4 1998, ISSN: 01485598. DOI: 10.1023/a:1022665802010.

[10] P. Wells, "Doppler studies of the vascular system," *European Journal of Ultrasound*, vol. 7, pp. 3–8, 1 February 1998, ISSN: 09298266. DOI: 10.1016/s0929-8266(98)00006-8. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0929826698000068.

[11] P. J. Fish, "Ultrasonic investigation of blood flow," *Proceedings of the Institution of Mechanical Engineers, Part H: Journal of Engineering in Medicine*, vol. 213, pp. 169–180, 3 March 1999, Pmid: 10420772. DOI: 10.1243/0954411991534898. [Online]. Available: https://doi.org/10.1243/0954411991534898.

[12] P. R. Hoskins, "A review of the measurement of blood velocity and related quantities using Doppler ultrasound," *Proceedings of the Institution of Mechanical Engineers, Part H: Journal of Engineering in Medicine*, vol. 213, pp. 391–400, 5 1999, Pmid: 10581966. DOI: 10.1243/0954411991535004. [Online]. Available: https://doi.org/10.1243/0954411991535004.

[13] T. Jansson, H. W. Peresson, and K. Lindström, "Estimation of blood perfusion using ultrasound," *Proceedings of the Institution of Mechanical Engineers, Part H: Journal of Engineering in Medicine*, vol. 213, pp. 193–193, 3 March 1999, ISSN: 0954-4119. DOI: 10.1243/0954411991534915. [Online]. Available: http://journals.sagepub.com/doi/10.1243/0954411991534915.

[14] J. A. Jensen, "Algorithms for estimating blood velocities using ultrasound," *Ultrasonics*, vol. 38, pp. 358–362, 1-8 March 2000, ISSN: 0041624x. DOI: 10.1016/s0041-624x(99)00127-4. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0041624X99001274.

[15] P. Munk, "Estimation of blood velocity vectors using ultrasound," Technical University of Denmark, 2000. [Online]. Available: https://findit.dtu.dk/en/catalog/537f0d537401dbcc1200be04.

[16] E. Picano, "Sustainability of medical imaging," *BMJ*, vol. 328, pp. 578–580, 7439 March 2004, ISSN: 0959-8138. DOI: 10.1136/bmj.328.7439.578. [Online]. Available: https://www.bmj.com/lookup/doi/10.1136/bmj.328.7439.578.

[17] A. Erguri, Y. Huang, X. Zhuang, O. Oralkan, G. Yarahoglu, and B. Khuri-Yakub, "Capacitive micromachined ultrasonic transducers: Fabrication technology," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 52, no. 12, pp. 2242–2258, 2005. DOI: 10.1109/TUFFC.2005.1563267.

[18] *AD8021 Low Noise, High Speed Amplifier for 16-Bit Systems*, Analog Devices, Inc., May 2006. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/AD8021.pdf (visited on April 10, 2023).

[19] E. Chérin, R. Williams, A. Needles, G. Liu, C. White, A. S. Brown, Y.-Q. Zhou, and F. S. Foster, "Ultrahigh frame rate retrospective ultrasound microimaging and blood flow visualization in mice in vivo," *Ultrasound in Medicine & Biology*, vol. 32, pp. 683–691, 5 May 2006, ISSN: 03015629. DOI: 10.1016/j.ultrasmedbio.2005.12.015. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0301562906000172.

[20] *EL7104 High Speed, Single Channel, Power MOSFET Driver*, Renesas Electronics, Jun. 2006. [Online]. Available: https://www.renesas.com/us/en/document/dst/el7104-datasheet.

[21] I. K. H. Tsang, B. Y. S. Yiu, D. K. H. Cheung, H. C. T. Chiu, C. C. P. Cheung, and A. C. H. Yu, "Design of a multi-channel pre-beamform data acquisition system for an ultrasound research scanner," IEEE, Sep. 2009, pp. 1840–1843, ISBN: 978-1-4244-4389-5. DOI: 10.1109/ultsym.2009.5441869. [Online]. Available: http://ieeexplore.ieee.org/document/5441869/.

[22] E. A. Aydın and İ. Güler, "Design Of Pic-controlled Pulsed Ultrasonic Transmitter For Measuring Gingiva Thickness," *Instrumentation Science & Technology*, vol. 38, pp. 411–420, 6 October

2010, ISSN: 1073-9149. DOI: 10.1080/10739149.2010.509149. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/10739149.2010.509149.

[23] P. R. Hoskins, "Haemodynamics and blood flow measured using ultrasound imaging," *Proceedings of the Institution of Mechanical Engineers, Part H: Journal of Engineering in Medicine*, vol. 224, pp. 255–271, 2 February 2010, Pmid: 20349818, ISSN: 0954-4119. DOI: 10.1243/09544119jeim572. [Online]. Available: http://journals.sagepub.com/doi/10.1243/09544119JEIM572.

[24] N. Matsuoka, D.-G. Paeng, R. Chen, H. Ameri, W. Abdallah, Q. Zhou, A. Fawzi, K. K. Shung, and M. Humayun, "Ultrasonic Doppler measurements of blood flow velocity of rabbit retinal vessels using a 45-MHz needle transducer," *Graefe's Archive for Clinical and Experimental Ophthalmology*, vol. 248, pp. 675–680, 5 2010, ISSN: 1435-702x. DOI: 10.1007/s00417-009-1298-9. [Online]. Available: https://doi.org/10.1007/s00417-009-1298-9.

[25] *TX810 8-Channel, Programmable T/R Switch for Ultrasound*, Texas Instruments, April 2010. [Online]. Available: https://www.ti.com/lit/gpn/tx810.

[26] K. L. Hansen, M. B. Nielsen, and J. A. Jensen, "In-vivo studies of new vector velocity and adaptive spectral estimators in medical ultrasound," 2011. [Online]. Available: https://orbit.dtu.dk/en/publications/in-vivo-studies-of-new-vector-velocity-and-adaptive-spectral-esti.

[27] J. A. Jensen, S. I. Nikolov, J. Udesen, *et al.*, "Recent advances in blood flow vector velocity imaging," IEEE, October 2011, pp. 262–271, ISBN: 978-1-4577-1252-4. DOI: 10.1109/ultsym.2011.0064. [Online]. Available: http://ieeexplore.ieee.org/document/6293382/.

[28] A. Sarvazyan, T. J. Hall, M. W. Urban, M. Fatemi, S. R. Aglyamov, and B. S. Garra, "An Overview of Elastography-An Emerging Branch of Medical Imaging," *Current Medical Imaging Reviews*, vol. 7, pp. 255–282, 4 November 2011, ISSN: 15734056. DOI: 10.2174/157340511798038684. [Online]. Available: http://www.eurekaselect.com/openurl/content.php?genre=article%5C&issn=1573-4056%5C&volume=7%5C&issue=4%5C&spage=255.

[29] T. A. Bigelow, A. Kellar, A. Tapia, F. Ferrer, J. Batcheler, J. Driggs, and R. Page, "Design Document for a Pulse Echo Ultrasound System," Iowa State University, December 2012.

[30] C.-C. Huang, P.-Y. Lee, P.-Y. Chen, and T.-Y. Liu, "Design and implementation of a smartphone-based portable ultrasound pulsed-wave doppler device for blood flow measurement," *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*, vol. 59, pp. 182–188, 1 January 2012, ISSN: 0885-3010. DOI: 10.1109/tuffc.2012.2171. [Online]. Available: http://ieeexplore.ieee.org/document/6138742/.

[31] C. A. Winckler, P. R. Smith, D. M. J. Cowell, O. Olagunju, and S. Freear, "The design of a high speed receiver system for an ultrasound array research platform," Ieee, October 2012, pp. 1481–1484, ISBN: 978-1-4673-4562-0. DOI: 10.1109/ultsym.2012.0370. [Online]. Available: http://ieeexplore.ieee.org/document/6562380/.

[32] J. A. Jensen, *Estimation of Blood Velocities Using Ultrasound: A Signal Processing Approach*, Third Edition. Department of Electrical Engineering, Technical University of Denmark, August 2013, ISBN: 9780521464840.

[33] E. Pirnia, "Blood velocities estimation using ultrasound," Lund University, Sep. 2013.

[34] *AD783 Complete Very High Speed Sample-and-Hold Amplifier*, Analog Devices, October 2014. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/AD783.pdf.

[35] J. E. Browne, "A review of doppler ultrasound quality assurance protocols and test devices," *Physica Medica*, vol. 30, no. 7, pp. 742–751, 2014, ISSN: 1120-1797. DOI: https://doi.org/10.1016/j.

ejmp.2014.08.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1120179714005171.

[36] *MD1213DB1 High Speed ±100V 2A Pulser*, Supertex, Inc., March 2014. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/md1213db1.pdf.

[37] R. K. Sagdiev, E. S. Denisov, Y. K. Evdokimov, M. G. Fazlyyyakhmatov, and N. F. Kashapov, "Phased array based ultrasound scanning system development," *IOP Conference Series: Materials Science and Engineering*, vol. 69, p. 012 012, 1 December 2014, ISSN: 1757-899x. DOI: 10.1088/1757-899x/69/1/012012. [Online]. Available: https://iopscience.iop.org/article/10.1088/1757-899X/69/1/012012.

[38] T. L. Szabo, *Diagnostic Ultrasound Imaging: Inside Out*, Second Edition. Elsevier, 2014, ISBN: 9780123964878. DOI: 10.1016/c2011-0-07261-7. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/C20110072617.

[39] K. K. Shung, *Diagnostic Ultrasound: Imaging and Blood Flow Measurements*, Second Edition. CRC Press, December 2015, ISBN: 978-1-4665-8264-4.

[40] L. Svilainis, A. Chaziachmetovas, and V. Dumbrava, "Half bridge topology 500 V pulser for ultrasonic transducer excitation," *Ultrasonics*, vol. 59, pp. 79–85, May 2015, ISSN: 0041624x. DOI: 10.1016/j.ultras.2015.01.014. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0041624X15000165.

[41] *AD8332 Ultralow Noise VGAs with Preamplifier and Programmable $R_{IN}$*, Analog Devices, May 2016. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/AD8331_8332_8334.pdf.

[42] *AD8333 DC to 50 MHz, Dual I/Q Demodulator and Phase Shifter*, Analog Devices, May 2016. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/AD8333.pdf.

[43] P. Govindan, B. Wang, P. Ravi, and J. Saniie, "Hardware and software architectures for computationally efficient three-dimensional ultrasonic data compression," *IET Circuits, Devices & Systems*, vol. 10, pp. 54–61, 1 January 2016, ISSN: 1751-858x. DOI: 10.1049/iet-cds.2015.0083. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1049/iet-cds.2015.0083.

[44] B. Wang and J. Saniie, "Ultrasonic Signal Acquisition and Processing platform based on Zynq SoC," vol. 2016-August, IEEE, May 2016, pp. 0448–0451, ISBN: 978-1-4673-9985-2. DOI: 10.1109/eit.2016.7535282. [Online]. Available: http://ieeexplore.ieee.org/document/7535282/.

[45] *MD1213 High-Speed Dual-MOSFET Driver*, Microchip Technology, Inc., Jun. 2017. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/20005713B.pdf.

[46] *PYNQ-Z1 Board Reference Manual*, Digilent, Inc., April 2017. [Online]. Available: https://digilent.com/reference/_media/reference/programmable-logic/pynq-z1/pynq-rm.pdf (visited on April 10, 2023).

[47] S. Ricci and V. Meacci, "Data-adaptive coherent demodulator for high dynamics pulse-wave ultrasound applications," *Electronics 2018, Vol. 7, Page 434*, vol. 7, p. 434, 12 December 2018, ISSN: 2079-9292. DOI: 10.3390/ELECTRONICS7120434. [Online]. Available: https://www.mdpi.com/2079-9292/7/12/434/htm%20https://www.mdpi.com/2079-9292/7/12/434.

[48] *BPF-C4R5+ Bandpass Filter 50Ω 2 to 7 MHz*, Mini-Circuits, August 2019. [Online]. Available: https://www.minicircuits.com/pdfs/BPF-C4R5+.pdf (visited on April 10, 2023).

[49] I. Song, J. Yoon, J. Kang, M. Kim, W. S. Jang, N. Y. Shin, and Y. Yoo, "Design and implementation of a new wireless carotid neckband doppler system with wearable ultrasound sensors: Preliminary

results," *Applied Sciences 2019, Vol. 9, Page 2202*, vol. 9, p. 2202, 11 May 2019, ISSN: 2076-3417. DOI: 10.3390/APP9112202. [Online]. Available: https://www.mdpi.com/2076-3417/9/11/2202.

[50] B. Wang and J. Saniie, "A High Performance Ultrasonic System for Flaw Detection," IEEE, October 2019, pp. 840–843, ISBN: 978-1-7281-4596-9. DOI: 10.1109/ultsym.2019.8926280. [Online]. Available: https://ieeexplore.ieee.org/document/8926280/.

[51] H. Ding, D. Yang, J. Xu, X. Chen, X. Le, Y. Wang, L. Lin, and J. Xie, "Pulsed Wave Doppler Ultrasound Using 3.7 MHz Pmuts Toward Wearable Blood Flow Measurements," IEEE, January 2020, pp. 400–403, ISBN: 978-1-7281-3581-6. DOI: 10.1109/mems46641.2020.9056430. [Online]. Available: https://ieeexplore.ieee.org/document/9056430/.

[52] B. Jana, R. Biswas, P. K. Nath, G. Saha, and S. Banerjee, "Smartphone-Based Point-of-Care System Using Continuous-Wave Portable Doppler," *IEEE Transactions on Instrumentation and Measurement*, vol. 69, pp. 8352–8361, 10 October 2020, ISSN: 0018-9456. DOI: 10.1109/tim.2020.2987654. [Online]. Available: https://ieeexplore.ieee.org/document/9064842/.

[53] *MD0101 High Voltage Protection T/R Switch with Clamp Diodes*, Microchip Technology, Inc., May 2020. [Online]. Available: https://ww1.microchip.com/downloads/aemDocuments/documents/APID/ProductDocuments/DataSheets/MD0101-High-Voltage-Protection-TR-Switch-with-Clamp-Diodes-Data-Sheet-20005916A.pdf.

[54] J. Cumps. (Jun. 13, 2021), [Online]. Available: https://community.element14.com/technologies/fpga-group/b/blog/posts/vhdl-pwm-generator-with-dead-time-the-design (visited on May 9, 2023).

[55] H. Ding, D. Yang, M. Qu, *et al.*, "A Pulsed Wave Doppler Ultrasound Blood Flowmeter by PMUTs," *Journal of Microelectromechanical Systems*, vol. 30, pp. 680–682, 5 October 2021, ISSN: 1941-0158. DOI: 10.1109/jmems.2021.3103756. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9515180.

[56] *ISL55111 Dual, High Speed MOSFET Driver*, Renesas Electronics, April 2021. [Online]. Available: https://www.renesas.com/us/en/document/dst/isl55110-isl55111-datasheet.

[57] M. Winder, A. J. Owczarek, J. Chudek, J. Pilch-Kowalczyk, and J. Baron, "Are We Overdoing It? Changes in Diagnostic Imaging Workload during the Years 2010–2020 including the Impact of the SARS-CoV-2 Pandemic," *Healthcare*, vol. 9, p. 1557, 11 November 2021, ISSN: 2227-9032. DOI: 10.3390/healthcare9111557. [Online]. Available: https://www.mdpi.com/2227-9032/9/11/1557.

[58] "Heart disease facts." (October 2022), [Online]. Available: https://www.cdc.gov/heartdisease/facts.htm (visited on February 23, 2023).

[59] M. Omura, R. Nagaoka, K. Yagi, K. Yoshida, T. Yamaguchi, and H. Hasegawa, "Characterization of blood mimicking fluid with ultrafast ultrasonic and optical image velocimeters," *Japanese Journal of Applied Physics*, vol. 61, Sg1067, Sg Jul. 2022, Publisher Copyright: © 2022 The Japan Society of Applied Physics., ISSN: 0021-4922. DOI: 10.35848/1347-4065/ac4ea9. [Online]. Available: https://iopscience.iop.org/article/10.35848/1347-4065/ac4ea9.

[60] "Altium Designer - PCB design software." (February 2023), [Online]. Available: https://www.altium.com/altium-designer.

[61] J. Cumps, Private communication, Talked about implementation of PWM generators on PYNQ hardware platforms, 2023.

[62] J. Hinrichs. "Firmware Repository." (March 2023), [Online]. Available: https://github.com/s163555/doppler_pwm (visited on April 12, 2023).

[63] J. Hinrichs. “Ultrasound AFE PCB.” (April 2023), [Online]. Available: https://github.com/s163555/ultrasoundsystem_pcb (visited on May 9, 2023).

[64] J. Hinrichs. “Xilinx Vivado Project.” (May 2023), [Online]. Available: https://github.com/s163555/pynq_ultrasound_pulser (visited on May 9, 2023).

[65] L. Li, L. Zhao, R. Hassan, and H. Ren, “Review on Wearable System for Positioning Ultrasound Scanner,” *Machines*, vol. 11, p. 325, 3 February 2023, ISSN: 2075-1702. DOI: 10.3390/machines11030325. [Online]. Available: https://www.mdpi.com/2075-1702/11/3/325.

[66] *Sample and hold Sampling Zero-order hold First-order hold Digital-to-analog converter*. [Online]. Available: https://www.pngwing.com/en/free-png-zueso (visited on April 10, 2023).

[4]

[4]

헌릭스 예페 19910722 오르후스 덴마크 대전, 대한민국

Technical University of Denmark 전기및전자공학부 (학사)

Tokyo Institute of Technology 전기및전자공학부 (교환학생)

Technical University of Denmark 전기및전자공학부 (석사)

한국과학기술원 전기및전자공학부 (석사)

Welltec A/S 테스트 공학자

Technical University of Denmark 국립 우주 연구소 조교

# Appendix A

# Gantt Chart

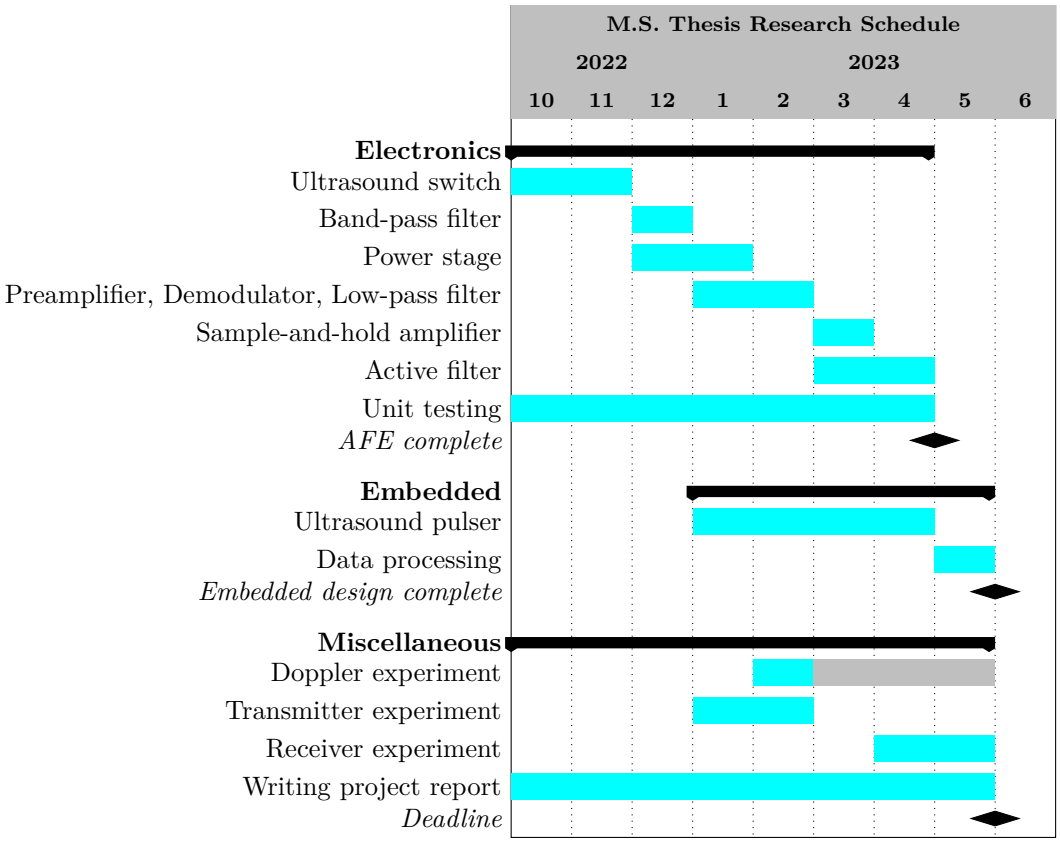



Figure A.1: Gantt chart of project schedule

# Appendix B

# Source Code

Latest sources and projects can be found at [62]–[64].

## B.1 Microcontroller Code

Listing B.1: main.c

```
/*
 * Copyright (c) 2023 JH
 *
 * SPDX-License-Identifier: Apache-2.0
 */

/**
 * @file Doppler PWM app
 */

#include <zephyr/kernel.h>
#include <zephyr/sys/printk.h>
#include <zephyr/device.h>
#include <zephyr/drivers/pwm.h>

static const struct pwm_dt_spec pwm_t1 = PWM_DT_SPEC_GET(DT_ALIAS(pwmch1));
//static const struct pwm_dt_spec pwm_ch1n = PWM_DT_SPEC_GET(DT_ALIAS(pwmch1n));
static const struct pwm_dt_spec pwm_t2 = PWM_DT_SPEC_GET(DT_ALIAS(pwmch2));
static const struct pwm_dt_spec pwm_t4 = PWM_DT_SPEC_GET(DT_ALIAS(pwmch4));
#define SLEEP_MSEC        100U

struct pwmcontrol {
        const struct device *svpwm;
}

void main(void)
{
        printk("Test af main - %s\r\n", CONFIG_BOARD);
        uint32_t pulse_width_t1 = 152U;
        uint32_t pulse_width_t2 = 1515U;
```

```
        uint32_t pulse_width_t4 = 50000U;
        int error = 0;

        printk("PWM-firmware API\n");
        if (!device_is_ready(pwm_t1.dev)) {
                printk("Error: PWM device %s is not ready\n",
                       pwm_t1.dev->name);
                return;
        }

        if (!device_is_ready(pwm_t2.dev)) {
                printk("Error: PWM device %s is not ready\n",
                       pwm_t2.dev->name);
                return;
        }
        if (!device_is_ready(pwm_t4.dev)) {
                printk("Error: PWM device %s is not ready\n",
                       pwm_t4.dev->name);
                return;
        }*/
        printk("Entering while loop\r\n");
        while (1) {

                error = pwm_set_pulse_dt(&pwm_t1, pulse_width_t1);
                if (error) {
                        printk("Error ch1 %d: failed to set pulse width\n", error);
                        return;
                }
                error = pwm_set_pulse_dt(&pwm_t2, pulse_width_t2);
                if (error) {
                        printk("Error ch1n %d: failed to set pulse width\n", error);
                        return;
                }
                error = pwm_set_pulse_dt(&pwm_t4, pulse_width_t4);
                if (error) {
                        printk("Error ch1n %d: failed to set pulse width\n", error);
                        return;
                }
                for(int i=0;i<1000;i++) {
                        k_sleep(K_MSEC(SLEEP_MSEC));
                }
                printk("Loop iteration\r\n");
        }
}
```

## B.2 Field Programmable Gate Array Code

# 1 PWM with generic dead band and duty cycle resolution

[5]:
```
from pynq import Overlay
ol=Overlay("pwm_ultrasound_pulser.bit")
from pynq import MMIO
RANGE = 8 # Number of bytes; 8/4 = 2x 32-bit locations which is all we need for␣
 ↪this example

duty_address = ol.ip_dict['axi_gpio_duty']['phys_addr']
duty_register = MMIO(duty_address, RANGE)
# Write 0x00 to the tri-state register at offset 0x4 to configure the IO as␣
 ↪outputs.
duty_register.write(0x4, 0x0) # Write 0x0 to location 0x4; Set tri-state to␣
 ↪output

band_address = ol.ip_dict['axi_gpio_band']['phys_addr']
band_register = MMIO(band_address, RANGE)
# Write 0x00 to the tri-state register at offset 0x4 to configure the IO as␣
 ↪outputs.
band_register.write(0x4, 0x0) # Write 0x0 to location 0x4; Set tri-state to␣
 ↪output

flags_address = ol.ip_dict['axi_gpio_flags']['phys_addr']
flags_register = MMIO(flags_address, RANGE)
# Write 0x00 to the tri-state register at offset 0x4 to configure the IO as␣
 ↪outputs.
flags_register.write(0x4, 0x0) # Write 0x0 to location 0x4; Set tri-state to␣
 ↪output

start_delay_address = ol.ip_dict['axi_gpio_start_delay']['phys_addr']
start_delay_register = MMIO(start_delay_address, RANGE)
# Write 0x00 to the tri-state register at offset 0x4 to configure the IO as␣
 ↪outputs.
start_delay_register.write(0x4, 0x0) # Write 0x0 to location 0x4; Set tri-state␣
 ↪to output

train_length_address = ol.ip_dict['axi_gpio_train_length']['phys_addr']
```

Figure B.1: Jupyter Notebook running on PYNQ Z1 1

```
train_length_register = MMIO(train_length_address, RANGE)
# Write 0x00 to the tri-state register at offset 0x4 to configure the IO as␣
 ↪outputs.
train_length_register.write(0x4, 0x0) # Write 0x0 to location 0x4; Set␣
 ↪tri-state to output

gate_delay_address = ol.ip_dict['axi_gpio_gate_delay']['phys_addr']
gate_delay_register = MMIO(gate_delay_address, RANGE)
# Write 0x00 to the tri-state register at offset 0x4 to configure the IO as␣
 ↪outputs.
gate_delay_register.write(0x4, 0x0) # Write 0x0 to location 0x4; Set tri-state␣
 ↪to output

def duty(duty):
    duty_register.write(0x00, duty)

def band(band):
    band_register.write(0x00, band)

def dutypct(duty):
    duty_register.write(0x00, round((0x1F*2)/(100/duty)))

def fire():
    flags_register.write(0x00, 1) # bit 0
    flags_register.write(0x00, 0)

def prime():
    flags_register.write(0x00, 2) # bit 1
    flags_register.write(0x00, 0)

def startdelay(startdelay):
    start_delay_register.write(0x0, startdelay);

def trainlength(trainlength):
    train_length_register.write(0x0, trainlength);

def gatedelay(gatedelay):
    gate_delay_register.write(0x0, gatedelay);
```

[7]:
```
duty(0x1F)
```

[8]:
```
dutypct(50)
```

[52]:
```
band(4)
```

[34]:
```
startdelay(20)
```

Figure B.2: Jupyter Notebook running on PYNQ Z1 2

[98]:
```
trainlength(127)
```

[ ]:
```
gatedelay(20)
```

[319]:
```
prime()
```

[329]:
```
fire()
```

[43]:
```
band(0)
startdelay(20)
trainlength(96)
gatedelay(20)
prime()
fire()
```

[ ]:
```
band(10)
startdelay(200)
trainlength(294)
gatedelay(200)
prime()
fire()
```

[ ]:
```
band(10)
startdelay(2000)
trainlength(294)
gatedelay(1000)
try:
    while True:
        prime()
        fire()
except KeyboardInterrupt:
    pass
```

[ ]:

[ ]:

Figure B.3: Jupyter Notebook running on PYNQ Z1 3

In [1]:
```
#from pynq.overlays.base import BaseOverlay
#from pynq.lib.arduino.arduino_analog import Arduino_Analog
#import matplotlib.pyplot as plt
#from time import sleep

#ol = BaseOverlay('base.bit')
#xadc = Arduino_Analog(ol.ARDUINO, [5])
# Select the sampling interval in the unit of ms.
#xadc.set_log_interval_ms(1)
#xadc.start_log()
# Wait for stacking data.
#sleep(1)
#v = xadc.get_log()
# print(v[0])
```

In [2]:
```
import time
from pynq.overlays.base import BaseOverlay
base = BaseOverlay("base.bit")
from pynq.lib.arduino import Arduino_Analog
from pynq.lib.arduino import ARDUINO_GROVE_A1

analog0 = Arduino_Analog(base.ARDUINO,ARDUINO_GROVE_A1)
a = []
t_start = time.time()
t_end = time.time() + 1
while time.time() < t_end:
    a.append(analog0.read()[0])
```

In [4]:
```
#%matplotlib inline
import matplotlib.pyplot as plt
from matplotlib.legend_handler import HandlerLine2D

plt.rcParams['figure.figsize'] = [10, 12]
plt.rcParams['figure.dpi'] = 100 # 200 e.g. is really fine, but slower

#line1, = plt.plot(range(len(a)), a,
#                  'ro', label="Voltage on ADC")
#plt.title('Analog to Digital Converter')
#plt.axis([0, len(a), 0.0, 3.3])
#plt.xlabel('Sample number')
#plt.ylabel('Voltage')
#plt.legend(loc=4,bbox_to_anchor=(1, -0.3),
#           ncol=2, borderaxespad=0.,
#           handler_map={line1: HandlerLine2D(numpoints=1),
#                        })
#plt.show()
#plt.savefig('adc.pdf', bbox_inches='tight')
```

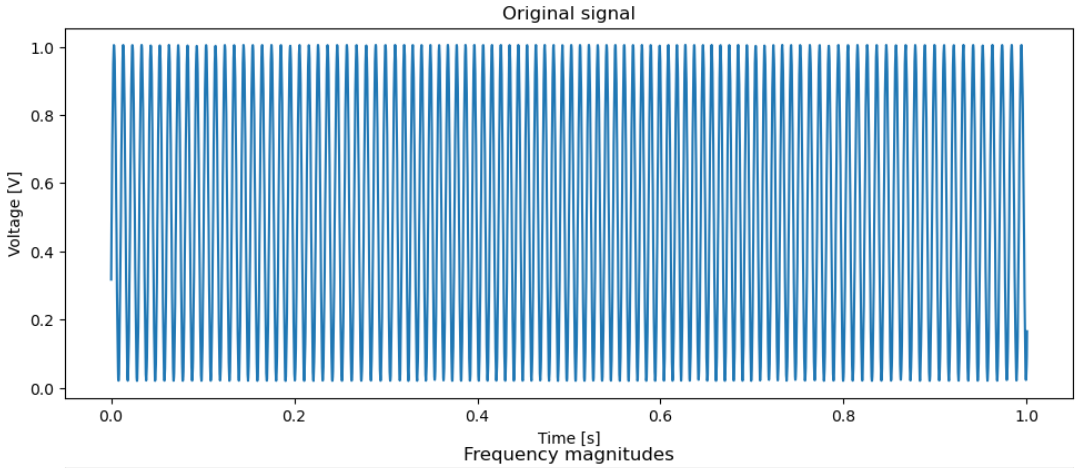
Figure B.4: XADC running on PYNQ Z1 1

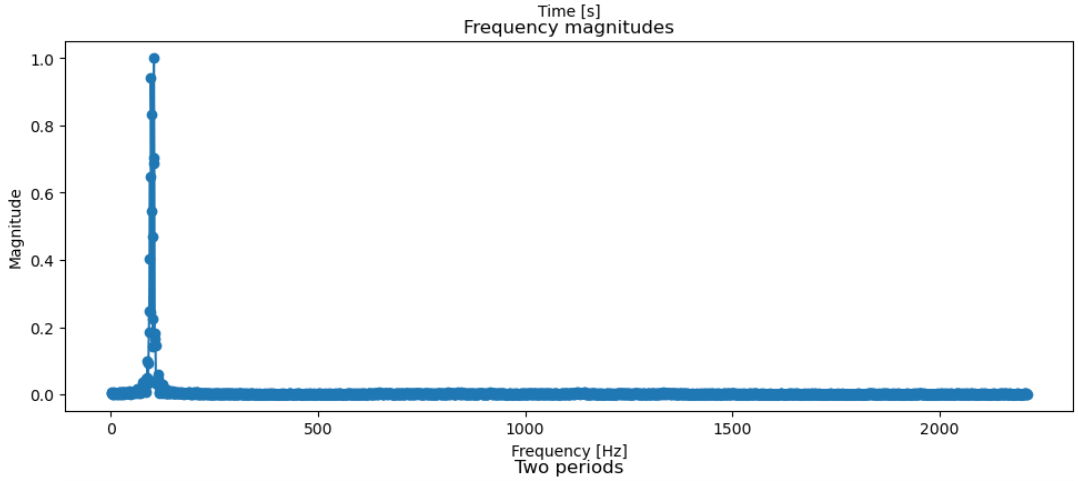
In [8]:

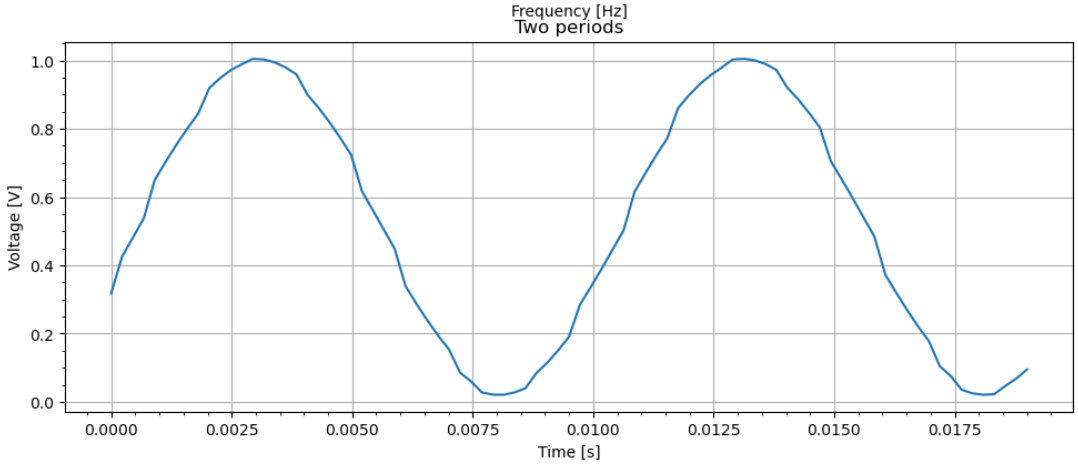
```
import numpy as np
import matplotlib.pyplot as plt
fs = round(1/((t_end-t_start)/len(a)))
print(f"Calculated samplerate fs = {fs} kS/s")
t = np.linspace(0, t_end-t_start, len(a))

fig, ax = plt.subplots(3, 1)
ax = ax.flatten()
ax[0].plot(t, a)
ax[0].set_title("Original signal")
ax[0].set_xlabel("Time [s]")
ax[0].set_ylabel("Voltage [V]")
plt.tight_layout()
plt.grid()
plt.minorticks_on()
#plt.show()

a_fft = np.fft.fft(a)           # Original FFT
a_fft = a_fft[1:]                 # Delete DC component
a_fft = a_fft[:round(len(t)/2)] # First half ( pos freqs )
a_fft = np.abs(a_fft)           # Absolute value of magnitudes
a_fft = a_fft/max(a_fft)        # Normalized so max = 1

freq_x_axis = np.linspace(1, fs/2, len(a_fft))
ax[1].plot(freq_x_axis, a_fft, "o-")
ax[1].set_title("Frequency magnitudes")
ax[1].set_xlabel("Frequency [Hz]")
ax[1].set_ylabel("Magnitude")
#plt.grid()
#plt.minorticks_on()
#plt.show()


f_loc = np.argmax(a_fft) # Finds the index of the max
f_val = freq_x_axis[f_loc] # The strongest frequency value

samplenums = round(2*(1/f_val)*fs)

ax[2].plot(t[0:samplenums],a[0:samplenums])
ax[2].set_title("Two periods")
ax[2].set_xlabel("Time [s]")
ax[2].set_ylabel("Voltage [V]")
#plt.grid()
#plt.minorticks_on()

plt.show()

print(f"The largest frequency component is f = {f_val}")
plt.savefig('fft.pdf', bbox_inches='tight')
```

```
Calculated samplerate fs = 4422 kS/s
```

Figure B.5: XADC running on PYNQ Z1 2




```
The largest frequency component is f = 103.95341474445952
<Figure size 1000x1200 with 0 Axes>
```

In [6]:
```
f0=5e6
f_d=200
theta=60*np.pi/180
c=1480
v=((f_val)*c)/(2*np.cos(theta)*f0)
print(f"The estimated velocity is v = {v}")
```

```
The estimated velocity is v = 0.030770210764360015
```

Figure B.6: XADC running on PYNQ Z1 3

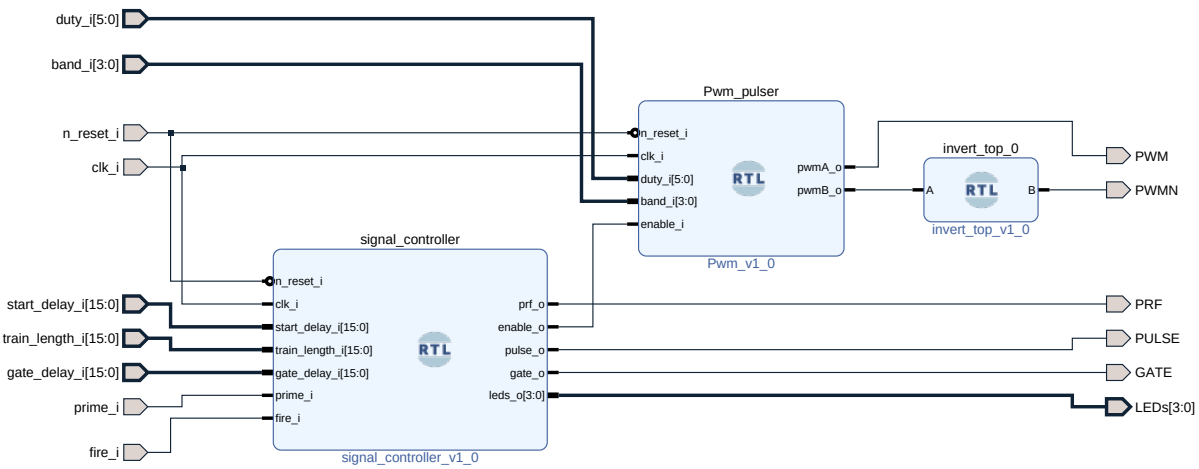



Figure B.7: Zynq Ultrasound Pulser block diagram

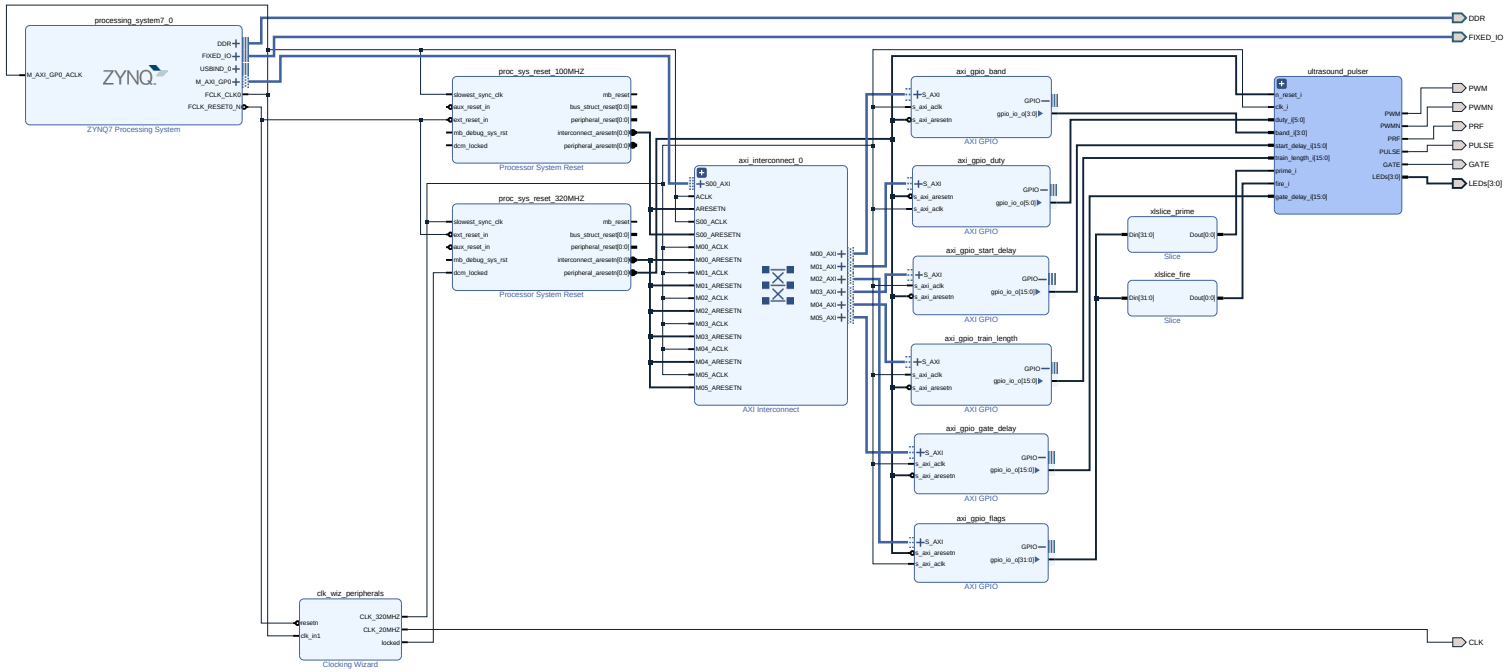



Figure B.8: ZYNQ Top Level block diagram

Listing B.2: Pwm.vhdl

```
1  --*********************************************************************
2  -- Module for generating repetitive pulses.
3  --*********************************************************************
4
5  library IEEE;
6  use IEEE.std_logic_1164.all;
7
8  package PulsePckg is
9
10     constant HI   : std_logic := '1';
11     constant LO   : std_logic := '0';
12     constant ONE  : std_logic := '1';
13
14     component Pwm is
15         generic (
16             resolution : integer := 6; -- bit width of the counter used for duty cycle
17             band_bits : integer := 4   -- bit width of the variable that holds the desired deadband
18         );
19         port (
20             n_reset_i : in  std_logic;          -- async reset
21             enable_i : in  std_logic;           -- enable the outputs
22             clk_i  : in  std_logic;             -- Input clock.
23             duty_i : in  std_logic_vector (resolution - 1 downto 0);      -- Duty-cycle input.
24             band_i : in  std_logic_vector (band_bits - 1  downto 0);      -- number of clock-ticks to
               ↪  keep both signals low before rising edge
25             pwmA_o  : out std_logic;            -- PWM output.
26             pwmB_o  : out std_logic             -- PWM output inverse.
27         );
28     end component;
29
30 end package;
31
32 --*********************************************************************
33 -- PWM module.
34 --*********************************************************************
35
36 library IEEE;
37 use IEEE.MATH_REAL.all;
38 use IEEE.std_logic_1164.all;
39 use IEEE.numeric_std.all;
40 use WORK.PulsePckg.all;
41
42 entity Pwm is
43     generic (
44         resolution : integer := 6;
45         band_bits : integer := 4
46     );
47     port (
48         n_reset_i : in  std_logic;          -- async reset
49         enable_i : in  std_logic;           -- enable the outputs
50         clk_i  : in  std_logic;             -- Input clock.
```

```
        duty_i : in  std_logic_vector (resolution - 1 downto 0);      -- Duty-cycle input.
        band_i : in  std_logic_vector (band_bits - 1 downto 0);      -- number of clock-ticks to keep
        ↪  both signals low before rising edge
        pwmA_o  : out std_logic;            -- PWM output.
        pwmB_o  : out std_logic             -- PWM output inverse.
    );
end entity;

architecture arch of Pwm is
    signal timer_r       : natural range 0 to 2**duty_i'length-1;
begin

    clocked: process(clk_i)
    begin

        if rising_edge(clk_i) then
            -- sync reset
            if n_reset_i = '0' then
                pwmA_o   <= LO;
                pwmB_o   <= LO;
                timer_r <= 0;
            else

                -- timer
                timer_r <= timer_r + 1;
                pwmA_o   <= LO;
                pwmB_o   <= LO;
                if enable_i = '0' then
                    pwmB_o   <= LO;
                    timer_r <= 0;
                else
                    -- output a
                    if timer_r <= unsigned(duty_i) and timer_r >= unsigned(band_i)  then
                        pwmA_o <= HI;
                    end if;
                    -- output b
                    if timer_r > to_integer(unsigned(band_i)) + to_integer(unsigned(duty_i)) then
                        pwmB_o <= HI;
                    end if;
                end if; -- enable

            end if; -- sync reset
        end if; -- rising_edge
    end process clocked;


end architecture;
```

Listing B.3: signalcontroller.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

```

```
4  package PulsePckg is
5
6      component signal_controller is
7          generic (
8              start_delay_resolution : integer := 6;  -- bit width of the counter used delaying pulse train
               ↪ start
9              train_length           : integer := 7;  -- bit width of the PWM burst train counter
10             gate_delay_resolution  : integer := 6;  -- bit width of the counter used delaying the gate
               ↪ signal start
11             counter_resolution     : integer := 7   -- big enought o hold the largest of above
12         );
13         port (
14             n_reset_i     : in   std_logic;                                                    -- async reset
15             clk_i         : in   std_logic;                                                    -- Input
               ↪ clock.
16             start_delay_i : in   std_logic_vector (start_delay_resolution - 1 downto 0);   -- delay
               ↪ before the pulse train.
17             train_length_i: in   std_logic_vector (train_length - 1 downto 0);             -- how many
               ↪ PWMs in a burst train.
18             gate_delay_i  : in   std_logic_vector (gate_delay_resolution - 1 downto 0);    -- delay
               ↪ before gate signal.
19             prime_i       : in   std_logic;                                                    -- allow cycle
               ↪ start
20             fire_i        : in   std_logic;                                                    -- start a
               ↪ cycle
21             prf_o         : out  std_logic;                                                    -- PRF
22             enable_o      : out  std_logic;                                                    -- enable pwm
23             pulse_o       : out  std_logic;                                                    -- PULSE
24             gate_o        : out  std_logic;                                                    -- GATE
25             leds_o        : out std_logic_vector (3 downto 0)
26         );
27     end component;
28
29 end package;
30 library IEEE;
31 use IEEE.STD_LOGIC_1164.ALL;
32 use ieee.numeric_std.all;
33
34
35 entity signal_controller is
36     generic (
37         train_length           : integer := 7;  -- bit width of the PWM burst train counter
38         start_delay_resolution : integer := 6;  -- bit width of the counter used delaying pulse train
           ↪ start
39         gate_delay_resolution  : integer := 6;  -- bit width of the counter used delaying the gate signal
           ↪ start
40         counter_resolution     : integer := 7   -- big enought o hold the largest of above
41     );
42     port (
43         n_reset_i     : in   std_logic;                                                    -- async reset
44         clk_i         : in   std_logic;                                                    -- Input clock.
45         start_delay_i : in   std_logic_vector (start_delay_resolution - 1 downto 0);   -- delay before
           ↪ the pulse train.
46         train_length_i: in   std_logic_vector (train_length - 1 downto 0);             -- how many PWMs
           ↪ in a burst train.
```

```
47          gate_delay_i  : in   std_logic_vector (gate_delay_resolution - 1 downto 0);     -- delay before
            ↪  gate signal.
48          prime_i       : in   std_logic;                                                 -- allow cycle
            ↪  start
49          fire_i        : in   std_logic;                                                 -- start a cycle
50          prf_o         : out  std_logic;                                                 -- PRF
51          enable_o      : out  std_logic;                                                 -- enable pwm
52          pulse_o       : out  std_logic;                                                 -- PULSE
53          gate_o        : out  std_logic;                                                 -- GATE
54          leds_o        : out std_logic_vector (3 downto 0)
55      );
56  end signal_controller;
57
58  architecture arch of signal_controller is
59      type states is (ready, delay, pulse, gate_delay, gate, done );
60      signal counter : integer range 0 to 2**counter_resolution-1;
61      signal state  : states;
62
63  begin
64      clocked: process(clk_i)
65      begin
66          if rising_edge(clk_i) then
67              -- sync reset
68              if n_reset_i = '0' then
69                  counter <= 0;
70                  state <= ready;
71                  -- reset output states
72                  prf_o <= '1';
73                  enable_o <= '0';
74                  pulse_o <= '0';
75                  gate_o <= '0';
76                  leds_o <= (others => '0');
77              else
78                  counter <= counter + 1;
79                  -- default output states
80                  prf_o <= '1';
81                  enable_o <= '0';
82                  pulse_o <= '0';
83                  gate_o <= '0';
84                  leds_o <= (others => '0');
85
86                  case State is
87                      when ready =>  -- start a pulse group as soon as the fire bit is set
88                          leds_o <= (0 => '1', others => '0');
89                          -- check if the caller has reset the prime bit
90                          if prime_i = '0' then
91                              -- if yes, start if the fire bit is set.
92                              if fire_i = '1' then
93                                  counter <= 0;
94                                  state <= delay;
95                              end if;
96                          end if;
97
98                      when delay =>
99                          leds_o <= (0 => '0', 1 => '1', others => '0');
```

```
100                         prf_o <= '0';
101                         if counter >= to_integer(unsigned(start_delay_i)) - 1 then
102                             counter <= 0;
103                             enable_o <= '1';
104                             state <= pulse;
105                         end if;
106 
107                     when pulse =>
108                         leds_o <= (0 => '1', 1 => '1', others => '0');
109                         enable_o <= '1';
110                         prf_o <= '0';
111                         if counter < to_integer(unsigned(train_length_i)) - 1 then  -- let the pulse
                            ↪  change sync with last edge of PWM
112                             pulse_o <= '1';
113                         elsif counter = to_integer(unsigned(train_length_i)) - 1 then
114                             pulse_o <= '1';
115                             counter <= 0;
116                             state <= gate_delay;
117                         end if;
118 
119                     when gate_delay =>
120                         leds_o <= (0 => '0', 1 => '0', 2 => '1', others => '0');
121                         if counter >= to_integer(unsigned(gate_delay_i)) - 1 then
122                             counter <= 0;
123                             state <= gate;
124                         end if;
125 
126                     when gate =>
127                         leds_o <= (0 => '1', 1 => '0', 2 => '1', others => '0');
128                         gate_o <= '1';
129                         if counter = to_integer(unsigned(train_length_i)) - 1 then
130                             counter <= 0;
131                             state <= done;
132                         end if;
133 
134                     when done =>
135                         leds_o <= (0 => '1', 1 => '1', 2 => '1', others => '0');
136                         -- check if the caller has reset the fire bit
137                         if fire_i = '0' then
138                             -- if yes, prime if the prime_bit is set.
139                             if prime_i = '1' then
140                                 counter <= 0;
141                                 state <= ready;
142                             end if;
143                         end if;
144                 end case;
145             end if; -- sync reset
146         end if; -- rising_edge
147 
148     end process clocked;
149 
150 end architecture;
```

Listing B.4: inv_top.vhdl

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity invert_top is
    Port ( A : in  STD_LOGIC;
           B : out  STD_LOGIC);
end invert_top;

architecture Behavioral of invert_top is
begin
    -- invert the signal from the push button switch and route it to the LED
    B <= not A;
end Behavioral;
```

## B.3 MATLAB Scripts

Listing B.5: US Simulation Master Script.m

```m
%% CIRCUIT: PA
clear; close all; clc;
path_LTSpice = '"C:/Program Files/LTC/LTspiceXVII/XVIIx64.exe"';
circuit_path = 'C:/Users/Jeppe\OneDrive - Danmarks Tekniske Universitet/10.
↪  semester/MSc_Thesis_Teams/Simulation/';
circuit_name = 'Tx_PA';
command = ' -Run -ascii -b ';
%% Simulate and import data
% Run LTSpice simulation
command = [path_LTSpice command '"' circuit_path circuit_name '.asc"'];
dos(command);

% Import data
raw_data = LTspice2Matlab([circuit_path circuit_name '.raw']);

% Display variables
fprintf('var \t name\n')
for i=1:raw_data.num_variables
    fprintf('%d \t\t %s \n', raw_data.selected_vars(i), raw_data.variable_name_list{i})
end
%% Plot data
% Trace numbers
var_va = 6;
var_vb = 1;
var_vo = 9;
var_irout = 26;

Open_TimeVect = raw_data.time_vect;
Open_va = raw_data.variable_mat(var_va,:);
Open_vb = raw_data.variable_mat(var_vb,:);
Open_var_vo = raw_data.variable_mat(var_vo,:);
```

```
Open_var_irout = raw_data.variable_mat(var_irout,:);

figure(1)
subplot(3,1,1)
hold on
plot(raw_data.time_vect*1e9, raw_data.variable_mat(var_va,:), 'k');
plot(raw_data.time_vect*1e9, raw_data.variable_mat(var_vb,:), '--k');
title(sprintf('Waveform %s and %s',
↪  raw_data.variable_name_list{var_va},raw_data.variable_name_list{var_vb}));
grid minor
ylabel(raw_data.variable_type_list{var_va});
xlabel('Time [ns]' );
xlim([min(raw_data.time_vect*1e9) max(raw_data.time_vect*1e9)])
legend(raw_data.variable_name_list{var_va}, raw_data.variable_name_list{var_vb},'location','best');
hold off
subplot(3,1,2)
hold on
plot(raw_data.time_vect*1e9, raw_data.variable_mat(var_vo,:), 'k');
title(sprintf('Waveform %s', raw_data.variable_name_list{var_vo}));
grid minor
ylabel(sprintf('%s [V]',raw_data.variable_type_list{var_vo}));
xlabel('Time [ns]');
xlim([min(raw_data.time_vect*1e9) max(raw_data.time_vect)*1e9])
%xlim([SimDelay,SimStop]);
%ylim([min(raw_data.variable_mat(var_opa_out,:)-raw_data.variable_mat(var_spk_m,:)),max(raw_data.variable_mat(var_opa_out,:)
%legend(raw_data.variable_name_list{var_vo},'location','best');
%hold off
%sgt = sgtitle('Open loop','Color','black');
%sgt.FontSize = 20;
subplot(3,1,3)
hold on
plot(raw_data.time_vect*1e9, -raw_data.variable_mat(var_vo,:).*raw_data.variable_mat(var_irout,:), 'k');
%title(sprintf('Waveform -%s %\\cdot% %s',
↪  raw_data.variable_name_list{var_vo},raw_data.variable_name_list{var_irout}),'Interpreter','latex');
title(sprintf('Waveform -%s*%s',
↪  raw_data.variable_name_list{var_vo},raw_data.variable_name_list{var_irout}));
grid minor
ylabel('Power [W]');
xlabel('Time [ns]' );
xlim([min(raw_data.time_vect*1e9) max(raw_data.time_vect*1e9)])
%legend(raw_data.variable_name_list{var_vo},'location','best');
hold off
%% CIRCUIT: Preamplifier
clear; close all; clc;
path_LTSpice = '"C:/Program Files/LTC/LTspiceXVII/XVIIx64.exe"';
circuit_path = 'C:/Users/Jeppe\OneDrive - Danmarks Tekniske Universitet/10.
↪  semester/MSc_Thesis_Teams/Simulation/';
circuit_name = 'Rx_preamp';
command = ' -Run -ascii -b ';
%% Simulate and import data
% Run LTSpice simulation
command = [path_LTSpice command '"' circuit_path circuit_name '.asc"'];
dos(command);

% Import data
```

```
raw_data = LTspice2Matlab([circuit_path circuit_name '.raw']);

% Display variables
fprintf('var \t name\n')
for i=1:raw_data.num_variables
    fprintf('%d \t\t %s \n', raw_data.selected_vars(i), raw_data.variable_name_list{i})
end
%% Plot data
% Trace numbers
var_input = 18;
var_opa_out = 14;
var_vga_vip = 7;
var_vga_vin = 6;
var_vga_oh = 15;
var_vga_ol = 16;

Open_TimeVect = raw_data.time_vect;
Open_opa_out = raw_data.variable_mat(var_opa_out,:);
Open_input = raw_data.variable_mat(var_input,:);
Open_vga_vip = raw_data.variable_mat(var_vga_vip,:);
Open_vga_vin = raw_data.variable_mat(var_vga_vin,:);
Open_vga_oh = raw_data.variable_mat(var_vga_oh,:);
Open_vga_ol = raw_data.variable_mat(var_vga_ol,:);

figure(1)
subplot(2,1,1)
hold on
plot(raw_data.time_vect, raw_data.variable_mat(var_input,:), 'k');
plot(raw_data.time_vect, raw_data.variable_mat(var_opa_out,:), '--k');
title(sprintf('Waveform %s and %s',
↪  raw_data.variable_name_list{var_input},raw_data.variable_name_list{var_opa_out}));
grid minor
ylabel(raw_data.variable_type_list{var_input});
xlabel('Time [sec]' );
xlim([0 3e-7])
legend(raw_data.variable_name_list{var_input},
↪  raw_data.variable_name_list{var_opa_out},'location','best');
hold off
%subplot(3,1,2)
%hold on
%plot(raw_data.time_vect, raw_data.variable_mat(var_vga_vip,:), 'k');
%plot(raw_data.time_vect, raw_data.variable_mat(var_vga_vin,:), '--k');
%title(sprintf('Waveform %s and %s',
↪  raw_data.variable_name_list{var_vga_vip},raw_data.variable_name_list{var_vga_vin}));
%grid on
%ylabel(raw_data.variable_type_list{var_opa_out});
%xlabel('Time [sec]' );
%xlim([SimDelay,SimStop]);
%ylim([min(raw_data.variable_mat(var_opa_out,:)-raw_data.variable_mat(var_spk_m,:)),max(raw_data.variable_mat(var_opa_out,:)
%legend(raw_data.variable_name_list{var_vga_vip},
↪  raw_data.variable_name_list{var_vga_vin},'location','best');
%hold off
%sgt = sgtitle('Open loop','Color','black');
%sgt.FontSize = 20;
subplot(2,1,2)
```

```
hold on
plot(raw_data.time_vect, raw_data.variable_mat(var_vga_oh,:), 'k');
plot(raw_data.time_vect, raw_data.variable_mat(var_vga_ol,:), '--k');
title(sprintf('Waveform %s and %s',
↪  raw_data.variable_name_list{var_vga_oh},raw_data.variable_name_list{var_vga_ol}));
grid minor
ylabel(raw_data.variable_type_list{var_opa_out});
xlabel('Time [sec]' );
xlim([0 3e-7])
%xlim([SimDelay,SimStop]);
%ylim([min(raw_data.variable_mat(var_opa_out,:)-raw_data.variable_mat(var_spk_m,:)),max(raw_data.variable_mat(var_opa_out,:)
legend(raw_data.variable_name_list{var_vga_oh},
↪  raw_data.variable_name_list{var_vga_ol},'location','best');
hold off
%% CIRCUIT: DEMODULATOR
clear; close all; clc;
path_LTSpice = '"C:/Program Files/LTC/LTspiceXVII/XVIIx64.exe"';
circuit_path = 'C:/Users/Jeppe\OneDrive - Danmarks Tekniske Universitet/10.
↪  semester/MSc_Thesis_Teams/Simulation/';
circuit_name = 'Rx_demod';
command = ' -Run -ascii -b ';
%% Simulate and import data
% Run LTSpice simulation
command = [path_LTSpice command '"' circuit_path circuit_name '.asc"'];
dos(command);

% Import data
raw_data = LTspice2Matlab([circuit_path circuit_name '.raw']);

% Display variables
fprintf('var \t name\n')
for i=1:raw_data.num_variables
    fprintf('%d \t\t %s \n', raw_data.selected_vars(i), raw_data.variable_name_list{i})
end
%% Plot data
% Trace numbers
var_lop = 9;
var_lon = 10;
var_vga_ol = 7;
var_vga_oh = 6;
var_out_i = 17;
var_out_q = 19;

figure(1)
%subplot(2,1,1)
hold on
plot(raw_data.time_vect, raw_data.variable_mat(var_vga_ol,:), 'k');
plot(raw_data.time_vect, raw_data.variable_mat(var_vga_oh,:), '--k');
title(sprintf('Waveform %s and %s',
↪  raw_data.variable_name_list{var_vga_ol},raw_data.variable_name_list{var_vga_oh}));
grid minor
ylabel(raw_data.variable_type_list{var_vga_ol});
xlabel('Time [sec]' );
xlim([1.5e-5 1.53e-5])
legend(raw_data.variable_name_list{var_vga_ol},
↪  raw_data.variable_name_list{var_vga_oh},'location','best');
```

```
hold off
figure(2)
%subplot(2,1,2)
hold on
plot(raw_data.time_vect, raw_data.variable_mat(var_lop,:), 'k');
plot(raw_data.time_vect, raw_data.variable_mat(var_lon,:), ':k');
grid minor
legend(raw_data.variable_name_list{var_lop}, raw_data.variable_name_list{var_lon},'location','best');
ylabel(raw_data.variable_type_list{var_lop});
xlabel('Time [sec]' );
xlim([1.5e-5 1.53e-5])
ylim([1.05 1.35])
hold off

figure(3)
hold on
%plot(raw_data.time_vect, movmean(raw_data.variable_mat(var_out_i,:),1000), 'k');
plot(raw_data.time_vect, raw_data.variable_mat(var_out_i,:), 'k');
%plot(raw_data.time_vect, movmean(raw_data.variable_mat(var_out_q,:),1000), '--k');
plot(raw_data.time_vect, raw_data.variable_mat(var_out_q,:), '--k');
title(sprintf('Waveform %s and %s',
↪  raw_data.variable_name_list{var_out_i},raw_data.variable_name_list{var_out_q}));
grid minor
ylabel(raw_data.variable_type_list{var_out_i});
xlabel('Time [sec]' );
%xlim([1.5e-5 1.5e-4])
legend(raw_data.variable_name_list{var_out_i}, raw_data.variable_name_list{var_out_q},'location','best');
hold off
%% CIRCUIT: dc-coupler TRANSIENT
clear; close all; clc;
path_LTSpice = '"C:/Program Files/LTC/LTspiceXVII/XVIIx64.exe"';
circuit_path = 'C:/Users/Jeppe\OneDrive - Danmarks Tekniske Universitet/10.
↪  semester/MSc_Thesis_Teams/Simulation/';
circuit_name = 'dc-coupler';
command = ' -Run -ascii -b ';
%% Simulate and import data
% Run LTSpice simulation
command = [path_LTSpice command '"' circuit_path circuit_name '.asc"'];
dos(command);

% Import data
raw_data = LTspice2Matlab([circuit_path circuit_name '.raw']);

% Display variables
fprintf('var \t name\n')
for i=1:raw_data.num_variables
    fprintf('%d \t\t %s \n', raw_data.selected_vars(i), raw_data.variable_name_list{i})
end
%% Plot data
% Trace numbers
var_in = 4;
var_out = 1;

figure(1)
hold on
```

```
plot(raw_data.time_vect, raw_data.variable_mat(var_in,:), '--k');
plot(raw_data.time_vect, raw_data.variable_mat(var_out,:), 'k');
grid minor
legend(raw_data.variable_name_list{var_in}, raw_data.variable_name_list{var_out},'location','best');
ylabel(raw_data.variable_type_list{var_in});
xlabel('Time [sec]' );
hold off
%% FREQ analysis
clear; close all; clc;
fidi = fopen('dc-coupler.txt');
Dc = textscan(fidi, '%f(%fdB,%f°)', 'CollectOutput',1);
D = cell2mat(Dc);
figure
sgtitle('\textbf{Bode Plot}','interpreter','latex');
subplot(2,1,1)
semilogx(D(:,1), D(:,2),'k')
ylabel('Amplitude [dB]','Interpreter','latex')
grid
subplot(2,1,2)
semilogx(D(:,1), D(:,3),'k')
ylabel('Phase [deg]','Interpreter','latex')
grid
xlabel('Frequency [Hz]','Interpreter','latex')
```

Listing B.6: Analysis.m

```
close all; clc; clear
index=6;
%% Setting
param_title= {'fontsize',18, 'fontname','Arial'};
param_label={'fontsize',15,'fontname','Arial'};

save_fig=0;
F_prf=1/99.99e-6;
M=5;
f0=5e6;
fres=5;
dB_scale=0;
padding = 1;
sampled_interpolation = 0;
interpolation_factor=100;

%% Get data
file_name1=['scope_' num2str(index) '_2.csv'];
file_name2=['scope_' num2str(index) '_4.csv'];
data1 = xlsread(file_name1);
data2 = xlsread(file_name2);
time=data1(3:length(data1)-3,1);
v1=data1(3:length(data1)-3,2);
v2=data2(3:length(data2)-3,2);
```

```
%% chose signal
signal = v1;


%% Bandpass filter before sampling
dt=mean(diff(time));
fs = 1/dt;
fn=fs/2;
[c, d]=butter(2, [3e6 7e6]/fn,'bandpass');
signal=filter(c,d,signal);

%% Power
power_signal = (rms(signal))^2;

%% t_start
 idx_start=min(find(v2>max(v2)*0.7));
 t_start = time(idx_start);

% idx_start = max(find(time<-411e-6));
% t_start = time(idx_start);
%% N count
count = t_start;
N=0;                                                %number of elements in sampled array
while count<time(end)
   count = count + 1/F_prf;
   N=N+1;
end

%% Cutting
 idx_nxt=min(find(time>(t_start+ 1/F_prf)));
 t_nxt = time(idx_nxt);
 Is_prf=1/(t_nxt-t_start);
 idx_length= idx_nxt - idx_start;
 signal_array = zeros(N, idx_length);
 idx_count = idx_start;
 height_offset =max(signal)*2;
 cut_time = (1:idx_length)*dt;
figure()
hold on
for i=1:N
    if (idx_count+idx_length-1 < length(signal))
        signal_array(i,:)= signal(idx_count:idx_count+idx_length-1);
        signal_array(i,:) = signal_array(i,:) + height_offset*(i-1);
        plot(cut_time*1e6, signal_array(i,:))
        xlabel('Time(\mus)',param_label{:})
% %         title('Cutted data',param_title{:})
        xlim([0 cut_time(end)*1e6])
        idx_count = idx_count+idx_length;
    end
end

hold off

figure()
plot(cut_time*1e6, sum(signal_array,1)/N);
```

```
xlabel('Time(\mus)',param_label{:})
title('Summed Data',param_title{:})


figure()
plot(cut_time*1e6, signal_array(1,:))
xlabel('Time(\mus)',param_label{:})
% title('Cutted data 1',param_title{:})

figure()
plot(time*1e6, signal)
xlabel('Time(\mus)',param_label{:})
ylabel('Voltage (V)',param_label{:})

 a=[6 7]*1e-6;
 dt = mean(diff(time));
 t_offset = a(1):10*dt:a(2);
 vel_avg=zeros(1,length(t_offset));
 vel_max=zeros(1,length(t_offset));
 for j=1:length(t_offset)
%% Sampling
time_sam = zeros(1,N);
signal_sam= zeros(1,N);
count= t_start+t_offset(j);
for i= 1:N
    idx= max(find(time<count));
    time_sam(i) = time(idx);
    signal_sam(i) = signal(idx);
    count = count + 1/F_prf;
end

%% Bandpass filter after sampling
dt_sam=mean(diff(time_sam));
fs_sam = 1/dt_sam;
fn_sam=fs_sam/2;
[c, d]=butter(3, [100 F_prf/2*0.8]/fn_sam,'bandpass');
signal_sam=filter(c,d,signal_sam);


%% plot in time domain
% figure()
% hold on
% plot(time*1e6, signal)
% for i=1:length(time_sam)
%     scatter(time_sam(i)*1e6, signal_sam(i),'r')
% end
% hold off
% xlabel('time(\mus)',param_label{:})
% % title(['time offset: ' num2str(t_offset(j)*1e6) '\mus'],param_title{:})
% if save_fig
%  saveas(gcf,['time offset= ' num2str(t_offset(j)*1e6) 'us.jpg'])
% end
%
% scatter(time_sam*1e6, signal_sam,'r')
% xlabel('Time(\mus)',param_label{:})
```

```
% ylabel('Voltage(V)',param_label{:})

%% Fourier transform
L = length(time_sam);
fs=1/mean(diff(time_sam));
if padding
    Lnew = round(fs/fres);
    P = Lnew-L;
    if P<0
        error('P should be bigger than 0. You should increse the ''fres'' ')
    end
    signal_sam = [signal_sam zeros(1,P)];
else
    Lnew=L;
end


Y= fft(signal_sam);
f = fs/Lnew*(0:(Lnew/2));
P2 = abs(Y/Lnew);
P1 = P2(1:Lnew/2+1);
P1(2:end-1) = 2*P1(2:end-1);
if dB_scale
    P1 = mag2db(P1); %dB scale conversion
end

avg_f = sum(f.*P1)/sum(P1);
idx1 = find(P1==max(P1));
avg_f = sum(f.*P1)/sum(P1);


%% Plot Fourier Transform
figure()
hold on
plot(f,P1)
plot(f(1),P1(1))
plot(f(2),P1(2))
plot(f(3),P1(3))
hold off
title(['FFT of time offset: ' num2str(t_offset(j)*1e6) '\mus'],param_title{:})
xlabel('frequency',param_label{:})
if dB_scale
    ylabel('Magnitude (dB)')
end
legend(['Max frequency: ' num2str(f(idx1)) 'Hz'],...
[' Velocity: ' num2str(1540*f(idx1)/f0/2*sqrt(2)) 'm/s'],...
[' Velocity: ' num2str(1540*avg_f/f0/2*sqrt(2)) 'm/s'])

if save_fig
 saveas(gcf,['FFT of time offset= ' num2str(t_offset(j)*1e6) 'us.jpg'])
end
 %% Sweep graph
vel_avg(j)=1540*avg_f/f0/2 *sqrt(2);
vel_max(j)= 1540*f(idx1)/f0/2* sqrt(2);
 end
```

```

figure();
hold on
scatter(t_offset*1e6,vel_avg*100)
scatter(t_offset*1e6,vel_max*100)
legend('Avg', 'Max')
% title(['Sweeped data     index: ' num2str(index)],param_title{:})
ylabel('velocity (cm/s)',param_label{:})
xlabel('time offset(\mus)',param_label{:})

%% Data
figure();
hold on
% scatter(t_offset*1540/sqrt(2)/2*1e3,vel_avg*100)
scatter(t_offset*1540/sqrt(2)/2*1e3,vel_max*100)
% legend('Avg', 'Max')
% title(['Sweeped data     index: ' num2str(index)],param_title{:})
ylabel('Velocity (cm/s)',param_label{:})
xlabel('Depth (cm)',param_label{:})
```

Listing B.7: demodulator.m

```
clc;clear;close all;

%out_t2 = readtable('demodulator/scope_204_2.csv','ReadVariableNames', false, 'HeaderLines', 1);
%out_t4 = readtable('demodulator/scope_204_4.csv','ReadVariableNames', false, 'HeaderLines', 1);
%in_t2 = readtable('demodulator/scope_206_2.csv','ReadVariableNames', false, 'HeaderLines', 1);
%in_t4 = readtable('demodulator/scope_206_4.csv','ReadVariableNames', false, 'HeaderLines', 1);
t2 = readtable('demodulator/tek0010ALL.csv','ReadVariableNames', false, 'HeaderLines', 20);
t1 = readtable('demodulator/tek0011ALL.csv','ReadVariableNames', false, 'HeaderLines', 20);
t1.Properties.VariableNames = ["time","ch1","ch2","ch3","ch4"];
t2.Properties.VariableNames = ["time","ch1","ch2","ch3","ch4"];
%%
figure(1)
hold on
%title('Demodulated signal $f_d=1~\mathrm{kHz}$','interpreter','latex');
plot(t1.time,movmean(t1.ch1,200),'k')
plot(t1.time,movmean(t1.ch4,200),'k-.','Color','[0.5 0.5 0.5]')
plot(t1.time,t1.ch2,'--k')
plot(t1.time,t1.ch3,':k')
grid minor
set(gca,'TickLabelInterpreter','latex')
ylabel('Voltage [V]','interpreter','latex');
xlabel('Time [sec]','interpreter','latex');
%xlim([-2e-4 2e-4])
legend('Demodulated signal $I$ 1~kHz','Demodulated signal $Q$ 1~kHz','Differential signal A
↪  5.001~MHz','Differential signal B 5.001~MHz 180\angle','Location','best','interpreter','latex')
hold off
% create a new pair of axes inside current figure
axes('position',[.65 .5125 .25 .25])
hold on
box on % put box around new pair of axes
indexOfInterest = (t1.time < 1.5e-7) & (t1.time > -1.5e-7); % range of t near perturbation
```

```
%indexOfInterest = (t < 11*pi/8) & (t > 9*pi/8)
%indexOfInterestp = indexOfInterest.';
plot(t1.time(indexOfInterest),t1.ch2(indexOfInterest),'--k') % plot on new axes
plot(t1.time(indexOfInterest),t1.ch3(indexOfInterest),':k') % plot on new axes
set(gca,'TickLabelInterpreter','latex')
axis tight
grid on
ylim([min(t1.ch2) max(t1.ch2)])
hold off
%%
figure(2)
%subplot(2,1,1)
hold on
%title('Received signal $f_d=5.3~\mathrm{MHz}$','interpreter','latex');
plot(t2.time,movmean(t2.ch2,10),':k')
plot(t2.time,movmean(t2.ch3,10),'--k')
plot(t2.time,t2.ch4,'k')
grid minor
ylabel('Voltage [V]','interpreter','latex');
xlabel('Time [sec]','interpreter','latex');
xlim([-0.15e-6 0.15e-6])
set(gca,'TickLabelInterpreter','latex')
legend('Differential signal A 5.001~MHz','Differential signal B 5.001~MHz 180\angle','Local Oscillator
↪  $L_{OSC4}=$ 20~MHz','Location','best','interpreter','latex')
hold off
%subplot(2,1,2)
%hold on
%title('Local Oscillator signal $L_{\mathrm{osc4}}=20~\mathrm{MHz}$','interpreter','latex');
%plot(t2.time,t2.ch3,'k')
%plot(t2.time,t2.ch4,'k')
%grid minor
%ylabel('Voltage [V]');
%xlabel('Time [sec]' );
%xlim([3e-6 3.2e-6])
%hold off
```

Listing B.8: preamplifier.m

```
clc;clear;close all;

%out_t2 = readtable('scope_208_2.csv','ReadVariableNames', false, 'HeaderLines', 1);
%out_t4 = readtable('scope_208_4.csv','ReadVariableNames', false, 'HeaderLines', 1);
%in_t2 = readtable('scope_208_1.csv','ReadVariableNames', false, 'HeaderLines', 1);
t1 = readtable('tek0012ALL.csv','ReadVariableNames', false, 'HeaderLines', 20);
t1.Properties.VariableNames = ["time","input","outa","outb"];


figure(1)
subplot(2,1,1)
title('AC coupled input signal','interpreter','latex');
hold on
plot(t1.time,t1.input,'k','DisplayName','Input')
%plot(out_t4.Var1,out_t4.Var2,'k','DisplayName','LOP')
```

```
grid minor
ylabel('Voltage [V]','interpreter','latex');
set(gca,'TickLabelInterpreter','latex')
%xlabel('Time [sec]' );
xlim([-0.2e-6 0.2e-6])
%l=legend('show');
hold off
subplot(2,1,2)
%figure(2)
hold on
title('Amplified differential signal with DC coupling','interpreter','latex');
plot(t1.time,movmean(t1.outa,10),'k')
plot(t1.time,movmean(t1.outb,10),'--k')
grid minor
ylabel('Voltage [V]','interpreter','latex');
xlabel('Time [sec]','interpreter','latex');
set(gca,'TickLabelInterpreter','latex')
xlim([-0.2e-6 0.2e-6])
legend('Differential signal A 5.001~MHz','Differential signal B 5.001~MHz
↪  180\angle','Location','best','interpreter','latex')
hold off
%set(l,'Interpreter','Latex');
```

Listing B.9: sampler.m

```
clc;clear;close all;

t1 = readtable('tek0001ALL.csv', 'ReadVariableNames', false, 'HeaderLines', 20);
t1.Properties.VariableNames = ["time","output","pulse","input"];

figure(1)
subplot(3,1,1)
hold on
title('Demodulated input waveform','interpreter','latex');
plot(t1.time,t1.input,'k')
set(gca,'TickLabelInterpreter','latex')
grid minor
ylabel('Voltage [V]','interpreter','latex');
%xlabel('Time [sec]' );
hold off
subplot(3,1,2)
hold on
title('Sample pulse input','interpreter','latex');
plot(t1.time,t1.pulse,'k')
set(gca,'TickLabelInterpreter','latex')
grid minor
ylabel('Voltage [V]','interpreter','latex');
%xlabel('Time [sec]' );
hold off
subplot(3,1,3)
hold on
title('Sampled demodulated output','interpreter','latex');
plot(t1.time,t1.output,'k')
```

```
set(gca,'TickLabelInterpreter','latex')
grid minor
ylabel('Voltage [V]','interpreter','latex');
xlabel('Time [sec]','interpreter','latex');
hold off
```

Listing B.10: transmitter.m

```
clc;clear;close all;

t1 = readtable('scope_202_1.csv', 'ReadVariableNames', false, 'HeaderLines', 1);
t2 = readtable('scope_202_2.csv','ReadVariableNames', false, 'HeaderLines', 1);
t4 = readtable('scope_202_4.csv','ReadVariableNames', false, 'HeaderLines', 1);

figure(1)
subplot(2,1,1)
hold on
title('PWM Input to gate driver','interpreter','latex');
plot(t1.Var1,movmean(t1.Var2,25),'k','DisplayName','INA')
plot(t2.Var1,movmean(t2.Var2,25),':k','DisplayName','INB')
set(gca,'TickLabelInterpreter','latex')
grid minor
ylabel('Voltage [V]','interpreter','latex');
xlabel('Time [sec]','interpreter','latex');
legend('show','interpreter','latex')
hold off
subplot(2,1,2)
hold on
title('Power stage output voltage','interpreter','latex');
plot(t4.Var1,t4.Var2,'k')
set(gca,'TickLabelInterpreter','latex')
grid minor
ylabel('Voltage [V]','interpreter','latex');
xlabel('Time [sec]','interpreter','latex');
hold off
```

Listing B.11: vna.m

```
clc;clear;close all;
%%
mag = readtable('TRACE01.CSV','ReadVariableNames', false, 'HeaderLines', 3);
phase = readtable('TRACE02.CSV','ReadVariableNames', false, 'HeaderLines', 3);
mag.Properties.VariableNames = ["x","y","zero"];
phase.Properties.VariableNames = ["x","y","zero"];

figure()
sgtitle('\textbf{Bode Plot}','interpreter','latex');
subplot(2,1,1)
hold on
%plot(mag.x,mag.y,'k')
semilogx(mag.x,mag.y,'k')
```

```
grid minor
set(gca,'TickLabelInterpreter','latex')
ylabel('Magnitude [dB]','interpreter','latex');
xlabel('Frequency [Hz]','interpreter','latex');
%xlim([900e3 11e6])
ylim([-15 1])
hold off
subplot(2,1,2)
hold on
%plot(phase.x,unwrap(phase.y),'k')
semilogx(phase.x,unwrap(phase.y),'k')
grid minor
set(gca,'TickLabelInterpreter','latex')
ylabel('Phase [deg]','interpreter','latex');
xlabel('Frequency [Hz]','interpreter','latex');
xlim([0 14e6])
hold off
```

Listing B.12: bpf datasheet s21.m

```
clc;clear;close all;
%%
import_table = readtable('BPF-C4R5+_Plus25DegC.txt','ReadVariableNames', false, 'HeaderLines', 10);
import_table.Properties.VariableNames = ["freq", "mag_s11", "phase_s11", "mag_s21", "phase_s21",
↪  "mag_s12", "phase_s12", "mag_s22", "phase_s22"];

freq = import_table{:,'freq'};
mag = import_table{:,"mag_s21"};
phase = import_table{:,"phase_s21"};
freq = freq.*1e6;
phase = unwrap(phase);
%%
figure()
sgtitle('\textbf{Bode Plot}','interpreter','latex');
subplot(2,1,1)
hold on
semilogx(freq,mag,'k')
grid on
set(gca,'TickLabelInterpreter','latex')
ylabel('Magnitude [dB]','interpreter','latex');
xlabel('Frequency [Hz]','interpreter','latex');
xlim([0 14e6])
ylim([-15 1])
hold off
subplot(2,1,2)
hold on
semilogx(freq,phase,'k')
grid on
set(gca,'TickLabelInterpreter','latex')
ylabel('Phase [deg]','interpreter','latex');
xlabel('Frequency [Hz]','interpreter','latex');
xlim([0 14e6])
hold off
```

Listing B.13: switch.m

```
clc;clear;close all;

t2 = readtable('230317/scope_114_2.csv','ReadVariableNames', false, 'HeaderLines', 1);
t4 = readtable('230317/scope_114_4.csv','ReadVariableNames', false, 'HeaderLines', 1);

figure(1)
subplot(2,1,1)
hold on
title('Transmitted signal','interpreter','latex');
plot(t4.Var1,t4.Var2,'k')
grid minor
ylabel('Voltage [V]','interpreter','latex');
%xlabel('Time [sec]','interpreter','latex');
set(gca,'TickLabelInterpreter','latex')
xlim([9.8942e-5 1.21698e-4])
hold off
subplot(2,1,2)
hold on
title('Received signal','interpreter','latex');
plot(t2.Var1,t2.Var2,'k')
grid minor
ylabel('Voltage [V]','interpreter','latex');
xlabel('Time [sec]','interpreter','latex');
set(gca,'TickLabelInterpreter','latex')
xlim([9.8942e-5 1.21698e-4])
hold off
```

Listing B.14: pulser.m

```
clc;clear;close all;

t1 = readtable('digital.csv', 'ReadVariableNames', false, 'HeaderLines', 1);
t1.Properties.VariableNames = ["time","PWM","PWMN","PULSE","GATE","PRF","CLK"];
%%
%figure(1)
fig = figure('units','inch','position',[0,10,10,2*3.3/3]);
subplot(6,1,1)
hold on
title('CLK','Interpreter','latex')
stairs(t1.time,t1.CLK,'k','DisplayName','Output')
set(gca,'TickLabelInterpreter','latex')
xlim([min(t1.time) max(t1.time)])
hold off
subplot(6,1,2)
hold on
title('PRF','Interpreter','latex')
stairs(t1.time,t1.PRF,'k','DisplayName','Input')
set(gca,'TickLabelInterpreter','latex')
```

```
xlim([min(t1.time) max(t1.time)])
hold off
subplot(6,1,3)
hold on
title('PWM','Interpreter','latex')
stairs(t1.time,t1.PWM,'k','DisplayName','Input')
set(gca,'TickLabelInterpreter','latex')
xlim([min(t1.time) max(t1.time)])
hold off
subplot(6,1,4)
hold on
title('PWMN','Interpreter','latex')
stairs(t1.time,t1.PWMN,'k','DisplayName','Input')
set(gca,'TickLabelInterpreter','latex')
xlim([min(t1.time) max(t1.time)])
hold off
subplot(6,1,5)
hold on
title('PULSE','Interpreter','latex')
stairs(t1.time,t1.PULSE,'k','DisplayName','Input')
set(gca,'TickLabelInterpreter','latex')
xlim([min(t1.time) max(t1.time)])
hold off
subplot(6,1,6)
hold on
title('GATE','Interpreter','latex')
stairs(t1.time,t1.GATE,'k','DisplayName','Input')
set(gca,'TickLabelInterpreter','latex')
xlim([min(t1.time) max(t1.time)])
xlabel('Time [s]','interpreter','latex');
hold off
```

# Appendix C

# Simulation Models

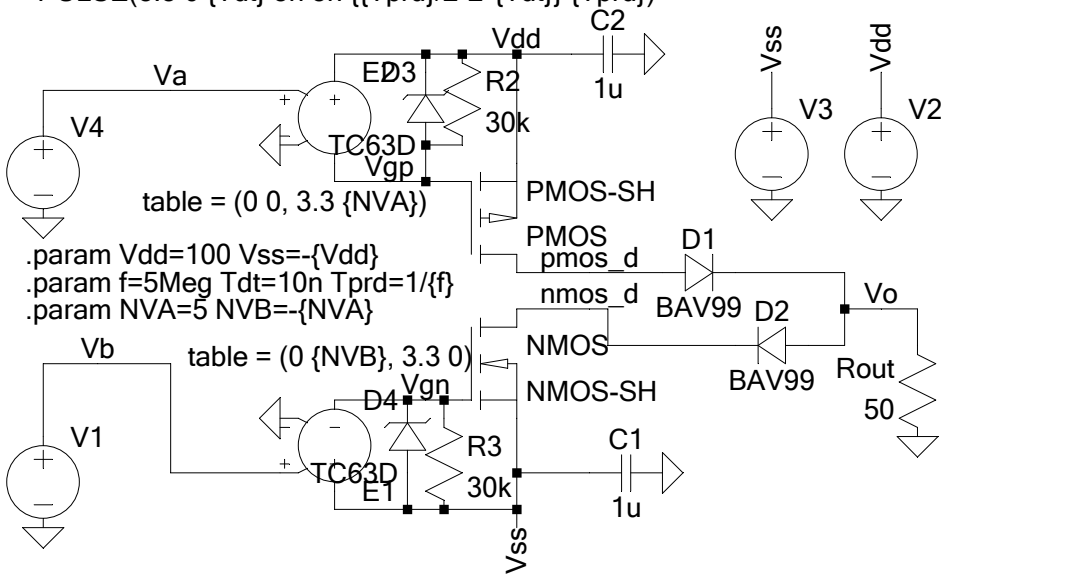



Figure C.1: LTspice model of transmitter

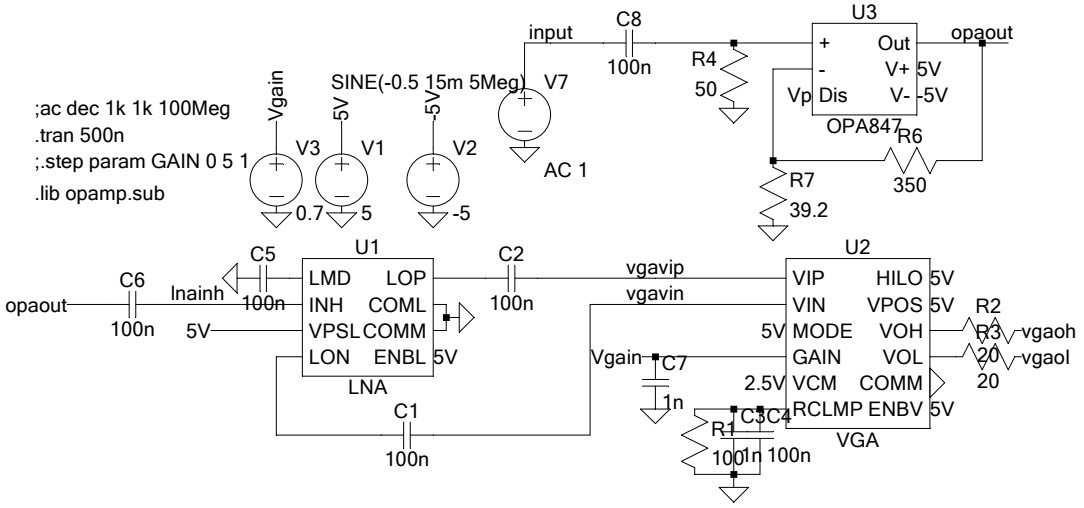



Figure C.2: LTspice model of preamplifier

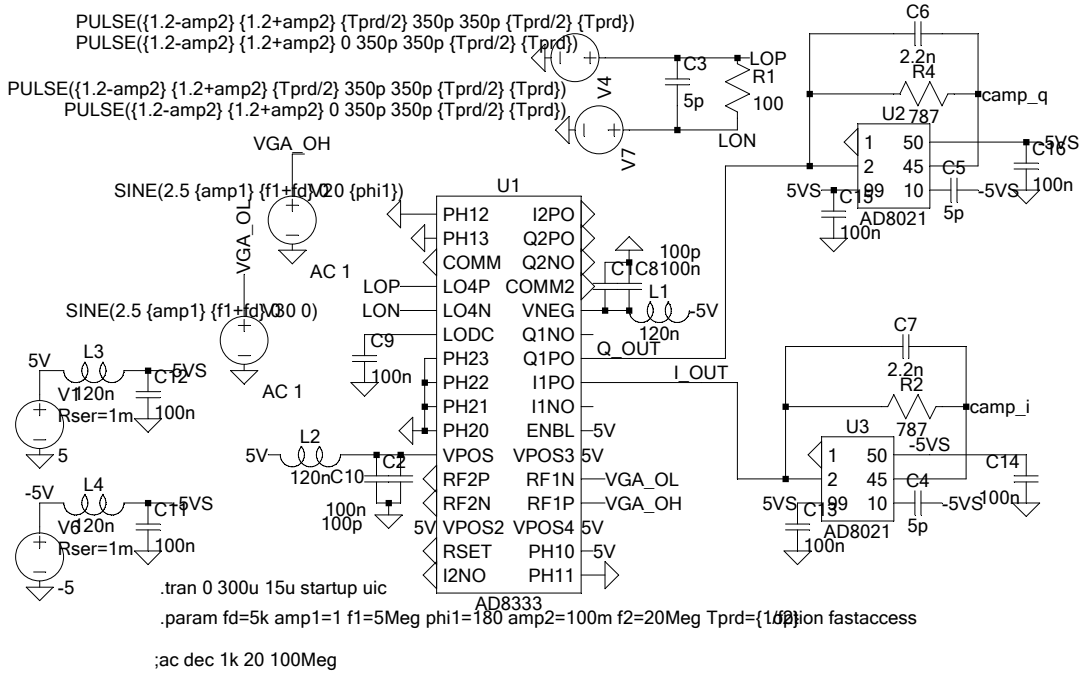



Figure C.3: LTspice model of demodulator

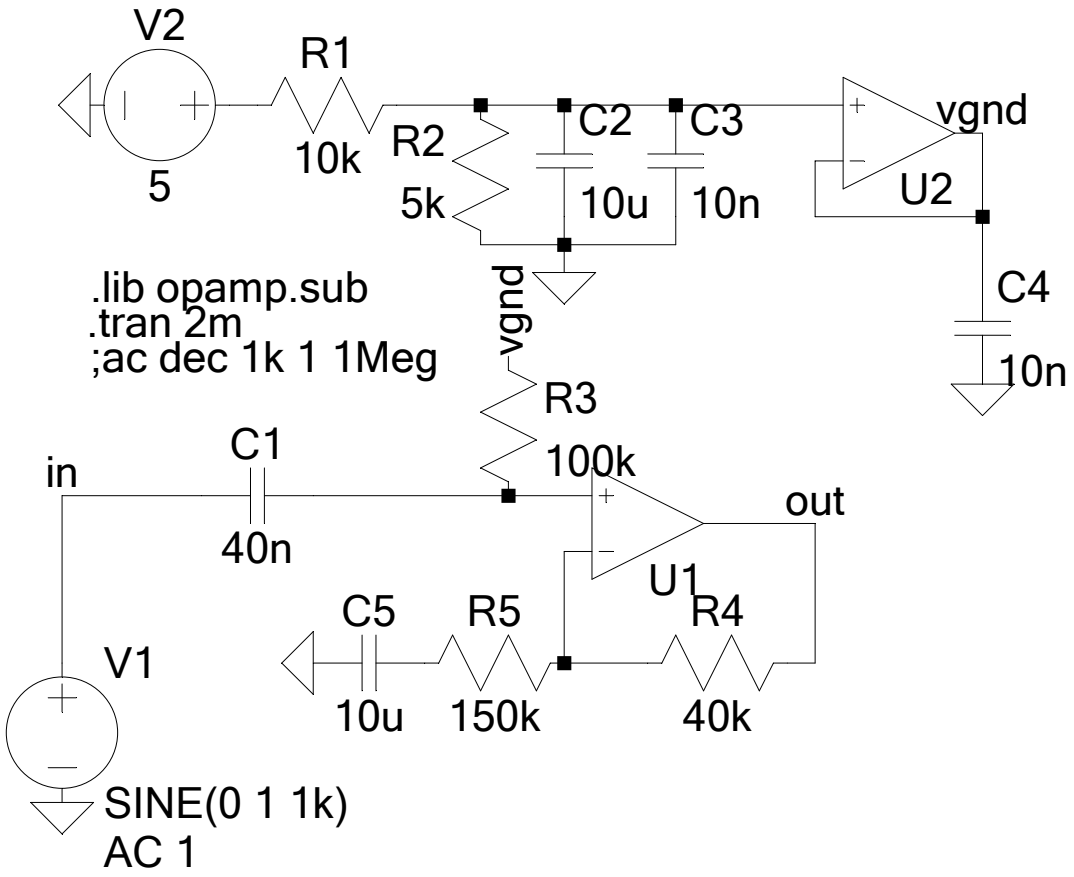



Figure C.4: LTspice model of PRF filter

# Appendix D

# Circuit Schematics

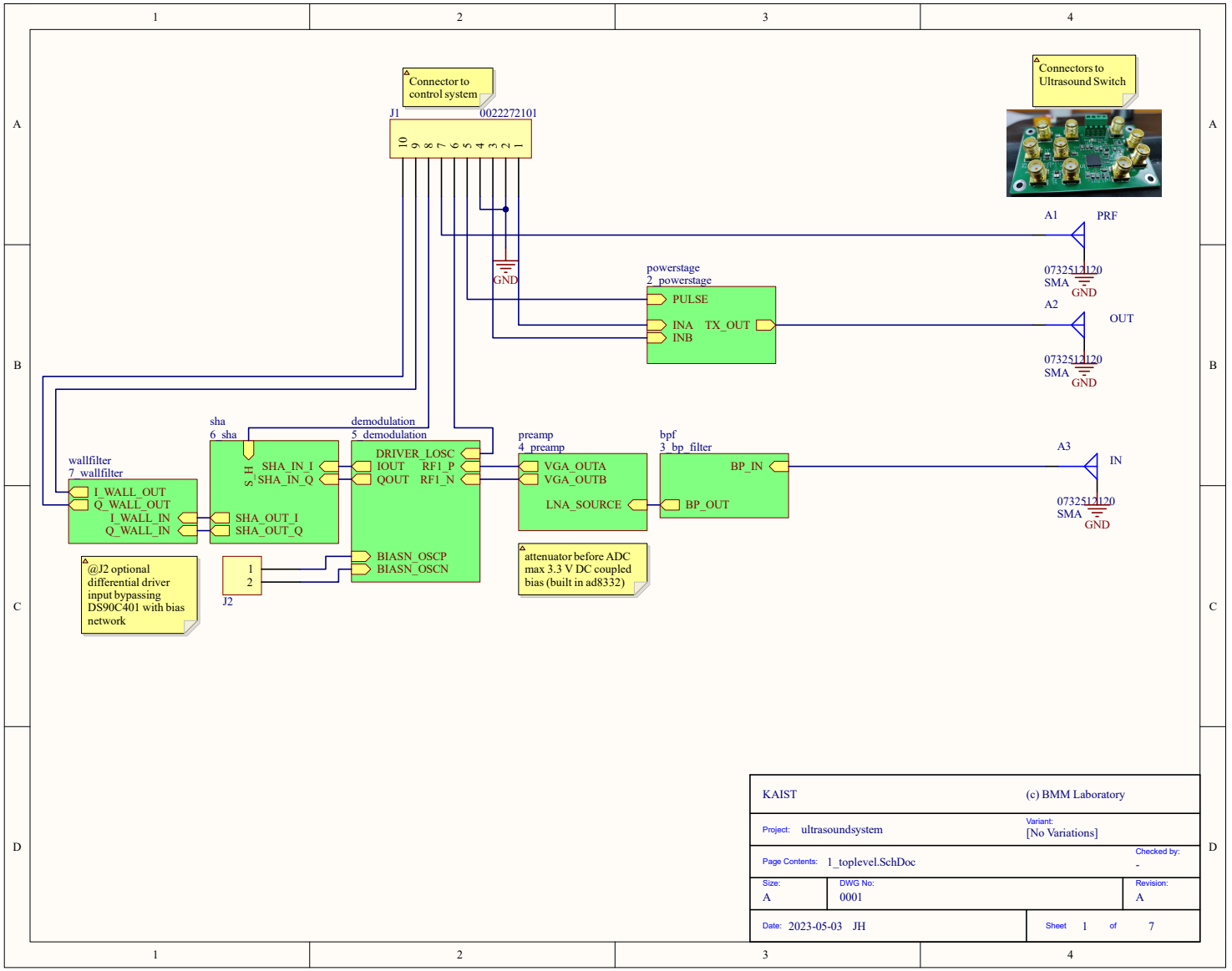



Figure D.1: AFE Top Level

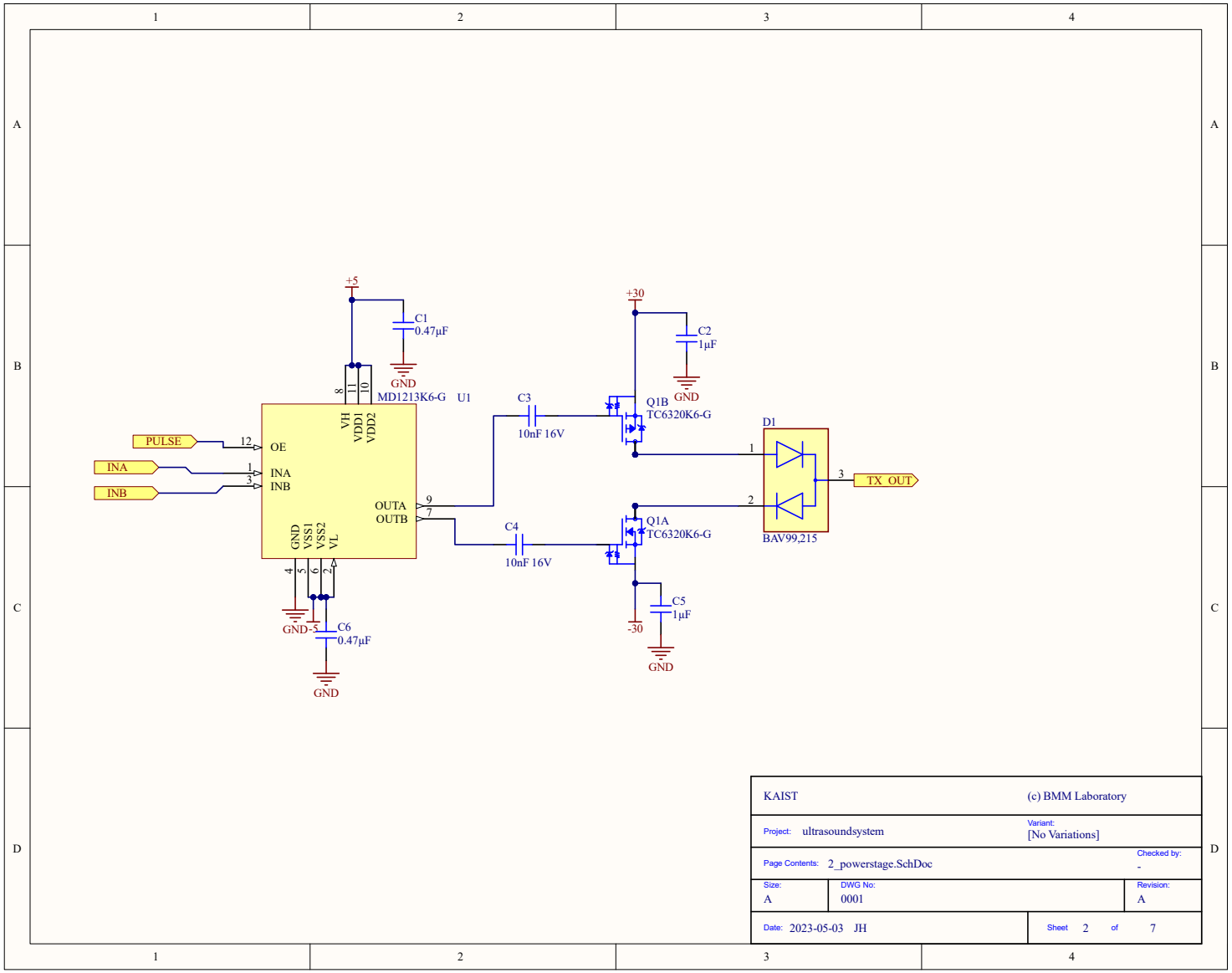



Figure D.2: AFE Power Stage

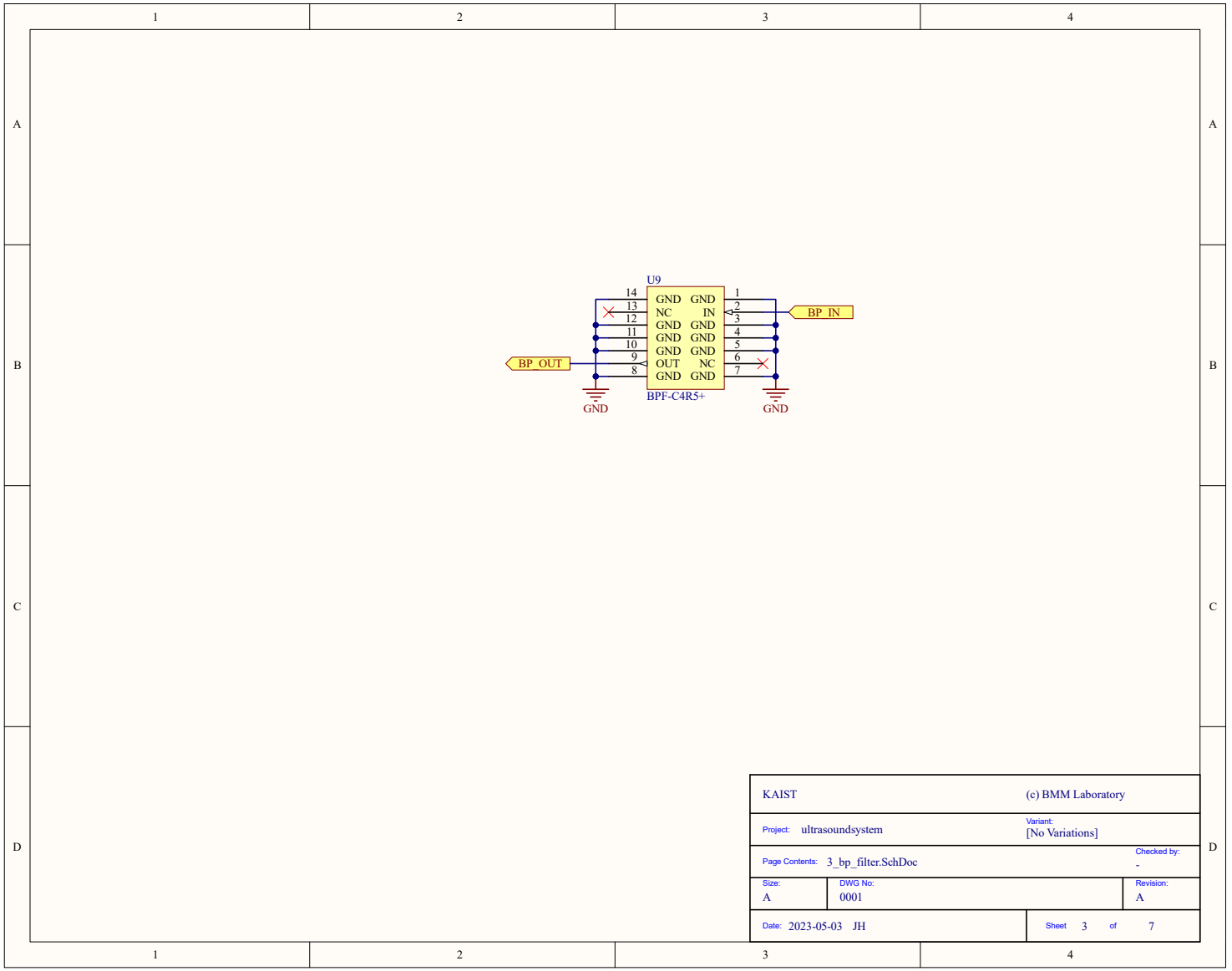



Figure D.3: AFE BPF

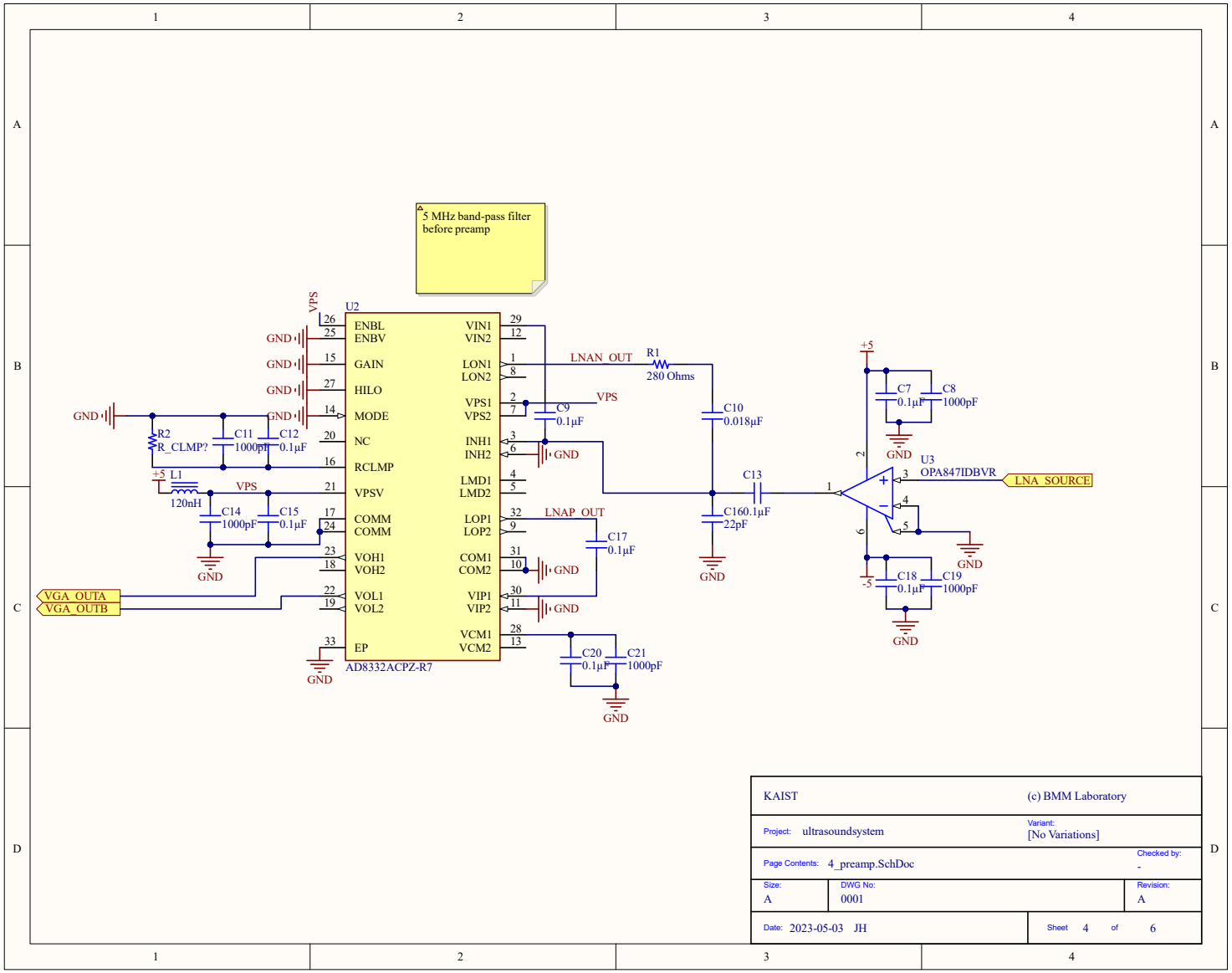



Figure D.4: AFE Preamplifier

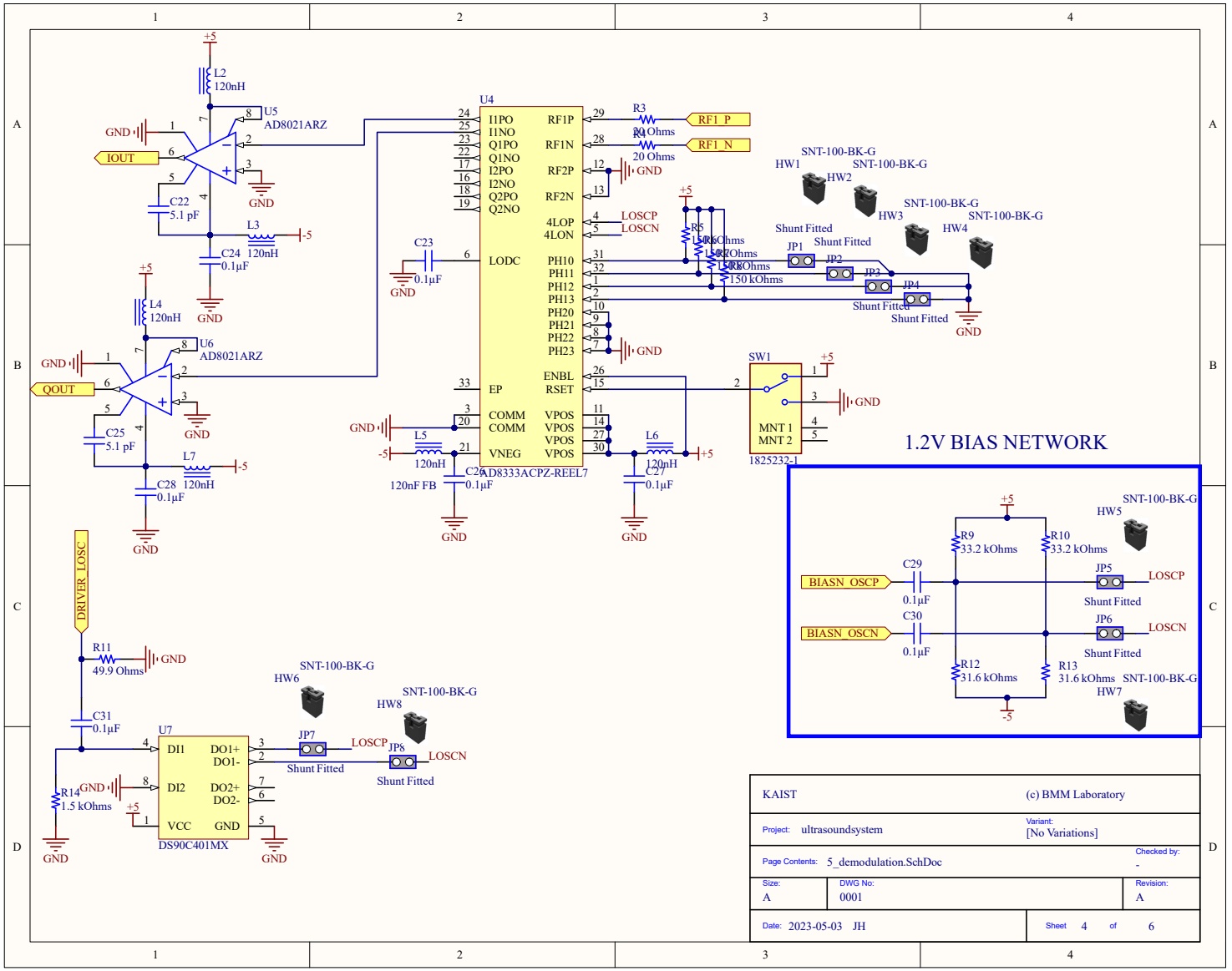



Figure D.5: AFE Demodulator

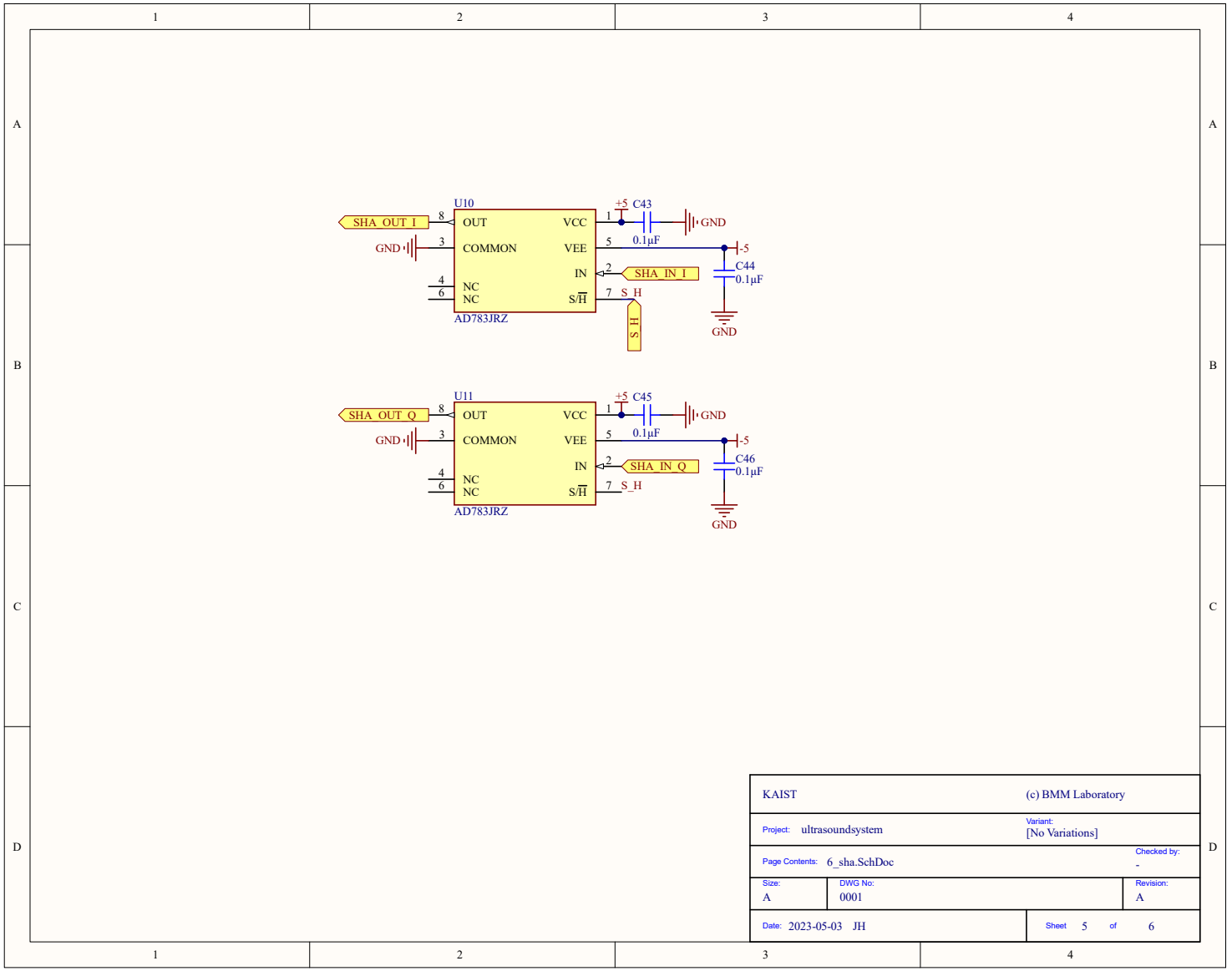



Figure D.6: AFE Sample and Hold Amplifier

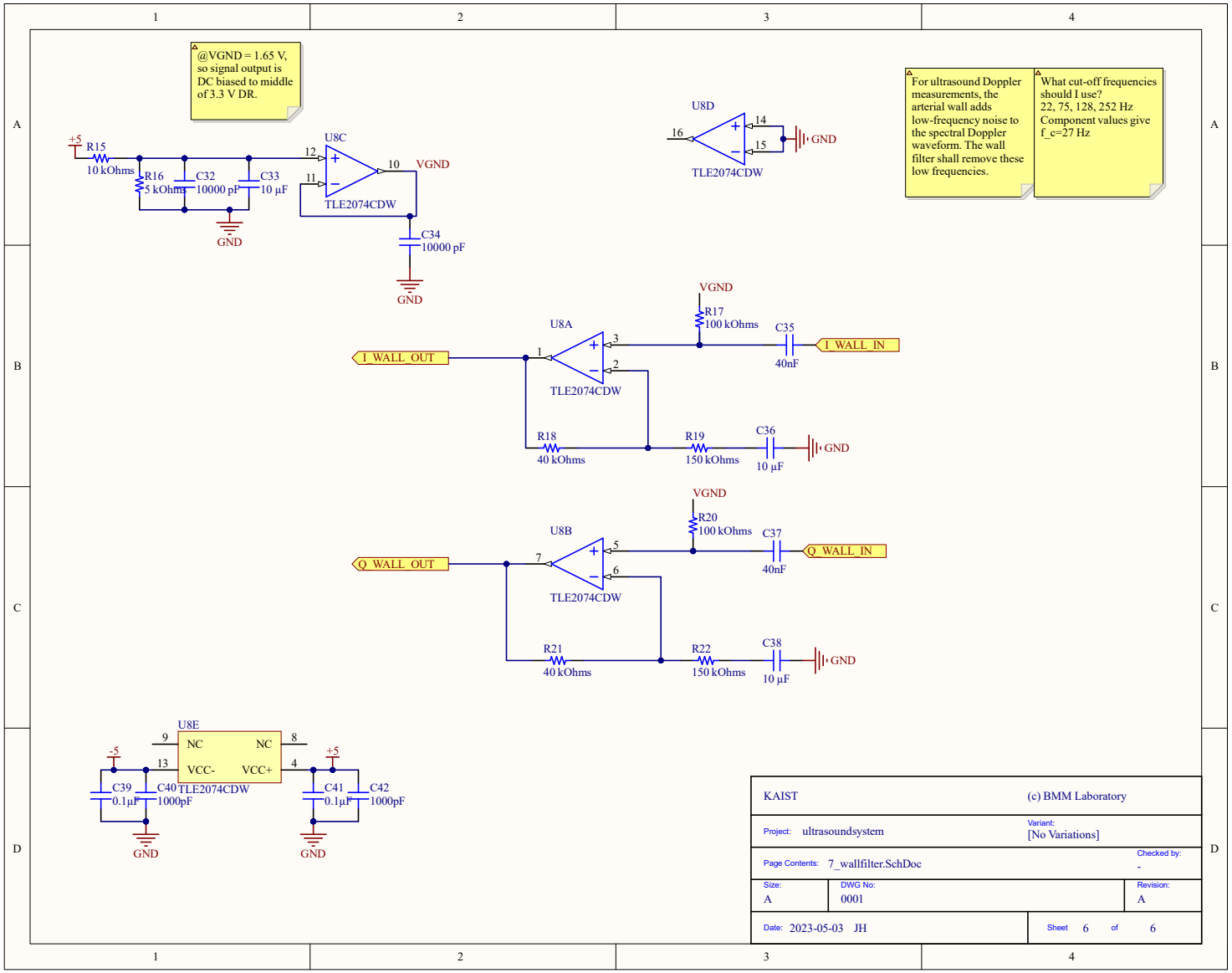



Figure D.7: AFE Wall Filter

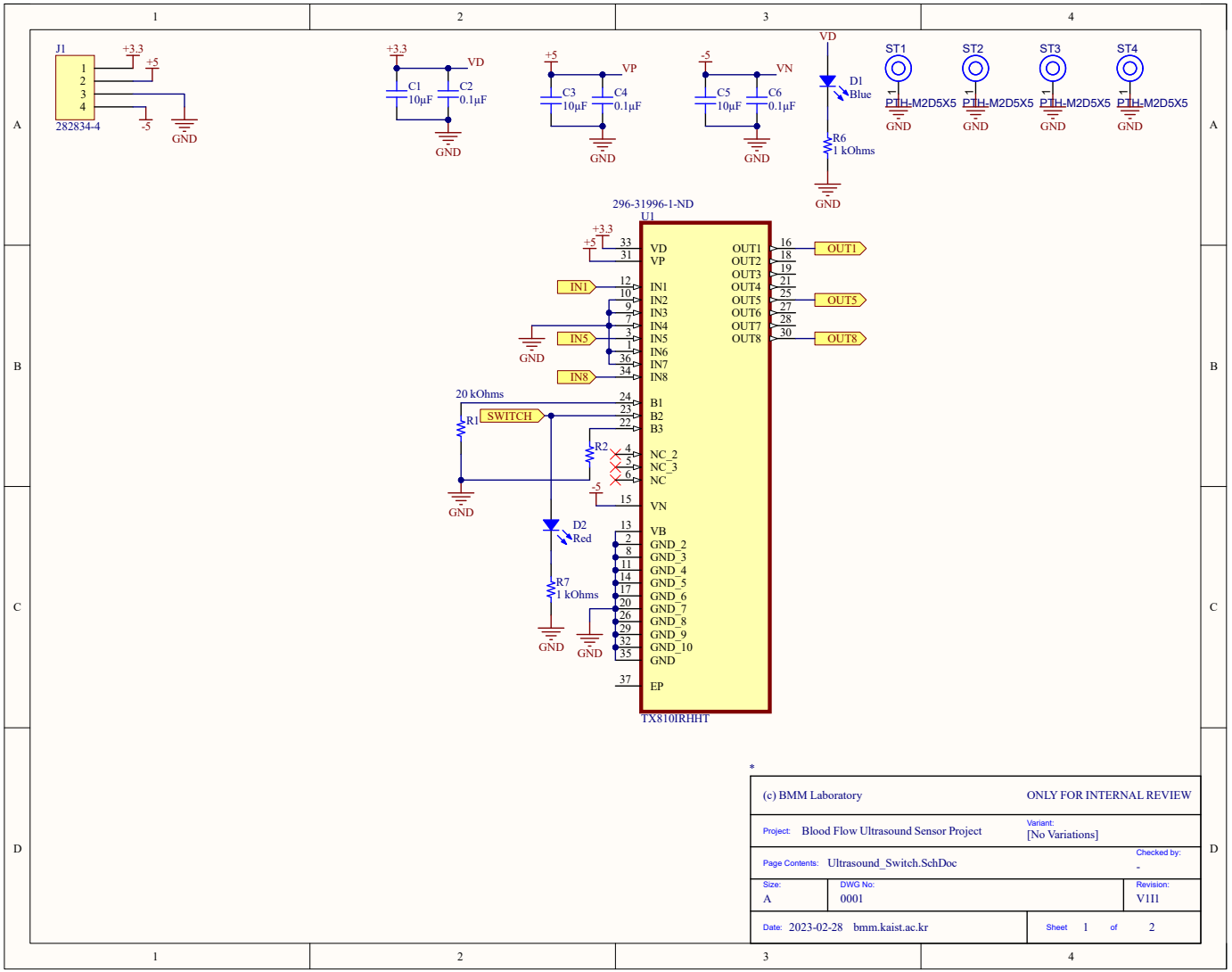



Figure D.8: UltrasoundSwitch Schematic A

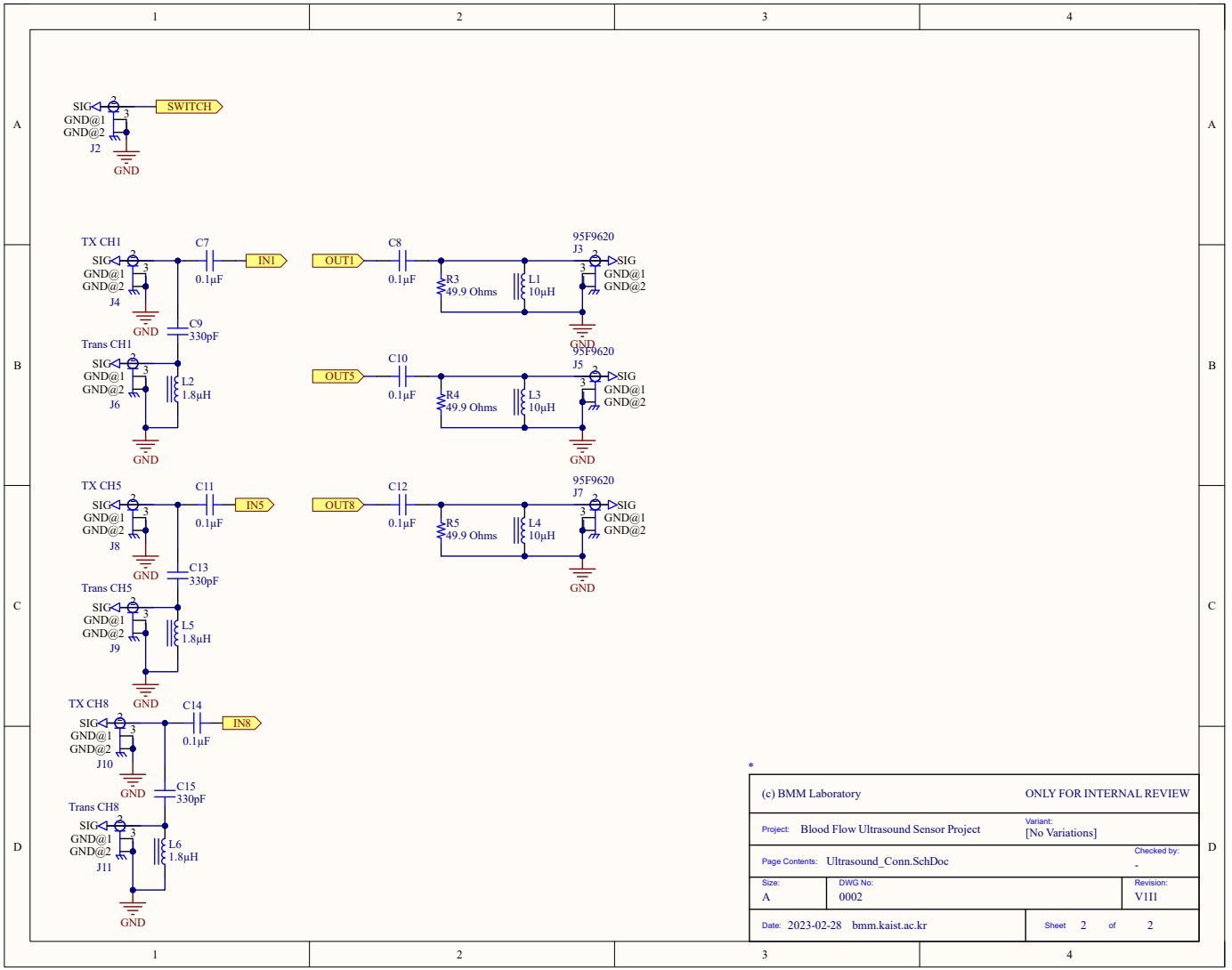



Figure D.9: UltrasoundSwitch Schematic B

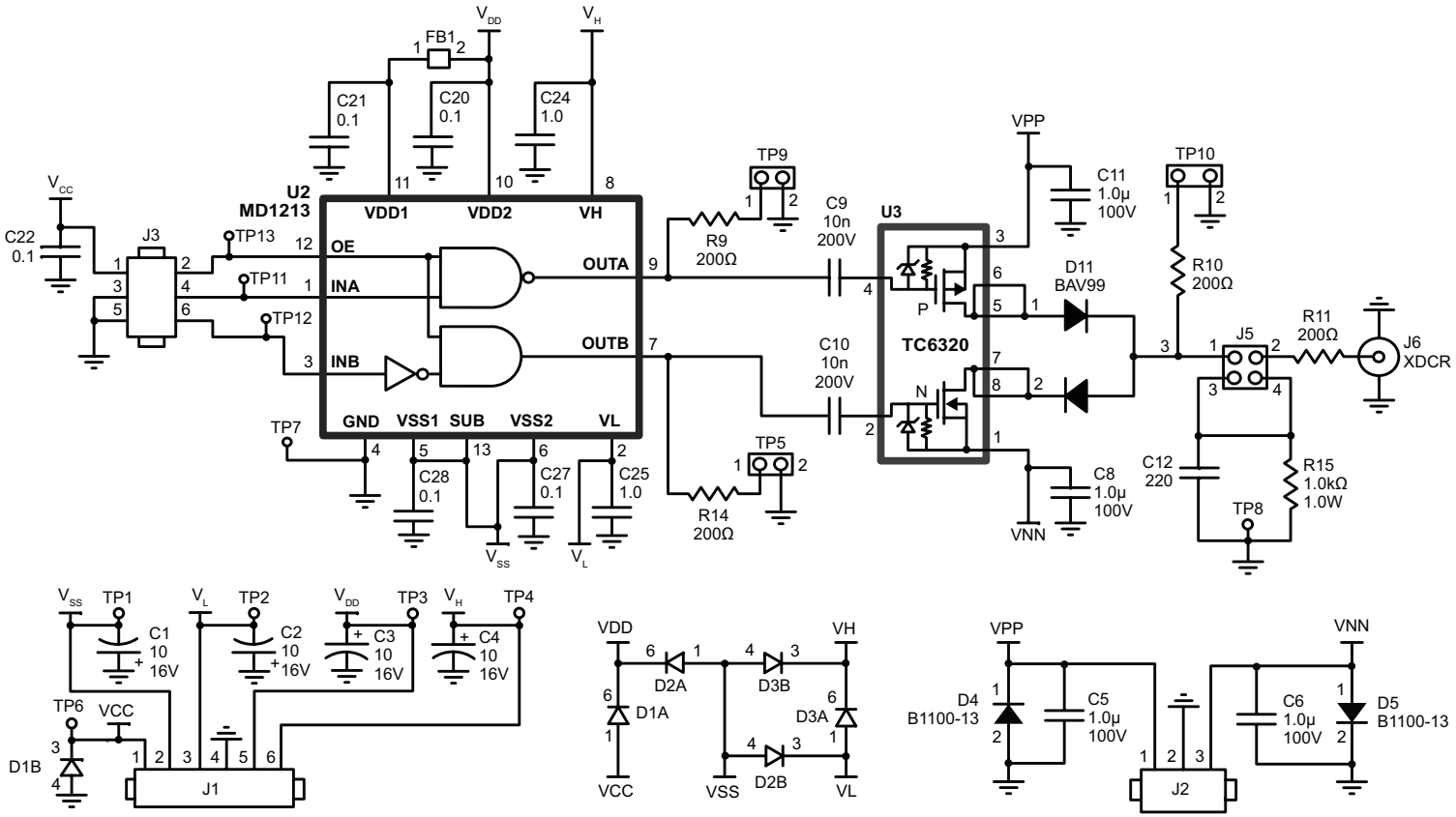



Figure D.10: MD1213DB1 Transmitter Schematic

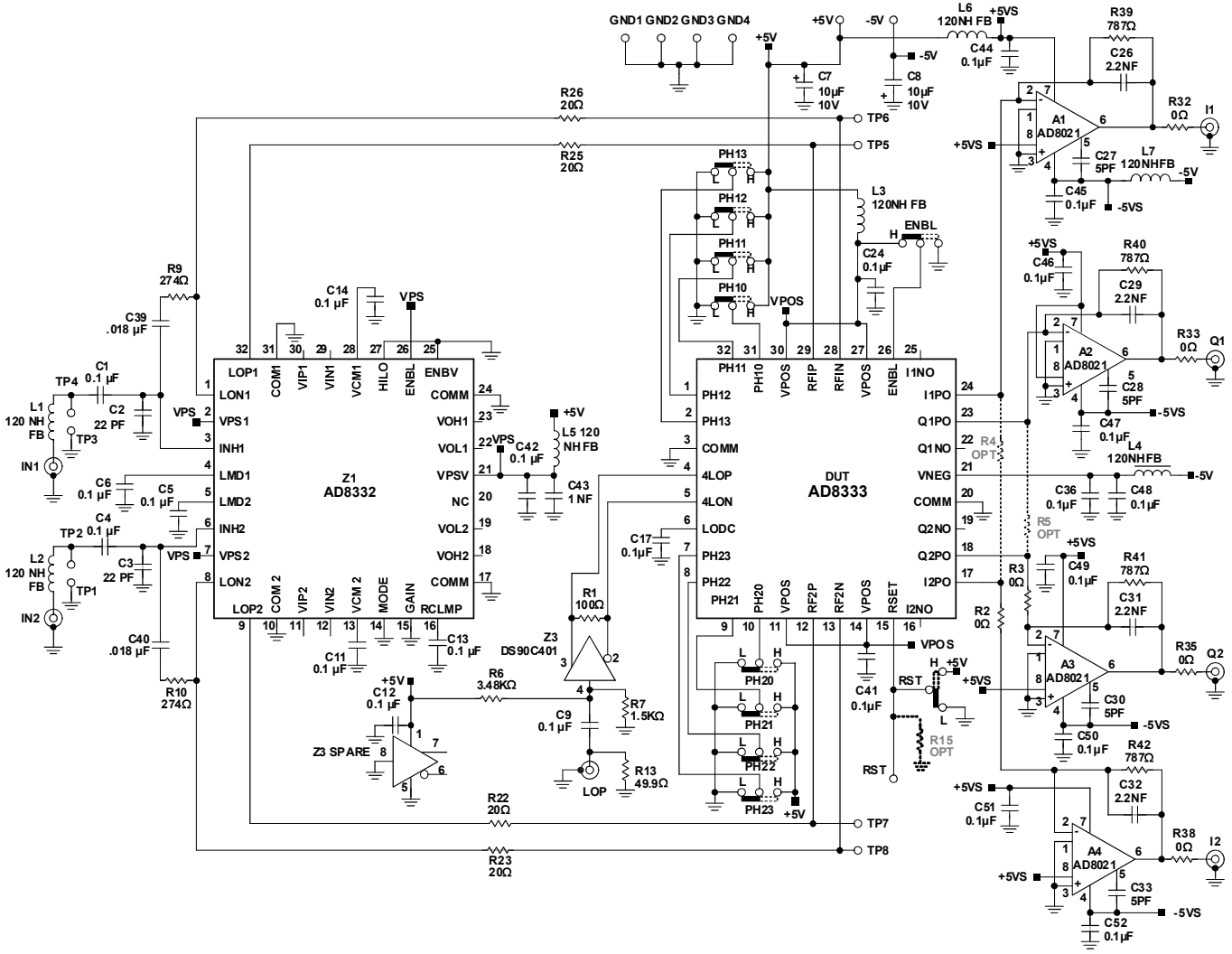



Figure D.11: AD8332 Preamplifier, AD8333 IQ Demodulator Schematic

# Appendix E

# Circuit CAD Assembly Documentation

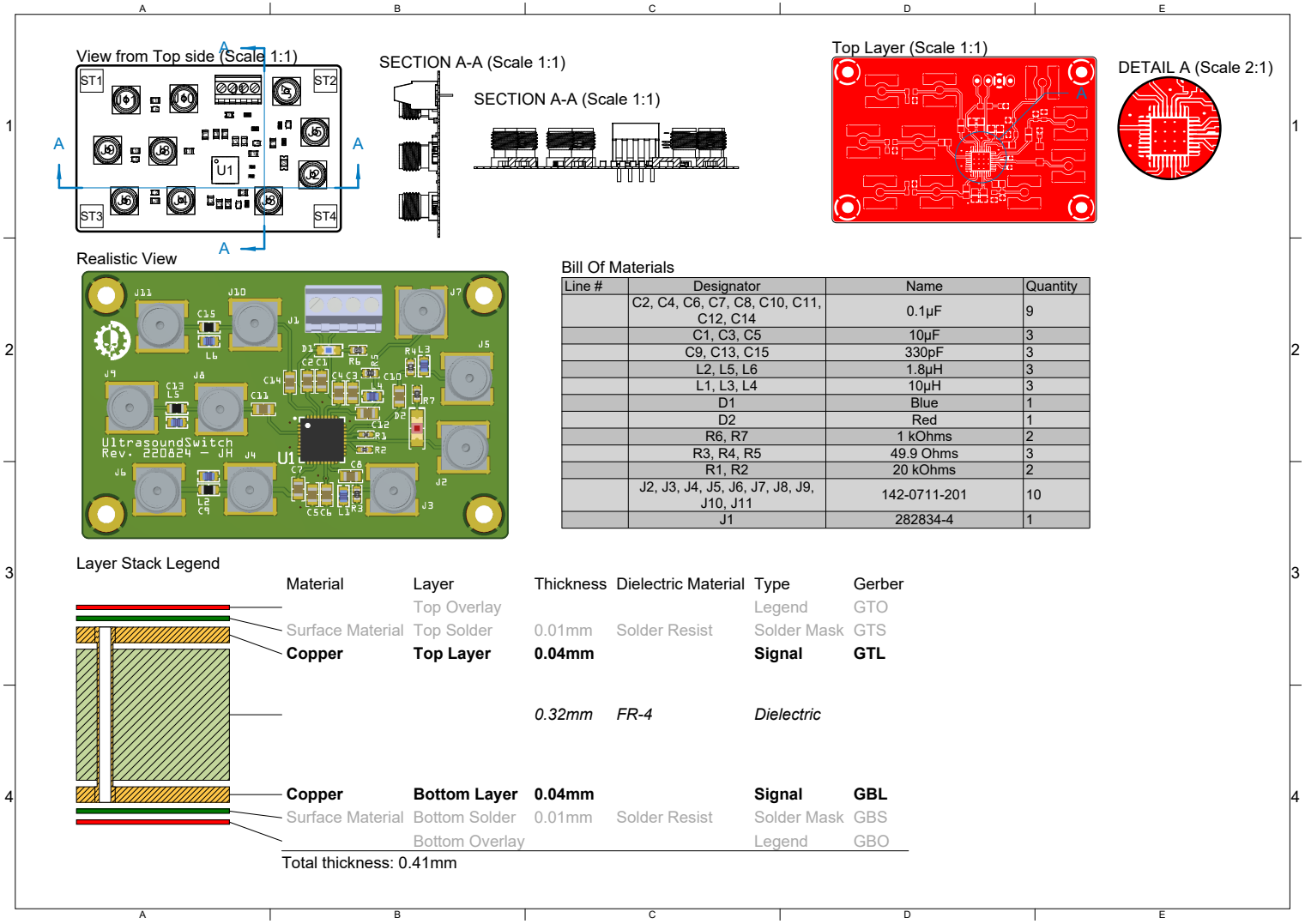



Figure E.1: UltrasoundSwitch Assembly Information

# Appendix F

# Instruments

Table F.1: List of instruments used for solder work

| **Function** | **Manufacturer** | **Model** |
|---|---|---|
| Visual inspection microscope | Leica | A60 |
| Manual soldering | Weller | WX2 |
| Heat gun | Thermaltronics | TMT-HA600-2 |
| Solder paste | Chip Quik | SMD291AX250T3 |
| Solder flux | Chip Quik | SMD291NL |
| Reflow oven | Puhui | T-962A |
| DMM | Fluke | 175 |

Table F.2: List of instruments used in experiments

| **Function** | **Manufacturer** | **Model** |
|---|---|---|
| DCPS 1 | RIGOL | DP832A 200W |
| DCPS 2 | Keysight | E3631A 80W |
| Function generator 1 | Keysight | 33500B |
| Function generator 2 | Tektronix | AFG3102 |
| DMM | Fluke | 175 |
| Transducer (PZT) | HAGISONIC | M715-SB-S 204 5 MHz |
| Transducer (CMUT) | BMM Creation | 6ch 3.3 MHz C.F. |
| RF Amplifier | Tomco | BT00100-AlphaS-CW |
| Oscilloscope 1 | Keysight | DSO-X 2024A |
| Oscilloscope 2 | Tektronix | MSO4054 |
| Physiological simulator | CIRS | Doppler String Phantom 043A |
| Vector Network Analyzer | Agilent | E5071B ENA Series Network Analyzer |