Expertise
and insight
for the future

Nhan Tran

Design and Implementation of a Signal Reading Device for an Analog Output Module

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology Degree Programme

Bachelor's Thesis

30 April 2021

Author	Nhan Tran
Title	Design and Implementation of a Signal Reading Device for an Analog Output Module
Number of Pages	46 pages + 1 appendix
Date	30 April 2021
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Smart Systems
Instructors	Marco Välimäki, Senior Hardware Specialist, EKE-Electronics Ltd. Keijo Länsikunnas, Senior Lecturer

An Analog Output Module (AOM) is a multichannel signal generating device provided by EKE-Electronics Ltd, and there is a need for monitoring the AOM channels concurrently during testing phases or customer project work at the company. In this project, a prototype of a device was designed and implemented to measure different signal types such as voltage, current, and the PWM signal from the AOM. The analog signals were read by being dropped on a resistor, the value of which could be altered depending on the signal type. This was done to produce a voltage as an input to an ADC integrated circuit. On the other hand, the AOM PWM signal could be measured by an input capturing the peripheral of a microcontroller chip.

The obtained results do not meet the desired requirements perfectly. The signal reading device is not entirely capable of detecting the smallest changes of the AOM signals. However, in conclusion, the results do satisfy the demand of testing and monitoring the AOM signals. The device is able to measure the AOM channels simultaneously, and the errors are insignificant enough so that the data is considered trustable.

Keywords	AOM, signal reader

Contents

List of Abbreviations

ADC	Analog to Digital Conversion. The process of sampling at a high rate a continuous analog signal and matching the samples to the closest discrete sequence of predefined values.
AOM	Analog Output Module. The modular device, developed by EKE-Electronics Ltd., which can generate industrial standard current loops, bipolar voltages, and PWM signals.
DAC	Digital to Analog Converter. The counter process of ADC which converts digital to analog signals.
HAT	Hardware Attached on Top. A type of add-on boards to be connected on top of Raspberry Pi computers through the computers' header array.
IC	Integrated Circuit. A semiconductor technology that embeds a functional circuit to a compact module called a chip to reduce complexity of circuit designing.
ICU	Input Capture Unit. An STM32 microcontroller's peripheral used for capturing PWM inputs.
LSB	Least Significant Bit. The lowest weighted bit of a binary number.
MISO	Master In Slave Out. An SPI communication line connecting from the master's input to the slave's output.
MOSI	Master Out Slave In. An SPI communication line connecting from the master's output to the slave's input.
MSB	Most Significant Bit. The highest weighted bit of a binary number.
PC	Personal Computer.
PCB	Printed Circuit Board. An electronic circuit built on a solid board.

PWM	Pulse Width Modulation. A modulation method that varies the duty cycle of a rectangular digital signal.
SCLK	Serial Clock. The SPI clock signal driven by the SPI master.
SPST	Single Pole Single Throw. A configuration of switches that only one terminal of a switch connects and disconnects to the other terminal of that switch.
UART	Universal Asynchronous Receiver Transmitter. A serial data transfer protocol widely used by microcontrollers.

1 Introduction

EKE-Electronics Ltd., a subsidiary of EKE Group, is a rolling stock company providing solutions for train system integration, train automation, onboard communication, safety improvement and remote monitoring. One of the products of EKE-Electronics is the Analog Output Module (AOM) which can generate industrial standard analog signals such as current loops and bipolar voltages.

The AOM has up to 24 output channels. In real fleets, it is typical that all the channels are utilized to control different subsystems. This results in the need of testing or monitoring multiple AOM channels simultaneously. Using a multimeter or an oscilloscope for this purpose would be impractical as they both have a very limited amount of measuring channels. Moreover, in the production process, testing AOMs by measuring their channels one after another with multimeters or oscilloscopes is also time and work consuming, and that would be even worse when the number of AOMs increases. For this reason, a device with a sufficient number of input channels and a single result displaying unit would be beneficial in saving time and workload when testing both the AOM in the production process and the software of the customer projects for EKE. Therefore, the goal of this project was to design and implement a signal reading device for the AOM. This device will also be referred to as the signal reader later in this document. The signal reader will measure all the AOM signal types, namely voltage, current and PWM before transmitting the results to a Raspberry Pi computer.

The signal reader was implemented as a proof concept on an add-on board for a Raspberry Pi computer in the form of a HAT board which means there is a PCB size constraint of roughly 65×56 mm for the signal reader [16]. Moreover, all 24 outputs of the AOM were divided into four identical groups, so this project consisted of designing and implementing both hardware and software for the signal reader to read only one group of channels (among four identical ones) from the AOM.

2 Requirements

2.1 Input Side

The signal reading device is cascaded to output group 1 of an AOM. This output group of the AOM contains two heavy duty and two regular analog channels, which are capable of driving up to 1 kΩ and 500 Ω of resistance in the current mode respectively. Each analog channel can be configured to be either current loop transmitters or bipolar voltage outputs. Additionally, there are two other digital pins dedicated to generating PWM signals [18].

The requirements are to detect the operating mode (current and voltage mode) of the AOM and read the corresponding signal. The operating mode detection is done purely by the device itself without user intervention. The reading process must be done fast enough so that the user can observe the real time output of the AOM. The data refresh rate is expected around 1 refresh per second. In addition, the reading device should be able to detect the smallest step of every AOM signal type which are voltage, current, and PWM.

When an AOM channel is configured as a voltage output, its signal varies from -12 to 12 V with the resolution of 1 mV per step. In turn, in the current mode, an AOM channel can source a current from 0 to 24 mA with the resolution of 1 µA. The signal reader's input range must be wide enough to accommodate this signal span, and it must also be precise enough to detect the smallest change of the AOM in both modes. Furthermore, the AOM PWM signal has the frequency range from 10 Hz to 10 kHz with the resolution of 0.1% and the output level of either 5 V or 15 V. This behaviour of the AOM must also be taken into account that the signal reading device is fast enough and has a required resolution to detect the minimum change of the AOM's PWM frequency within its range. The signal reader is also required to adapt with both PWM logical 1 voltage levels.

The device is intended to be an HAT extension board of a Raspberry Pi computer meaning that it will use the power supply from the Pi [15]. Therefore, the risk of over voltage or reverse voltage is considered as out of scope and shall be simplified in this thesis. Instead, another voltage issue is that if the AOM heavy duty channels are configured as current outputs, and are open circuited, their voltage can rise up to 24.5 V thanks to the

AD5755 digital to analog drivers inside the AOM [2, 37]. The signal reader must, therefore, have a solution to deal with such a high voltage. Moreover, if an AOM voltage-configured channel is shorted, the AOM itself has an ability to handle the error by entering the safe mode, and the voltage channel is then disabled. This feature should of course be tested by the signal reader. In other words, there should be a mechanism to simulate the short circuit condition for the voltage channels of the AOM to verify that the AOM channel can handle the error.

2.2 Output Side

All the reading results from all the channels of the AOM need to be transferred to a Raspberry Pi computer via serial connection. The user can run a shell from the Raspberry Pi computer to interface with the AOM signal reader. The data produced by the output of the signal reader is as follows: an individual analog channel operating mode, current and voltage readings, PWM channels' frequency, period and duty cycle. The sending process must be done fast enough so that the user can observe the real time output of the AOM. The expected pace is 1 second between data refreshments.

3 Device Functional Blocks

The signal reader is divided into three main blocks: input, processing, and output blocks. The input block consists of a select-channel subblock, a convert-to-voltage subblock, and a read-voltage subblock. The processing block is a microcontroller doing mode detection and calculations for the measurements. The output block is a virtual one which can be considered as the connection between the processing block and the Raspberry Pi computer. The distribution of the subblocks is better illustrated in Figure 1. The input block first selects AOM channels one by one to route the signal through. It then converts the selected signal into a voltage level called V_{AOM}. The processing block then calculates the AOM measurements based on V_{AOM} before transmitting the results to a Raspberry Pi computer through a shell for displaying to the user.

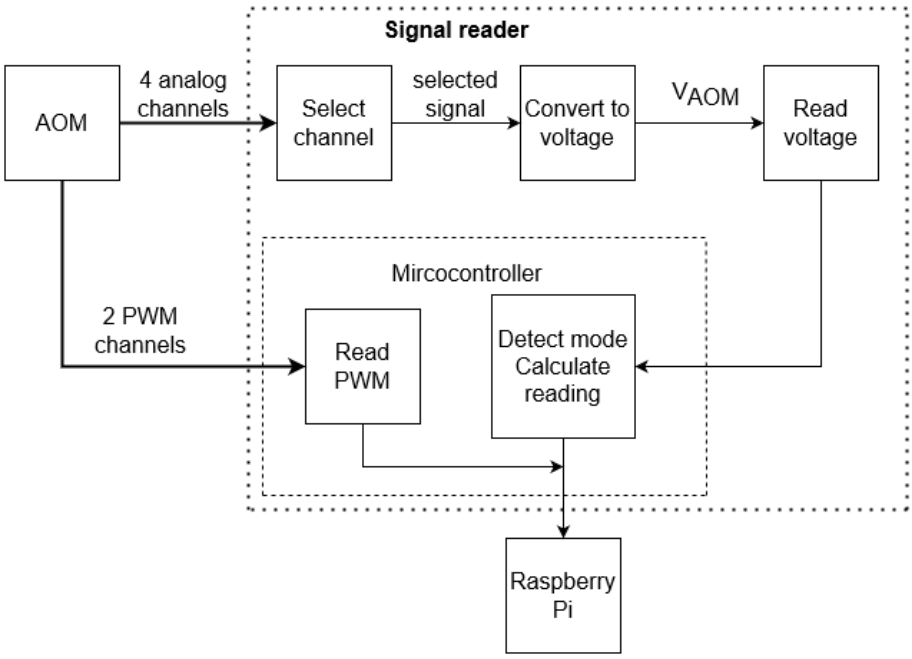

Figure 1: Three main blocks of the AOM signal reading device

The channel selecting block in Figure 1 is a multiplexer (also known as MUX) that allows only one selected AOM signal at a time to pass through to the voltage converting block [3, 1]. This next block is a system of different-value resistors connecting from the output of the MUX to the ground plane. Only one of these resistors is activated or closed circuited at a time depending on whether the AOM channel is in the current or voltage mode. The other resistors will be open circuit or disconnected from the MUX output. What is produced afterwards is a voltage level (V_{AOM} in Figure 1) readable to an ADC chip at the next block. The voltage read by the ADC is then transferred to a microcontroller where the data is used to detect the AOM operating mode or to be simply displayed to the user later. The PWM channels, on the other hand, can be read directly by the microcontroller. The AOM PWM outputs, however, can be configured to generate 15 V for logic level 1 and 0 V for level 0 logic, resulting in the need for a voltage limiting circuit to clamp the 15 V level to a safe value accepted by the microcontroller.

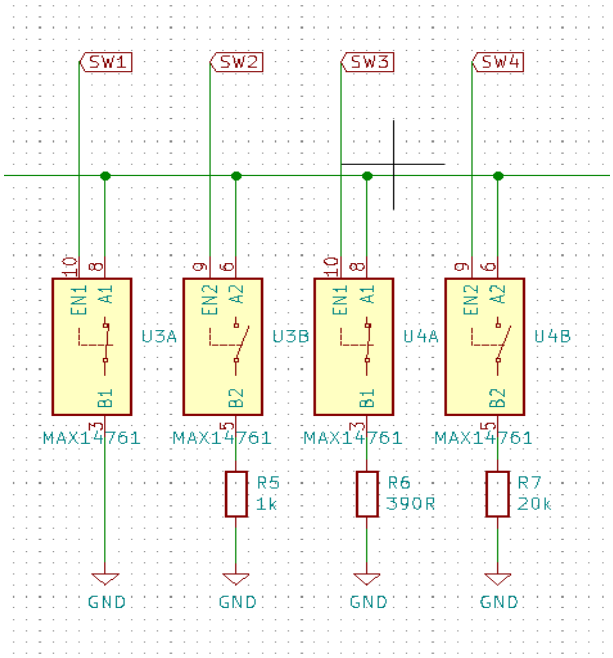

Figure 2: Voltage converting block

As mentioned, the voltage converting block consists of four different resistors 0 Ω, 1 kΩ, 390 Ω, and 20 kΩ connecting from the output of the multiplexer (the previous block) to the ground by turning on corresponding switches as shown in Figure 2 to detect the AOM operating modes and reading the values from it. Only one appropriate resistor is activated depending on the detected mode to result a voltage for the next block. During operation, the AOM voltage can be negative. Thus, regular MOSFET switches are not suitable for controlling the resistors because MOSFETs relies on the gate to source voltage to turn on or off. Analog switches will be employed for this block instead due to their independence from drain, source voltages. More detail will be discussed in the next section. At the last stage of the signal processing chain is the microcontroller for which the STM32F417 will be used. It has a 12-bit ADC peripheral which is insufficient for the resolution of the AOM signals. For that reason, a separate ADC chip is required. Successive Approximation Converter (SAR ADC) and Sigma-Delta ADC are the two most popular types of ADC chips in the market for the time being. The next section will also address the two ADC types and explain why SAR ADC is utilized for the signal reader.

4 Background

4.1 Analog Switch vs MOSFET

4.1.1 MOSFET

In most applications where there is only the presence of positive voltages, MOSFETs are commonly used to switch on or off other electronic components. Nevertheless, as mentioned in the previous section, there is a key feature that analog switches are more preferable to MOSFETs in this project, which is explained below.

A Metal Oxide Semiconductor Field Effect Transistor (MOSFET) is an electronic component consisting of a Source and a Drain terminal [17]. It can block or allow current to flow through the two terminals depending on the voltage applied to the third terminal called the Gate. MOSFETs are categorized into N-channel and P-channel types. For N-channel MOSFETs, when the voltage between the Gate and Source terminal (V_{GS}) is higher than the threshold voltage, typically 3 V or logical 1 level, the transistor will be turned on, and the current can pass through the Drain and Source terminal. When V_{GS} is lower than the threshold voltage (V_{GSTH}), depicting the logical 0 level, the MOSFET is turned off blocking the current flow across the transistor. On the other hand, P-channel works based on the opposite manner of V_{GS}. That is, for V_{GS} being higher and lower than V_{GSTH}, the MOSFET is switched off and on respectively. The behaviour of both types of MOSFETs described above can be considered as a normally open switch or enhancement type. In practice, there is another form of MOSFETs called depletion type. However, that is out of the scope of this project and hence will not be discussed.

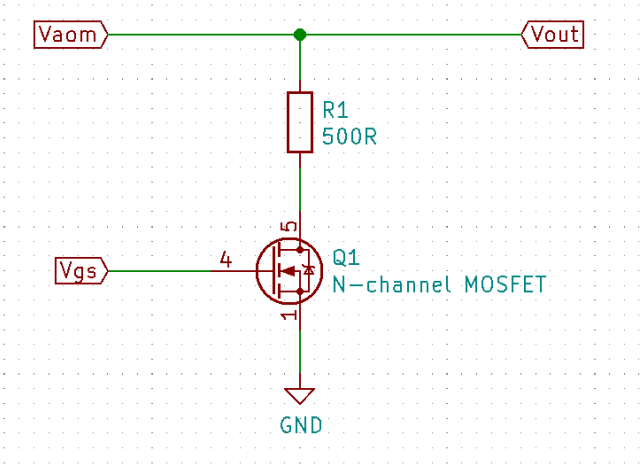

Figure 3: A MOSFET acting as a switch for positive input voltage only

The problem with MOSFETs is that they cannot switch negative voltages. Figure 3 shows that when V_{GS} is higher or lower than V_{GSTH}, the MOSFET is turned on or off respectively and hence it connects or disconnects the resistor to the AOM output channel. This is only true when the voltage from the AOM is positive. When the AOM is in voltage mode where its voltage can be negative, the MOSFET cannot block the signal due to the parasitic diode between the Source and Drain gate. When V_{AOM} is -12V for example, and the MOSFET Q1 is required to disconnect the R1 resistor by pulling V_{GS} to logical 0 level. In that case, Q1 still fails to be off because of the parasitic diode conducting electric current from the ground to the AOM channel. For this reason, the alternative solution is using analog switches.

4.1.2 Analog Switch

Analog switches are also driven by voltage. They inherit while improving MOSFETs' characteristics by having a P-channel connected in parallel with an N-channel MOSFET. The Source and Drain terminal of the P-type transistor are connected to the Source and Drain of the N-type transistor. Their Gate terminals are complementary to each other by the help of a logical NOT gate which is the left side of Figure 4. The right side of the figure is the P and N channel MOSFET. [3, 2.]

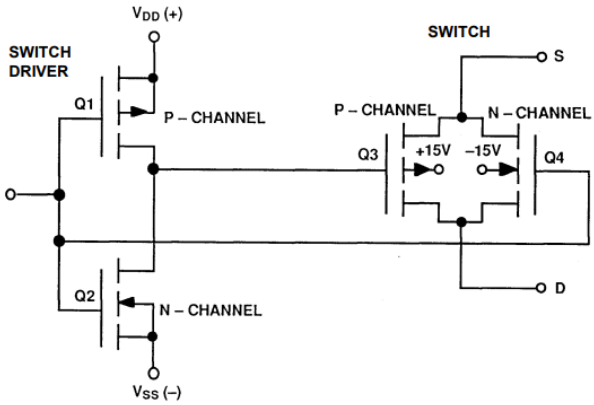

Figure 4: Components of an analog switch. Reprinted from Analog Devices, Inc [3].

Analog switches overcome MOSFETs' disadvantage by having parasitic or body diodes organized in a different way as illustrated in Figure 5. The diodes' locations are where the leakage currents exist. To be exact, the diodes are the source of the leakage depicted in the figure. This arrangement enables analog switches to work both when the Source terminal voltage is larger and smaller than the Drain terminal's because the switches rely on the controlling the Gate voltage with respect to the ground rather than the Gate to Source voltage like MOSFETs do. Despite having a solution to the parasitic diode issue, in practice, analog switches introduce leakage currents that might shift the signal creating an offset error at the output of the switch. On resistance (R_{ON}) – the resistance between the Source and Drain terminals of the analog switches when it is turned on – is also a contributing factor to the errors of the signal path. [3, 5.]

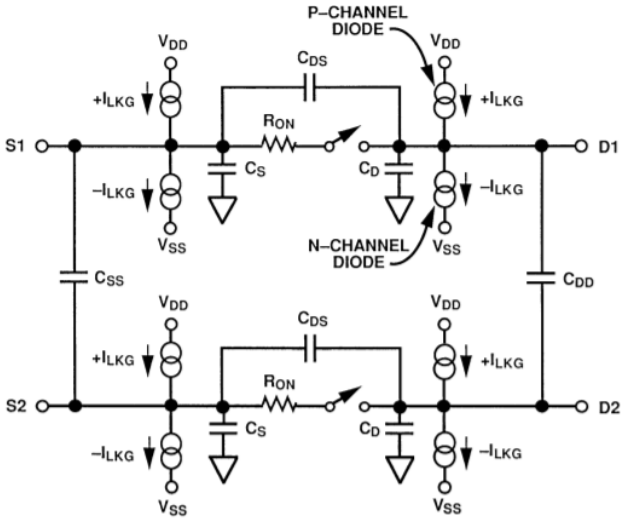

Figure 5: Equivalent circuit of an analog switch. Reprinted from Analog Devices, Inc [3].

It is important to assume that an input signal is applied to the Source terminal of the analog switch in Figure 5, and an output voltage is expected at the Drain terminal. Ideally, when the switch is closed, the output and input are exactly the same. However, in reality, there exists on-resistance between the Source and Drain terminal of the switch distorting the output signal. In addition, the on-resistance usually varies with temperature and input voltage resulting in nonlinearity at the output. This can be significantly improved by having a voltage buffer IC in the signal path because the buffer often has an input impedance of up to megaohms. This will greatly reduce the effect of the on-resistance. Detailed design will be described in section 5.

4.1.3 Multiplexer (MUX)

The circuit design of the signal reader involves a multiplexer for selecting the AOM channel. Multiplexers employ analog switches having one common terminal so that different input can be routed to a single output and vice versa [3, 1]. (See Figure 6.) By using analog switches as building blocks, multiplexers also inherit the error source from the switches which are leakage currents and on-resistance.

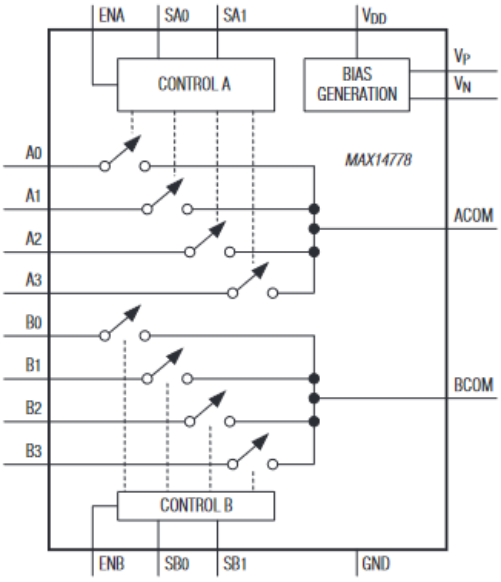

Figure 6: Function blocks of a typical MUX. Reprinted from Maxim Integrated, Inc [4].

By providing different combinations of logic levels to the control inputs (SA0 and SA1 of unit A in the figure), a specific terminal from A0 to A3 will be routed to the common terminal ACOM. The same principal also applies for unit B of the multiplexer.

4.2 Analog to Digital Converter

4.2.1 Successive Approximation Converter

Successive Approximation Register Converter, or SAR ADC, is one of the most common ADC types available in the market for the time being. This type of ADC has low power consumption, and latency since the ADC continuously samples the input signal and does the conversions [6]. This makes SAR ADC suitable for measuring multiple signals simultaneously. The SAR converter only performs one sample for each conversion, and the sampling process is triggered by a starter signal. As a result, this ADC type is also usually put in use in applications that require precise control of when the samplings occur. Moreover, many models of the SAR ADC architecture are provided in small size packages which are also suitable for space constrained PCBs, for example, a circuit board that is attached on top of a Raspberry Pi computer. [6.]

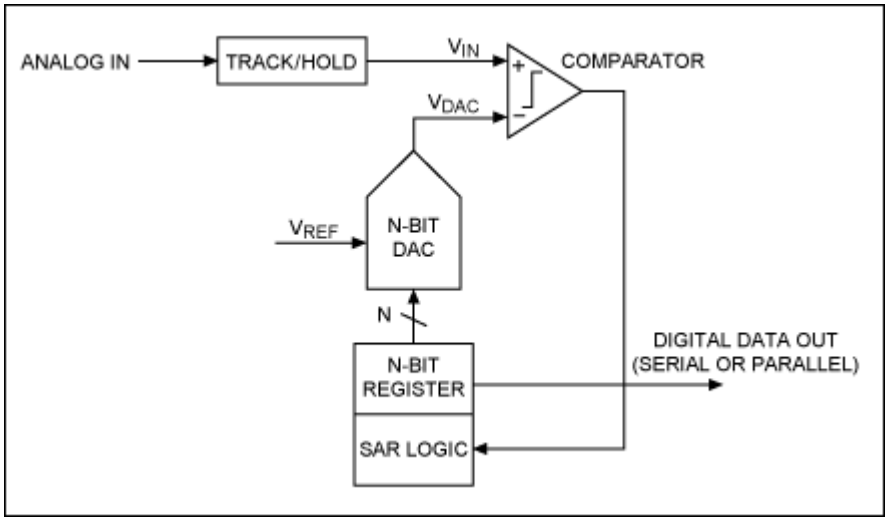

Figure 7: SAR ADC function blocks. Reprinted from Maxim Integrated [5].

Figure 7 above illustrates elemental function blocks of a SAR ADC device. It has a hold register to store the value of input voltage to be measured. This voltage is then compared with V_{DAC} – a voltage produced by a Digital to Analog Converter (DAC) element. In general, the DAC block is to replicate the input voltage with the help of the comparator. It keeps adjusting its value converging to the input as long as the comparator perceives a difference between V_{IN} and V_{DAC}. The digital value of the DAC block that produces the closest voltage to the input voltage is then the measurement result. The DAC block is governed by the content of the N-bit register which is afterwards the result register of the

SAR ADC. To be specific, at first, all the bits in the register are reset to 0s, except the MSB which is set to 1. This register setup yields a voltage at the output of the DAC (VDAC) being VREF/2. After that, VDAC is compared with VIN to check whether VIN is higher. If this true, the MSB is kept as 1. Otherwise, it is reset to 0. After that, the SAR logic block moves the interest to the second MSB of the N-bit register. This bit is also forced to 1 and is combined with the found MSB. The DAC voltage is produced according to the value of the N-bit register, and the procedure is repeated until the LSB value is found. The content of the N-bit register is the AD conversion result. [5.]

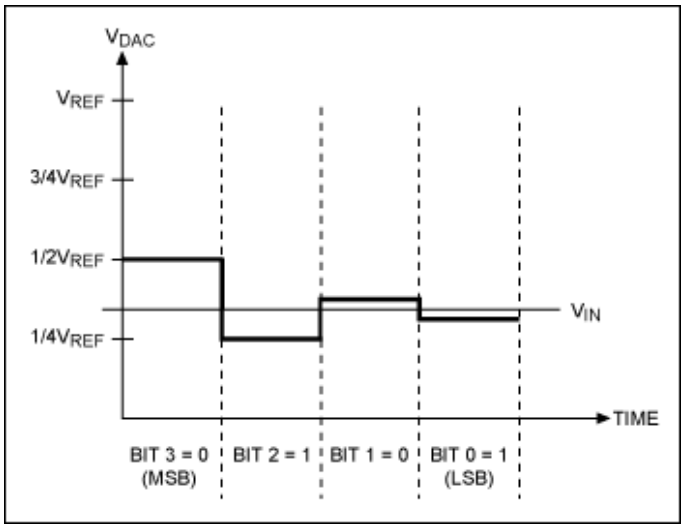

Figure 8: Decision made to generate the N-bit register value to replicate input voltage. Reprinted from Maxim Integrated [5]

Figure 8 describes how the DAC voltage is generated according to the 4-bit register. At first, the register value is 1000. VDAC is then VREF/2 and is greater than VIN resulting in the MSB reset to 0. Secondly, the next MSB is set to 1, so the register value is 0100 which is 4 in decimal or VREF/4. This voltage is lower than VIN, and, of course, the currently considered bit is 1. Similarly, the logic repeats until all four bits are found. [5.]

4.2.2 Sigma-delta ADC

Sigma-delta ADC uses over sampling, digital filtering, and noise shaping to achieve high SNR performance. The other advantages of this ADC type are also high resolution and integration, and low power consumption. When an analog signal is sampled and digitalized by an ADC in general, the output signal is introduced with some distortion since a finite number of samples is used to rebuild a continuous signal of infinite values. [7.] This

distortion also known as the noise floor of the output can be visualized by FFT analysis in Figure 9. The name noise floor depicts that the noise frequencies are distributed equally from zero to half of the sampling frequency.

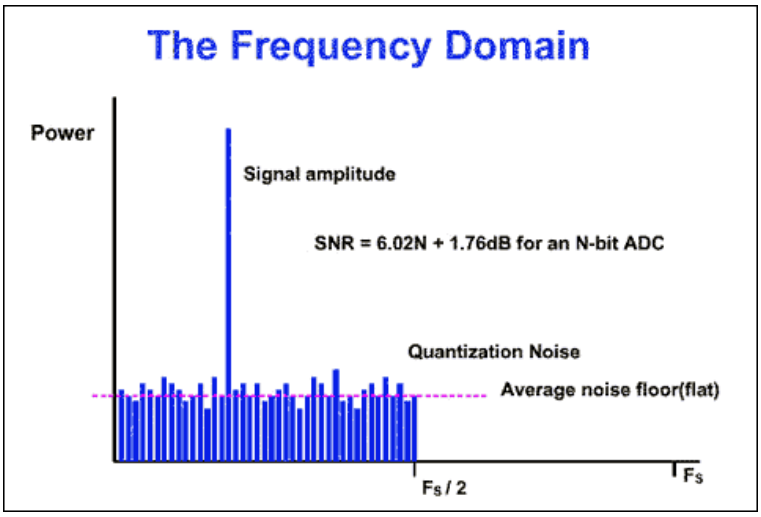

Figure 9: FFT of digital output. Reprinted from Maxim Integrated [7].

In Sigma-Delta ADCs, the noise magnitudes are greatly reduced since the sampling frequency is raised to a considerably high value, and the noise power is hence spread over a much wider frequency range. As shown in Figure 10, the noise attenuated by over sampling is then filtered even further by a digital lowpass filter. The part of the noise, colored in red, which has higher frequency than the cut-off frequency of the filter will be supressed leaving only the signal and a small amount of the low frequency noise. [7.]

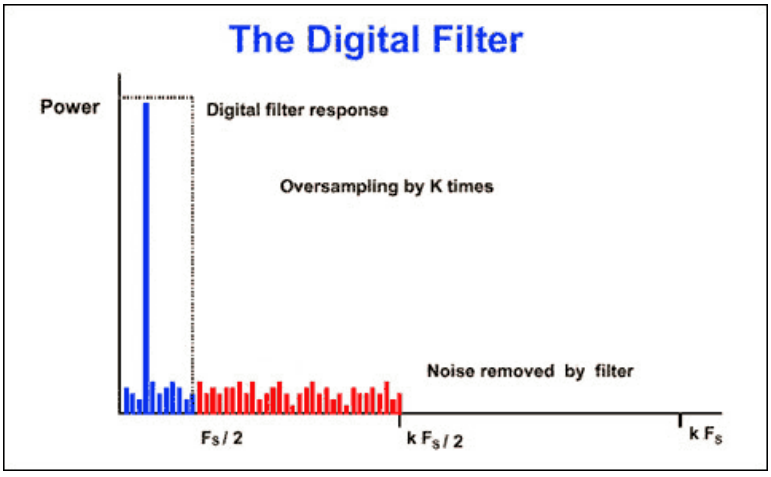

Figure 10: Reduced noise magnitude is further filtered. Reprinted from Maxim Integrated [7].

The remaining noise power discussed above can be eliminated even more with a noise shaping technique in which the input signal will be fed into a sigma-delta modulator as described in Figure 11. This modulator acts as a lowpass filter to the signal and a highpass filter to the sampling noise, so what is produced at its output is a low frequency sampling noise portion being distributed in a higher frequency area. [7.]

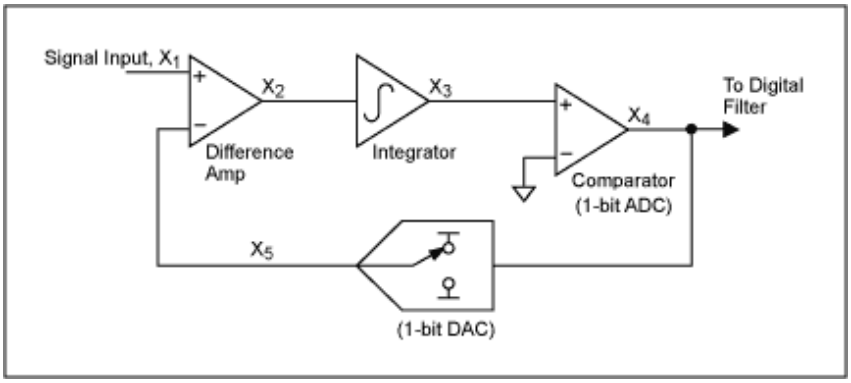

Figure 11: Sigma-delta modulator. Reprinted from Maxim Integrated [7].

Figure 12 shows the noise distribution at the output of the integrator. If this signal is fed into the digital filter mentioned in Figure 10, the remaining noise power is even less [7]. This is to prove that Sigma-delta ADC is a considerable option for high SNR performance.

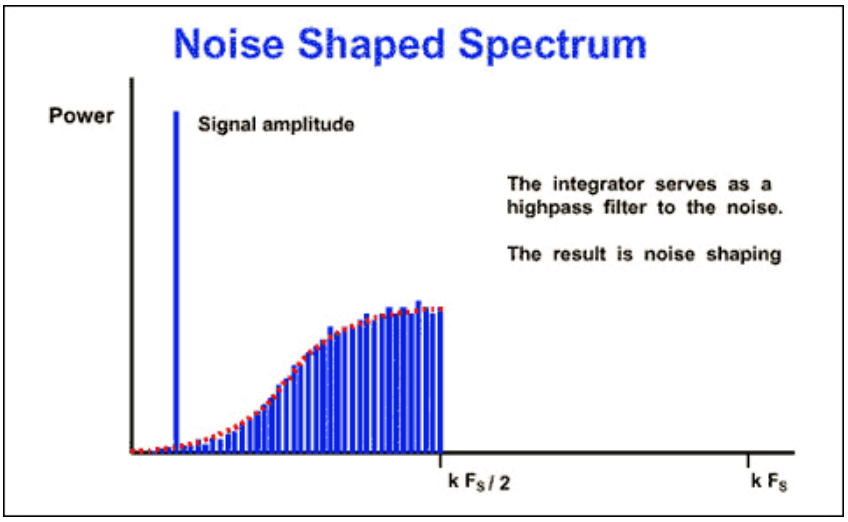

Figure 12: The noise distribution created by the integrator. Reprinted from Maxim Integrated [7].

Despite the outstanding performance of Signa-Delta ADCs in signal-to-noise ratio, the latency of this type of ADC is quite high since Signa-Delta ADC does not have sample-

and-hold mechanism as SAR ADC does. It will take a longer time for a Signa-Delta chip to read multiple AOM channels simultaneously than its competitor – SAR ADC. That is the reason SAR ADC will be applied in this project.

5 Block Design and Interface Definition

5.1 Input Block

Figure 13 shows the circuitry to process four AOM current or voltage signals to a voltage readable to the ADC block. Starting from the AOM outputs, 1 out of 4 of the signals is routed by a MAX14778 multiplexer. The most striking feature of this IC to be adopted is the support for a wide input signal range up to ±25 V. Moreover, there are two multiplexer units in 1 single 5 mm × 5mm package. [4, 1.] Although only one unit is currently employed, it is advantageous for future development when more AOM channels are to be monitored. The unused MUX B of the multiplexer will be disabled by grounding the ENB pin with a 10 kΩ resistor, and bypass capacitors are also needed for the stability of the IC. The capacitor values are selected as recommended by the datasheet [4, 10]. At the input side of the MUX, there are four transzorb diodes SMAJ18A to ground any transient voltage that is more than 18 V for each AOM channel before it enters and damages the MUX. Cascaded to the multiplexer is the voltage converting block which takes both the current and voltage signal and produces a voltage level at its output. By alternating three different resistors, 390 Ω, 1 kΩ, and 20 kΩ, the signal reader can adapt to the AOM operating mode. The software is implemented so that only one of them is active (connecting the signal to the ground by closing the corresponding analog switch) at a time. Even though there is a variety of different types of analog switches in the market, the ones that support over-the-rail signal voltage are quite limited. An impressive candidate for a large signal range application like this project is the MAX14761 - a double SPST analog switch having decent performance such as low on resistance (2 Ω maximum), on resistance flatness of 5.1 mΩ, and leakage current, wide input signal range. MAX14761 is also a compact dual single pole/single throw (SPST) switch. It is available in a 10-pin TDFN package with the size of 3 mm × 3mm. [9, 2.] Two packages will be needed because three switch units are required for the resistors and one for simulating the short circuit condition for the AOM in which the AOM enters a safe state.

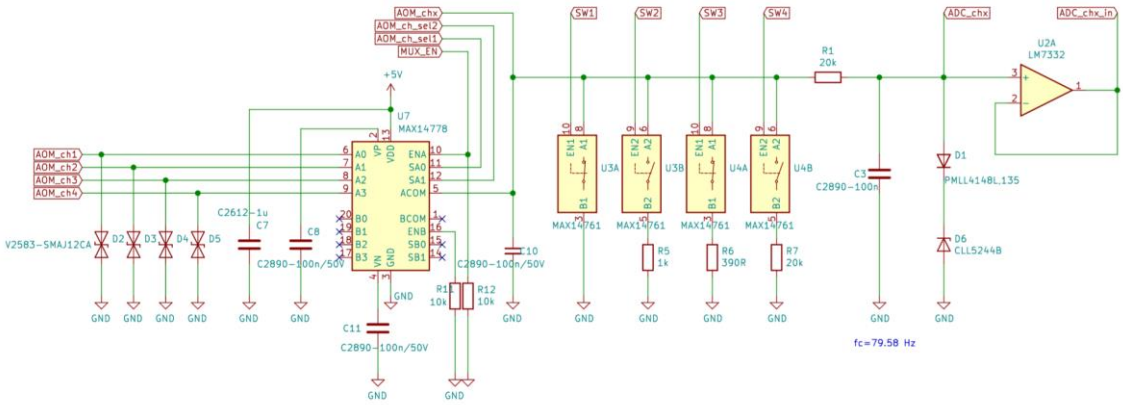

Figure 13: From 4 current/voltage signals to 1 readable voltage

To continue with the signal flow, the voltage from the active resistor described previously is passed through an RC lowpass filter (R1 and C3) to trip off the unwanted high frequency noise before being fed to a voltage limiting unit. The filter's cut-off frequency is calculated by the following formula.

$$f_c = \frac{1}{2\pi RC} \approx 80\ Hz$$

The circuit is followed by a Zener diode D6 - CLL5244B to clamp the AOM voltage to 14 V in case the AOM produces higher voltage than the signal reader's safety limit. This is further discussed in the next section. The purpose of D1 is to block the current from the ground to the signal wire ADC_chx when the AOM is outputting a negative voltage in the voltage mode. The signal chain is ended by a buffer op-amp LM7332 for safety in case the AOM channel is overloaded. Again, this buffer is chosen due to its support for high dynamic range input with typical input range from -15 to 15 V [10, 1].

5.1.1 Reading Current Signal

Within one AOM output group, there are two channels which can provide 24 mA to a 1 kΩ resistors in the current mode. On the other hand, the remaining analog channels are capable of driving up to 500 Ω resistors also at 24 mA. According to Ohm's law presented by equation (1), the resistor value will be selected to be 390 Ω. Switch 3 in Figure 13 will

be turned on routing the current through R6 to the ground. As a result, the voltage drop between the plus and minus pin of the current channels is from 0 to 9.36 V as the AOM outputs 0 to 24 mA of current. Similarly, the Ohm's law also shows that the voltage drop will increment/decrement 0.39 mV for every 1 µA of change in the output current.

$$V = I \times R \quad (1)$$

When the current channels are over driven by being applied with a resistance greater than the rated value or even open circuit, the voltage at AOM channels 3 and 4, which are capable of providing 24 mA to a 500 Ω resistor, is driven to the maximum value of 12 V. On the other hand, channels 1 and 2, which can drive up to 1 kΩ at 24 mA, generate about 24 V. This voltage is limited to a safe number of 14 V by the Zener diode mentioned in the previous section unless it will damage the ADC channel of the signal reader. To sum up, for the input current from 0 to 24 mA and a step change of 1 µA, the produced voltage after going through the input block is from 0 to 9.36 V with the step of 0.39 mV.

In practice, active components like the transzorb and Zener diodes, multiplexers, analog switches, buffer, and the ADC in Figure 13 introduce a considerable amount of leakage current as an error to the reading results. That is because in the current mode, the AOM channels output constant current signals without caring if the produced current gets lost during the path to the signal reader. During the operation of an AOM channel, the total leakage current from the AOM to the processing block involves the multiplexer the on-leakage current of input and output pin, the on-leakage current of one analog switch and off-leakage current from the three others since there is only one conducting analog switch at a time. The summation of the listed leakages can be considered as a constant offset leakage since it makes the signal reader measurements drift away from the true values. Fortunately, it can be compensated by software. The process is quite simple. First, the AOM is configured to generate zero current. The signal is then measured by the signal reader. The readings are expected to be non-zero due to the offset current. This non-zero value is the offset error of the design, and in the future measurements, the final results will be the subtraction of the crude readings from the AOM by the offset. For example, when the AOM is putting out 0 mA, the signal reader perceives that as 20 mA. This number is the offset current. Hence, in later reading, if the measurement is 30 mA, the true value is in fact 30 mA – 20 mA = 10 mA.

5.1.2 Reading Voltage Signal

The AOM voltage channels can be monitored directly by the signal reader. The driving capacity of each AOM channel is 10 mA. Therefore, the minimum resistance between the plus and minus pin of the AOM is 1 kΩ. In fact, the output spans from -12 to 12 V with a 1 mV step instead of just -10 to 10 V due to the over range ability of the AOM. Hence, in case of over range operation, the resistance between the plus and minus channel must be at least 1.2 kΩ so that the current is within a driving capacity of 10 mA.

In the voltage mode, the active resistor value is 20 kΩ, and switch 4 in Figure 13 is turned on so that resistor R7 is connected while the other switches are turned off to disable the corresponding resistors. Fortunately, leakage currents have no effects to the signal received by the processing block. This is because the AOM channel, which is a voltage source in this case, will regulate its current to make sure the voltage is the same as the designated value. In other words, the input block's input voltage is equal to its output voltage.

$$V_{in} = V_{out}$$

That is the theory. In practice, the multiplexer offers an on resistance between its input and output terminal when a MUX channel is selected. To make it worse, this resistance is dependent on temperature and other factors for example the input voltage of the multiplexer. This dependency is often referred to as on resistance flatness. [4, 3.] As a result, even if the voltage at the multiplexer input is regulated, the output side voltage can be different than that. Being similar to the offset in the current reading mode, the voltage difference can be eliminated by calibration when the AOM is outputting zero voltage signals. Later in the testing phase of the project, it is found that the ADC chip also offers an error proportional to the AOM voltage signal which means the so-called gain error will change linearly with the input voltage [11, 1]. This type of error escalates when the internal reference voltage (4.096 V) of the ADC is selected. In this project, the internal reference selection cannot be done differently because the supply voltage of the ADC is ±15 V. Choosing the available 5 V from the Raspberry Pi will result in the ADC input span from -15 to 15 V, which is the same as the power supply, and is not a good practice. Due to the linearity of the error, it can also be calibrated by software. The procedure will be

addressed in the testing section later when the slope of the line is known by collecting measured values corresponding to AOM signals.

5.1.3 Mode Detection

The operating mode detection of a channel can be done by assuming that the AOM is in the voltage mode and closing switch 4 as shown in Figure 13. If the AOM is actually in the current mode, the 20 kΩ resistor R7 surely overloads the AOM current channel, and the maximum capable voltage will be drawn from the AOM channels. This maximum voltage is typically 24.5 V for channels 1 and 2 and 15 V for the other analog channels. With the help of the voltage clamp circuit, the mentioned voltage is kept under 15 V as discussed in 5.1.1. This is distinguishable from the voltage mode because in that case, the AOM can only output maximum 12 V even in the over range condition. If, on the other hand, the voltage received from the AOM channels is lower than 12 V, it is still uncertain to tell the operating mode yet. Further action can be done by activating switch 2 to enable the 10 kΩ resistor. If the voltage drop on the active resistor retains the same as the last voltage drop, the AOM channel is in the voltage mode. Otherwise, it is in the current mode.

5.1.4 Reading PWM Signal

The PWM reading method is using the microcontroller's timers. There are two independent timers needed. The first timer measures the time between two consecutive rising edges to obtain the period while the second timer is used to get an interval between a rising and the next falling edge for the duty cycle. There is still an issue with the AOM's PWM high level voltage of 15 V. This level is far beyond the safe margin of the microcontroller STM32F417 input. The high voltage can be overcome by utilizing Zener diodes D7 and D8 (see Figure 14) the Zener voltage of which is 3.3 V. This means the voltage at the cathodes of these diodes is regulated to 3.3 V. Resistors R14 and R15 are to limit the current through the diodes.

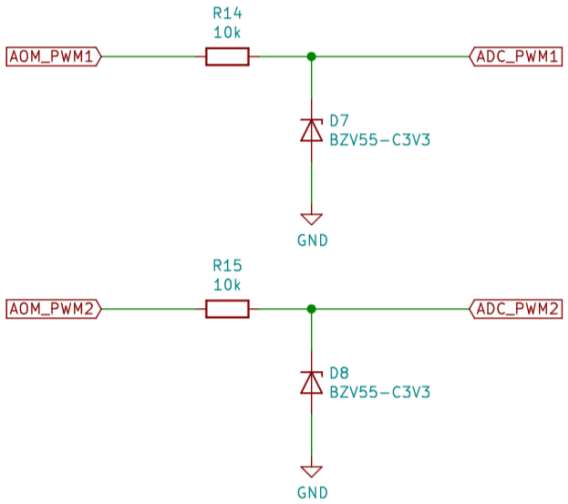

Figure 14: PWM reading circuit

The microcontroller STM32F417 has a dedicated peripheral for reading PWM signals called ICU (Input Capture Unit) driver which will ease the pain of manually setting up the timers and reading count values in the timers' registers. The software configuration to enable this peripheral is further described in this paper in a later section.

5.2 Processing Block

5.2.1 Analog to Digital Converter Block

The selected ADC device for this project was MAX1301. It is available in a 20 pin TSSOP package offering four reading channels in which only one is used [12, 1]. The remaining three channels will be useful for the future development of the project when all four channels of the AOM are to be monitored. Again, this ADC IC was chosen for its wide input dynamic range. Having an option of input signal range from -3×V_{REF} to 3×V_{REF}, and the reference voltage of 4.096 V, the dynamic range of the ADC will be from -12.288 V to 12.288 V. [12, 1.] The MX1301 has a 16-bit resolution, so within the full-scale range, there are 65536 different ADC codes. This means the step size is:

$$\frac{12.288 - (-12.288)}{65536} = 0.375\ mV$$

The smallest change of AOM signals in the voltage mode is 1 mV, and 1 µA in the current mode will result in 0.39 mV as discussed previously in section 5.1.1. The ADC's resolution is just enough to handle the AOM signal step change in the current mode.

5.3 Output Block

The output block utilizes the UART protocol to both transmit measurements to and receive commands from the Raspberry Pi's shell. As already mentioned in section 3, the output block is required to send data fast enough for data visualization, so it would be beneficial to identify the data payload size in every operating mode. With a given UART baud rate, the time required to transmit a data packet, which is the minimum duration between transmissions, can then be determined. Data will be presented on the shell console in two tables, analog and PWM. The user interface is shown below.

```
Live analog signals:
!-------------------------------------------------------------!
| Channel    | 1          | 2          | 3          | 4          |
|------------|------------|------------|------------|------------|
| Mode       | Voltage    | Current    | No signal  | No signal  |
|------------|------------|------------|------------|------------|
| Value      |  11.999 V  | 12.345 mA  | N/A        | N/A        |
!-------------------------------------------------------------!
Live PWM signals:
!-----------------------------------!
| Channel      | 1        | 2        |
|--------------|----------|----------|
| Frequency Hz | 10000    | 10000    |
|--------------|----------|----------|
| Period    ms | 099.1    | 099.1    |
|--------------|----------|----------|
| Duty Cycle % | 077.7    | 066.6    |
!-----------------------------------!
Enter 'q' to quit.
```

In the current loop mode, the current varies from 0 to 24 mA with the resolution of 1 µA. The measurement of each AOM current channel has, therefore, nine characters in the form of "AB.XYZ mA" and "Current" for the *Value* and *Mode* row respectively. The column of channel 2 in the *Live analog signals* table above is a good example for this. Similarly, the voltage output mode readings range from -12 to +12 V with a resolution of 0.5 mV, and the format is "-AB.XYZ" or "+AB.XYZ mV" for the *Value* row and "Voltage" for the *Mode* row. The worst case, where the maximum payload is reached, happens when all the channels are of the voltage type. There will be 68 bytes to be transmitted for the analog table in this case. Lastly, for the PWM channels, the data includes a frequency

maximum value of which is 10000 Hz and a duty cycle with the largest percentage of 100% and 0.1% of resolution. For the PWM table, the fix payload to be transferred is 46 bytes for both channels. The whole display will need 114 bytes in total.

Each character is transmitted by a UART frame which normally contains 1 start bit, 5 to 9 data bits, 1 parity bit and 1 stop bit [13, 1]. In this project, the payload is configured to 1 byte for each UART frame and the parity bit is suppressed. This results in a 10-bit UART frame. There will be no parity bit attached to the frame because communication between the output block and the Raspberry Pi occur in real time. There will be a decent traffic in the communication channel, so the probability for the user to witness the error data on the display is considerably low. With 114 characters embedded in 10-bit frames, there will be a total of 1140 bits for every transmission from the output block to the Raspberry Pi. If a baud rate of 115200 kbps is selected, the time required to send that payload is calculated as below.

$$Transmission\ time = \frac{payload\ size}{baud\ rate}$$

$$Transmission\ time = \frac{1140}{115200} = 11.81\ ms$$

This transmission time is also the minimum duration required between transmissions. In this project, the time between transmissions is chosen to be 100 ms since it clearly satisfies the 11.8 ms requirement, and the cycle time of the AOM program is 100 ms as well.

6 Schematics and Software Implementation

6.1 Schematics and Component Selection

Figure 15 shows the early stage of the input block where the signals from the AOM channels are multiplexed by MAX14778 so that an interested channel is selected among four channels. First, 4 transzorb diodes SMAJ18AC will ground any transient voltage exceeding 19 V at the 4 AOM channels [14, 2]. The rated power of every transzorb diode is 400 W [14, 1]. Next, the multiplexed signal, in the form of voltage or current, is then input to the analog switch MAX14761. This results in a voltage at AOM_chx which is later low-pass filtered by an RC circuit (C3 and R1). The cut-off frequency of this filter is approximately 80 Hz to suppress high frequency noise while keeping the low frequency signal. The purpose of the Zener diode CLL5244B is to limit the voltage at ADC_chx in case a high capability current channel from the AOM is open circuited by the analog switches. These channels can produce up to 24.5 V at ADC_chx turning the general diode PMLL4148L into the forward bias state. A voltage of 14.8 V is created by the combination of diode PMLL4148L's forward voltage and the Zener voltage of diode D6. This voltage is a safe limit for the buffer LM7332 the functionality of which is to prevent the AOM channels from being overloaded. The approach of using a general purpose and a Zener diode instead of a transzorb diode is that the breakdown voltage response of transzorbs is not as sharp as that of Zeners.

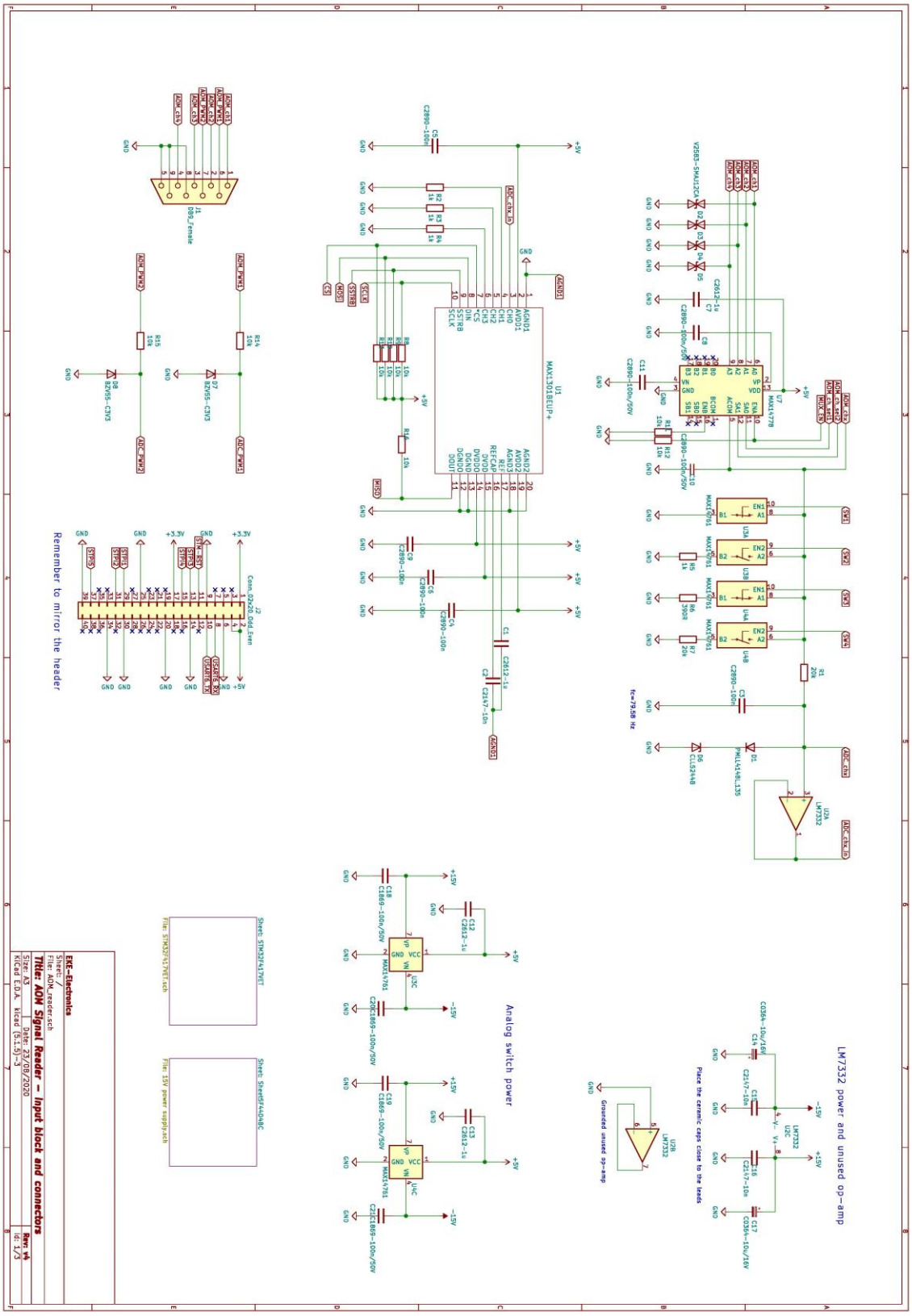

Figure 15: Input processing schematic

The buffered signal ADC_chx_in is input to the ADC MAX1301BEUP as illustrated in Figure 16. The ADC chip operates with a single 5 V power supply, but it can support an input swing from -3×V_{REF} to 3×V_{REF} where V_{REF} is the ADC's internal reference, the voltage being 4.096 V. Only 1 input channel of the ADC is used in this project, and the other 3 channels are grounded by 1 kΩ resistors for stability. MAX1301 is also a good option for future development of this thesis project where the remaining output groups of the AOM are taken into use. In addition to the bypass capacitors at the ADC's power pins, the pull-up resistors for the SPI bus are needed to create a default state for the SPI lines when its master, the STM32 microcontroller, is reset.

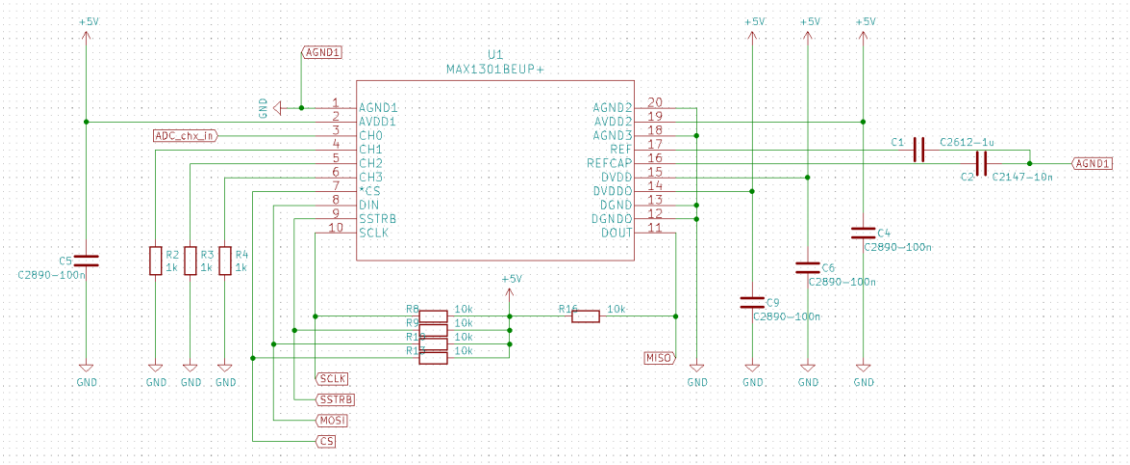

Figure 16: Analog to digital converter MAX1301BEUP

The AOM inputs including the voltage/current and PWM signals are acquired and fed to the multiplexer MAX14778 by the DB9 connector J1 in Figure 17. The header J2 is where the whole circuit board gets the power supply from. The 3.3 V and 5 V supply are provided by the Raspberry Pi computer through the header J2. The UART communication between the Raspberry Pi and the STM32 microcontroller also happens through this header. Moreover, for debugging purposes, 5 GPIO pins are connected between the two entities. In the middle of the figure is how the PWM signals are processed and input into the STM32. Since the AOM PWM signal is 15 V for logic 1, it must be limited to a safe level for the STM32, 3.3 V. The Zener diodes BZV55-C3V3 are used for this purpose as they offer 3.3 V of limiting voltage. The 10 kΩ resistors are to limit the current from the AOM channels to the ground when the Zener diodes conduct.

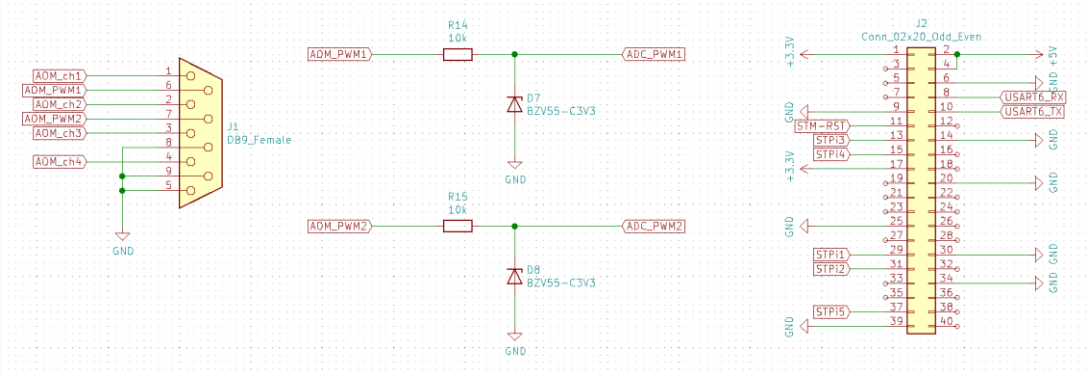

Figure 17: Input and output connectors

Figure 18 shows how the STM32F417VET microcontroller connects with the reset controller TPS3703A and the crystal oscillator HC3325. The reset controller monitors the voltage supply of the STM32 and resets the microcontroller if the supply voltage is out of the safe margin longer than 200 ms. This time period is selected by using a 10 kΩ pull-up resistor at pin 3 – CT or Capacitor Time. Pin 6 (Manual Reset) is controlled by a GPIO pin from the Raspberry Pi so that the STM32 can be actively reset by the Raspberry Pi. In addition to the reset controller, an external crystal oscillator HC3325, whose frequency is 25 MHz, is required because the built-in LC oscillator of the STM32 is not so precise.

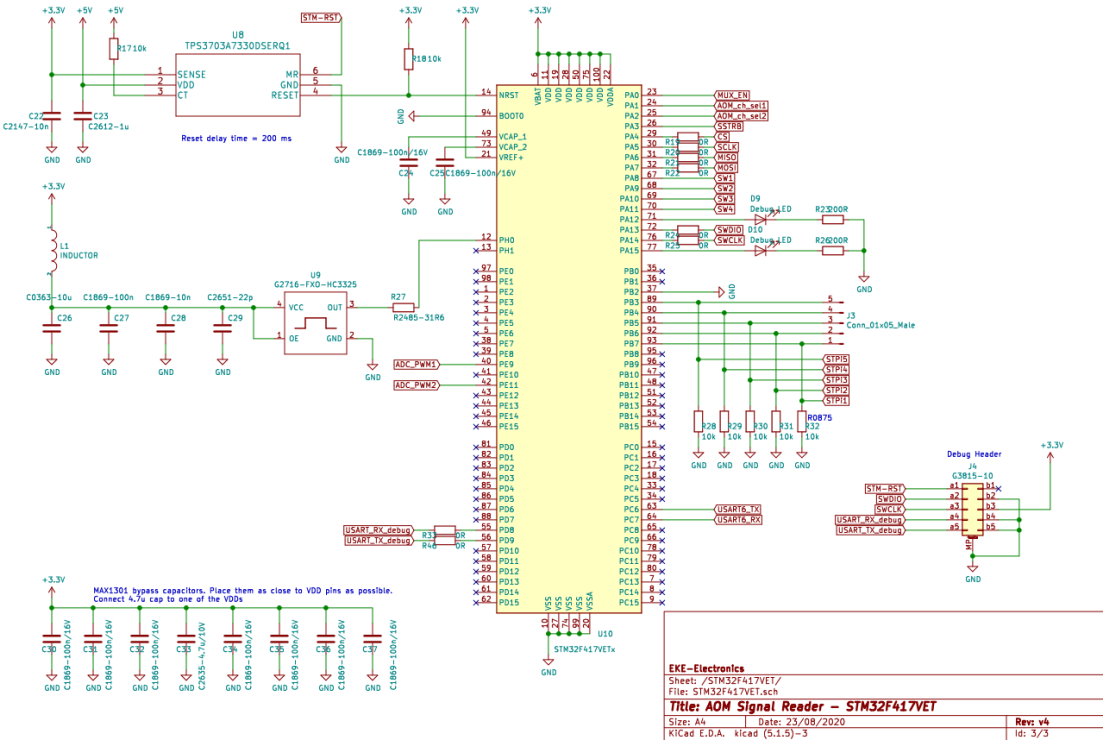

Figure 18: STM32F417VET microcontroller

The SPI bus of the STM32 is connected to the SPI of the ADC chip MAX1301 so that the STM32 acts as the master and the ADC operates as the slave. For debugging purposes, there are 2 STM32 GPIO pins connected to 2 LED lights and 5 GPIO pins connecting from the STM32 to the Raspberry Pi. Last but not least, the STM32 - Raspberry Pi communication is done through USART6_TX and USART_RX. Those two wires go from the STM32 to the header J2 which is compatible with the output header of Raspberry Pi.

6.2 Software Implementation

6.2.1 Overall Descriptions

The operation of the whole circuit is governed by the software embedded in the microcontroller STM32F417. Figure 19 outlines the structure of the entire software. The STM32 controls the closed/opened state of the analog switches to enable/disable the corresponding resistors and decides the AOM operating mode. Furthermore, it reads the ADC data from the MAX1301 to calculate the current or voltage depending on the AOM signal type. The STM32 also communicates with the Raspberry Pi computer through a shell where the user can enter different commands.

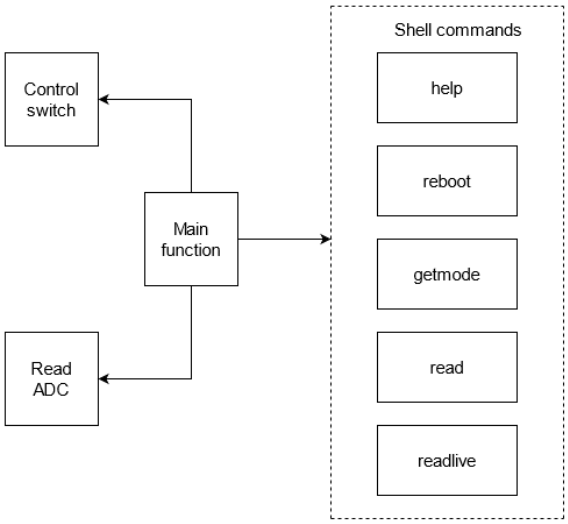

Figure 19: Software element diagram

The software implementation for multiplexing the AOM channel as well as controlling the analog switches is simple and straightforward. Thus, the code shall not be presented in this paper.

6.2.2 Reading ADC Data

The ADC IC MAX1301 is quite a simple yet special. It can be interfaced with by the SPI protocol, and it has only two registers for analog configuration and mode selection. The 1-byte analog configuration register holds the information for channel to read from, differential or single-ended input signal, and input range. Channel 0, the single-ended signal type and the dynamic range of $\pm 3V_{REF}$ are the configuration information to be written to the register [12, 14]. The mode control register only stores its mode of operation for which the mode of using an external clock from the STM32 microcontroller is employed [12, 23].

With the control, configuration being pre-set, a 16-bit reading result will be available after a conversion start byte. The result is output at the MISO pin of the slave MAX1301 from the start of the third byte. As a result, in order to obtain the full result bytes, 4 bytes of SCLK signal must be provided. This way of working can be visualized by the timing diagram in Figure 20.

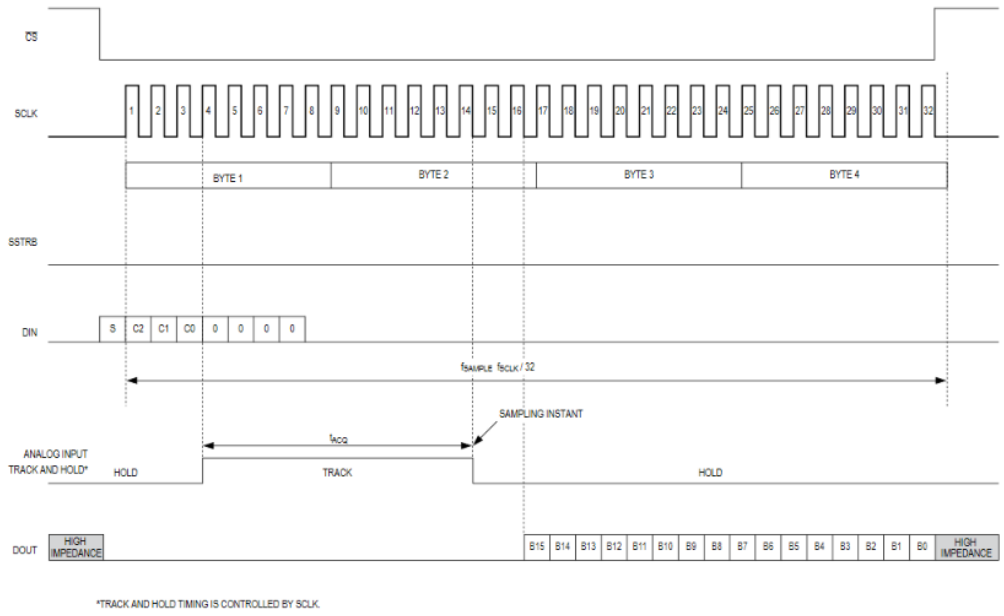

Figure 20: Timing diagram of reading a conversion from MAX1301 [12, 15]

The software function for reading ADC data from the MAX1301 takes no input argument and returns a measurement result as an integer number. It first starts the SPI with the configuration of a 2 MHz clock signal, clock polarity being low in idle, data being sampled at rising edge and output at falling edge, most significant bit first. Then, the analog configuration byte followed by the mode control byte is sent to the MAX1301 from the microcontroller. Right after that, a 4-byte SPI exchange function is called to obtain the ADC data. The exchange starts on the third byte from when the start conversion byte is received by the MAX1301, and the received bytes from the ADC are stored in an 8-bit buffer (rxbuf). The first received data byte will, therefore, be shifted 8 bits to the left and added with the second received data byte to get the 16-bit result value – misocode. This result is still crude and needs more calculation to get the voltage produced by the AOM because the ADC chip MAX1301 outputs a code from 0x0000 to 0xFFFF for input voltage from $-3V_{REF}$ to $+3V_{REF}$. The full-scale span ($\pm 3V_{REF}$) is divided into 65536 steps, so the voltage of $-3V_{REF}$ and $+3V_{REF}$ corresponds to -32768 and 32767 respectively. The relationship becomes an output code from 0x0000 to 0xFFFF for input code from -32768 to 32767. The formulas to calculate the voltage at input resistors produced by the AOM are as follows:

$$Input\ code = Output\ code - 32768$$

$$Input\ voltage = Input\ code \times \frac{3V_{REF} - (-3V_{REF})}{65536}$$

The code snippet showing the implementation of the ADC data reading function is in the appendix. The function name is readADC().

6.2.3 Mode Detecting Shell Command - Getmode

The user can interact with the signal reader through a shell running in the Raspberry Pi. One of the different commands the user can query the signal reader is detecting the AOM's operating mode. The procedure is the same as the mode detecting method mentioned in section 5.1.3. All 4 AOM channels are looped through, and the getmode() function is called for each channel. The results will also be displayed on the console. The getmode() function takes an argument about the channel from which the operating mode is obtained. Based on that, it controls the MUX to activate the channel for the signal to get to the input resistor. The function closes the analog switches and reads the produced voltage with the help of the readADC() function in the previous section. Depending on how the input voltage is generated, the function decides the AOM operating mode accordingly. Since the getmode() function is later used by other shell commands, it does not print anything to the console unless the argument --verbose" is provided, e.g. *getmode --verbose*. The verbose argument is just for debugging purposes, and thus can be ignored in normal use. The software implementation of the mode detecting command can be found at the appendix section.

6.2.4 Signal Reading Shell Command - Read

The AOM voltage or current signal can be retrieved by the command *read [signal] [channel]*, where [signal] can be either "voltage", "current", or "pwm" and [channel] can be from "1" to "4". It is important to note that this command does nothing related to mode detection which means the AOM operation mode of the interested channel must be known in advance. The function first acquires the signal type to read and the interested channel. If the argument is "voltage", it activates both the MUX channel corresponding to the input and the analog switch of the 20 kΩ resistor. After reading the input voltage twice with a delay of 20 milliseconds in between, the second ADC data received is the actual voltage of the AOM. The result of the first read is not usable because, in the idle state, the AOM is isolated from the signal reader. The input voltage is pulled to around 13 V by the voltage buffer U2 – LM7332 which is not correct, and the reading result (of around 13 V) persists there in the ADC register until it is read by the microcontroller. This is a known

issue that should be improved in future development if higher data throughput is needed. Then, an important part of the calculation is the error compensation. For the voltage mode, the error source is the offset and gain error voltage noted in section 5.1.2. In the program, the quantity found for this error voltage by testing is around 21 mV which is symbolized by CALIB_VOL. Then the gain error is compensated by multiplying with a constant coefficient as shown in the code snippet below. Again, the cmd_read() function is also called by another function. That is the reason for a user argument called "--result-only". When this argument is provided, only one number is printed with a voltage unit letter "V". Otherwise, more comprehensive words are shown as in the code snippet (see Appendix 1).

When the user inputs "current" as the argument for the read command, the same procedure is repeated but the 390 Ω resistor is activated instead, and the read voltage is divided by the resistance by Ohm's law for the current. The offset error compensation is accomplished by the calibcurrent() function. Depending on the current measurement, this function subtracts the measurement by a specific offset found during testing. The "--result-only" argument also works for this case in the same way as reading voltages.

Last but not least, if the argument "pwm" is applied, the ICU peripheral of the microcontroller is started for the provided channel. The default configuration is 10 MHz for the peripheral clock frequency and the PWM input is at logic 1 when active. When the frequency of the PWM signal is lower than 120 Hz, the ICU driver is too fast to read input which means there will be overflows in the ICU counters. This situation is fortunately detected by the ICU callback functions. When this happens the overflow callbacks will change the configuration, lowering the ICU frequency to 100 kHz. This is effective from the next iteration of the PWM reading until the measured PWM frequency is detected to be higher than 160 Hz where the configuration is reverted to the default (10 MHz ICU clock frequency). The readPWM() function only configures and initializes processes for measuring the PWM signal. Instead, the period and duty cycle counters are available with reading values when the callback functions icuGetWidthX() and icuGetPeriodX() are called. The displaying of the results also has two options, in the normal and "result-only" mode. The content of the signal reading function is shown in the appendix as the command cmd_read().

6.2.5 Live Signal Reading Shell Command - Readlive

The live signal reading shell command is called to read and display all types of AOM signals from all the channels. The command is run by readlive. To have the measurements in real-time, the function loops through all the AOM analog and PWM channels. It reads the signals according to the detected mode by calling the getmode and *read* command described in sections 6.2.3 and 6.2.4 respectively. In this function, escape sequences are utilized to move the console cursor around the display to print template tables where data is filled in. After the setup phase, which consists of hiding the blinking cursor, moving it to the origin, clearing the screen and printing the template tables, all the analog channels are scanned to detect the modes. The observed modes are then shown in the table where appropriate before the program calls the *read* command for each channel depending on the detected mode. The obtained signals are also to fill in the tables. After that, the PWM table is updated with the measured PWM signals' duty cycle and frequency/period. The procedure is repeated continuously every 400 ms until the user enters a terminating character 'q' because there is a delay of 100 ms for reading each analog channel. The implementation of the live signal reading command and template table printing function can be seen in the appendix as the command cmd_readlive() and the function printtemplate().

7 Prototyping and Testing

7.1 Prototyping

In addition to the first page of the project schematic illustrated in Figure 15, the two remaining pages describe the ±15 V power supply circuit, the microcontroller STM32F417 and its supporting ICs such as the reset manager U8 – TPS3703A and the external 25 MHz crystal U9 - HC3325. The ±15 V power supply circuit is a ready-made design at EKE Ltd. and is reused in this project with the author's permission. There are also some LEDs for debugging purposes and some GPIO connections between the STM32 and the Raspberry Pi which were planned at the schematic designing phase in case there is need, yet those GPIO connections are not used for the time being.

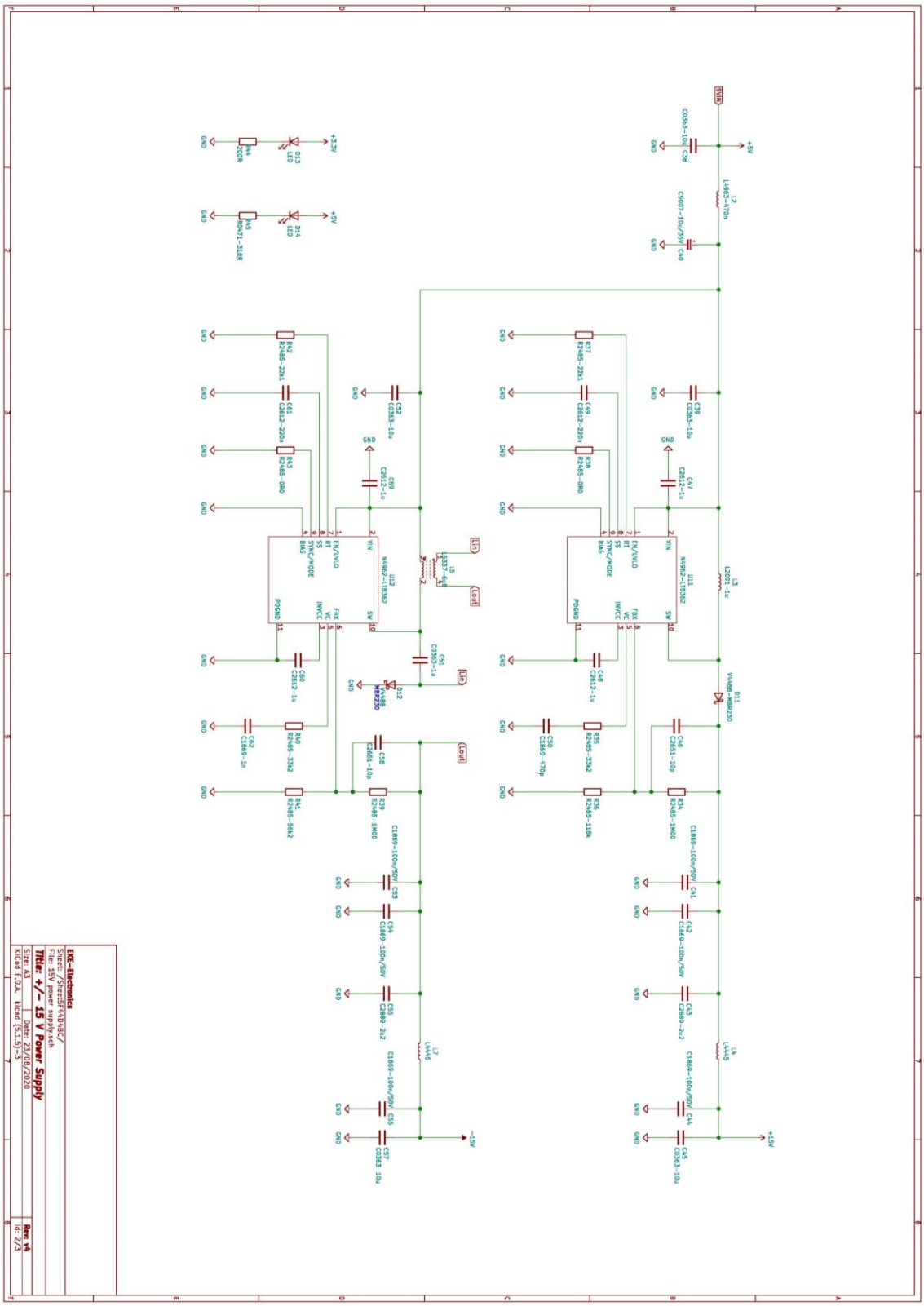

Figure 21: +/- 15 V power supply

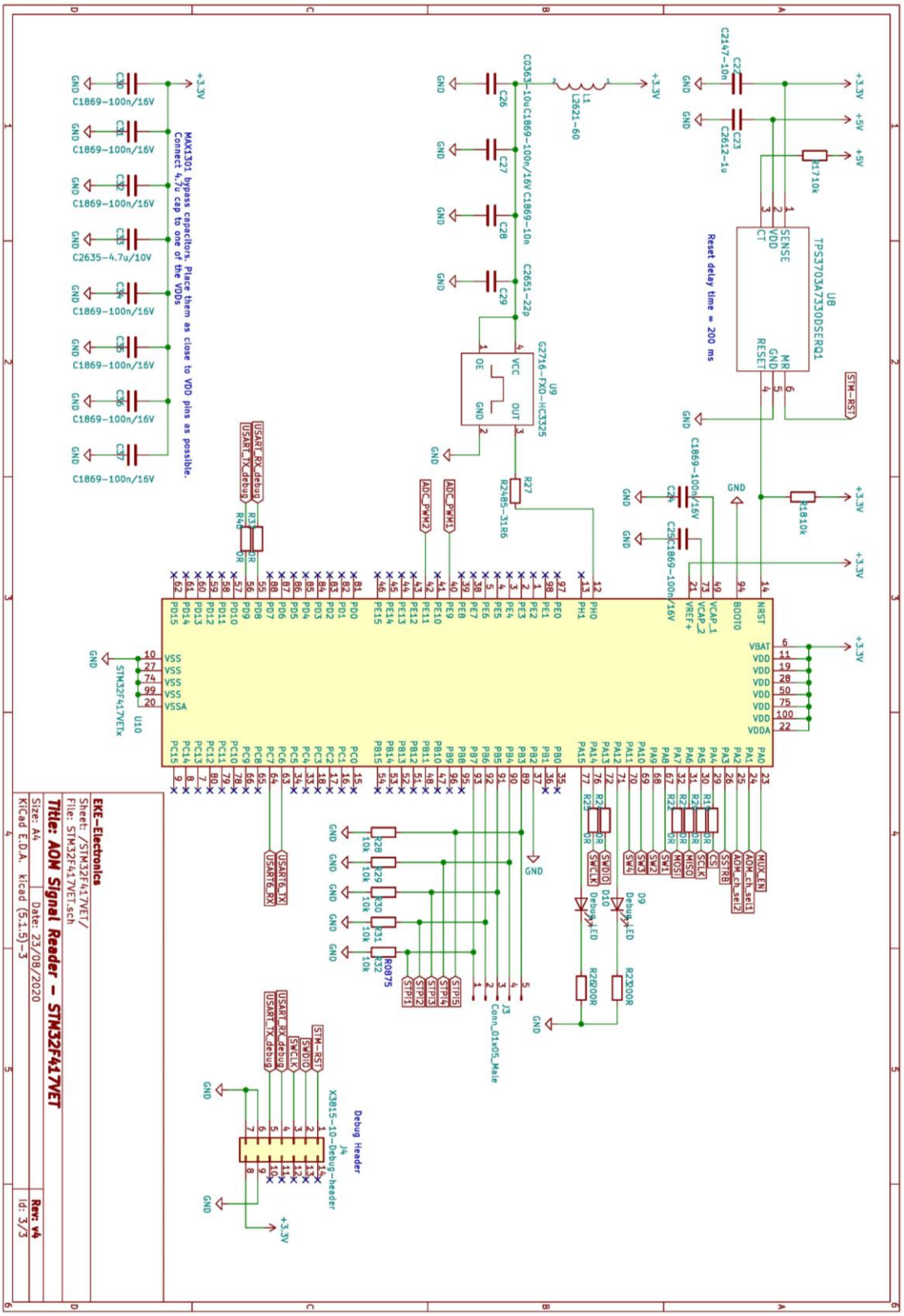

Figure 22: Microcontroller and its supporting units

With the schematic being ready, the prototyping process from layout design to PCB component assembly is reported starting from Figure 23 to Figure 28. When the board is manufactured and all the electronic components are gathered, the prototyping starts with applying soldering paste to the board. This is made easy by the stencil that is provided along with the board by the manufacturer shown in Figure 24. When that is completed, all the components are placed onto their footprints on the board according to the layout. Many of the components are so tiny that a microscope is necessary (see Figure 25 and Figure 26).

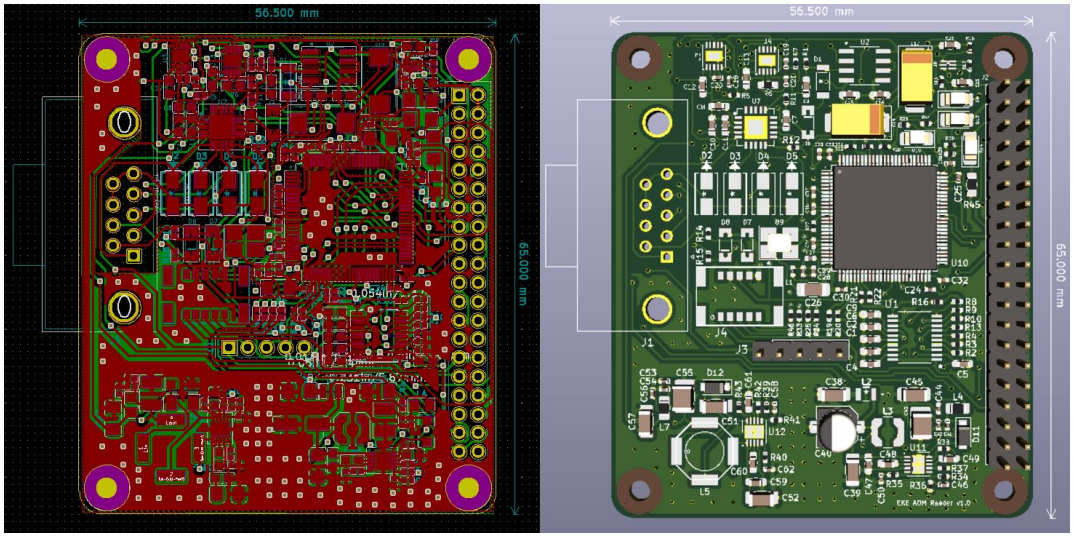

Figure 23: PCB layout design and 3D view

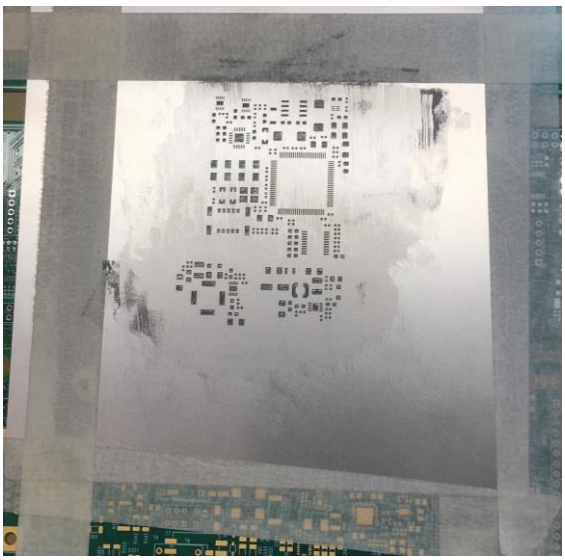

Figure 24: Applying soldering paste with the help of a stencil

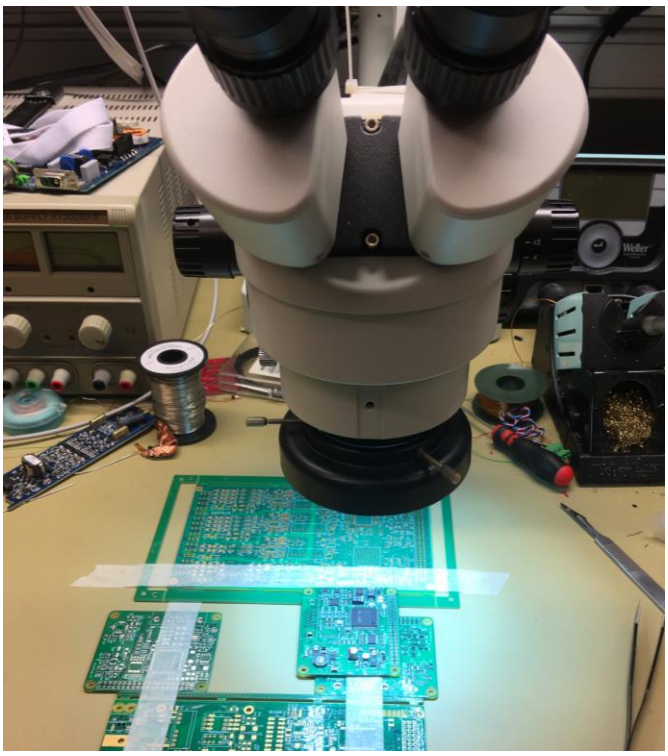

Figure 25: All the components put in place

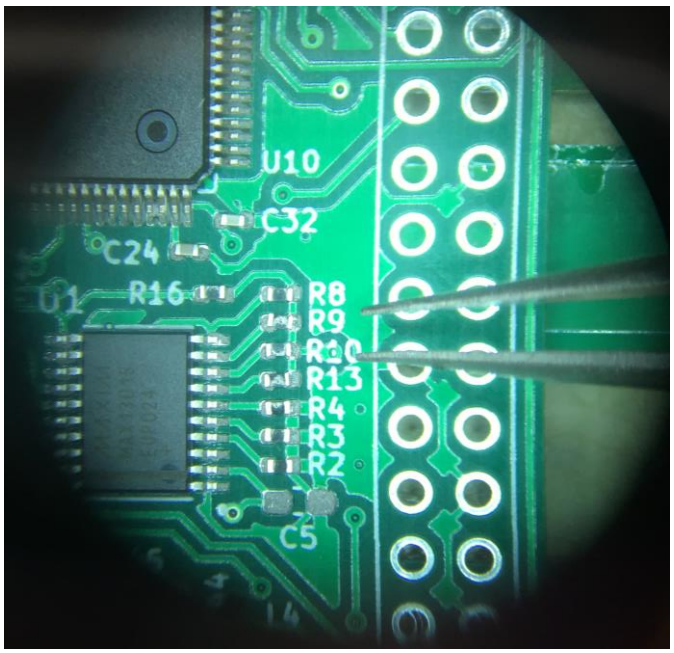

Figure 26: Soldering tiny components using a microscope

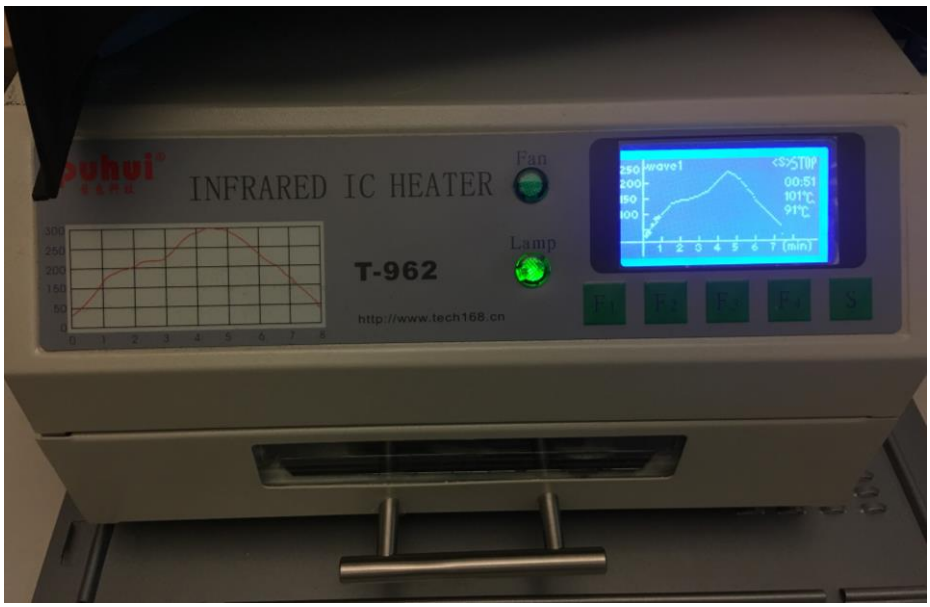

Figure 27: The board being heated inside a reflow heater for the solder paste to work

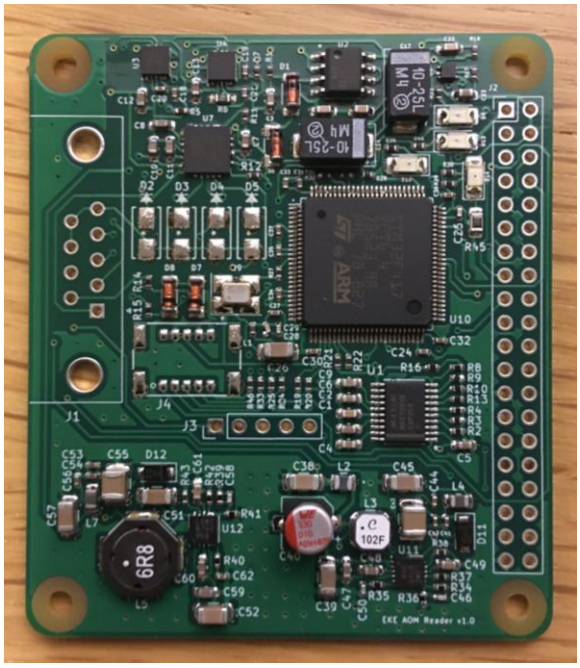

Figure 28: The ready circuit after the reflow soldering

The process continues by putting the PCB, with all the electronic components in place, into a reflow heater. The applied heat is varied in a certain way illustrated by the graph in Figure 27. This function of heat over time will melt the soldering paste in a controlled manner, and when everything cools down afterwards, the components are soldered to

the board as shown in Figure 28. Some big components like connectors and diodes can be soldered later by a soldering iron.

7.2 Testing and Measurement Compensation

7.2.1 Voltage compensation

Testing is done by configuring the AOM to output different voltage and current values. For convenience, all four channels of the AOM will generate the same value, for example 1000, 2000 or 3000 mV as shown in the *AOM* column of Table 1 and Table 3. It is observed that voltage readings (see Table 1) of all four channels do not have noticeable difference. For that reason, the signal reader channels' measurements will be combined into average values. The error between the AOM source signal and the measured voltages is also taken into account for compensation. Drawing a line graph from Table 1 shows that the relationship between the signal reader measurements and the true AOM data linear, and the measurement graph is a line displaced and slightly rotated from the data of the AOM. Because of the overwhelming amplitude of the data compared to the error, the error is drawn alone versus the unity gain line whose equation is y = x for ease of visualization (see Figure 29). This error is also called gain error because it varies depending on the AOM input signal. To sum up, the measurement graph, which is the combination of the linear AOM output and the linear error, can be approximated as a line passing two points, A(-12000,-11996) and B(12000,12038), and the objective is to shift the line and rotate it so that it passes two points, A'(-12000,12000) and B'(12000,12000), by compensation. Since the measurement line is now passing vector $\overrightarrow{AB} = (2400,24034)$, its normal vector is $\vec{n} = (-24034,2400)$. The measurement graph which has the normal vector $\vec{n}$ and which passes point A can be represented in 2-dimentional metric by the following equation:

$$-24034(x + 1200) + 24000(y + 11996) = 0$$

$$or\ y = \frac{24034}{24000}x + 21$$

From the equation above, it is obtained that the offset error is 21 mV and the gain error is 24034/24000. The compensation process is simply reversing the equation so that it

becomes the equation y = x. Of course, this is done by subtracting the y (the signal reader measurements) by 21 and by multiplying it by 24000/24034.

Table 1: AOM voltages read by the signal reader

Types	AOM	Signal reader	Error
Value (mV)	-12000	-11996	4
	-11000	-10994	6
	-10000	-9993	7
	-9000	-8991	9
	-8000	-7989	11
	-7000	-6987	13
	-6000	-5986	14
	-5000	-4986	14
	-4000	-3984	16
	-3000	-2983	17
	-2000	-1981	19
	-1000	-980	20
	0	19	19
	1000	1021	21
	2000	2023	23
	3000	3025	25
	4000	4026	26
	5000	5027	27
	6000	6028	28
	7000	7030	30
	8000	8031	31
	9000	9032	32
	10000	10034	34
	11000	11036	36
	12000	12038	38

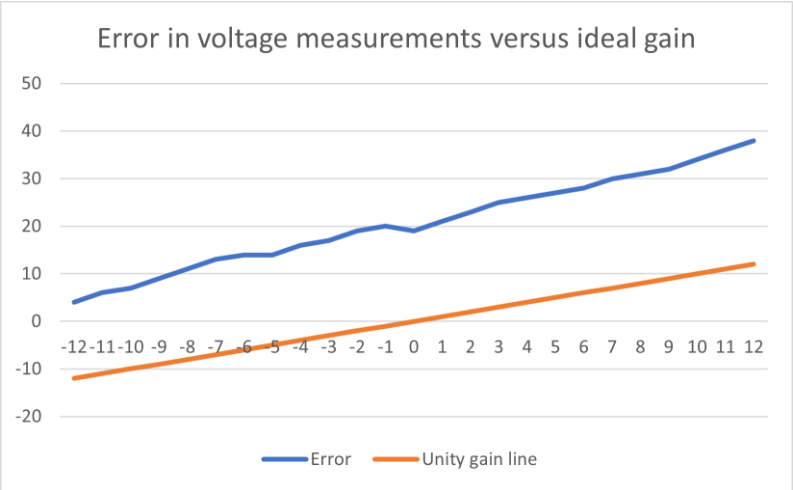

Figure 29: Voltage measurement errors (gain error) compared to ideal gain

After applying compensation, voltage readings are improved noticeably with the maximum error of 2 mV. Table 2 shows the compensation results for different AOM signal values in millivolts. The highest and lowest magnitudes of the measurements among four channels of the signal reader are listed.

Table 2: Signal reader measurement after compensation

AOM signal	Highest measurement	Lowest measurement
-12000	-12000	-11998
-11000	-11000	-10.998
-10000	-9998	-9998
-9000	-8998	-8998
-8000	-7999	-7998
-7000	-6999	-6998
-6000	-5999	-5998
-5000	-4999	-4998
-4000	-3999	-3997
-3000	-2999	-2997
-2000	-1998	-1998
-1000	-999	-998
0	1	0
1000	1001	1000
2000	2002	2000

3000	3000	3000
4000	4001	4000
5000	5000	5000
6000	6001	6000
7000	7001	7000
8000	8001	8001
9000	9003	9001
10000	10001	10001
11000	11003	11001
12000	12001	12001

It is observed that measurements persist at certain values most of the time as long as the AOM signals do not change. Although there are occasional fluctuations within a 2 mV range, this is considered acceptable behaviour since noise cannot be completely avoided.

7.2.2 Current Compensation

Current compensation is done similarly to compensation for voltage that includes setting up the AOM's four channels to generate different constant currents and reading the currents at the signal reader side. However, in this case, the error behaves in a noisier way which means the signal reader measurements fluctuate between the values shown in Table 3. For ease of testing and compensating, the error in the measurements is simplified based on the observation of what values are mainly shown during testing. For example, when the AOM current is 7000 µA, the signal reader oscillates from 7056 to 7058 and to 7061 µA, but the error of 56 µA is selected because 7056 is the main value shown from the reading. In addition, a software filter that calculates the average current of each channel is also implemented to reduce fluctuation, although that delays the correct current to be shown on the terminal. It is observed from Table 3 that the error current only depends on the leakage current of the circuit. The error virtually retains its value over the AOM signal span from 0 to 24000 µA and only changes due to different leakage current – input current relationship of the electronic components of the circuit. Hence, current compensation only involves applying different offset current compensation for different AOM input currents.

Table 3: AOM currents retrieved by the signal reader

Channel	AOM	Signal reader	Simplified error
Value (µA)	0	53-56	56
	1000	1053-1056	56
	2000	2053-2056	56
	3000	3053-3056	56
	4000	4053-4056	56
	5000	5056-5058	56
	6000	6056-6058	56
	7000	7056-7058-7061	56
	8000	8056-8058	56
	9000	9056-9058-9061	56
	10000	10056-10058	56
	11000	11053-11056-11058	56
	12000	12053-12056-12059	56
	13000	13053-13056	56
	14000	14053-14057	56
	15000	15051-15053	53
	16000	16048-16051-16053	53
	17000	17048-15051-17053	51
	18000	18048-18051	51
	19000	19046-19048-19051-19053	50
	20000	20046-20048-20051	48
	21000	21043-21046-21048-21051	48
	22000	22046-22048-22051	48
	23000	23046-23048-23054	48
	24000	24046-24048-24055	48

After applying the compensation and doing a retest, the current reading becomes more presentable yet slightly less performant because of the software filter. The filter reduces fluctuations by continuously calculating the average values of the measurements for each channel. While this algorithm reduces the variations, it also delays how long the signal reader can read correct AOM signals since the averaged measurements are slowing reaching the real AOM data.

Table 4: Current measurements after compensation

Channel	AOM	Highest measurement	Lowest measurement
Value (µA)	0	2	0
	1000	1002	1000

	2000	2000	1998
	3000	3000	2999
	4000	3999	3998
	5000	5000	4999
	6000	6000	5999
	7000	6999	6998
	8000	8000	7999
	9000	8999	8997
	10000	10002	10000
	11000	11001	10999
	12000	12001	12000
	13000	13001	12999
	14000	14000	13999
	15000	15000	14997
	16000	16002	16001
	17000	17002	17001
	18000	18001	17999
	19000	19001	19000
	20000	20002	20000
	21000	21002	21000
	22000	22001	21999
	23000	23001	22999
	24000	24001	23999

Although the fluctuations are greatly suppressed by the software filter, there are still variations in the current measurements with the maximum error up to 3 µA occasionally. The highest and lowest reading values among the four channels are listed in Table 4.

7.2.3 Reading PWM test

Two issues were found after the PWM reading test. The first problem was an unexpected error of the duty cycle counter value. For example, a 50% duty cycle AOM PWM signal with the frequency of 10 kHz would result in the period counter value of 100 and the duty cycle counter value of 50 at the signal reader side. In reality, 58 was the value of the duty cycle counter of the signal reader. This was debugged to be the slew time of the AOM PWM signal from 15 V to V_{IL} (low level input voltage) of the STM32 microcontroller. To be more specific, it is necessary to reuse the example above where the AOM PWM frequency is 10 kHz, which means a period of 0.1 ms and the duty cycle being 50%. From t = 0 to t = 0.05 ms, the PWM voltage is at the high level or 15 V. When t = 0.05 ms, the

voltage starts to drop from 15 V to V_{IL} (the voltage where the STM32 considers it as a low logic level). It is important to denote this time interval as the slew time t_{slew}. During the time from t = 0.05 ms to t = 0.05 ms + t_{slew}, the STM32 still consider its input to be high logic level. Thus, the STM32 keeps increasing the duty cycle counter. Fortunately, this issue is easily overcome by subtracting a constant compensation from the duty cycle counter as the slew time remain intact regardless of the duty cycle and the frequency of the PWM signals.

The second issue is more challenging which is that the period and frequency counter of the STM32 get overflowed for the PWM signals the frequency of which is less than around 100 Hz. Although the ICU driver of the STM32 microcontroller is capable of detecting an overflow, trying to correct the PWM reading is not necessarily more practical than lowering the ICU driver frequency since the AOM measurements are to be streamed in real-time to the console. Specifically, the frequency is decreased from 10 MHz to 100 kHz when the overflow occurs, which implies that the AOM is currently generating low frequency PWM. On the other hand, the ICD driver frequency is reversed back to 10 MHz when the read PWM frequency is higher than 160 Hz.

Table 5: PWM measurement results

AOM frequency (Hz)	**Measured frequency (Hz)**	**AOM duty cycle channel (%)**	**Measured duty cycle channel 1 (%)**	**AOM duty cycle channel 2 (%)**	**Measured duty cycle channel 2 (%)**
10	10	9.9	9.9	67.4	67.4
50	50	9.9	9.9	64.5	64.5
100	100	9.9	9.9	61.1	61.1
200	200	9.9	9.9	62.7	62.7
1000	1000	9.9	9.9	75.1	75.1
2000	2000	9.9	9.9	15.1	15.1
3333	3333	78.9	78.9	75.1	75.1
5004	5002.5	9.9	9.9	71.9	71.9
6789	6788.8	12.0	12.0	83.6	83.6
6000	5998.8	77.7	77.6	7.5	7.4
7000	6997.9	77.7	77.7	75.5	75.4
8000	8000	77.7	77.7	85.0	85.0

9000	9000.9	77.7	77.7	93.3	93.2
10000	10000	9.9	9.9	11.4	11.4

Afterwards, the PWM reading results prove to be trustable. However, the performance of reading PWM signals is quite dependent on the source signals' frequency because there are rounding errors in calculating the measured frequencies. The tested PWM measurements are shown in Table 5 above.

8 Conclusion

To sum up, the project's goal was to create a device to simultaneously monitor the AOM signals with a precise resolution. The device was expected to detect the AOM channels' operating mode and recognize the smallest changes in the AOM signals. Afterwards, a signal reader consisting of a circuit board and embedded software were implemented. The device routes the signal from each AOM channel to an ADC chip for data measurements. The reading results are then transferred to a Raspberry Pi computer and are displayed through a shell command interface. In terms of precision, the signal reader does not quite fulfill the requirement to detect the smallest changes in the AOM signals. The voltage reading has the maximum error of 2 mV which doubles the minimum step size of the AOM's voltage. For current measurement, the maximum error is a little worse with 3 µA, which appears occasionally. There are also frequent fluctuations of 2 µA away from the AOM values. The PWM capture, in some cases, has the worst precision among the reading of AOM's other signal types due to rounding errors. In many other cases without rounding errors, the PWM reading results are retrieved perfectly. Although the ambitious goals are not fully achieved in that the signal reader cannot detect every smallest change of the AOM signals, the end result of this project is acceptable and can be utilized for its dedicated purpose, which is testing multiple channels of the AOM.

In the future, the signal reader could be improved in many ways. The current input block, multiplexing and converting various signals to readable voltage, can be duplicated to enable the signal reader to read from the remaining channel groups. Another enhancement can be increasing the precision while suppressing the measured signal fluctuations.

References

1 Analogue Output Module (AOM) [online]. Espoo, Finland: EKE-Electronics Ltd; 18 June 2018.
URL: https://www.eke-electronics.com/analogue-output-module-aom. Accessed 29 April 2020.

2 Quad Channel, 16-Bit, Serial Input, 4 mA to 20 mA and Voltage Output DAC, Dynamic Power Control [online]. Norwood, MA, USA: Analog Devices, Inc.
URL: https://www.analog.com/media/en/technical-documentation/datasheets/AD5755.pdf. Accessed 29 April 2020.

3 Analog Switches and Multiplexers Basics [online]. Norwood, MA, USA: Analog Devices, Inc.
URL: https://www.analog.com/media/en/training-seminars/tutorials/MT-088.pdf. Accessed 12 May 2020.

4 MAX14778 Dual ±25V Above- and Below-the-Rails4:1 Analog Multiplexer [online]. San Jose, CA, USA: Maxim Integrated, Inc.; July 2020.
URL: https://datasheets.maximintegrated.com/en/ds/MAX14778.pdf. Accessed 15 January 2021.

5 Understanding SAR ADCs: Their Architecture and Comparison with other ADCs [online]. San Jose, CA, USA: Maxim Integrated, Inc.; 2 October 2001.
URL: https://www.maximintegrated.com/en/design/technical-documents/tutorials/1/1080.html. Accessed 23 June 2020.

6 SAR ADCs vs. Delta-Sigma ADCs: Different Architectures for Different Application Needs [online]. Dallas, TX, USA: Texas Instruments, Inc.
URL: https://training.ti.com/ADCwebinar. Accessed 23 June 2020.

7 Sigma-Delta ADCs [online]. San Jose, CA, USA: Maxim Integrated, Inc.; 31 January 2003.
URL: https://www.maximintegrated.com/en/app_notes/index.mvp/id/1870. Accessed 27 June 2020.

8 Curran, Ryan. Exploring Different SAR ADC Analog Input Architectures [online]. Norwood, MA, USA: Analog Devices, Inc.
URL: https://www.analog.com/en/technical-articles/exploring-different-sar-adc-analog-input-architectures.html. Accessed 22 May 2020.

9 MAX14759/MAX14761/MAX14763 Above- and Below-the-Rails Low On-Resistance Analog Switches [online]. San Jose, CA, USA: Maxim Integrated, Inc.; August 2012.
URL: https://datasheets.maximintegrated.com/en/ds/MAX14759-MAX14763.pdf. Accessed 15 January 2021.

10 LM7332 Dual Rail-to-Rail Input and Output30-V, Wide VoltageRange,High Output,Operational Amplifier [online]. Dallas, TX, USA: Texas Instruments, Inc.; March 2013.
URL: https://www.ti.com/lit/ds/symlink/lm7332.pdf. Access 18 January 2021.

11 The ABCs of Analog to Digital Converters: How ADC Errors Affect System Performance [online]. San Jose, CA, USA: Maxim Integrated, Inc.
URL: https://www.maximintegrated.com/en/design/technical-documents/tutorials/7/748.html. Access 10 November 2020.

12 MAX1300/MAX1301 8- and 4-Channel, ±3 x VREF Multirange Inputs, Serial 16-Bit ADCs [online]. San Jose, CA, USA: Maxim Integrated, Inc.; December 2011.
URL: https://datasheets.maximintegrated.com/en/ds/MAX1300-MAX1301.pdf. Accessed 15 January 2021.

13 Basic of UART [online]. URL: https://openlabpro.com/guide/basics-of-uart/. Access 12 June 2020.

14 SMAJ5.0A Thru SMAJ300CA Surface Mount Transient Voltage Suppressor [online]. SMC Diode Solutions.
URL: https://www.digikey.fi/htmldatasheets/production/1914723/0/0/1/SMAJ-Series.pdf/. Access 12 November 2020.

15 Adams, James. Introducing Raspberry Pi HATs [online]. Cambridge, England: Raspberry Pi Foundation; 31 July 2014.
URL: https://www.raspberrypi.org/blog/introducing-raspberry-pi-hats. Accessed 15 January 2021.

16 Adams, James. Introducing Raspberry Pi HATs [online]. Cambridge, England: Raspberry Pi Foundation; 16 March 2018.
URL: https://github.com/raspberrypi/hats/blob/master/hat-board-mechanical.pdf. Accessed 15 January 2021.

17 The MOSFET [online]. Cambridge, MA, USA: Electronics Tutorial.
URL: https://www.electronics-tutorials.ws/transistor/tran_6.html. Accessed 15 January 2021.

18 Pulse Width Modulation [online]. Cambridge, MA, USA: Electronics Tutorial.
URL: https://www.electronics-tutorials.ws/blog/pulse-width-modulation.html. Accessed 15 January 2021.

Code Snippets

```
/*****************************************************************************/
//! \brief Read voltage at ADC channel 0
//!
//! Precondition: the MUX channel and analog switch must be enabled in ad-
vance
//! @param none
//!
//! @return int: voltage (mV)
//
/*****************************************************************************/
int readADC(void)
{
    double voltage;
    int misocode, signedcode, millivolt;

    spiAcquireBus(&SPID1);          /* Acquire ownership of the bus.    */
    spiStart(&SPID1, &spicfg);      /* Setup transfer parameters.       */

    spiSelect(&SPID1);              /* Slave Select assertion.          */
    spiSend(&SPID1, 1, analog_cfg); /* Send analog input configuration  */
    spiUnselect(&SPID1);            /* Slave Select de-assertion.       */

    spiSelect(&SPID1);
    spiSend(&SPID1, 1, mode_select);        /* Send mode selection byte */
    spiUnselect(&SPID1);

    spiSelect(&SPID1);
    spiExchange(&SPID1, 4, start_conv, rxbuf); /*Atomic transfer operations*/
    spiUnselect(&SPID1);
    spiReleaseBus(&SPID1);                     /* Ownership release.       */

    misocode = rxbuf[2] << 8;
    misocode |= rxbuf[3];

    signedcode = misocode - 32768;
    voltage = signedcode * 6 * 4.096 / 65536;
    millivolt = (int) (voltage * 1000);

    return millivolt;
}
```

```
/*****************************************************************************/
//! \brief Get mode command
//!
//!
//
/*****************************************************************************/
void cmd_getmode(BaseSequentialStream *chp, int argc, char *argv[])
{
    if (argc == 1)
    {
        if (strcmp(argv[0], "--verbose") == 0)
        {
            verbose = TRUE;
        }
    }

    for(int i = 1; i < 5; i++)
    {
        chxmode[i-1] = getmode(i);
        channelmode[i-1] = chxmode[i-1];
    }
    PRINT("Channel: 1\t    2\t       3\t  4\n\r");
    PRINT("Mode:    %s %s %s %s\n\r",
          modetxt[chxmode[0]], modetxt[chxmode[1]], modetxt[chxmode[2]],
          modetxt[chxmode[3]]);

}
```

```
/*****************************************************************************/
//! \brief Get operating mode of an AOM channel
//!
//! @param unsigned short: AOM channel number (1-4)
//!
//! @return int: enum current/voltage mode
//
/*****************************************************************************/
int getmode(unsigned short channel)
{
    int millivolt, lastmvolt;

    sel_MUXch(channel);
    //Asume current mode, enable 20 kOhm resistor
    en_analogSW(RESISTOR20K);
    millivolt = readADC();
    chThdSleepMilliseconds(200);
    millivolt = readADC();
    lastmvolt = millivolt;

    if (verbose)
    {
        PRINT("Channel %d:\n\r", channel);
        PRINT("Voltage over 20 kOhm: %d\n\r", millivolt);
    }

    if (checkhigher(millivolt, 10000, TOLERANCE) == TRUE || checklower(milli-
volt, -10000, TOLERANCE) == TRUE)
    {
        if (verbose)
        {
            PRINT("Current mode detected\n\r");
        }
        return currentmode;
    }
    else if (checkequal(millivolt, 20, 2) == TRUE)
    {
        if (verbose)
        {
          PRINT("No signal\n\r");
        }
        return nosignal;
    }
    else //Can either be current or voltage mode
    {
        //Enable 10 kOhm resistor
        en_analogSW(RESISTOR10K);
        millivolt = readADC();
        chThdSleepMilliseconds(200);
        millivolt = readADC();
```

```
            if (verbose)
            {
                PRINT("Voltage over 10 kOhm: %d\n\r", millivolt);
            }

            if(checkequal(millivolt, lastmvolt, THRESHOLD) == TRUE)
            {
                if (verbose)
                {
                    PRINT("Voltage mode detected\n\r");
                }
                return voltagemode;
            }
            else
            {
                if (verbose)
                {
                    PRINT("Current mode detected\n\r");
                }
                return currentmode;
            }
        }
    }
```

```
/*****************************************************************************/
//! \brief Signal reading command
//!
//!
//
/*****************************************************************************/
void cmd_read(BaseSequentialStream *chp, int argc, char *argv[])
{
    (void) argv;
    (void) argc;
    uint8_t channel;
    int millivolt;

    if (argc == 2 || argc == 3)
    {

        if (strcmp(argv[2], "--result-only") == 0)
        {
            result_only = TRUE;
        }
        else
        {
            result_only = FALSE;
        }

        if (strcmp(argv[0], "voltage") == 0)
        {
            channel = strtol(argv[1], NULL, 10);
            if (channel >= 1 && channel <= 4)
            {
                sel_MUXch(channel);
                en_analogSW(RESISTOR20K);
                /* The first read is garbage because of hw */
                millivolt = readADC();

                /* Wait a bit for the voltage to settle */
                chThdSleepMilliseconds(100);
                millivolt = readADC();
                millivolt -= CALIB_VOL;
                voltage = ceil(((double) millivolt) * 12000.0f / 12017.0f);
                voltage = voltage / 1000.0f;

                if (!result_only)
                {
                    PRINT("read: Voltage: %2.3f V\n\r", voltage);
                }
                else PRINT("%7.3f V", voltage);

            }
            else PRINT("Channel must be from 1 to 4\n\r");
        }
        else if (strcmp(argv[0], "current") == 0)
        {
```

```
        channel = strtol(argv[1], NULL, 10);
        if (channel >= 1 && channel <= 4)
        {
            sel_MUXch(channel);
            en_analogSW(RESISTOR390R);

            /* The first read is garbage because of hw */
            millivolt = readADC();

            /* Wait a bit for the voltage to settle */
            chThdSleepMilliseconds(100);
            millivolt = readADC();
            current = ((double) millivolt) / 390.0f;

            calibcurrent(channel, &current);

            if (fast_forward[channel-1])
            {
                last_current_avr[channel-1] = current;
                fast_forward[channel-1] = FALSE;
            }
            current_avr[channel-1] = ((current -
                                       last_current_avr[channel-1])/n) +
                                       last_current_avr[channel-1];

            last_current_avr[channel-1] = current_avr[channel-1];
            if (!result_only)
            {
                PRINT("read: Current: %2.3f mA\n\r", current);
            }
            else PRINT("%7.3f mA", current_avr[channel-1]);
        }
        else PRINT("Channel must be from 1 to 4\n\r");
    }
    else if (strcmp(argv[0], "pwm") == 0)
    {
        channel = strtol(argv[1], NULL, 10);
        if (channel == 1)
        {
            if(highfreqPWM1 == TRUE)
            {
                readPWM(true, 1, &icucfg1);
            }
            else
            {
                readPWM(false, 1, &icucfg1_low);
            }
        }
        else if (channel == 2)
        {
            if(highfreqPWM2 == TRUE)
            {
                readPWM(true, 2, &icucfg2);
```

```
                }
                else
                {
                    readPWM(false, 2, &icucfg2_low);
                }
            }
            else PRINT("Channel must be either 1 or 2\n\r");
            if(lastfrequency1 >= 160.0)
            {
                highfreqPWM1 = TRUE;
            }
            if(lastfrequency2 >= 160.0)
            {
                highfreqPWM2 = TRUE;
            }
        }
        else
        {
            PRINT("Incorrect first argument\n\r");
            PRINT("USAGE: read voltage {channel(1-4)}\n\r");
            PRINT("       read current {channel(1-4)}\n\r");
            PRINT("       read pwm {channel(1|2)}\n\r");
        }
    }
    else
    {
        PRINT("USAGE: read voltage {channel(1-4)}\n\r");
        PRINT("       read current {channel(1-4)}\n\r");
        PRINT("       read pwm {channel(1|2)}\n\r");
    }

}
```

```
/*****************************************************************************/
//! \brief Live signal reading command
//!
//!
//
/*****************************************************************************/
void cmd_readlive(BaseSequentialStream *chp, int argc, char *argv[])
{
    char *cmd_readmode[3] = {"voltage", "current", "pwm"};
    char *cmd_readchannel[4] = {"1", "2", "3", "4"};
    uint8_t run_modedetection[4] = {TRUE, TRUE, TRUE, TRUE};
    /* For passing to cmd_read */
    char *argv_read[3] = {"", "", "--result-only"};
    /* To quit the command */
    uint8_t quit = 'a';

    if (argc == 1)
    {
        if (strcmp(argv[0], "--verbose") == 0)
        {
            ver = TRUE;
        }
    }
    /* Hide the cursor for better display */
    PRINT("\033[?25l");
    /* Clear the screen */
    PRINT("\033[2J");
    /* Move the cursor to the origin */
    PRINT("\033[0;0H");
    printtemplate();

    /* Get AOM channel modes and fill in analog table */
    do
    {
        /* Detect operating mode for analog table */
        for (int i = 0; i < 4; i++)
        {
            if (run_modedetection[i] == TRUE)
            {
                channelmode[i] = getmode(i+1);
                if (channelmode[i] != nosignal)
                {
                    run_modedetection[i] = FALSE;
                }
                movetoslot(1, i+1, 'a');
                PRINT("%s", modetxt[channelmode[i]]);
            }
        }
        /* Move the cursor to the origin */
        PRINT("\033[0;0H");
        /* Loop through analog table */
        for (int i = 0; i < 4; i++)
        {
```

```
            if (channelmode[i]==voltagemode || channelmode[i]==currentmode)
            {
                argv_read[0] = cmd_readmode[channelmode[i]];
                argv_read[1] = cmd_readchannel[i];
                movetoslot(2, i+1, 'a');

                cmd_read(consoleStream, 3, argv_read);
            }
        }
        argv_read[0] = cmd_readmode[2];          /* read pwm */
        for (int i = 1; i < 3; i++)              /* Loop through pwm table */
        {
            movetoslot(1, i, 'p');
            argv_read[1] = cmd_readchannel[i-1];
            cmd_read(consoleStream, 3, argv_read);
        }
        quit = sdGetTimeout(&SD3, MS2ST(100));   /* 100 ms timeout */
    } while (quit != 'q');
    PRINT("\033[?25h");                          /* Show the cursor */
    PRINT("\033[20;0H");                 /* Move the cursor to the origin */
}

/*****************************************************************************/
//! \brief Print template table
//!
//!
//
/*****************************************************************************/
void printtemplate(void)
{
    //For some reason, the cursor moves to (1,1) after PRINT("\033[0;0H")
    PRINT("Live analog signals:\n\r");
    PRINT("-------------------------------------------------------!\n\r");
    PRINT("| Channel  | 1        | 2        | 3        | 4        |\n\r");
    PRINT("|----------|----------|----------|----------|----------|\n\r");
    PRINT("| Mode     | Voltage  | Current  | No signal| No signal|\n\r");
    PRINT("|----------|----------|----------|----------|----------|\n\r");
    PRINT("| Value    | +11.999 V| 12.345 mA| N/A      | N/A      |\n\r");
    PRINT("!-------------------------------------------------------!\n\r");

    PRINT("Live PWM signals:\n\r");
    PRINT("!---------------------------------!\n\r");
    PRINT("| Channel    | 1        | 2        |\n\r");
    PRINT("|------------|----------|----------|\n\r");
    PRINT("| Frequency  | 10000 Hz | 10000 Hz |\n\r");
    PRINT("|------------|----------|----------|\n\r");
    PRINT("| Period     | 099.1 ms | 099.1 ms |\n\r");
    PRINT("|------------|----------|----------|\n\r");
    PRINT("| Duty Cycle | 077.7 %c | 066.6 %c |\n\r", '%', '%');
    PRINT("!---------------------------------!\n\r");
    PRINT("Enter 'q' to quit.\n\r");
}
```