Preface

This report is a result of the master thesis carried out in second semester of the 5th and final year of a master study at the institute of Engineering Cybernetics, IME faculty at the Norwegian University of Science and Technology. The project is a continuance of the autumn project carried out in the previous semester. The complete report for the autumn project is found digitally in appendix C.

The master thesis assignment is provided by Thelma AS in collaboration with the institute of Engineering Cybernetics. Thelma provided the specification as well as an office space with access to assembly and test equipment.

This master thesis included development, assembly and testing of a complete system. This is a major task to complete in only five months and focus has therefore been on testing the system as a design concept rather than implementing all details of a final product. Suggestions for further development have though been included at the end of this report.

I would like to thank my teaching supervisor Jo Arve Alfredsen for his great effort and guidance throughout the project. I would also like to express gratitude towards Åge Grønningsæter at Thelma for his help and support.

Finally I would like to thank Atmel Norway and John Olav Horrigmo at the institutes' component service for providing equipment, components and tools.

Trondheim
31st of July 2009

Stian Orø Moen

Summary

A telemetry buoy is a standalone device used to receive acoustic pulse-position modulated signals from digital tags attached to fish. The buoy is submerged in the sea and will store any valid signals from tagged fish. The received information typically includes an ID and can include information such as salinity level, depth, temperature, acceleration data and more. The goal of this master thesis was to design and validate a system for a low power telemetry buoy which utilizes digital signal processing.

This thesis is a continuance of the work, done by the same author, in the project described in *Acoustic telemetry buoy* project report [4]. The goal of the previous project was to answer the question "*is it possible to design a low power acoustic telemetry buoy which satisfies at least one year operation and 100 000 stored receptions based on digital signal processing?"*. The conclusion was that it should be possible to design as system which meets the requirements using the UC3B microcontroller. In addition it is recommended to design an external reception detector to wake up the microcontroller and initiate reception.

While the previous project focused on the digital part of the system the master thesis focuses on designing a complete system. The main building blocks that have been designed are

- Variable gain amplifier
- Active analogue band pass filter
- Reception detector
- MCU with external flash

Several design concepts have been discussed for each block whereas one complete system prototype have been fully designed, assembled and tested. The system has been proved to work by successfully receiving signals from a standard Thelma fish tag.

The prototype has been thoroughly tested and adjustments to improve performance have been suggested. One of the most important criteria is low power consumption and the minimum of one year operation. Current measurements and energy consumption calculations conclude with an expected operation time of 1.51 years. Calculations show that the expected operating time is further increased to 2.21 years if the new Atmel AT32UC3L microcontroller is used in the final design. The design fulfils the other provided requirements with good margin.

The final conclusion is therefore that the system design provided in this report, together with the suggested improvements and the UC3L microcontroller, provides Thelma with a good basis for developing the leading standalone ultra low power acoustic telemetry buoy on the market.

Contents

PREFACE ..... **I**

SUMMARY ..... **II**

CONTENTS ..... **III**

1. INTRODUCTION **1**

1.1 PROJECT DEFINITION 1
1.2 BACKGROUND 1
1.3 EARLIER WORK 2
1.4 AVAILABLE SOLUTIONS 2

2. DESIGN PRINCIPLES **4**

2.1 UNDERSAMPLING 4
2.2 GENERAL LOW-POWER DESIGN CONSIDERATIONS 6
2.3 SPECIFIC POWER CONSUMPTION CONSIDERATIONS 7

3. SYSTEM OVERVIEW **9**

4. ANALOGUE DESIGN **10**

4.1 SIGNAL PROPERTIES 10
4.1.1 Transmission protocol for Vemco fish-tags *11*
4.2 AUTOMATIC GAIN CONTROL AMPLIFIER 13
4.2.1 Variable gain amplifier IC *13*
4.2.2 Variable gain amplifier using digital potentiometers *13*
4.2.3 Design solution *18*
4.3 FILTER DESIGN 19
4.3.1 Potentiometer noise removal filter *19*
4.3.2 The intuitive filter solution *23*
4.3.3 Butterworth filter *25*
4.3.4 Chebyshev filter *29*
4.3.5 Design solution *33*
4.4 RECEPTION DETECTOR 37
4.4.1 Detecting a signal using an RMS to DC converter *37*
4.4.2 Detecting a signal using a product detector *37*
4.4.3 Detecting a signal using a diode detector *38*
4.4.4 Detecting a signal using a modified diode detector *38*
4.4.5 Design solution *40*
4.5 COMPONENT SELECTION 42
4.5.1 Operational amplifier *43*
4.5.2 Digital potentiometer *43*

5. DIGITAL DESIGN **45**

5.1 CALCULATING AND SETTING THE GAIN IN THE AGC LOOP 45
5.2 SAMPLING AND FILTRATION 46
5.3 DECODING THE SIGNAL 46
5.4 STORING THE RECEIVED DATA 46
5.5 RECOVERING THE STORED DATA 47

6. PROTOTYPE DESIGN **48**

6.1 POWER DISTRIBUTION 48
6.2 PREAMP 49
6.3 VARIABLE GAIN AMPLIFIER AND FILTER 50
6.4 RECEPTION DETECTOR 51
6.5 CONNECTORS AND HEADERS 52
6.6 MCU AND FLASH MEMORY 52
6.7 PCB LAYOUT 53

7. FIRMWARE **56**

7.1 ADC_FUNCTIONS ... 58
7.2 CLOCK_FUNCTIONS ... 59
7.3 DSP_FUNCTIONS ... 59
7.4 EIC_FUNCTIONS ... 59
7.5 RTC_FUNCTIONS ... 59
7.6 SPI_FUNCTIONS ... 60
7.7 TC_FUNCTIONS ... 60
7.8 USART_FUNCTIONS ... 60
7.9 TEST_FUNCTIONS ... 61
7.10 MAIN.C ... 61

8. TEST AND MEASUREMENT PROCEDURES ... 62

8.1 FUNCTIONALITY TESTS ... 62
8.1.1 Power distribution ... 63
8.1.2 Preamp ... 63
8.1.3 Variable gain amplifier and filter ... 64
8.1.4 Reception detector ... 64
8.1.5 USB circuit ... 65
8.1.6 MCU and flash memory ... 65
8.1.7 Complete system test ... 66
8.2 PERFORMANCE TESTS AND MEASUREMENTS ... 67
8.2.1 Variable gain amplifier and filter ... 67
8.2.2 Reception detector ... 71
8.2.3 MCU and flash memory ... 73

9. TEST RESULTS AND MEASUREMENT DATA ... 74

9.1 ACCURACY OF THE RESULTS ... 74
9.2 FUNCTIONALITY TEST RESULTS ... 75
9.3 PERFORMANCE TEST AND MEASUREMENT RESULTS ... 78
9.3.1 Variable gain amplifier and filter test results ... 78
9.3.2 Reception detector test results ... 81
9.3.3 MCU and flash memory test results ... 83

10. DISCUSSION ... 84

10.1 ANALOGUE DESIGN ... 84
10.1.1 Band pass filter ... 84
10.1.2 Variable gain amplifier ... 85
10.1.3 Reception detector ... 87
10.2 DIGITAL DESIGN ... 88
10.2.1 MCU and Flash memory ... 88
10.3 COMBINED DESIGN ... 90

11. FURTHER DEVELOPMENT ... 92

11.1 THE CHOICE OF MICROCONTROLLER ... 92
11.2 DESIGN IMPROVEMENTS ... 93
11.3 ADDITIONAL FUNCTIONALITY ... 94
11.4 DEVELOPMENT OF NEW MODULATION SCHEMES ... 94

12. CONCLUSION ... 97

13. APPENDIX LIST ... 98

14. BIBLIOGRAPHY ... 98

1. Introduction

This chapter describes the project definition with associated prototype requirements as well as the background for the project. Furthermore a short summary of the results of the previous acoustic telemetry buoy project is provided in section 1.3 followed by a short summary of the similar products available.

1.1 Project definition

The goal of this master thesis is to design and validate a system for a low power telemetry buoy which utilizes digital signal processing and fulfils the requirements specified in the following.

The telemetry receiver should meet the following specifications:

- Store at least 100 000 receptions with individual time stamps
- Operating time of at least one year using one D-cell Tadiran 19Ah lithium battery [23]
- Handle acoustic receptions of modulated signals with carrier frequencies ranging from 60 kHz – 80 kHz.

In addition to the specifications it is desired to have the following features:

- High level of digital control to create a versatile and easy modifiable product
- Possibility to support new forms of modulation schemes without hardware modifications

1.2 Background

A telemetry buoy is a standalone device used to receive acoustic pulse-position modulated signals from digital tags attached to fish. The buoy is submerged in the sea and will store any valid signals from tagged fish. The received information typically includes an ID and can include information such as salinity level, depth, temperature, acceleration data and more.

Automatic receiver buoys play an important role for the applicability of acoustic telemetry in large scale studies of fish and other species' behaviour in the ocean. The placement of several telemetry buoys in strategic positions provides the ability to monitor behavioural patterns for long periods of time.

At the time of writing Thelma AS has a manual operated telemetry receiver. This is used to manually observe tagged fish for research purposes. In many situations it is neither practical nor possible to have people operating such a device. This is typically research where records are done in a period of several weeks or months. It is desired to have a stand-alone device that can be left for a longer period of time that records the presence of tagged fish. Some solutions exist, but the device must typically be fetched to retrieve the recorded information. Thelma wishes to develop their own product with improved performance compared to the competing products, and the ability to add more user friendly data retrieval through GSM communication.

1.3 Earlier work

This master thesis is a continuance of the work, done by the same author, in the project described in *Acoustic telemetry buoy* project report [4]. This section provides a short summary of the key observations made in the project. The complete report is found in appendix C.

The goal of the previous project was to answer the question "*is it possible to design a low power acoustic telemetry buoy which satisfies at least one year operation and 100 000 stored receptions based on digital signal processing?"*

The conclusion reads:
"The final conclusion is therefore that the AT32UC3B microcontroller should be used for further design. With this controller it should be possible to design an acoustic telemetry buoy that meets the requirements given in section 1.1. If a good external trigger for reception can be constructed this will provide the lowest energy consumption. The system should though incorporate digital filtration to allow compatibility with various frequencies and modulation schemes."

In addition to the final conclusion several forms of modulation schemes where discussed. The most important observations with respect to modulation are quoted below:

"Kilfoyle and Baggeroer states in an IEEE report [8] that the frequency content of an underwater telemetry signal remains largely contained within its original band whereas the amplitude and phase of the signal can vary widely in both time and space. This observation naturally concludes that modulation schemes using pulse position or frequency are of the most interest."

"For acoustic telemetry this modulation form can provide a higher throughput than PPM due to its better immunity against multipath distortion if the same frequency is not repeated too frequently. It will not add much complexity for the fish tag, but the receiver must be able to separate the different frequencies which will add complexity to the digital or analogue signal processing."

The battery to be used is specified in section 1.1 is also studied and it is shown that to be able to utilize the battery the system must operate at a voltage area of 2 – 3.7V. Even with this requirement the total capacity of the battery is reduced to about 13Ah resulting in an estimated total energy of 163800 J which leads to a maximum average current consumption of 1.48 mA for the complete system.

1.4 Available solutions

There are currently two other vendors of telemetry buoys; VEMCO and SONOTRONICS. The latest telemetry buoy from each vendor is the VR2W and the SUR respectively. The following will provide a short description of these products.

VR2W
The VEMCO produced VR2W have the following features:

- 8 MB of memory
- 1 000 000 detections
- 15 months operating time

- Receiver frequency: 69.0 kHz
- Bluetooth communication
- Real time clock

SUR – Submersible Ultrasonic Receiver
The SONOTRONICS produced SUR have the following features:

- 1 MB flash memory
- 100 000 detections
- 7-12 months operating time using two batteries
- 15 selectable frequencies ranging 30 kHz – 150 kHz
- RS-232 communication
- Real time clock
- Ping and response function to check if the SUR have any data

Note that the SUR will not listen to all 15 frequencies at the same time. It will listen for a particular frequency for two seconds, then it will power down for one second, power up and listen to the next frequency. In addition a one second delay is added after all 15 frequencies have been scanned. As a result a particular frequency will be checked every 46 seconds. With these delays the receiver will have an operating time of seven months. It is possible to increase the delays to achieve up to twelve months operating time.

2. Design principles

This chapter covers the hardware and software design techniques and principles used to design the acoustic telemetry buoy.

2.1 Undersampling

The most common form of digitizing an analogue signal is sampling according to the Nyquist theorem in equation (2.1).

$$f_s > 2 \cdot f_{\max} \tag{2.1}$$

Where
f_s is the sampling frequency
f_{max} is the highest frequency of the analogue signal

To avoid violating the theorem a low-pass filter must be used as in Figure 2-1.

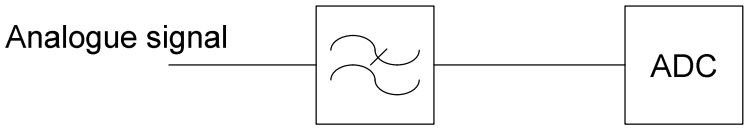


Figure 2-1 - Nyquist sampling with low pass filtration

If the criteria in (2.1) is violated aliasing will occur. Aliasing refers to an effect that causes different continuous signals to be indistinguishable when sampled. As an example a 100 Hz sinusoidal signal is sampled with a sampling frequency f_s = 80 Hz. The resulting sampled signal is illustrated in Figure 2-2. Notice that when the 100 Hz sinusoidal is sampled, the result is a 20 Hz sinusoidal. It is therefore not possible to distinguish the alias of the 100 Hz signal from an original 20 Hz signal.

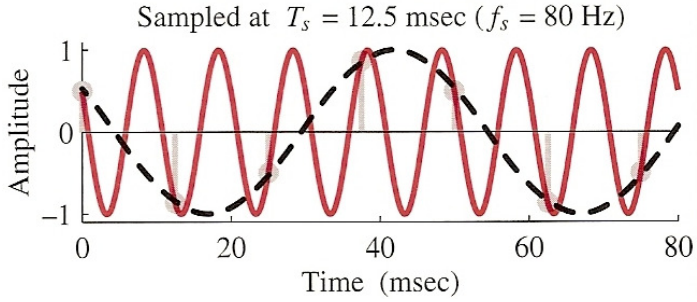


Figure 2-2 - Aliasing example, adapted from [3]

If all other frequencies are removed by a filter as in Figure 2-3, we can utilize the alias. Knowing that no frequencies outside the area $f \in [80Hz, 120Hz]$ will occur we know that the 20Hz digitized signal must originate from a 100 kHz analogue signal. This technique is known as undersampling or band pass sampling.

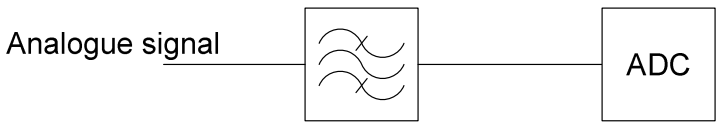


Figure 2-3 Undersampling with band pass filtration

According to undersampling theory [4], if the sampling rate f_s satisfies (2.2), the original signal can be reconstructed without any loss.

$$\frac{2f_H}{k} \le f_s \le \frac{2f_L}{k-1} \tag{2.2}$$

Where k is an integer satisfying

$$1 \le k \le \left\lfloor \frac{f_H}{f_H - f_L} \right\rfloor \tag{2.3}$$

Where
f_s is the sampling frequency
f_h is the highest frequency component of the signal
f_l is the lowest frequency component of the signal

Since a lower frequency is used over the same period of time without loss of information the result is fewer samples and therefore a more power efficient solution. The signal to noise ratio (SNR) typically decreases when reducing the number of samples, but since the band pass filter also reduces the bandwidth of the noise the SNR remains unchanged.

When receiving acoustic signals from fish tags, not all frequencies are of interest. As specified the section 1.1, only frequencies $f \in [60kHz, 80kHz]$ are relevant. Assuming that a band pass filter as in Figure 2-3 removes all other frequencies we can calculate the valid values of k:

$$1 \le k \le \left\lfloor \frac{80 \cdot 10^3}{60 \cdot 10^3} \right\rfloor \tag{2.4}$$
$$1 \le k \le 4$$

By using (2.2) the valid sample frequencies that do not produce overlapping of aliases are:

Table 2-1 - Valid sample frequencies

k	lowest f_s [Hz]	highest f_s [Hz]
1	160000	∞
2	80000	120000
3	53333,33333	60000
4	40000	40000

Notice that k = 1 provides the more common Nyquist frequency and k = 4 is the absolute minimum undersampling frequency where $f_s = 2 \cdot B$. By using a sampling frequency of 40 kHz the frequency aliases will occur as shown in Figure 2-4.

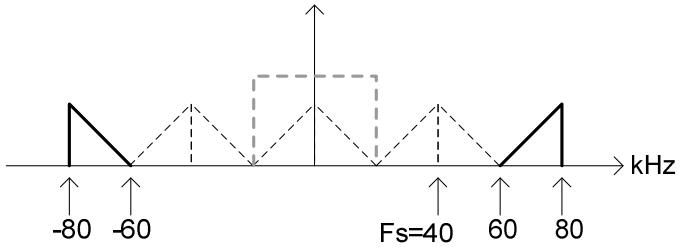


Figure 2-4 - Frequency spectrum when undersampling with fs = 40 kHz

Notice that both the negative and the positive frequency components are aliased. When undersampling, the frequency band will be aliased with an offset equal to the sampling frequency. The band is drawn with an incline with increasing frequency. The dotted lines are the resulting aliases. When using $f_s = 40$ kHz we get the inverse frequency band in 0-20 kHz since this is an alias of the negative frequency band. This will not present a problem as long as it is compensated for in software.

The latter example results in frequency bands with no separation. This will in theory require an optimal band pass filter to avoid aliasing. Some aliasing may still be permitted as long as this only contains signal levels below the noise floor and/or signals below the dynamic range of the ADC. Choosing a sampling frequency $f_s = 58$ kHz will provide more separation as shown in Figure 2-5. Some of the original frequencies will occur at two aliased frequencies, but since they don't overlap they can be removed using a digital filter.

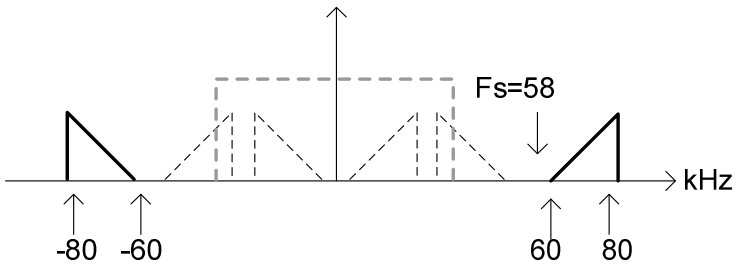


Figure 2-5 – Frequency spectrum when undersampling with fs = 58 kHz

2.2 General low-power design considerations

In addition to choosing the component with the lowest power consumption there are several other factors that must be taken into consideration. The highest level of integration usually results in the lowest power consumption. In an embedded system this often means choosing a microcontroller with many of the needed features inbuilt. Connecting "components" in an internal circuit instead of on a PCB will in most cases result in less leakage, provide the possibility of running the components closer to the voltage limits and a higher grade of optimization.

The most fundamental low-power requirement for any embedded system is that the system must be interrupt driven. This will allow the processor to sleep whenever the CPU is not needed for computation. Delay routines of the type

```
for(int i = 0; i < 200; i++){
//do nothing
}
```

are therefore strongly prohibited in a low-power application. The solution is to use a timer with an associated interrupt to wake the processor when the delay has expired. Modules that can relieve the CPU will also contribute to more time in sleep modes. This can for instance be a communication module, DMA controller, timer, etc.

Computation prohibits the CPU or MCU from being in a sleep mode. It is therefore vital that the computation time is at a minimum. This requires not only a high frequency and a powerful CPU instruction set, but that the program is constructed and optimized for the specific CPU. To achieve this, great knowledge about the instruction set and the compiler is required. Optimization settings for the compiler must be set, but without knowledge about the CPU the

program will never be optimal. An example is using floating-point numbers on a CPU without hardware support for this. All computation will therefore have to be done in software which will lead to a large increase in computation time.

Oscillators will also contribute to the overall power consumption. The designer must not only consider the frequency, but also the type of oscillator. The typical oscillators are internal RC oscillator, external clock, external crystal and external resonator. Factors that must be considered are

- Power consumption
- Start-up time
- Stability (jitter)
- Accuracy

Studying for instance the XMEGA manual [25] we find that the internal RC oscillator has the shortest start-up time and is for many microcontrollers the oscillator that requires the least amount of power. The downsides are that an RC oscillator typically will be more inaccurate and have a grater amount of jitter.

Since a real-time clock often is required in an embedded system many microcontrollers feature an ultralow-power, low frequency oscillator designed especially for 32.768 kHz quartz crystals. The low frequency usually excludes the possibility of using this oscillator as a system clock. On AVR and AVR32 microcontrollers it can though be used as reference to do runtime calibration of the internal RC oscillator to achieve greater accuracy.

Most microcontrollers have the ability of connecting an external crystal, external clocks are therefore seldom used. The external clock option can though be used if there are several other devices that require an oscillator. The devices can then share the same clock.

Unwanted oscillations may also contribute to an increase in power consumption. It is therefore important to use appropriate decoupling and insure a stable digital level on unused inputs. A high input resistance is also important to ensure that a minimum of current is consumed.

2.3 Specific power consumption considerations

This section describes considerations, specified by the manufacturer, for minimizing power consumption in a microcontroller. The devices studied are limited to Atmel AVR XMEGA, Atmel AVR32 UC3 and Texas Instruments MSP430.

Sleep modes enables the microcontroller to shut down unused modules to save power. When the device enters sleep mode, program execution is stopped. Interrupts or reset is used to wake the device again. Moreover, the individual clock to unused peripherals can be stopped during normal operation to save power.

The wake-up time for the device is dependent on the sleep mode and the frequency of the main clock source. The start-up time for the system clock source must be added to the wake-up time for sleep modes where the clock source is stopped. The ability to get into and out of the low-power modes and process data quickly is crucial because current is wasted by the CPU waiting for the clock to become stable [24]. Some MCUs have a two-stage clock wake-up providing a low-frequency clock to the CPU while a high-frequency clock is being

stabilized. On these devices the CPU may be operational in a short time, but running inefficient due to the low frequency.

For the XMEGA [25] the power reduction register provides a method to stop the clock to individual peripherals. This can be used in sleep modes to reduce overall power consumption significantly. On some Texas Instruments devices [24] the peripherals have the ability to disable themselves automatically when not in use. For more complex devices like the AVR32 UC3 [26] the ability to control the clock is at a much more detailed level. The synchronous clock generator can adjust the performance of the system, according to the current requirements, by switching between three different clock sources; internal RC-oscillator, PLL0 and oscillator0. Depending on the developers design it may be better to scale the clock instead of switching the source. This can be done in most MCUs “on-the-fly”. In addition to scaling and switching the synchronous clock for the CPU, the clock for each of the internal buses can be scaled down when the bus is not utilized completely. The DMA controller allows the bus to work at a different speed then the CPU. Hence having a DMA can lower the power consumption both in active and sleep mode [24].

Leakage current is sometimes overlooked when choosing a low-power MCU, but it must be considered for the most demanding low-power applications [24]. For the UC3 [26] all pins that are not connected externally to pull-ups, pull-downs, ground or power should be left as inputs, but with the internal pull-up enabled. This will ensure a stable digital level while reducing the input current with the internal resistor and thereby ensure the lowest possible power consumption

3. System overview

As described in section 1.3 the project report *Acoustic telemetry buoy* [4] concludes that the microcontroller AT32UC3B should be used together with a reception detector in the development of the telemetry buoy application. It also states that the telemetry buoy should incorporate some form of digital signal processing to allow compatibility with various frequencies and modulation schemes.

Figure 3-1 shows the system design concept. To be able to utilize the benefits of undersampling the system incorporates an analogue band-pass filter. Furthermore it features an automatic gain control amplifier. This is used to adjust the sensitivity of the receiver and provide a means of adjusting the signal to a suitable level before performing sampling. The reception detector will act as both a wakeup source for the microcontroller and a means of measuring the signal level. This signal can then be used as feedback in the AGC loop.

The filter, variable gain amplifier and reception detector will always be on, these blocks will therefore have to be designed for ultra low power consumption. The microcontroller will control the analogue gain in the variable gain amplifier, perform sampling, digital signal processing, decoding and storage of the reception if found valid.

The hydrophone will be selected based on the physical shape of the buoy and is not a topic of this report. The amount of gain in the fixed gain stage will depend on the choice of hydrophone and is therefore not covered.

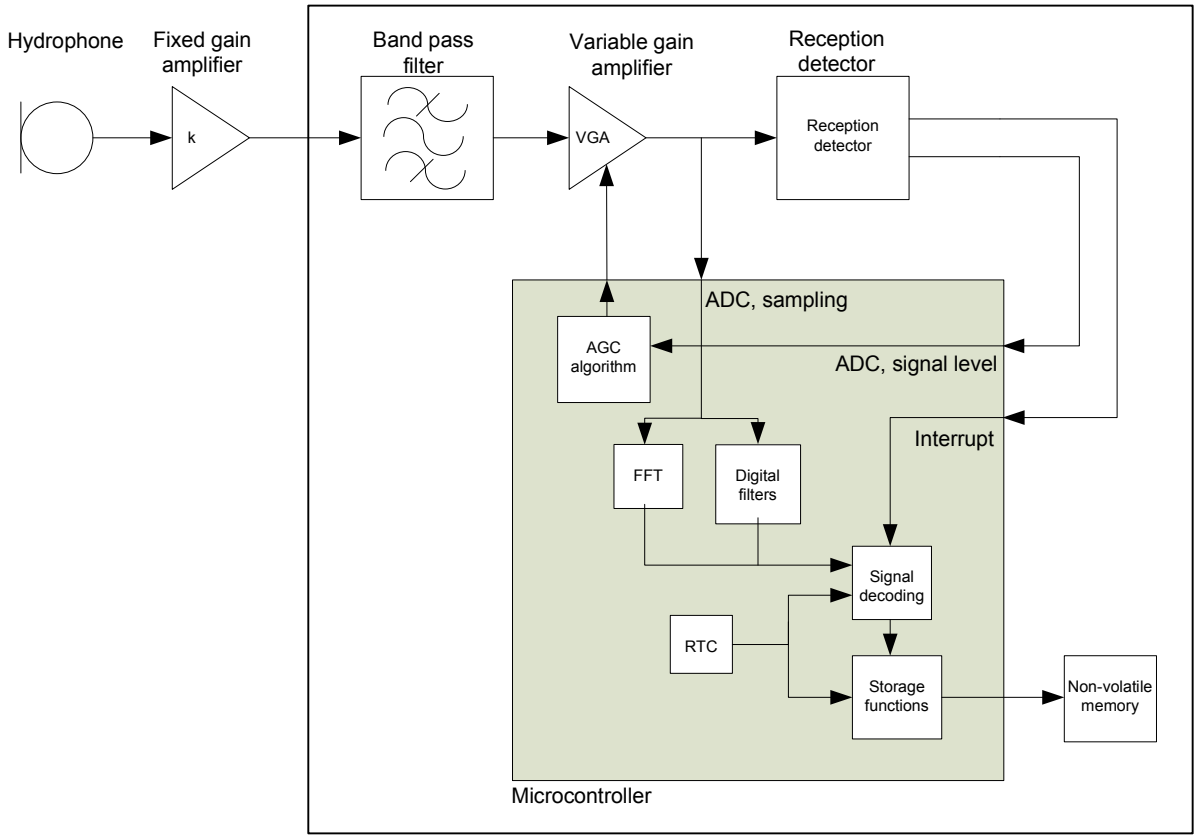


Figure 3-1 - System overview

4. Analogue design

This chapter describes the analogue design of the telemetry buoy. The analogue circuit consists of three main blocks:

- Variable gain control amplifier
- Band pass filter
- Signal detector.

The following sections will discuss several implementation methods for each of the blocks. The *Design Solution* section describes the choice for implementation for each of the described methods.

4.1 Signal properties

Before designing the analogue circuit it is important to determine the properties of the signal it should handle. Thelma's fish tags typically output a signal of about 150 dB (relative to 1μP at 1m) [9]. The ambient noise in the spectra of interest (60kHz to 80kHz) in sea water is dominated by thermal noise and sonic noise produced by weather. The SINTEF report *Fish telemetry* manual [8] provides the equation

$$TL = SL - (NL + 10\log B) - DT \tag{4.1}$$

Where
SL: transmitter source level (dB)
TL: transmission loss (dB)
B: bandwidth of channel
NL: noise level per unit bandwidth
DT: detection threshold

The result TL ultimately results in the dynamic range of the analogue circuit. The TL level will provide the highest signal level above the noise floor. Signals below the noise floor cannot be retrieved and are therefore of no interest. As shown in Figure 4-1 the lowest ambient noise level is about 20 dB. Assuming a bandwidth of the signal of 500Hz and that all signals above the noise floor can be retrieved the maximum dynamic range is found:

$$DR = 150 - (20 + 10\log 500) - 0 = 103dB \tag{4.2}$$

The AGC circuit should therefore have a dynamic gain range of 103 dB.

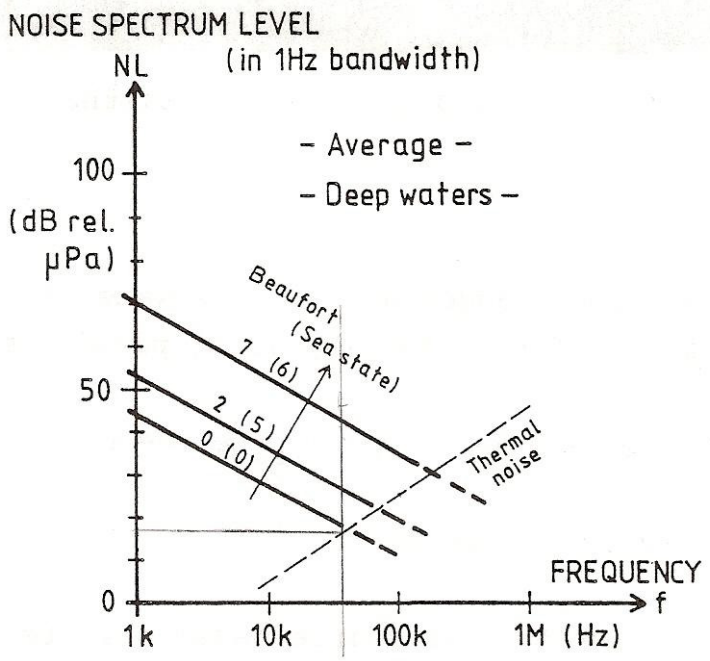


Figure 4-1 - Noise in the sea, adapted from [8]

4.1.1 Transmission protocol for Vemco fish-tags

The telemetry buoy should be designed to receive signals defined by Vemco. This section provides a short description gathered from the report *Acoustic telemetry buoy* [4] written by the same author as this report.

Existing digital acoustic fish tags use pulse position modulation (PPM) to send information. The package includes an ID and can include specific information such as temperature, depth, salinity, heart rate, checksum and more.

The most common format that is used is defined by Vemco. The description is provided by Thelma and is in essence as follows

- Differential pulse-position modulated signal
- Pulse length T_p = 10ms for all pulses
- Timeslot T_s = 20ms, pulses occur in the middle of the timeslot
- There are M = 4 bits coded by each pulse, this gives 2^4 = 16 time slots
- A transmission starts with two synchronization pulses. The difference in time between the pulses T_{sync} corresponds to the amount of data that will be transmitted i.e. the number of expected pulses.
- A guard time T_g it added between all pulses to compensate for any multipath or echo distortions.
- A transmission ends with an 8-bit CRC checksum.
- All pulses are detected on a rising edge.
- Timing is always referred to the rising edge of a pulse.

Figure 4-2 gives an example of a transmission of the data 30 and checksum 5. In addition to the listed specifications the following applies for the example:

- T_g = 380ms, guard time to suppress multipath distortion
- T_{sync} = 360ms, it is assumed that this means that one byte + CRC will be transmitted

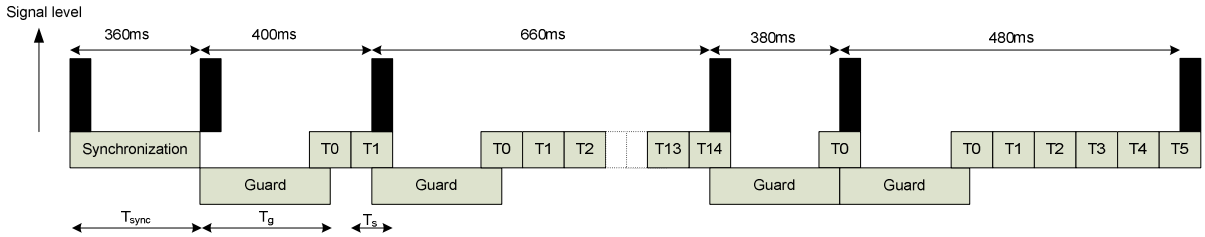


Figure 4-2 - Example of a Vemco DPPM packet

The time delays between the pulses are calculated by the transmitter as in Table 4-1.

Table 4-1 - Composition of a DPPM Vemco signal

Delay between	Delay components	Information	Delay
Pulse 1-2	Synchronization time $T_{sync.}$	Defined by protocol, specifies a total of 6 pulses i.e. 1 byte data	360ms
Pulse 2-3	Guard time T_g + 1 x Timeslot T_s,	Data[7:4] = 0001_2	400ms
Pulse 3-4	Guard time T_g + 14 x Timeslot T_s, data:	Data[3:0] = 1110_2	660ms
Pulse 4-5	Guard time T_g + 0 x Timeslot T_s	CRC[7:4] = 0000_2	380ms
Pulse 5-6	Guard time T_g + ¨5 x Timeslot T_s	CRC[3:0] = 0101_2	480ms
		SUM	2280ms

4.2 Automatic gain control amplifier

The automatic gain control (AGC) circuit consists of a variable gain stage, measurement and control stage as shown in Figure 4-3. The control stage may be either analogue or digital. For the telemetry buoy a digital approach is chosen to be able to alter the settings without modifying any hardware. The digital approach also provides the possibility of using digital potentiometers to adjust the gain.

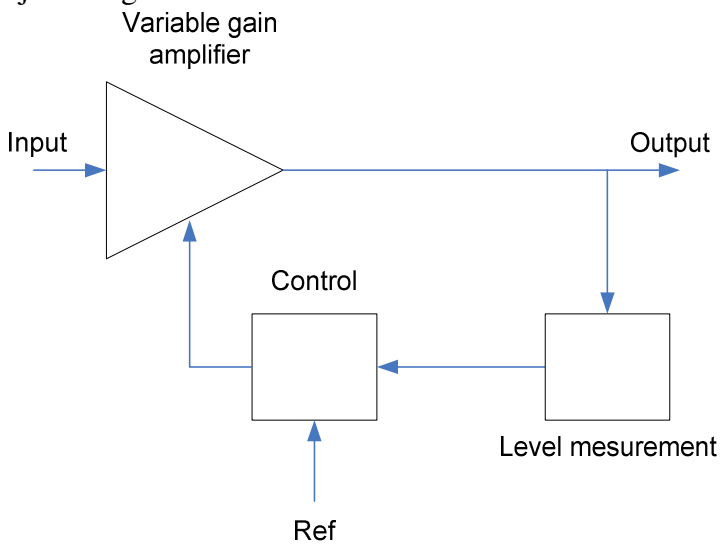


Figure 4-3 - AGC system overview

The level measurement can be done in several ways. These are described in section 4.4 Reception detector. The variable gain stage can be realised in several ways. The following sections will cover realisation by using a variable gain amplifier IC and by using operational amplifiers together with digital potentiometers.

4.2.1 Variable gain amplifier IC

The *intelligent hydrophone buoy* designed by Jan Eyolf Bjørnson [7] uses a variable gain amplifier (VGA) IC (AD604). Analog devices is the main supplier of VGA's. The devices offer a high dynamic range, linar amplification and high bandwidth. The downside is the rather high current consumption. The device requiring the least power (AD600) will draw a supply current of 14 mA. The AGC provides a signal to the reception detector. The VGA will therefore have to be enabled at all times. A current consumption of 14mA is considerably over the maximum limit of 1.48 mA as specified in 1.3. This makes the VGA IC unsuitable for the battery powered telemetry buoy.

4.2.2 Variable gain amplifier using digital potentiometers

Digital potentiometers alter the resistance by connecting the "wiper" to the resistors through a set of CMOS transistors. The control signal varies from protocols like I^2C and SPI to basic inputs such as step up and step down. The most common types have 256 or 32 discrete steps. The downside of using digital potentiometers is the tolerance. The end-to-end resistance typically has a tolerance of ±25% to ±30% [11]. This is manly due to process variations.

Maxim application notes [11] and [12] describe four principles for connecting a digital potentiometer in an operational amplifier connection. The configurations are described in the following sections.

4.2.2.1 Traditional audio control configuration

The traditional volume control shown in Figure 4-4 features a potentiometer at the input followed by an amplifier with fixed gain. This configuration will provide a linear gain. The downside is that the signal is scaled down before being amplified. For small signals this will result in a reduction of the signal to noise ratio (SNR). Noise introduced by the potentiometer or after the potentiometer will be more significant since the amplification is always at maximum after the signal scaling.

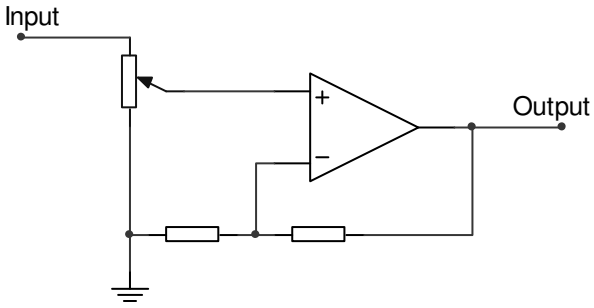


Figure 4-4 - Traditional volume control

4.2.2.2 Non-inverting configuration with one additional resistor

The circuit diagram is shown in Figure 4-5. This configuration provides a linear gain, but the gain is dependent on the end-to-end resistance of the digital potentiometer. The gain will therefore have the same tolerance as the potentiometer. This may not be a problem in an AGC application if the tolerance is taken into consideration at the design stage.

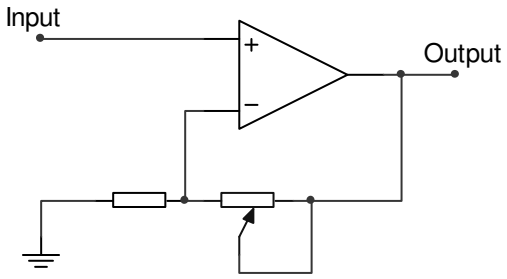


Figure 4-5 - Non-inverting amplifier with linear gain

4.2.2.3 Non-inverting configuration without additional resistors

The circuit diagram is shown in Figure 4-6. This configuration eliminates the gain tolerance introduced by the end-to-end tolerance of the digital potentiometer. The gain adjustment will though be non-linear. If we define $N \in [0,1]$ as the position of the wiper and G as the gain we get the expression

$$G(N) = 1 + \frac{N}{1-N} \tag{4.3}$$

The result is plotted in Figure 4-7.

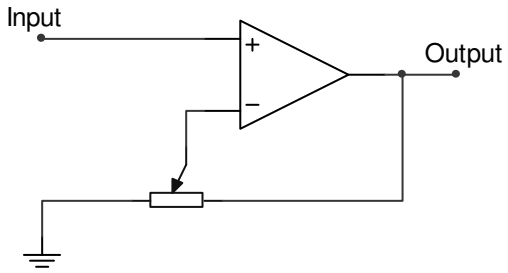


Figure 4-6 - Non-inverting amplifier with non-linear gain

Note that this connection will result in an infinite gain when N = 1. The gain must therefore be limited either in software or by external resistors

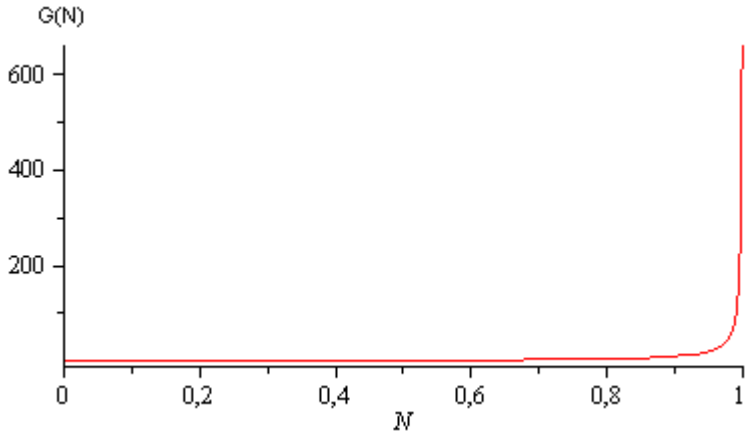


Figure 4-7 - Gain as a function of N

4.2.2.4 Potentiometer in a positive feedback configuration

The circuit diagram is shown in Figure 4-8. In this configuration the potentiometer is supplying the operational amplifier with positive feedback in addition to the negative feedback via the fixed resistors. If we define K_n as the negative feedback fraction and K_p as the positive feedback fraction it can be shown [12] that the gain of the circuit is

$$G = \frac{1 - K_n}{K_p - K_n} \tag{4.4}$$

Notice that the system becomes unstable when the positive feedback fraction K_p exceeds the negative feedback fraction K_n. It must therefore be ensured in software or with an additional resistor that $K_p < K_n$. If a software approach is chosen we get an amplifier which is not dependent on the end-to-end tolerance of the potentiometer.

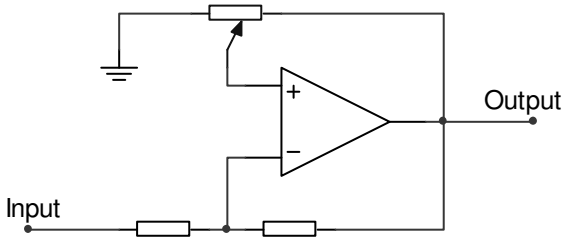


Figure 4-8 - Positive feedback configuration

The gain is non-linear as shown in Figure 4-7. Here K_n is set to 2/3.

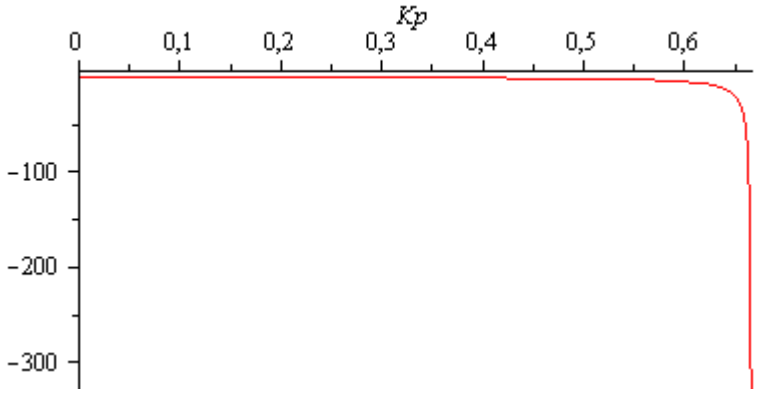


Figure 4-9 - Gain as function of Kp

4.2.2.5 Non-idealities of a digital potentiometer

When designing a variable gain amplifier (VGA) using digital potentiometers it is important to understand the non-idealities of digital potentiometers. This section addresses the most important factors.

End-to-end tolerance

An important factor is the tolerance of the end-to-end resistance. If the AGC design is a traditional control loop this may not present a problem. For a low power design it is desired to avoid using a continuous control loop. A more effective approach is to measure the signal strength and calculate the correct gain. This approach requires that the gain has little tolerance or that some sort of lookup table is generated at production. This lookup table may be hard to generate and it is therefore preferable to use a configuration which eliminates that the gain is dependent on the end-to-end tolerance. This leaves the *traditional audio gain*, *the non-inverting design without additional resistors* and the *positive feedback* configurations.

Discrete resistance value

As opposed to traditional potentiometers the digital potentiometer has fixed resistance steps. Typical values are 32, 64, 128 and 256 steps. If a higher resolution is needed several potentiometers can be used.

Frequency dependent operation

The datasheet for each individual potentiometer specifies a -3dB crossover frequency. For a 100k potentiometer this value is typically in the area of 40 kHz – 80 kHz. This is due to the stray capacitance between the wiper and ground [13] as shown in Figure 4-10.

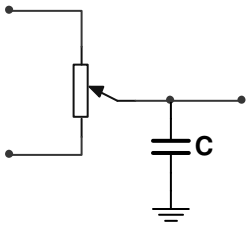


Figure 4-10 - Non-ideal digital potentiometer

The introduction of a capacitance in the circuit will lead to frequency dependent gain. This may present a problem even when the signal is below the given bandwidth of the potentiometer. If we consider the configuration described in section 4.2.2.3 and view the potentiometer as two resistors we get the connection as in Figure 4-11. The dotted line encapsulates the digital potentiometer.

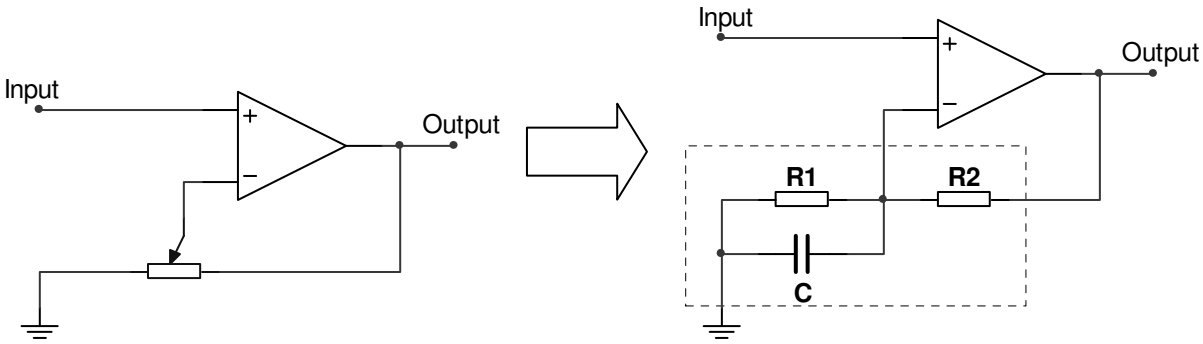


Figure 4-11 - Non-inverting configuration with non-ideal potentiometer

The transfer function is as follows

$$G = H(s) = 1 + \frac{R_2}{Z} = 1 + \frac{R_2}{\frac{R_1}{1 + sR_1C}} = \frac{R_1R_2Cs + R_2}{R_1} \tag{4.5}$$

The bode plot for (4.5) is shown in Figure 4-12. R1 = R2 = 50k, C = 60 pF. Notice that for frequencies higher than 50 kHz the gain is non-linear with respect to frequency. It is therefore vital that this capacitance is taken into consideration, especially when using the digital potentiometer in a feedback loop.

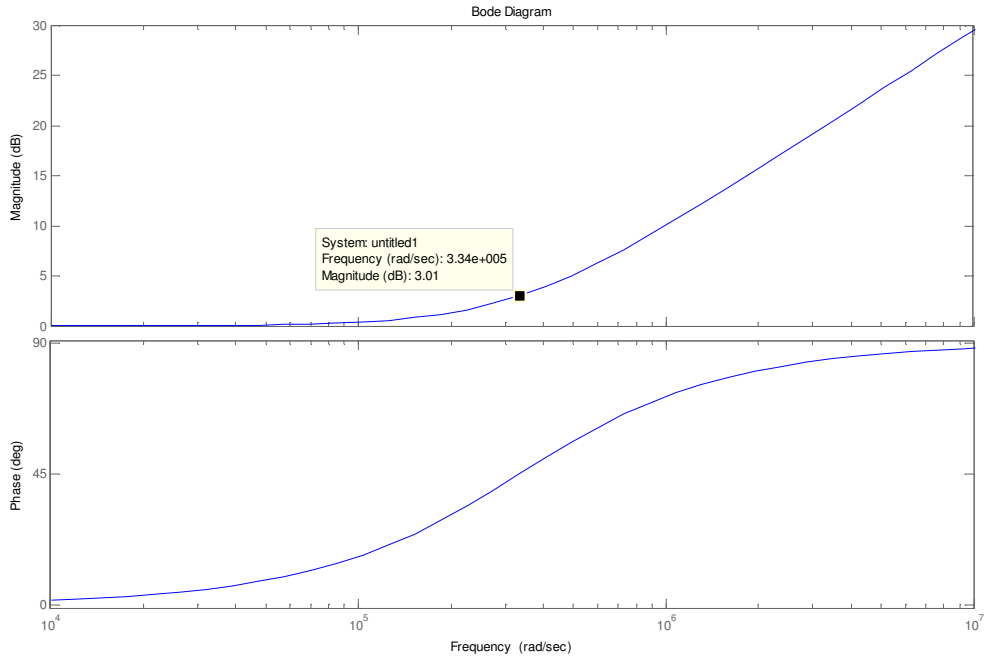


Figure 4-12 - Bode plot of gain transfer function

4.2.3 Design solution

Of the previously described techniques, only the digital potentiometer approach is able to provide a satisfactory result with respect to low power. It is preferable that the VGA has linear and known gain. This will enable the traditional automatic gain controller to set the correct gain with only one measurement. The traditional volume control circuit described in section 4.2.2.1 is therefore chosen as the base point for the design. To achieve a good gain resolution the system will use three digital potentiometers. This will provide $256^3 = 16.7 \cdot 10^6$ steps..

The main disadvantage of the circuit is that the amplifier it self will always be at full gain where the signal is being damped before being amplified. At small signal levels this can lead to a problem where noise generated inside or after the digital potentiometer is being amplified to an extent where the original signal cannot be retrieved. This can be internally generated noise, SPI crosstalk and noise due to change of gain setting. By introducing a filter between the digital potentiometer and the amplifier noise outside the spectra for interest is damped.

Another challenge is DC drift. As specified in section 4.1 the amplifier should have a dynamic range of minimum 103dB. This gives us a maximum gain of

$$G_{\max} = 10^{\frac{103}{20}} = 141254 \tag{4.6}$$

With such a large gain even the smallest DC offset can make the system saturate and thereby fail. Introducing a high pass filter will remove any DC offset. This will be discussed further in section 4.3 *Filter design*. To achieve this gain several operational amplifiers will be used in a cascade configuration, please refer to section 4.5.1 *Operational amplifier* on page 43 for details. The following section will suggest specific circuit solutions for the filter combined with the amplifier.

4.3 Filter design

To avoid aliasing when undersampling as described in section 2.1 on page 4 it is vital that the signal is limited in bandwidth. This requires a band pass filter that will limit the bandwidth of the signal. The following section describes a filter that removes DC offset and noise generated in the potentiometer.

4.3.1 Potentiometer noise removal filter

To remove DC offset, noise generated in the potentiometer and noise related to changing the setting of the potentiometer a simple first order filter is implemented as depicted in Figure 4-13. To minimize the power consumption we wish to only add passive components to this filter. Due to the high tolerances of coils it is desired to avoid using these when possible. The filter design is therefore concentrated around first order RC filters.

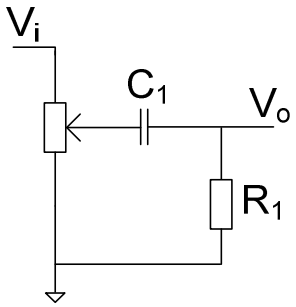


Figure 4-13 - Digital potentiometer with first order high pass filter

When choosing filter component values it is important to consider the frequency dependent operation of the digital potentiometer. In this application the wiper resistance and capacitance must be taken into account. Figure 4-35 shows the more detailed circuit where R_H and R_L are dependent on the wiper position, R_W is the wiper resistance and C_W is the wiper capacitance.

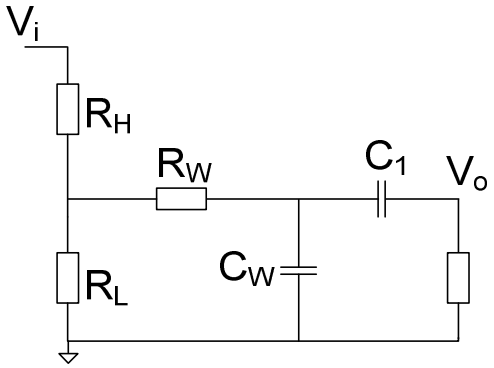


Figure 4-14 - Non-ideal potentiometer with first order high pass filter

To fully understand the behaviour of the circuit the transfer function $H_f(s) = \frac{V_o}{V_i}$ must be derived.

To analyse the circuit several impedances must be calculated. Figure 4-15 shows the different impedances.

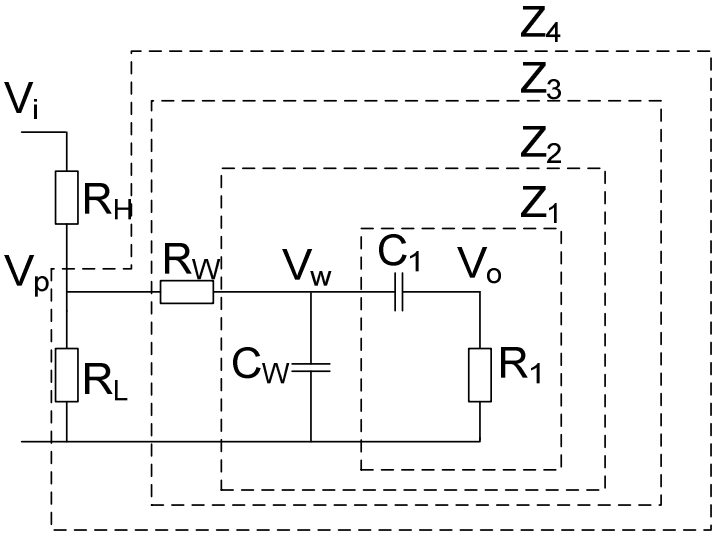


Figure 4-15 - Impedances of the potentiometer and filter circuit

The impedances are derived in an ascending order.

$$Z_1 = \frac{1}{sC_1} + R_1 = \frac{sR_1C_1 + 1}{sC_1} \tag{4.7}$$

$$Z_2 = X_{C_W} \parallel Z_1 = \frac{\frac{1}{sC_W} Z_1}{\frac{1}{sC_W} + Z_1} = \frac{R_1C_1s + 1}{s(C_1 + C_W + R_1sC_WC_1)} \tag{4.8}$$

$$Z_3 = R_W + Z_2 = R_W + \frac{R_1C_1s + 1}{s(C_1 + C_W + R_1sC_WC_1)} \tag{4.9}$$

$$Z_4 = \frac{R_L Z_3}{R_L + Z_3} = \frac{R_L\left(R_W + \frac{R_1C_1s + 1}{s(C_1 + C_W + R_1sC_WC_1)}\right)}{R_L + R_W + \frac{R_1C_1s + 1}{s(C_1 + C_W + R_1sC_WC_1)}} \tag{4.10}$$

Using these expressions the transfer functions can be derived using simple voltage divide calculations. All voltages are referred to Figure 4-15.

$$\frac{V_p}{V_i} = \frac{Z_4}{Z_4 + R_H} \tag{4.11}$$

$$\frac{V_W}{V_p} = \frac{Z_2}{R_W + Z_2} \tag{4.12}$$

$$\frac{V_o}{V_W} = \frac{R_1}{R_1 + \frac{1}{sC_1}} \tag{4.13}$$

$$\frac{V_o}{V_i} = \frac{V_p}{V_i} \cdot \frac{V_W}{V_p} \cdot \frac{V_o}{V_W} = \frac{Z_4}{Z_4 + R_H} \cdot \frac{Z_2}{R_W + Z_2} \cdot \frac{R_1}{R_1 + \frac{1}{sC_1}} \tag{4.14}$$

Inserting equations (4.7), (4.8), (4.9) and (4.10) into (4.14) gives the final transfer function

$$H(s)=\frac{V_o}{V_i}=\frac{R_L R_1 C_1 s}{C_W C_1 R_1\left(\left(R_L+R_W\right)R_H+R_L R_W\right)s^2+\left(\left(\left(C_1+C_W\right)R_L+\left(R_W+R_1\right)C_1+R_W C_W\right)R_H+\left(\left(R_W+R_1\right)C_1+R_W C_W\right)R_L\right)s+\left(R_L+R_H\right)} \quad (4.15)$$

The frequency response of (4.15) is plotted in Figure 4-16. The end-end resistance, wiper resistance and wiper capacitance are all typical numbers from the datasheet for the MAX5401EKA. R_1 and C_1 are chosen to provide a corner frequency of 60 kHz for the RC connection. The values are as follows

- R_1 = 100 kΩ
- C_1 = 26.53 pF
- $R_H = R_L$ = 50 kΩ
- C_W = 25 pF
- R_W = 250 Ω

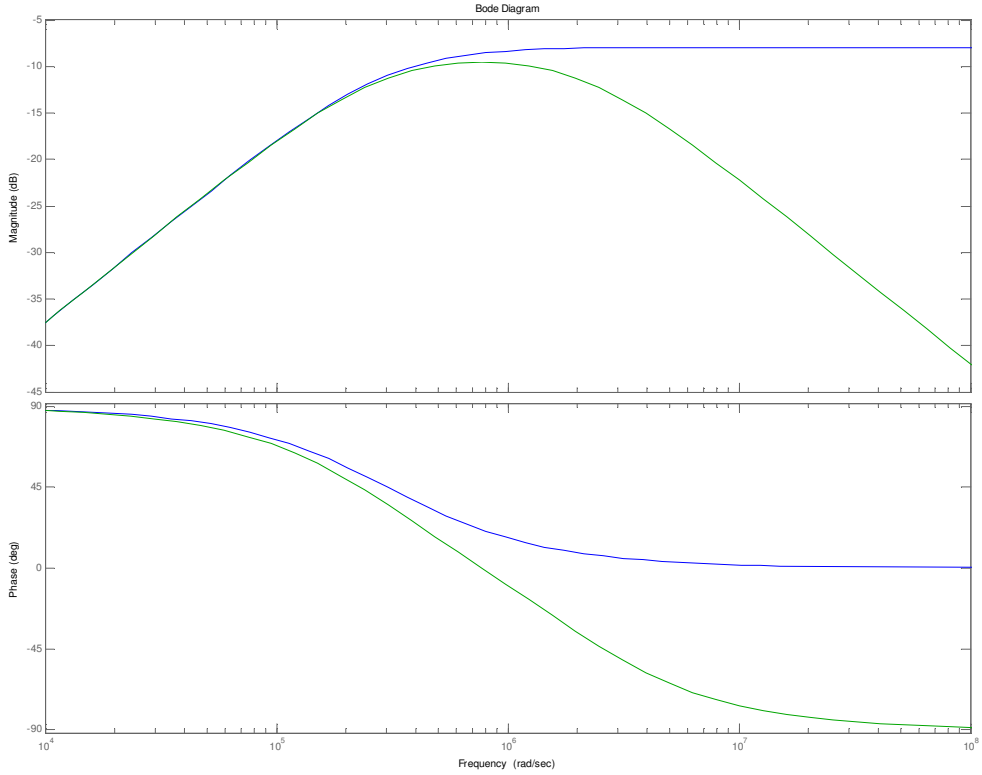


Figure 4-16 - Bode response of both ideal and non-ideal potentiometer and filter circuit

The blue curve represents the frequency response with an ideal potentiometer. The green curve represents the frequency response with a non-ideal potentiometer. Notice that the response is only affected in the upper frequency area. For the given component values the non-ideal behaviour adds an additional pole at f_2 = 329 kHz. The other pole lies at f_1 = 46 kHz. It is important to notice that the poles will vary with potentiometer position.

Using MATLAB the cut-off frequencies or roots are plotted in Figure 4-17. The maximum value is 255 due to the 8-bit resolution. The high frequency pole at f_2 will not affect the frequency band of interest and is not discussed further. The low frequency pole f_1 varies with

about 15 kHz. This may represent a problem and components should be chosen so that this variation is kept at a minimum. The variation will increase with decreasing value of R_1 and consequently increasing value of C_1. Increasing R_1 to 500 kΩ and C_1= 5.3 pF results in a variation of 3.6 kHz as shown in Figure 4-18. The upper cross frequency is now decreased to 268 kHz, but still out of the frequency band of interest and can therefore be ignored.

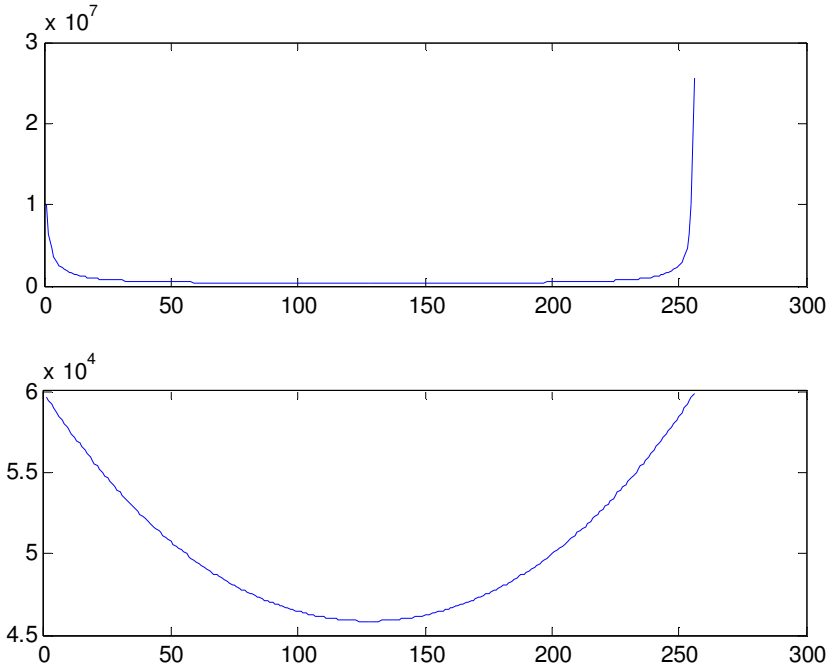


Figure 4-17 - Root frequencies in Hz as function of potentiometer position with R_1 = 100kΩ, C_1 = 26.5pF

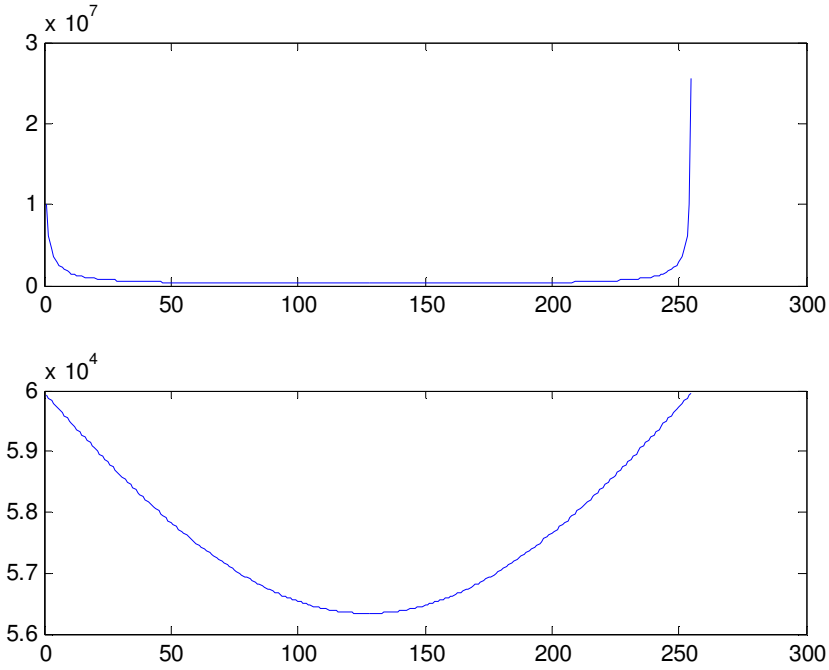


Figure 4-18 - Root frequencies in Hz as function of potentiometer position with R_1 = 500kΩ, C_1 = 5.3pF

4.3.2 The intuitive filter solution

The intuitive filter solution is based on cascading several equal filters to gain a steep roll off. To limit the bandwidth of the signal at the high end, all operational amplifier stages are designed as a first order non-inverting active low pass filter with a gain of 25. The circuit diagram is shown in Figure 4-19.

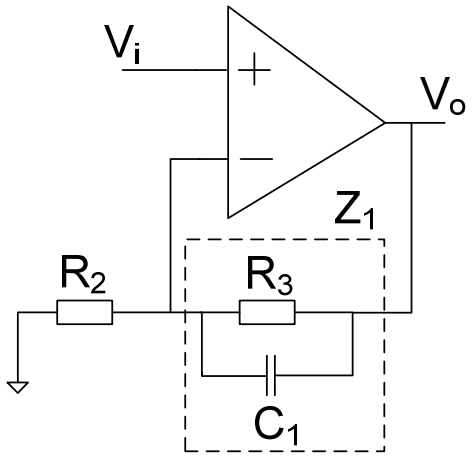


Figure 4-19 - Active low pass filter circuit diagram

An expression for Z_1 is needed to derive an expression for the gain

$$Z_1 = R_3 \,||\, X_{C_2} = \frac{R_3 \cdot \frac{1}{sC_2}}{R_3 + \frac{1}{sC_2}} = \frac{R_3}{R_3 C_2 s + 1} \tag{4.16}$$

The expression for the total gain is derived.

$$G(s) = 1 + \frac{Z_1}{R_2} 1 + \frac{\frac{R_3}{R_3 C_2 s + 1}}{R_2} = 1 + \frac{R_3}{R_2 R_3 C_2 s + R_2} \tag{4.17}$$

As with the high pass filter it is desired to use large resistors to minimize the power dissipation. The crossover frequency should be 80 kHz. The resistor ratio is equal to the fixed pass band gain

$$\begin{aligned} G_{passband} &= 1 + \frac{R_3}{R_2} = 25 \\ \Rightarrow R_3 &= 24 \cdot R_2 \end{aligned} \tag{4.18}$$

Capacitor values are available in fewer values than resistors. A standard capacitor value of 4.7pF is therefore chosen for C_2. The expression for the crossover frequency is used to calculate the resistor values. The constant gain of one is neglected. The absolute value is found

$$|G(s)| \approx \left| \frac{R_3}{R_2 R_3 C_2 s + R_2} \right| = \frac{R_3}{\sqrt{(R_2 R_3 C_2 \omega_0)^2 + R_2^{\,2}}} \tag{4.19}$$

The crossover frequency is found when |G(s)| equals the pass band gain divided by the square root of two.

$$|G(s)| = \frac{1}{\sqrt{2}} \cdot 24 \tag{4.20}$$

By combining (4.19) and (4.20) the expression for the crossover frequency is found.

$$\begin{aligned} &\frac{R_3}{\sqrt{(R_2 R_3 C_2 \omega_0)^2 + R_2^{\,2}}} = \frac{1}{\sqrt{2}} \cdot 24 \\ &R_3^{\,2} = 288\left[(R_2 R_3 C_2 \omega_0)^2 + R_2^{\,2}\right] \end{aligned} \tag{4.21}$$

(4.18) is inserted to (4.21)

$$\begin{aligned} &(24R_2)^2 = 288R_2^{\,2}\left[(R_3 C_2 \omega_0)^2 + 1\right] \\ &576 = 288\left[(R_3 C_2 \omega_0)^2 + 1\right] \\ &288 = 288(R_3 C_2 \omega_0)^2 \\ &1 = R_3 C_2 \omega_0 \\ &\Rightarrow R_3 = \frac{1}{C_2 \omega_0} \end{aligned} \tag{4.22}$$

By inserting C_2 = 4.7pF and using (4.18) the resistor values are found.

$$\begin{aligned} &R_3 = \frac{1}{2\pi f C_2} = \frac{1}{2\pi \cdot 80 \cdot 10^3 \cdot 4.7 \cdot 10^{-12}} = 423.3k\Omega \\ &R_2 = \frac{1}{24} \cdot R_2 = 17637\Omega \end{aligned} \tag{4.23}$$

Figure 4-20 shows the frequency response of the filters and the combined circuit where four Low pass filters with gain, three potentiometers and four high pass filters as described in section 4.3.1 are cascaded. The high pass filter from section 4.3.1 is used for both noise removal as described earlier and bandwidth limitation of the signal. The values used are:

R1 = 473.68kΩ
R2 = 17637Ω
R3 = 423.3kΩ
C1 = 5.6pF
C2 = 4.7pF

Where several connections of the same type are used they are identical with respect to both connection diagram and component values. The frequency response is simulated using a Matlab script found in appendix 2.

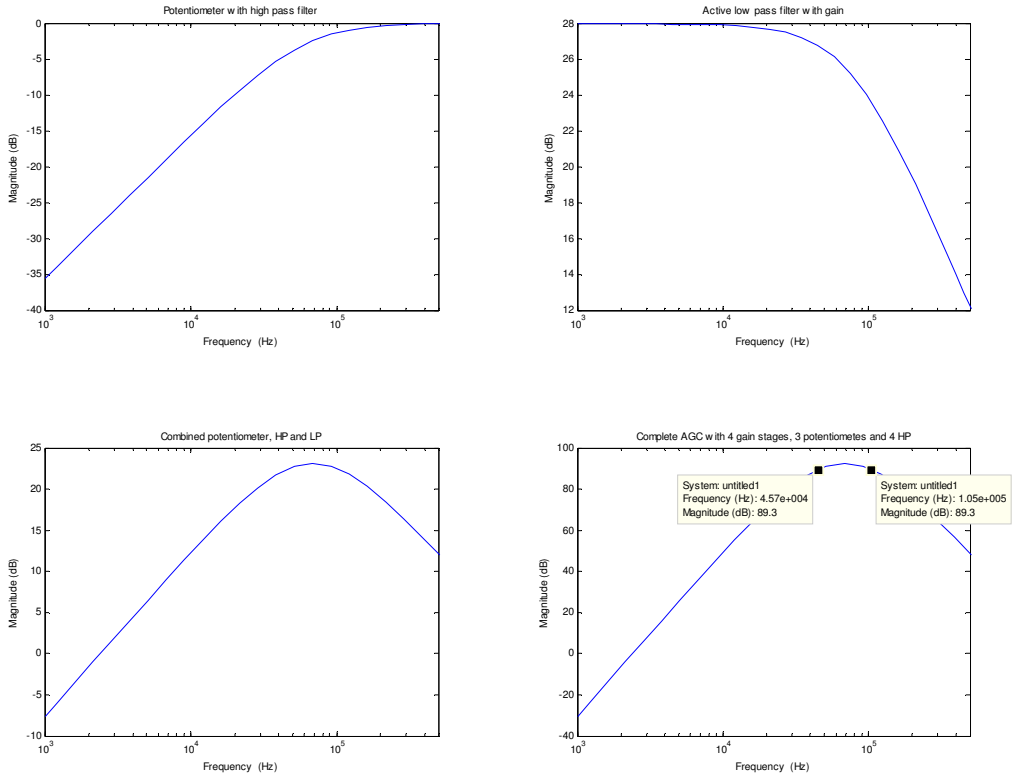


Figure 4-20 - Bode plot of filters and combined circuit

The complete circuit provides a pass band gain of 92dB and a bandwidth of 60 kHz. The requirement is 103 dB gain with a 20 kHz bandwidth. Obviously this configuration is not suitable. A filter with a steeper roll off is needed. The following sections will describe a more complex filter design with better performance.

4.3.3 Butterworth filter

The Butterworth filter is designed to have a frequency response which is as flat as mathematically possible in the passband. Its poles are arranged as evenly spread complex conjugated poles in a circle. The radius of the circle determines the crossover frequency. An example of a 4^{th} order Butterworth filter pole placement is shown in Figure 4-21. The placement of the zero points determines whether it is a high or low pass filter.

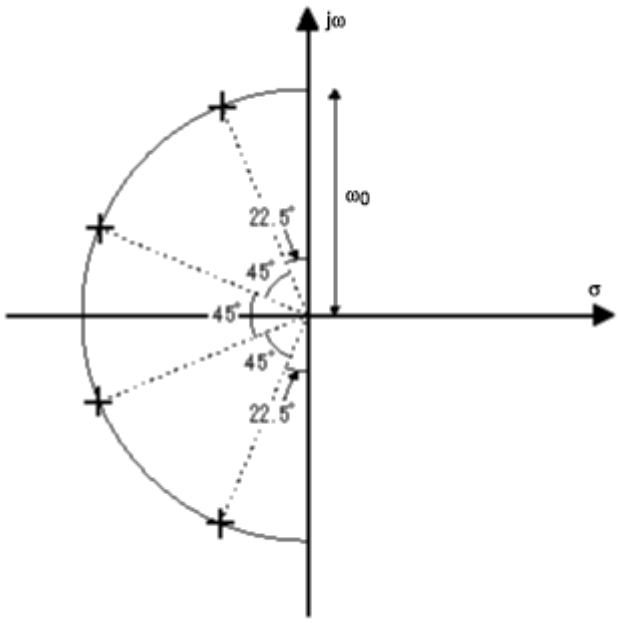


Figure 4-21 - Butterworth filter pole arrangement

The general transfer function for a Butterworth low pass filter is [14]

$$T(s) = \frac{K \cdot \omega_0^N}{(s-p_1)(s-p_2)\cdots(s-p_N)} \tag{4.24}$$

Where
K is a constant
N is the order of the filter
p is the complex pole
ω_0 *is the crossover frequency*

A general second order low pass filter is given [14] by

$$T(s) = \frac{a}{s^2 + s\frac{\omega_0}{Q} + \omega_0^2} \tag{4.25}$$

The poles are described as the complex position in Figure 4-21. By comparing the denominators of (4.24) and (4.25). For a second order filter we find that

$$\begin{aligned} (s-p_1)(s-p_2) &= s^2 + s\frac{\omega_0}{Q} + \omega_0^2 \\ s^2 - s(p_1+p_2) + p_1 \cdot p_2 &= s^2 + s\frac{\omega_0}{Q} + \omega_0^2 \end{aligned} \tag{4.26}$$

Since p_1 and p_2 are complex conjugated and can be written as

$$\begin{cases} p_1 = \omega_0 \angle \alpha \\ p_2 = \omega_0 \angle -\alpha \end{cases} \tag{4.27}$$

The product yields

$$p_1 \cdot p_2 = \omega_0 \cdot \omega_0 \angle(\alpha - \alpha) = \omega_0^2 \tag{4.28}$$

Therefore (4.26) reduces to

$$-(p_1 + p_2) = \frac{\omega_0}{Q}$$
$$\Rightarrow Q = -\frac{\omega_0}{p_1 + p_2} = -\frac{\omega_0}{2 \cdot real(p)} \tag{4.29}$$

Where real(p) is the common real part of the complex poles. Notice that the closer the poles are to the imaginary axis the higher Q value. Poles closer to the imaginary axis results in a less stable filter and hence a high Q value filter is more likely to oscillate than a filter with low Q value.

For analogue implementation of the filter, the Sallen and Key configuration is chosen as a demonstration and offers a second order filter using only one operational amplifier. The low pass circuit is shown in Figure 4-22. A high pass circuit is gained by swapping the capacitors with resistors and vice versa.

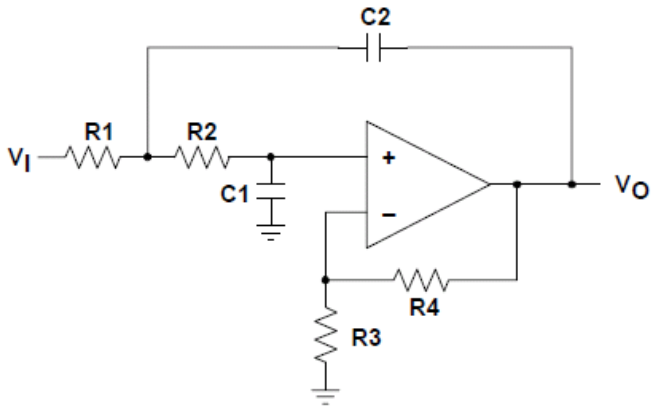


Figure 4-22 - Sallen and key configuration

The transfer function, derived by Texas Instruments in application note *Analysis of the Sallen-key Architecture* [15], is given in equation (4.30).

$$H(s) = \frac{V_o}{V_i} = \frac{K}{s^2\left(R_1R_2C_1C_2\right) + s\left(R_1C_1 + R_2C_1 + R_1C_2\left(1-K\right)\right) + 1} \tag{4.30}$$

By comparing (4.30) with the general equation (4.25) we find that

$$R_1R_2C_1C_2 = 1 \tag{4.31}$$
$$R_1C_1 + R_2C_1 + R_1C_2\left(1-K\right) = \frac{\omega_0}{Q} \tag{4.32}$$
$$\omega_0^2 = 1 \tag{4.33}$$

Note that for the general Sallen and Key expression the crossover frequency is 1 [rad/s]. The component values will first be calculated for this frequency and then scaled to fit the desired crossover frequency.

Since the resistors are available in more values than capacitors the capacitor values are chosen as a base point. The capacitors are chosen to be equal.

$$C_1 = C_2 = C \tag{4.34}$$

Solving (4.31), (4.32) and (4.33) with the insertion method gives the following solution

$$R_2 = \frac{1}{2} \cdot \frac{\omega_0 C_1 - \sqrt{\omega_0^2 C_1^2 - 4Q^2 C_2 C_1 k - 4C_1^2 Q^2 + 4C_1^2 Q^2 k}}{C_1 Q (C_2 + C_1)} \tag{4.35}$$

$$R1 = \frac{1}{R_2 C_1 C_2} \tag{4.36}$$

For this design it is desired to have a fourth order filter low pass filter and a fourth order high pass filter to get the proper damping of frequencies outside the pass band. A fourth order low pass filter filter is realized by calculating the poles for a fourth order Butterworth filter and using two Sallen and Key configurations with different Q values/different pole sets in cascade. A Matlab script found in appendix 1 was used to calculate the component values and simulate the frequency response of the circuit.

The frequency response of the complete Sallen and Key implemented Butterworth filter and the filter designed in section 4.3.2 is shown in Figure 4-23. The component values for the Butterworth filter are calculated by finding the four poles for each filter and using two values for (4.29), and calculating (4.34), (4.35) and (4.36) for each Q value or complex conjugated pole set.

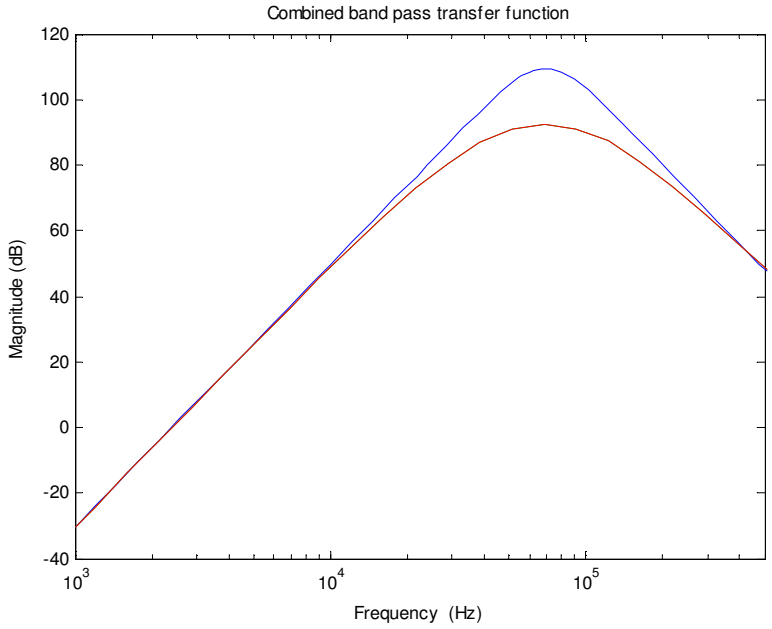


Figure 4-23 - Frequency response of Butterworth and intuitive filter

The Butterworth is clearly an improvement with respect to gain and roll off. A steeper roll off is still desired. The next section will therefore describe the Chebyshev filter.

4.3.4 Chebyshev filter

The Chebyshev filter is known for its steep roll off [14]. The cost of the steeper roll off is ripple in the passband. For many applications such as audio tone control this is not acceptable due to noticeable distortion, but for the telemetry buoy design the signal consist of digital pulses and a slight variation between the different frequency bands is tolerable as long as the receive threshold is set accordingly.

The steeper roll off is achieved by placing the poles closer to the imaginary axis as shown in Figure 4-24.

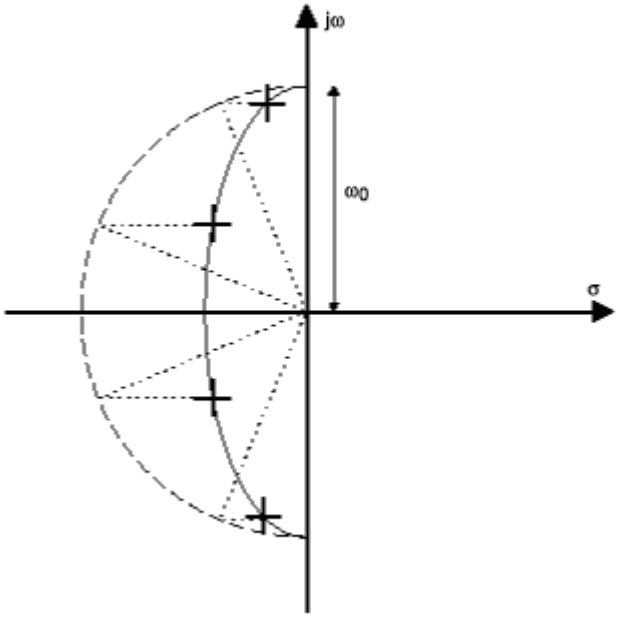


Figure 4-24 - Chebyshev filter pole placement

The general transfer function for a Chebyshev filter is [14]

$$T(s) = \frac{K \cdot \omega_P^N}{\varepsilon \cdot 2^{N-1}(s-p_1)(s-p_2)\cdots(s-p_N)} \tag{4.37}$$

Where
K is a constant
N is the order of the filter
p is the complex pole
ω_0 *is the crossover frequency*
and

$$\begin{aligned} |T(j\omega_P)| &= \frac{1}{\sqrt{1+\varepsilon^2}} \\ \Rightarrow \varepsilon &= \sqrt{\frac{1}{|T(j\omega_P)|^2} - 1} \end{aligned} \tag{4.38}$$

In (4.38) the $|T(j\omega_P)|$ is the maximum ripple in the passband which is defined at the design stage. For this design the allowed ripple is set to 1 dB. Note that it follows that the crossover frequency ω_P will be defined by -1 dB and not the usual -3dB. Due to the steep roll off this detail is not considered in the calculations.

The same script as for the Butterworth filter in appendix 1 is used for simulation and the same equations for calculating resistor values (4.34), (4.35) and (4.36) are therefore used for both the Butterworth and the Chebyshev filter only scaling the gain factor and changing the pole placement. Figure 4-25 shows the response of the intuitive, Butterworth and Chebyshev filter. It is clear that the Chebyshev filter offers the steepest roll off and the narrowest bandwidth.

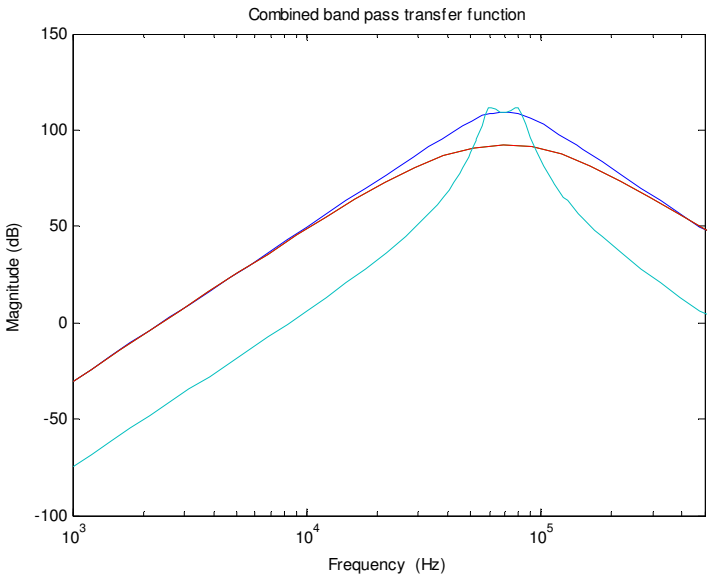


Figure 4-25 - Frequency response of Chebyshev, Butterworth and the intuitive filter

To provide low power consumption it is desired to use as few active components as possible as well as high resistor values to reduce the power dissipation. The Sallen and Key configurations uses one operational amplifier and is versatile with respect to component values**.** The example implementation is therefore based on this implementation.

As described earlier, due to high amplification it is important to remove any DC drift and low pass noise due to changes in the potentiometer setting. A high pass filter as described in section 4.3.1 is therefore inserted after every digital potentiometer.

To avoid that this filter and the potentiometer has substantial impact on the Chebshev filter a unity gain amplifier is placed after each digital potentiometer stage. It is important to notice that in order to use high resistor values the capacitors much have small values. The unity gain amplifier will ensure that the wiper capacitance, described in section 4.2.2.5, of the digital potentiometer does not have impact on the Sallen and Key configuration. The complete configuration is depicted in Figure 4-26.

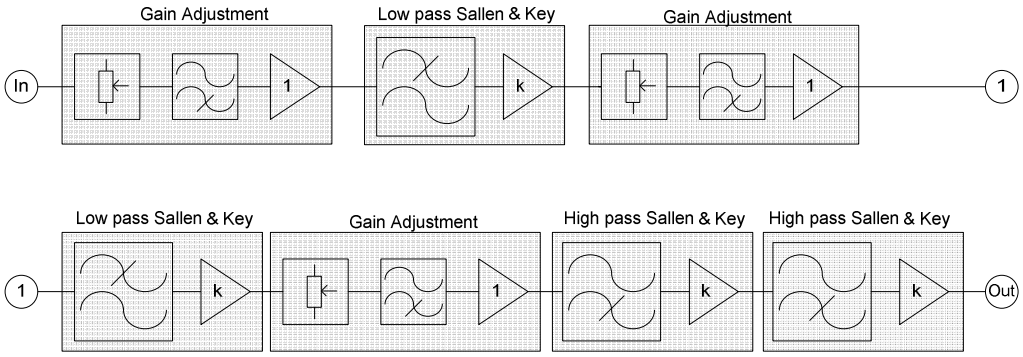


Figure 4-26 - Variable gain amplifier and filter block diagram

The *Gain Adjustment* block is realised as in Figure 4-27. To ensure that the high pass filter does not interfere with the Chebyshev filter the crossover frequency is selected to be as low as 10 kHz. Selecting a capacitor value of C_2 = 150pF yields a resistor value of

$$R_2 = \frac{1}{\omega_0 C} = \frac{1}{2\pi 10 \cdot 10^3 \cdot 150 \cdot 10^{-12}} = 106.1k\Omega \tag{4.39}$$

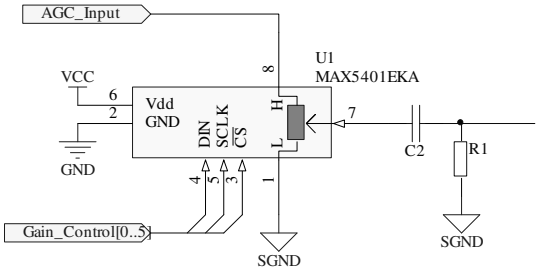


Figure 4-27 - Gain Adjustment block realisation

The *low and high pass Sallen and Key filters* are realised as previously described in section 4.3.3 on page 25. The component values are calculated with the Matlab script provided in appendix 1. Note that the component references in the script and the schematics may deviate due to automatic annotation with the design program.

The frequency response for the complete circuit in Figure 4-26 is shown in Figure 4-28. The green line shows the complete circuit at full amplification, the blue shows the original Chebyshev filter with amplification. Notice that the insertion of digital potentiometers does not affect the frequency band of interest.

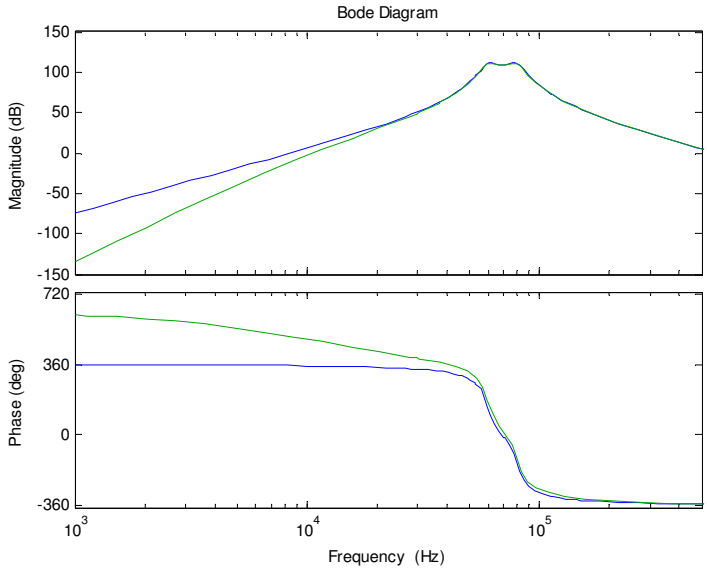


Figure 4-28 - Frequency response of variable gain circuit with Chebyshev filtration, potentiometers at max amplification

With the intuitive filter solution in section 4.3.2 the setting of the digital potentiometers affected the crossover frequencies. Figure 4-29 shows the frequency response with different gain settings. It is easy to see that the impact on the crossover frequencies is minimal. The Matlab script for the simulations is provided in appendix 4.

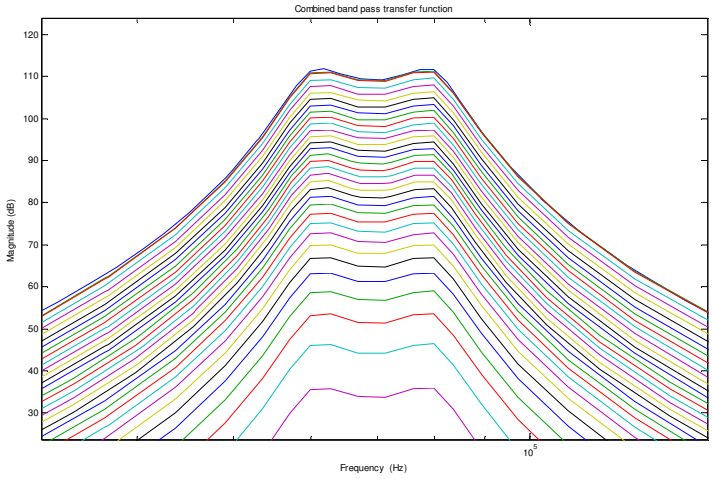


Figure 4-29 - Frequency response with different gain settings

An important factor is the linearity of the gain. Figure 4-30 shows the gain plotted as a function of the potentiometer setting of one of the potentiometers. As a result of the internal wiper capacitance of the potentiometer the gain is not perfectly linear. The result is though quite close and is regarded as satisfactory.

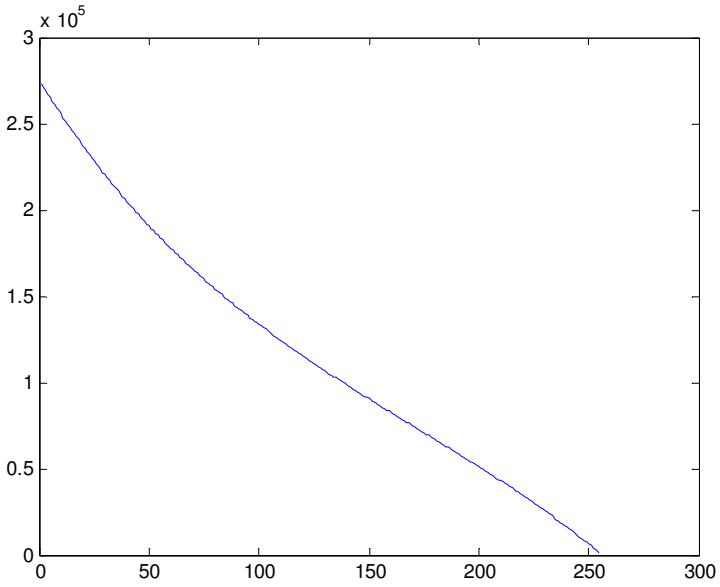


Figure 4-30 - Gain vs. digital potentiometer setting

4.3.5 Design solution

As described in the previous section the Chebyshev filter provides the best solution with respect to roll off. There is though one major drawback with the Chebyshev solution when designing a filter for low power. The Chebyshev filter requires high Q values which requires an operational amplifier with high performance when applying high frequencies.

Faulkner and Grimbleby informs in the IEEE article *Active filters and gain-bandwidth product*[16] that it has become a more or less accepted practice to concentrate attention on the question of sensitivity to variations in passive components rather than to variations of amplifier gain. The implication is that the amplifier gain can be assumed to be arbitrarily high. When, on the other hand, the need arises for an engineering design to meet a given practical specification, it usually becomes clear that the primary problem is the relation between circuit configuration and required amplifier performance in terms not of DC gain, but of gain-bandwidth product.

It is important to notice that the effect of finite gain-bandwidth product on filter performance cannot be calculated by assuming the gain to be real. Neglecting the phase angle leads, for instance, to the conclusion that the Q factor is always reduced as the frequency is increased whereas in fact for many configurations the Q factor increases

Faulkner and Grimbleby provide two important equations for calculating the actual crossover frequency for the Sallen-Key and the Rauch configurations:

$$\begin{aligned} \omega_0 &= \omega_L\left(1-\frac{1}{2}B\cdot Q_L\left(\frac{\omega_L}{\omega_T}\right)\right) \\ \frac{1}{Q} &= \frac{1}{Q_L}\left(1-\frac{1}{2}B\cdot Q_L\left(\frac{\omega_L}{\omega_T}\right)+B\cdot Q_L^2\left(\frac{\omega_L}{\omega_T}\right)^2\right) \end{aligned} \tag{4.40}$$

Where

ω_0 *is the actual crossover frequency*

Q *is the actual Q factor*

ω_L *is the crossover frequency with an ideal operational amplifier*

Q_L *is the Q factor with an ideal operational amplifier*

ω_T *is the Gain-Bandwidth product of the operational amplifier*

B *is the gain of the amplifier i.e. B = 1 + R1/R2 for non-inverting configuration*

The power consumption for an operational amplifier (opamp) increases with higher gain-bandwidth product. This means that for low power applications (4.40) must always be considered in order to design a circuit which is optimal with respect to power consumption and filter characteristics.

We see that in order to use an opamp with low gain-bandwidth product we need to compensate by using low Q values and low gain. The following procedure is used to calculate the required gain-bandwidth product of the opamp:

1. Define a maximum allowable deviation in crossover frequency due to the finite gain-bandwidth product.
2. Define a filter gain
3. Calculate the Q values of the filter, for Butterworth: use (4.29)
4. Use (4.40) to find the minimum ω_T

If the calculated ω_T results in an opamp with too high power consumption the Q value or the gain will have to be lowered.

The calculation procedure is performed on the low-pass Chebyshev filter calculated in section 4.3.4:

1. The maximum allowable deviation is defined as 300Hz
2. Gain is set to 7.
3. Q values are calculated with the Matlab script in appendix 1 and are: Q1 = 8.95, Q2 = 3.71.
4. (4.40) is rewritten to

$$\omega_T = \frac{1}{2}\frac{B \cdot Q_L \cdot \omega_L}{1-\frac{\omega_0}{\omega_L}} \tag{4.41}$$

By defining

$$\omega_0 = \omega_L - \delta \tag{4.42}$$

Where
δ *is the deviation in frequency.*

The gain bandwidth products become

$$\begin{aligned}\omega_{T,1} &= \frac{1}{2}\cdot\frac{B \cdot Q_L \cdot \omega_L}{1-\frac{\omega_L+\delta}{\omega_L}} = \frac{1}{2}\cdot\frac{7 \cdot 8.95 \cdot 80 \cdot 10^3}{1-\frac{80 \cdot 10^3 - 300}{80 \cdot 10^3}} = 668Mhz \\ \omega_{T,2} &= \frac{1}{2}\cdot\frac{B \cdot Q_L \cdot \omega_L}{1-\frac{\omega_L-\delta}{\omega_L}} = \frac{1}{2}\cdot\frac{7 \cdot 3.71 \cdot 80 \cdot 10^3}{1-\frac{80 \cdot 10^3 - 300}{80 \cdot 10^3}} = 277Mhz\end{aligned} \tag{4.43}$$

Operational amplifiers with a gain-bandwidth product as high as (4.43) draws currents of 5-10 mA. This is a too high power consumption for the telemetry buoy application. The filter performance must be reduced to lower the power consumption.

As a compromise the design will use a Butterworth configuration which has much lower Q values. In addition we see that amplification in the filter comes at a much greater cost than amplification outside the filter. The realisation of the filter will therefore have unity gain.

The design of an analogue filter can be made more effective by using available software tools. Texas instruments provide a filter design program *FilterPro* for free. The program suggests standard component values for circuit types such as Sallen and Key, MFB Single-Ended and MFB fully differential. The user can specify crossover frequencies, order of filter and choose from filter types such as Butterworth, Chebyshev, Bessel, Linear Phase and more. In addition to providing the schematic and the component values the program also specifies the required gain-bandwidth product for the opamps. It is not documented how this is done, but tests indicate that the same procedure is used with 0.3% deviation. The user interface is shown in Figure 4-31.

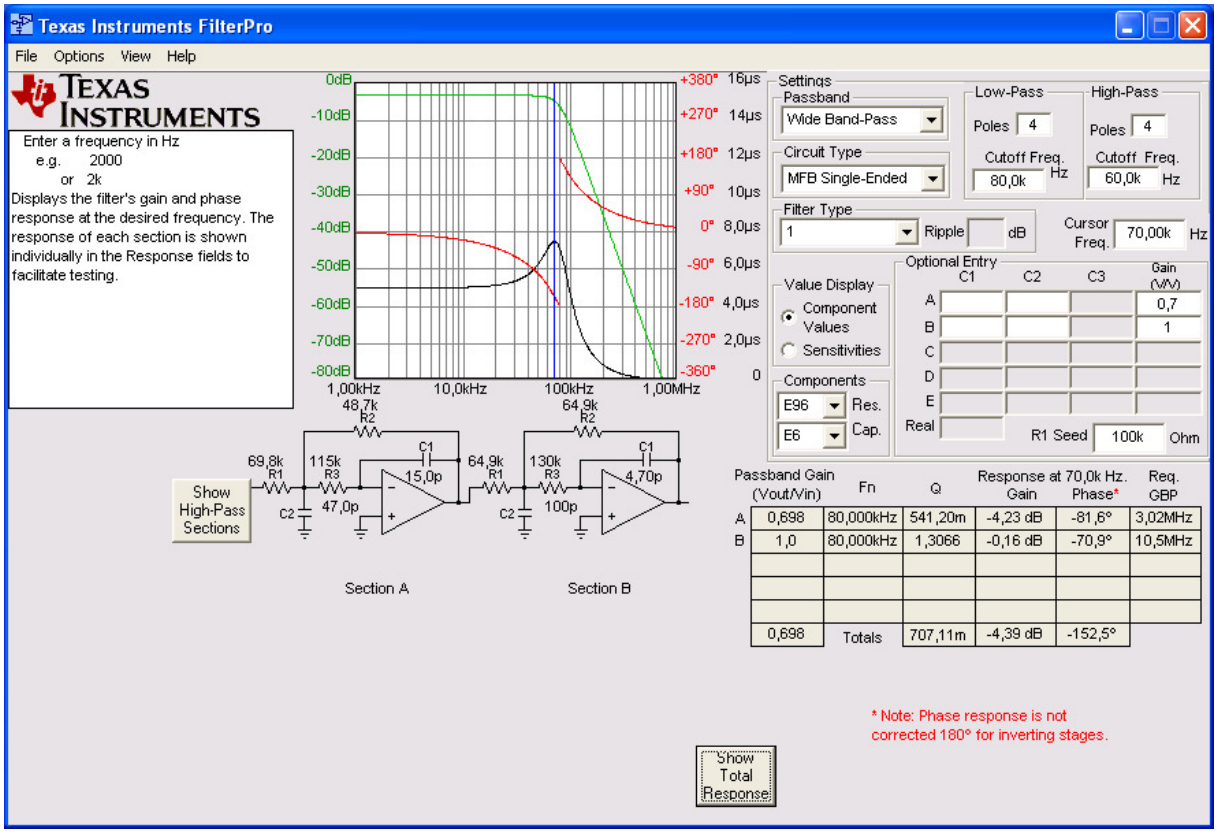


Figure 4-31 - FilterPro user interface

The *multiple feedback* (MFB) configuration provides a second order filter with only one operational amplifier as with the Sallen and Key configuration. The difference is that the MFB requires less gain-bandwidth of the opamp [17]. The implementation will therefore be based on the MFB configuration. The configuration of a second order low pass filter is shown in Figure 4-32. The high pass filter configuration is found by changing all resistors with capacitors and vice versa. The procedure for calculating the component vales is the same as for the Sallen and Key configuration described earlier only using a different transfer function

equation for the implementation. This time we will though use the *FilterPro* program to find the component values. Screenshots of the design is found in appendix 4.

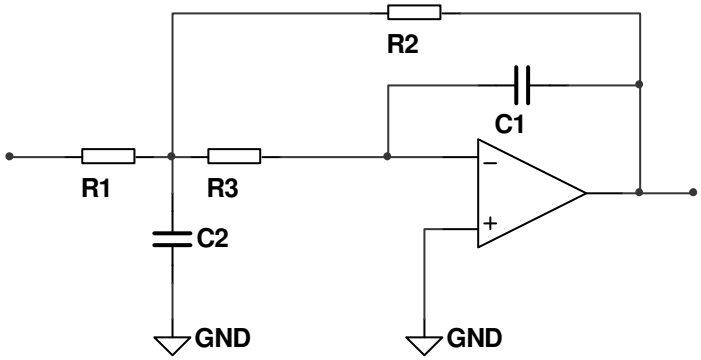


Figure 4-32 - Multiple feedback configuration

The configuration of the complete variable gain amplifier with filter will be similar to the one presented in section 4.3.4 and is shown in Figure 4-33. The main difference is the filter configuration and since the filter gain is unity, the gain is moved to operational amplifiers outside the filter. All gain blocks have a high pass filter in front. This has two main functions; it will remove much of the low frequency noise emitted by the digital potentiometer when changing the setting, secondly it will remove any DC offset which can lead to saturation of the opamp. Note that some of the filter blocks have gain not equal to unity. This is to be able to utilize operational amplifiers with a gain-bandwidth product of 3 MHz and 12 MHz. The total gain of the filter is though unity. The filter will use two amplifiers with 3 MHz gain-bandwidth product and two amplifiers with 12 MHz gain-bandwidth product.

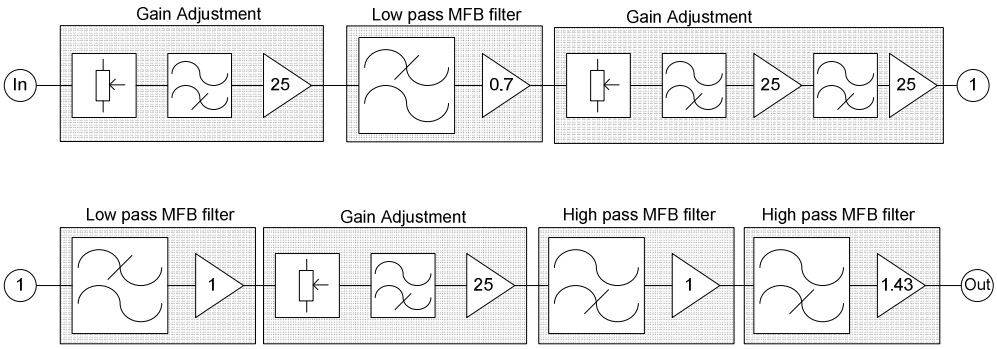


Figure 4-33 - Filter and variable gain amplifier block diagram

The complete maximum gain of the ideal circuit is

$$G_{max,dB} = 20\log(25^4) = 111.8dB \tag{4.44}$$

Please refer to the schematics in appendix 5b for a detailed circuit diagram with component values.

Simulating the complete circuit in Multisim 10.1 provides the bode plot given in Figure 4-34.

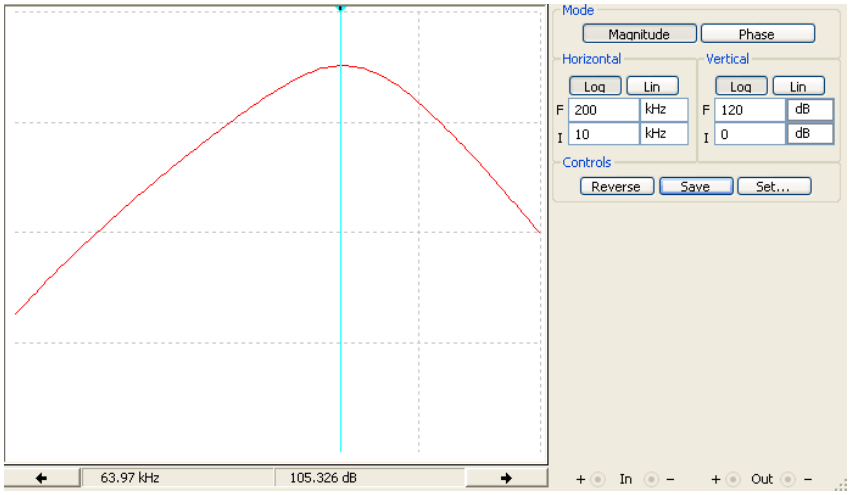


Figure 4-34 - Simulated bode plot of variable gain amplifier and filter

4.4 Reception detector

There are two principles for detecting a modulated signal pulse; measuring the envelope of the carrier frequency or measure the power in the carrier frequency band. This section describes one technique for measuring the power of a signal, two techniques for detecting the envelope of the signal and finally a modification which results in a third way of measuring the power.

4.4.1 Detecting a signal using an RMS to DC converter

In high frequency amplitude-shift-keying (ASK) radio reception applications a logarithmic amplifier (logamp) is often used to demodulate the signal by producing a DC signal proportional to the logarithm of the power of the input signal [5]. These circuits typically operate with frequencies up to several GHz consuming currents of 4 – 68mA (Analog Devices).

High frequency is not relevant to the telemetry buoy application, but a similar technique can be utilised using a RMS to DC converter. This IC produces a DC output signal directly proportional to the root-mean-square of the input signal. These devices are typically used in applications such as true RMS multimeters. An example of such a device is the AD737 which consumes a current of only 160µA handling frequencies up to 100 kHz [6]. The downside of using such a converter is that it requires both positive and negative power supply voltage. This will require an additional regulator to the system which will contribute to an increase in overall power consumption.

4.4.2 Detecting a signal using a product detector

The product detector produces an output signal by calculating the product of the input signal and a local oscillator. Alternatively the input signal can be multiplied with its inverse signal as for the *intelligent hydrophone buoy* designed by Jan Eyolf Bjørnson [7]. Assuming that the input signal is a sinusoidal as in equation (4.45)

$$V_{in} = A\sin(\omega t) \tag{4.45}$$

The output signal W becomes

$$W = \frac{-A\sin(\omega t)\times A\sin(\omega t)}{U} = -\frac{A^2}{2U}(1-\cos(2\omega t)) \tag{4.46}$$

By low pass filtering the output W we get a signal $W_L = -A^2/(2U)$ which clearly is negative proportional to the square of the input amplitude.

This technique can provide a very fast response time, it is though important to limit this response time with the low pass filter to reduce the vulnerability to transient distortion. The downside of this technique is that the analogue multiplier typically requires a current of more than 6 mA. To be able to use this technique in the telemetry buoy design a sampling scheme must be designed where the multiplier is deactivated when not reading the signal level. This excludes the possibility of having the reception detector as a wake-up source for the microcontroller.

4.4.3 Detecting a signal using a diode detector

The simplest form of envelope detector is the diode detector shown in Figure 4-35. If the capacitor and the resistor is chosen correctly V_{out} will follow the envelope of the input signal V_{in}. Some distortion in the form of ripple voltage will occur as the capacitor discharges. As long as the ripple lies below our threshold voltage this ripple is not of relevance to this application. In AM modulated audio signals however this is an important disadvantage of the envelope detector.

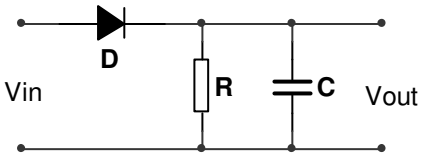


Figure 4-35 - Diode detector

When the input signal is larger than the sum of the capacitor voltage and the diode drop voltage, assuming the constant voltage drop model for the diode, the output voltage will be equal to the input voltage minus the diode drop

$$\begin{aligned} &V_{out} = V_{in} - V_D \\ &for \\ &V_{in} + V_D \geq V_{out} \end{aligned} \tag{4.47}$$

For this application the envelope detector should be used to measure signal strength. When signal strength reaches a certain threshold limit this will trigger a reception. The envelope detector will also serve as a level measurement in the AGC loop. It is therefore important that short pulses and transient voltages are suppressed to avoid a high rate of false triggers and level references.

4.4.4 Detecting a signal using a modified diode detector

A modification is made where the capacitor is charged through a resistor (R2). This introduces a low pass filter which will suppress transients and provide a more accurate signal level representation as the voltage power is a result of signal over time instead of instantaneous voltage. The circuit is shown in Figure 4-36.

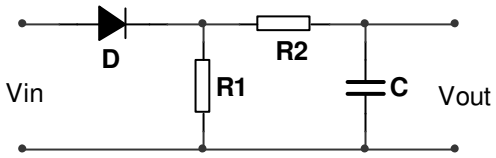


Figure 4-36 - Modified diode detector

When the input signal exceeds the capacitor voltage and the diode drop voltage, again assuming the constant voltage drop model for the diode and a constant input voltage, the differential equation is derived in (4.48).

$$\begin{aligned} i_{R2} &= i_C \\ \frac{v_i - v_o}{R} &= C\frac{dv_o}{dt} \end{aligned} \tag{4.48}$$

The solution of (4.48) is

$$v_o(t) = (v_o(0) - v_i)e^{-\frac{t}{RC}} + v_i \tag{4.49}$$

This is only valid when the input signal is constant. If the input signal to the circuit in Figure 4-36 is a sinusoidal without a DC component the input voltage will be constant equal to zero for the negative half period. Note that for discharge $R = R_1 + R_2$.

For the positive half period the input signal is described by

$$v_i(t) = A\sin(\omega t) \tag{4.50}$$

The differential equation is derived as follows

$$\begin{aligned} i_{R2} &= i_C \\ \frac{A\sin(\omega t) - v_o}{R} &= C\frac{dv_o}{dt} \end{aligned} \tag{4.51}$$

The solution is

$$v(t) = C_1 \cdot e^{-\frac{t}{RC}} - \frac{A\left(\omega RC\cos(\omega t) - \sin(\omega t)\right)}{1+\omega^2R^2C^2} \tag{4.52}$$

C_1 is found by defining the initial condition as v(0)

$$\begin{aligned} v(0) &= C_1 e^{-\frac{0}{RC}} - \frac{A\left(\omega RC\cos(\omega\cdot 0) - \sin(\omega\cdot 0)\right)}{1+\omega^2R^2C^2} \\ C_1 &= v(0) + \frac{A\omega RC}{1+\omega^2R^2C^2} \end{aligned} \tag{4.53}$$

This gives us the final expression for the positive half period

$$v(t) = \left[v(0) + \frac{A\omega CR}{1+\omega^2R^2C^2}\right]e^{-\frac{t}{RC}} - \frac{A\left(\omega RC\cos(\omega t) - \sin(\omega t)\right)}{1+\omega^2R^2C^2} \tag{4.54}$$

When the input signal is a sinusoidal with a DC component equal to zero, (4.49) will be used when the signal is negative and (4.54) when the signal is positive that is

$$\text{(4.49) for } 0 < \omega t < \pi$$
$$\text{(4.54) for } \pi < \omega t < 2\pi$$

The model is to complex to determine R_1, R_2 and C directly. Simulations are therefore needed to determine the values.

4.4.5 Design solution

The modified diode detector is chosen for final design. Its simplicity and low power feature makes this circuit the most suitable for the telemetry buoy application. The circuit is simulated in Multisim 10.l. A test signal is generated as shown in Figure 4-37. The circuit outputs a sinusoidal of 69 kHz, 1Vpp, 0V DC, in bursts of 1ms.

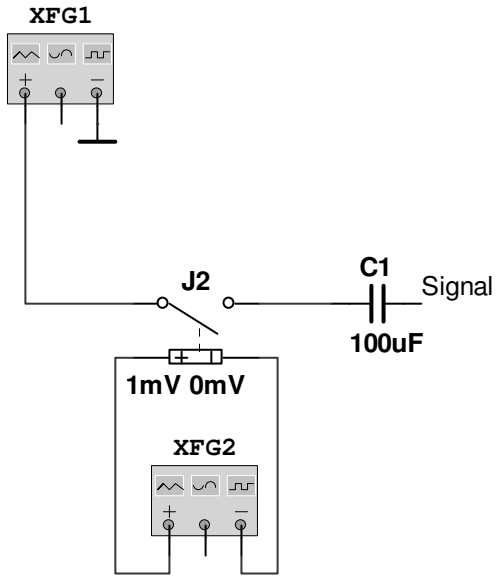


Figure 4-37 - Generating test signal

The detector circuit in Figure 4-38 uses a Schottky diode to provide a threshold voltage as low as possible. For small signals this voltage drop may still be crucial. R5 and R6 are therefore added to provide a bias for the diode. Unlike capacitors, resistors are available in most values and accuracies. A standard capacitor value is therefore chosen as a base point for the design.

To provide a short detection time while still suppressing transient voltages R1 is chosen so that the voltage is at 63% of maximum after 5 cycles of the 69 kHz. R2 is chosen so that the level falls at a sufficient rate while wasting a minimum of current. Assuming 1 V peak and a sinusoidal input the total current consumption of the circuit is

$$I = \frac{Vp}{2R\sqrt{2}} = \frac{1}{2 \cdot 10 \cdot 10^3 \cdot \sqrt{2}} = 35.4\mu A \qquad (4.55)$$

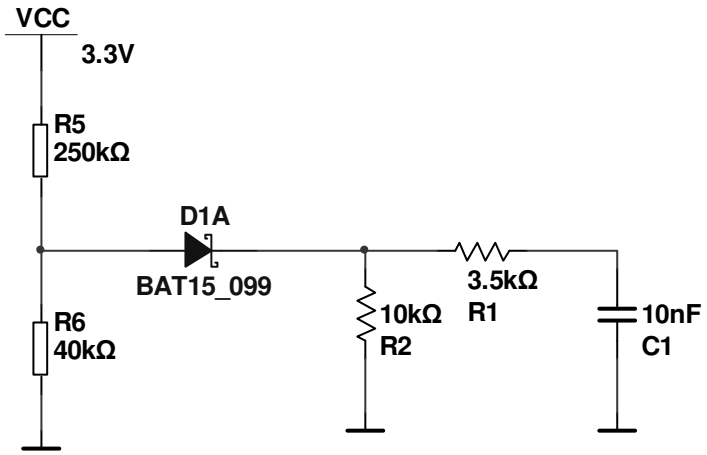


Figure 4-38 - Modified diode detector

The output level will be limited by the peak level of the signal. It is desired to amplify this signal to utilize the full voltage range. A simple low pass filtration is also added to remove ripple. The amplification circuit is shown in Figure 4-39. To minimize the power consumption the resistors should be as large as possible while preserving the characteristics of the operational amplifier. The downside of using large resistors is that the circuit is more vulnerable to electrical noise. It is therefore important to consider this when designing the PCB layout.

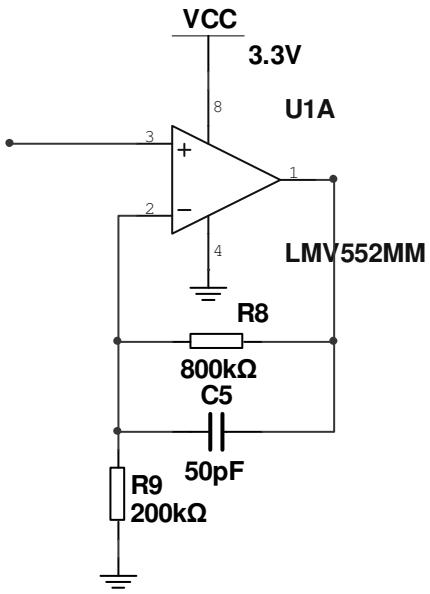


Figure 4-39 - Signal amplifier with low-pass filtration

Finally a digital signal is produced by a comparator to generate an external interrupt request to the MCU. The circuit is simulated with a fixed compare value. In the final design circuit this voltage will be set by the MCU to provide an adaptive solution in addition to the AGC. The circuit is shown in Figure 4-40.

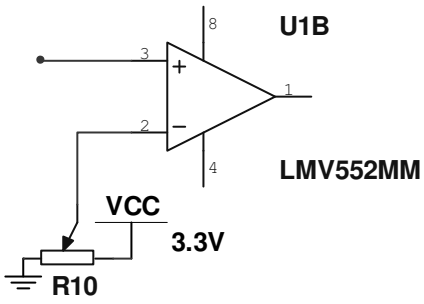


Figure 4-40 - Comparator circuit

The simulated results are shown in the two following scope plots. The channels are connected:

1. Original signal around zero (Figure 4-37)
2. Rectified signal (Figure 4-38)
3. Amplified output (Figure 4-39)
4. Comparator output (Figure 4-40)

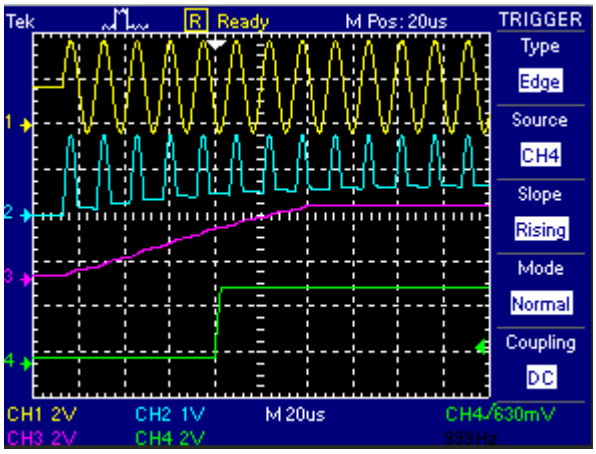


Figure 4-41 – Simulation scope plot: Modified diode detector trig

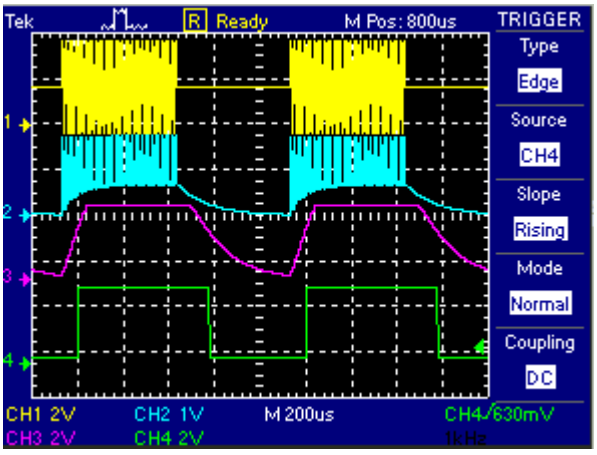


Figure 4-42 - Simulation scope plot: Modified diode detector

4.5 Component selection

This section covers the component selection. The most vital components are the operational amplifiers and the digital potentiometer.

4.5.1 Operational amplifier

For this application the most important factors of the operational amplifier are power consumption, gain bandwidth product and rail to rail output.

Section 4.3.5 describes the requirements for the amplifiers used in the filter circuit. It is required both operational amplifiers with a gain-bandwidth product of minimum 3 MHz and 12 MHz. To minimize the overall power consumption this design will use two different operational amplifiers for the filter circuit.

The *LMV552MM* operational amplifier manufactured by National Instruments operates at an extremely low power consumption of 34μA per amplifier. It features rail to rail output and a gain bandwidth product of 3MHz.

The highest signal frequency is 80 kHz. By stretching the gain bandwidth product to its specified value all amplifier elements will damp the highest signal with 3 dB. To avoid this, the calculations are based on a maximum frequency of 100 kHz. This gives us a maximum gain per amplifier of

$$G_{\max} = \frac{GainBandwidth}{f_{\max}} = \frac{3 \cdot 10^6}{100 \cdot 10^3} = 30$$
$$G_{\max,dB} = 20\log(G_{\max}) = 20\log(30) = 29.5dB \tag{4.56}$$

Four amplifiers are therefore needed to achieve the minimum of 103 dB gain. A gain of 25 per amplifier is chosen to provide a good dynamic range while staying well within the gain bandwidth limitation. The total gain is

$$G_{dB} = 4 \cdot 20\log(25) = 112dB \tag{4.57}$$

The *LMV562MM* operational amplifier manufactured by National Instruments operates at a power consumption of only 110uA while providing a gain-bandwidth of 12 MHz. It follows that this is one of the markets best amplifier with respect to gain-bandwidth / power consumption ratio. It also features rail-to-rail output and operates at voltages down to 2.7 V.

The design will therefore use four *LMV552MM* for gain, two for the active filter and two for the reception detector, in all six amplifiers. It will use two *LMV562MM* for the active filter.

4.5.2 Digital potentiometer

As described in section 4.2.2.5 it is important to consider the bandwidth and the wiper capacitance of the potentiometer. It is desired to have a potentiometer with as low wiper capacitance as possible to allow the potentiometer to have a high end-to-end resistance and thereby dissipating a minimum of power.

It is also desired to have as large a resolution as possible, this is typically 8-bit. If there is more than one potentiometer per package it is vital that the cross-talk between the channels is much lower than the gain between the channels.

The interface should be SPI or I^2C to ensure a correct gain setting. The interface frequency should be high to avoid interference with the signal band. The current consumption of a CMOS digital potentiometer is typically a few μA, but must be taken into consideration when choosing the potentiometer.

The 100 kΩ digital potentiometer *MAX5401EKA* is chosen for its very low wiper capacitance of 25pF while consuming only 5μA. At 50% setting it will provide a bandwidth of

$$BW = \frac{1}{2\pi 50 \cdot 10^3 \cdot 25 \cdot 10^4} = 127 kHz \tag{4.58}$$

It also features 8-bit resolution with a SPI compatible interface.

The design will therefore use three *MAX5401EKA* for the AGC circuit and one for the reception detector, in all four *MAX5401EKA*.

5. Digital design

This chapter includes a brief description of the digital hardware design of the telemetry buoy. For extensive details please refer to the project report *Acoustic telemetry buoy* [4] which is the basis for the whole digital design. The report [4] concludes that the Atmel AVR32 UC3 series of microcontrollers should be used for further design.

The design is concentrated around the Atmel microcontroller AT32UC3B1256. Key features are [19]:

- 48 pin package
- 75 DMIPS at 60 Mhz
- 7 peripheral DMA channels
- 10-bit internal ADC
- 256K Bytes internal flash memory
- 32K Bytes internal SRAM
- Universal Serial Bus 2.0
- 3.3V Operation

The main tasks of the digital part of the system are:

- Calculation and setting the gain in the AGC loop
- Sampling and digital filtration of the analogue signal
- Decoding of the received signals
- Storing received data with time stamps
- Providing the stored data to the user

The following sections provide an overview of the listed tasks. Please refer to chapter 7 for a description of the implemented firmware.

5.1 Calculating and setting the gain in the AGC loop

Section *4.4 Reception detector* describes a way of providing an analogue DC signal which is proportional to the amplitude of the filtered signal. This signal can be converted to a digital number using the internal ADC of the AT32UC3B microcontroller. If the signal is too strong the amplifier might saturate. It is therefore important that the microcontroller ensures that the signal is below the saturation limit of the amplifiers when measuring the signal level.

The amplifier gain is set by setting the wiper position of the digital potentiometers via the SPI. It is important that the MCU waits until the level signal has stabilized after altering the gain setting before converting the signal using the internal ADC.

A simplified functional schematic of the variable gain amplifier is shown in Figure 5-1. The wiper position, from low to high, is proportional to the input number. The equation for the gain is therefore

$$G = \frac{V_o}{V_i} = \frac{n_1}{256} \cdot 25 \cdot \frac{n_2}{256} \cdot 625 \cdot \frac{n_3}{256} \cdot 25$$
$$G = \frac{n_1 n_2 n_3}{256^3} \cdot 390625 \qquad (5.1)$$

Where
n^x is the 8-bit digital position of the wiper of potentiometer x

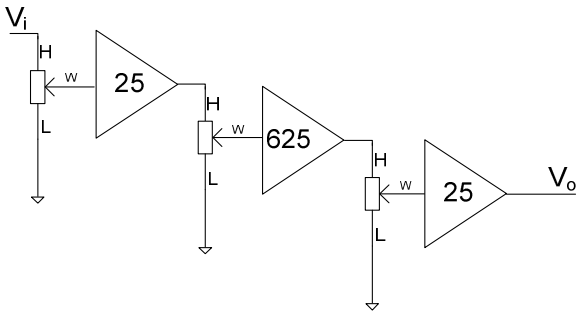


Figure 5-1 - Simplified variable gain amplifier

The gain is though somewhat lower due to the inserted filtration and using non-ideal operational amplifiers. Using the simulation result provided in Figure 4-34 on page 37, the maximum gain is found to be 105dB. The new expression becomes

$$\begin{aligned} G_{\max} &= 10^{\frac{105}{20}} = 178828 \\ G &= \frac{n_1 n_2 n_3}{256^3} \cdot 178828 = \frac{n_1 n_2 n_3}{94.34} \end{aligned} \tag{5.2}$$

5.2 Sampling and filtration

The sampling is done with the internal ADC. The detector offers a comparator configuration which is connected to an interrupt pin on the AT32UC3B. This configuration allows the microcontroller to be in a sleep mode when not receiving. The signal will be undersampled as described in section 2.1 on page 4. The frequency can be further filtered digitally before determining if the signal is a valid pulse of the correct frequency.

5.3 Decoding the signal

The AT32UC3B microcontroller has several internal timers which can run either with the system clock or the 32 kHz external or internal oscillator as clock source. These timers can be used with either the ADC and a firmware routine or the interrupt triggered by the reception detector to determine the pulse position. When the time separation of the pulses is determined the data can be retrieved. Each transmission ends width a CRC check and this must be validated before storing the data.

5.4 Storing the received data

The telemetry buoy should be able to store 100 000 receptions with ID, data and time stamps. Assuming an 8-bit ID, 16 bits of data and a time stamp of 48 bits the complete memory must be a minimum of 7.2 Mbit. To ensure that the data is not lost due to a system failure or removal of the battery the memory should be non-volatile. It is also important that the memory unit requires a minimum of power to store data. The latter requires a low current consumption as well as a short write time.

A serial flash AT45DB321 is chosen for this project. Key features are:

- 32 MBit of storage

- SPI™ interface
- 2.7-3.6V operation
- 25uA standby current
- 5uA Deep power down current
- 12mA write operation current
- 3ms page programming time (512 bit)

5.5 Recovering the stored data

Some form of interface must be provided for recovering the stored data. This interface might be Bluetooth, USB, RS232, GSM, Argos or other. It is preferable that the system can be adapted to any of these communication forms. The USB interface is chosen for this project. A UART to USB bridge FT232R is used for the communication. The AT32UC3B also features an inbuilt USB interface. This can be adapted for the final product, but the complexity of use makes it unsuitable for a prototype where it is to be used with debugging as well as recovering the stored data.

6. Prototype design

A prototype was designed in order to test both the analogue and digital hardware designs as well as having a realistic platform for testing the firmware. The prototype is arranged as six blocks:

- Preamp
- Variable gain amplifier and filter
- Reception detector
- Connectors and headers
- Power distribution
- MCU and Flash memory

The system is structured as in Figure 6-1.

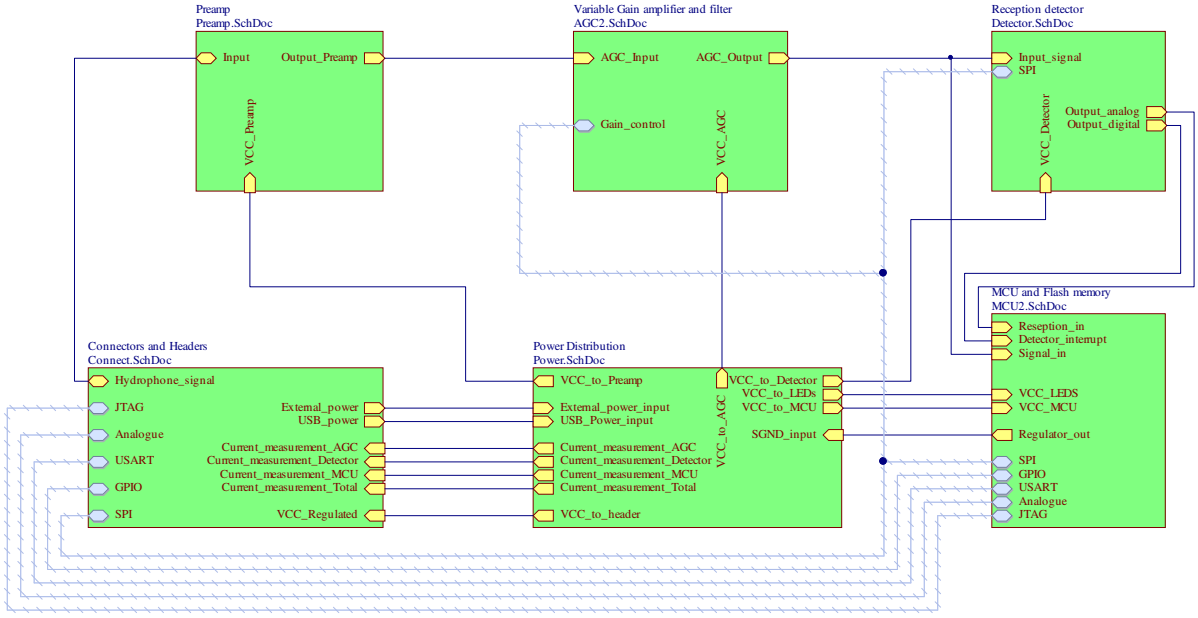


Figure 6-1 - Schematic overview

These hardware blocks will be described in the following sections. For a compete documentation please refer to the complete schematics, bill of materials and assembly diagram provided in appendix 5-7.

6.1 Power distribution

The schematic for the power distribution is shown in Figure 6-2. The power can be supplied from the USB or from an external DC power supply. The system is protected against wrong polarization with Q1. A P-channel MOSFET is used instead of a simple diode to eliminate any voltage drop. A fuse is mounted to protect the circuit against overload and short circuit.

The AT32UC3B requires a supply voltage of 3.3V. To limit the number of voltage regulators and the power loss that follows, the system voltage is chosen to be 3.3V for both the analogue and the digital design. The linear regulator LMS8117AMP is chosen for the prototype. It is important to notice that for the final product the system will require a buck-boost switching regulator carefully chosen to suit the power requirements of the final product. As for the prototype the voltage regulator is rather oversized for handling USB communication, LEDs, debugging etc.

The input voltage is limited by the voltage regulator U12 and is ranging from 4.5 – 12V DC. The current is limited to 0.45A by the transistor Q1.

To measure the current the prototype features four current shunt monitors INA138NA. The device amplifies the voltage drop over the series resistor with a gain of 100. This enables real-time current monitoring using an oscilloscope. The system offers three individually monitored power sections:

- Variable gain amplifier and filter
- Reception detector
- MCU and flash

In addition the sum of the three sections can be monitored to measure the current of the complete system. Debugging features and additional circuitry such as the preamp and LEDs are bypassed the monitors.

Signal ground is created by using a secondary voltage regulator LP3992IMF of 1.5V. As an alternative the internal 1.8V voltage regulator of the AT32UC3B can be selected as the signal ground. This will eliminate two external components and reduce the power consumption. The regulator is chosen with a jumper on P2. U17 can be shut down using a jumper on P3.

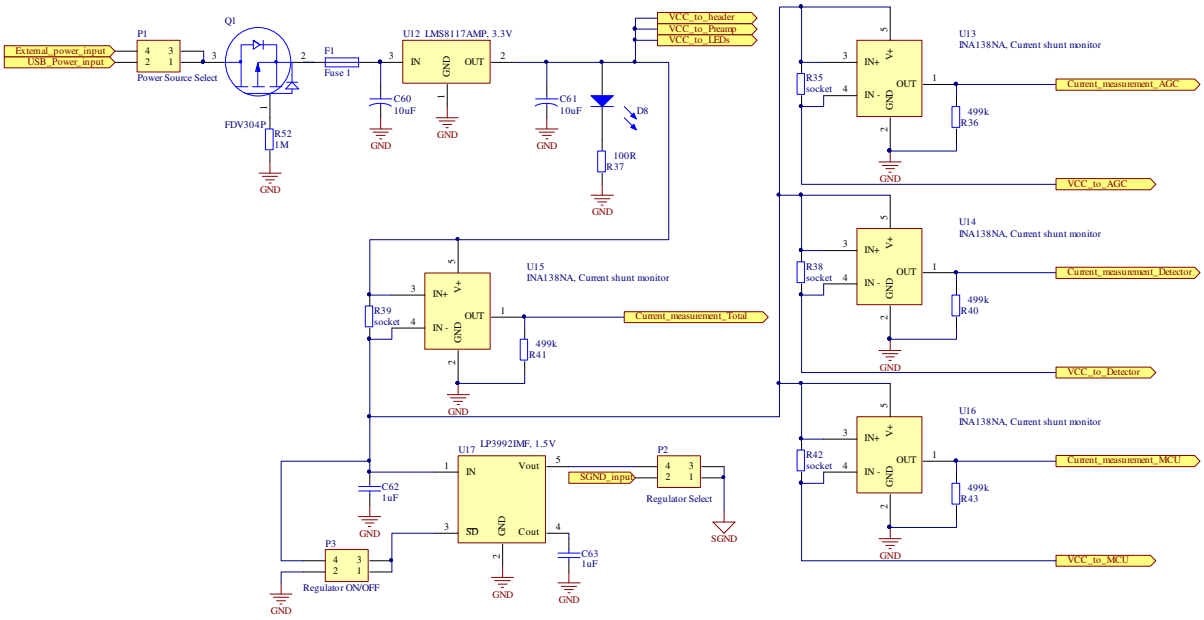


Figure 6-2 - Power distribution schematic

6.2 Preamp

The schematic for the preamp is shown in Figure 6-3. This block is added to provide support for various hydrophones, both passive and active. For a final design the preamp should have fixed gain suited for a specific hydrophone element. The choice of such an element is dependent on physical size and shape of the final product and is therefore not the scope of this prototype design.

The preamp provides possibilities such as

- Variable signal damping with R44
- Variable signal amplification with R45
- Choice of output signals
    - Direct
    - Damped
    - Amplified
    - Grounded
- Power source for an active two wire hydrophone

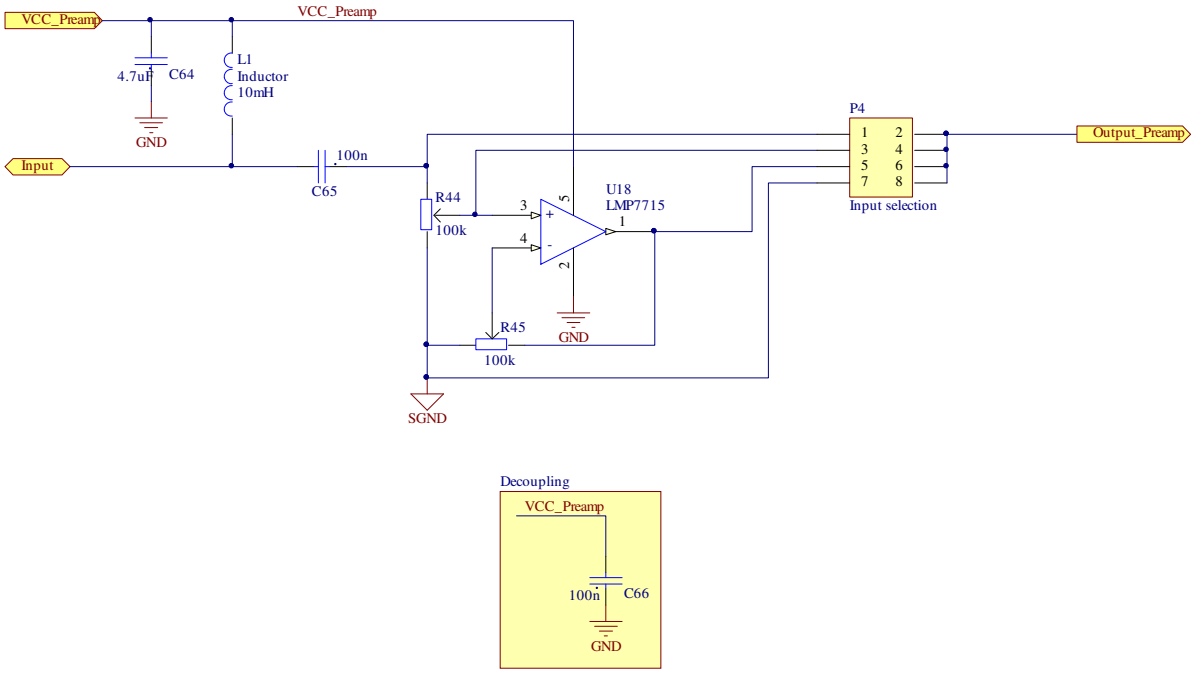


Figure 6-3 – Preamp schematic

6.3 Variable gain amplifier and filter

The background for the design of the variable gain amplifier and the filter is covered in section 4.2 and section 4.3 respectively. The schematic for the combined circuit is shown in Figure 6-4. The AGC_intput is connected to the output of the Preamp. The AGC_output is connected to the ADC input of the MCU and the reception detector.

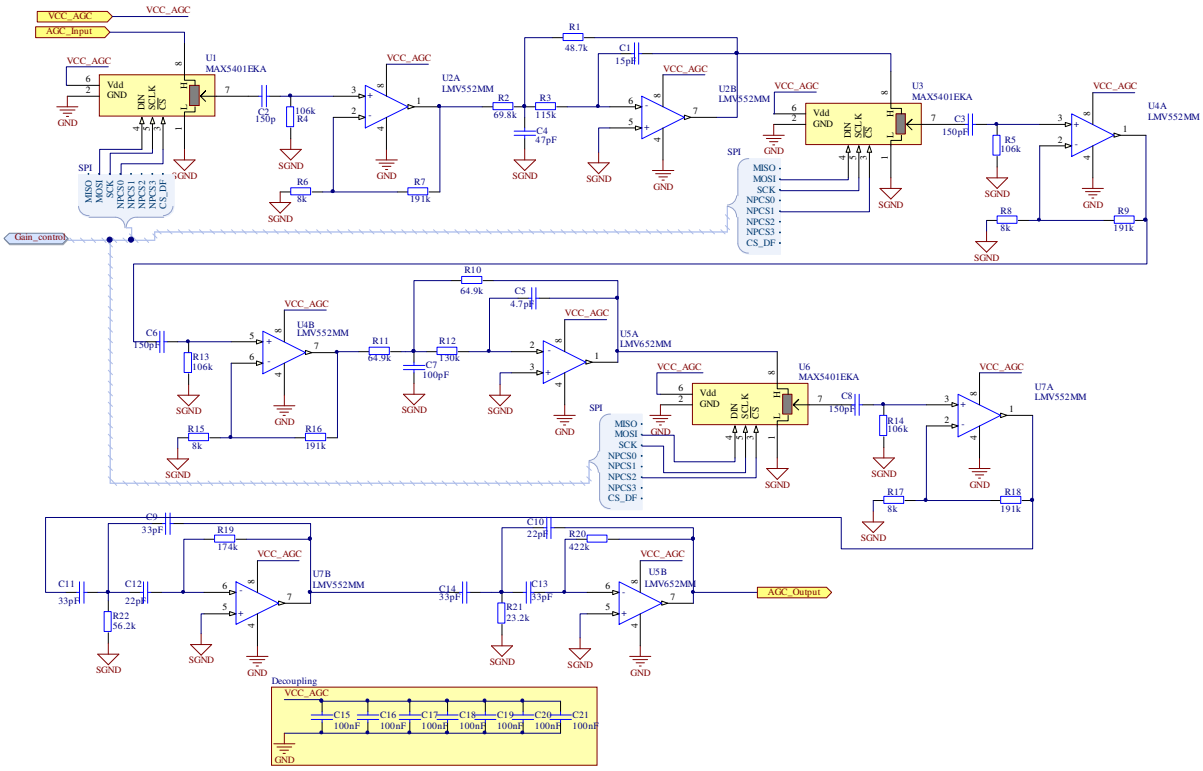


Figure 6-4 - Variable gain amplifier and filter schematic

6.4 Reception detector

The design of the reception detector is covered in section 4.4 on page 37. The schematic is shown in Figure 6-5 - Reception detector schematic.

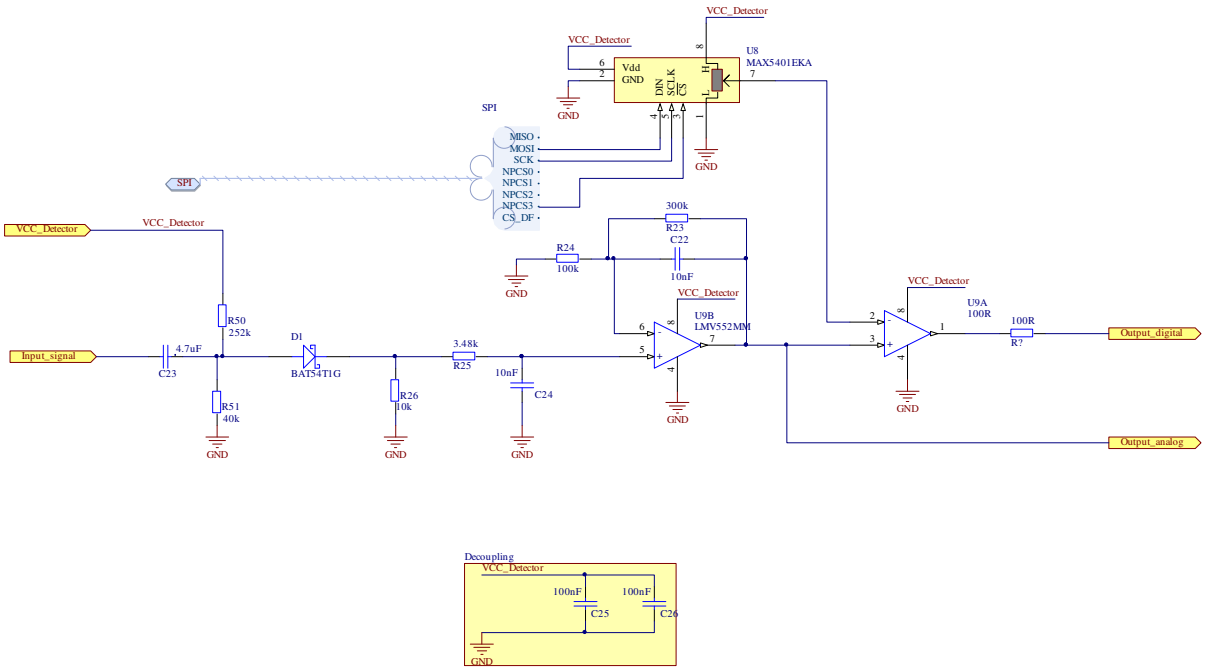


Figure 6-5 - Reception detector schematic

6.5 Connectors and headers

The schematic for the connectors and headers block is shown in Figure 6-6. The block includes the following headers:

- Power input – 4.5 – 12V external voltage input
- Current sense – output from the current shunt monitors
- Hydrophone – input signal from the hydrophone
- JTAG – programming and debugging of the AT32UC3B
- GPIO – connected in parallel with the LEDs for measurement and connection of external circuitry
- SPI and USART – for debugging purpose and possible external circuity
- Analogue – connected to the ADC inputs of the AT32UC3B, for debugging and possible additional inputs
- Power output – for potential additional external circuitry and measurements

In addition to headers this block includes a USB connector with a USB to UART bridge connected to the USART1 of the AT32UC3B. This is to be used for debugging and analysis of the results. The USART signals are connected through a DIP switch. This is to ensure that the USB does not draw any current from the microcontroller when performing current measurements. Series resistors are due to the different operating voltage of U19 and the microcontroller.

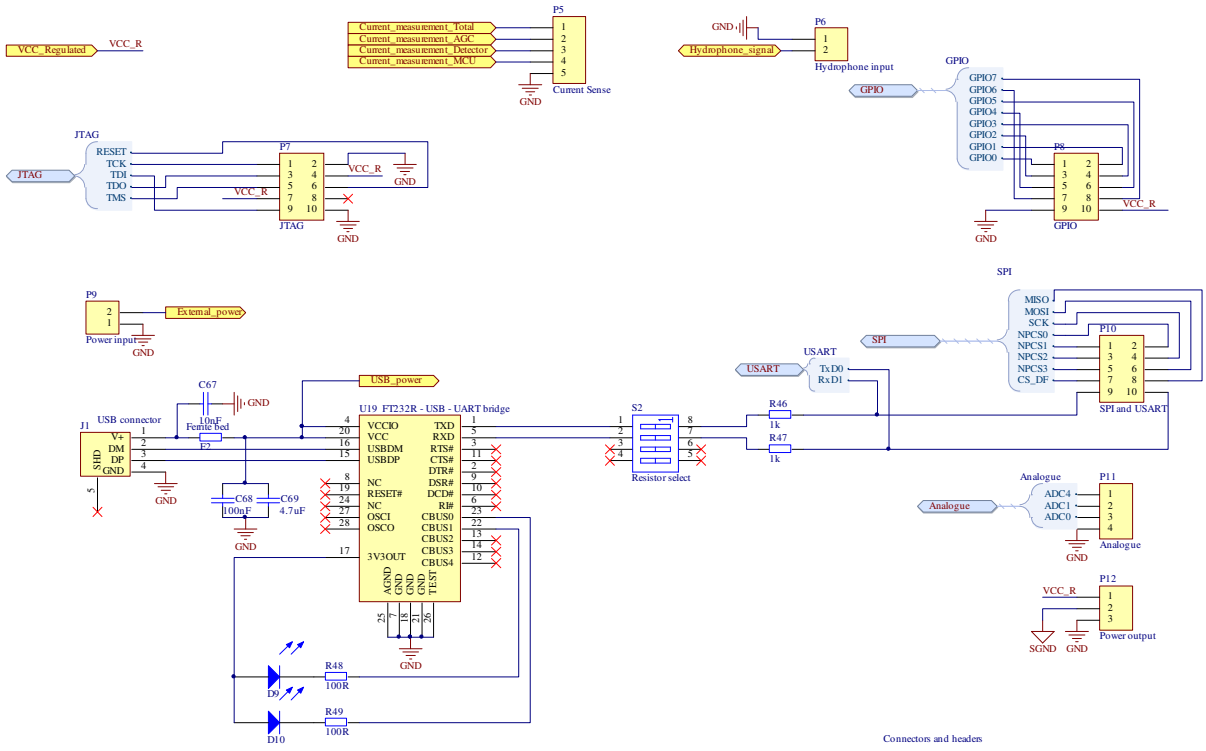


Figure 6-6 - Connections and headers schematic

6.6 MCU and Flash memory

The schematic for the MCU and flash block is shown in Figure 6-7. This block includes the digital design described in chapter 5. In addition to the microcontroller and the serial flash the

block includes six LEDs for indication and three switches for debugging. The LEDs can be disconnected from the microcontroller to avoid interference when performing current measurements.

The AT32UC3B1256 is decoupled according to the recommendations given by Atmel in application note *AVR32715: AVR32 UC3B Schematic Checklist* [20].

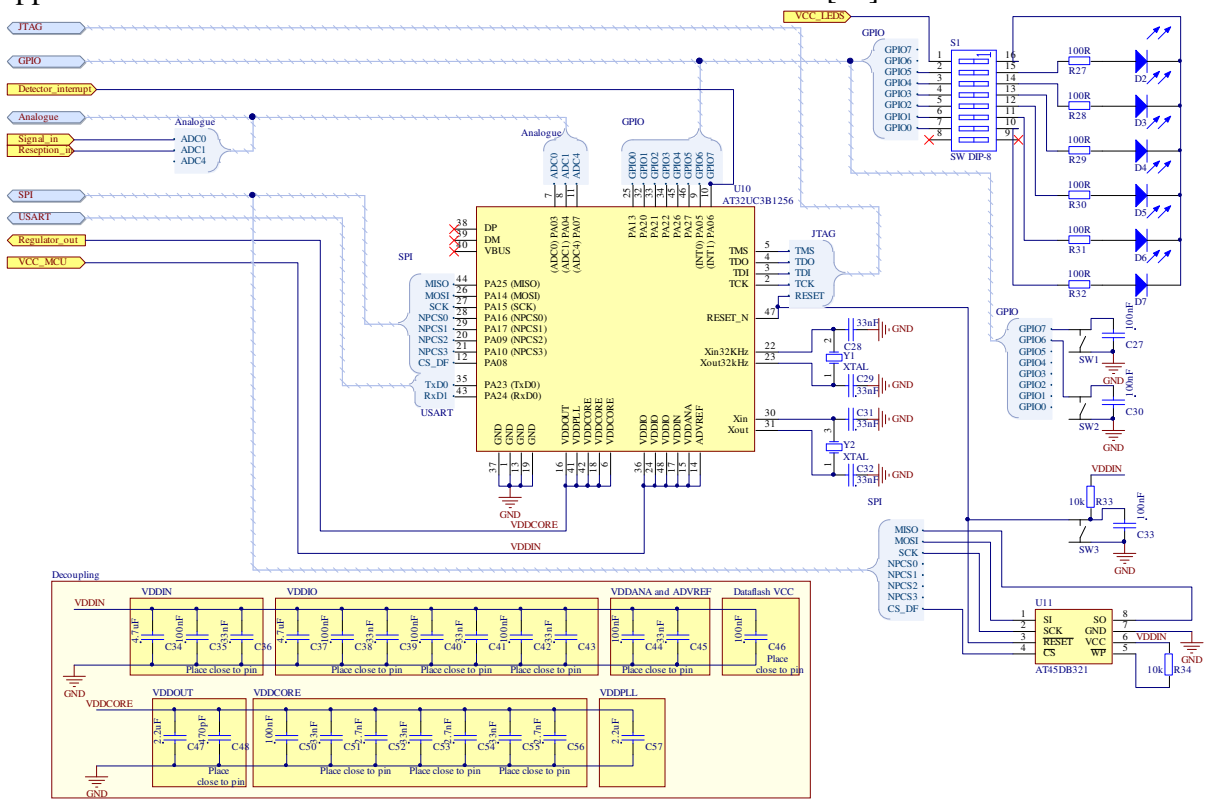


Figure 6-7 - MCU and Flash memory schematic

6.7 PCB layout

For the physical prototype a four layer PCB is used. The two inner layers are reserved for ground and VCC where the ground layer is a large plane whereas the VCC is divided into the different power sections described in section 6.1 on page 48. The layers gnd, vcc, top and silkscreen are shown in Figure 6-8. The bottom layer is reserved for digital signals while the top layer consists only of analogue signal paths. Having the gnd and vcc planes in between will help reduce the influence of digital noise in the analogue circuitry. Placing the vcc and ground planes in the inner layers also provides a decoupling effect helping reduce the noise further.

All the components are grouped on the PCB as in the schematic pages. This provides a good overview as well as ensuring noise immunity when separating the digital and analogue circuits. Notice that the digital circuits are placed to the right while the analogue are placed far to the left. This spacing will ensure a minimum of digital noise impact on the analogue circuitry.

The gerber files for the PCB are available in appendix A.

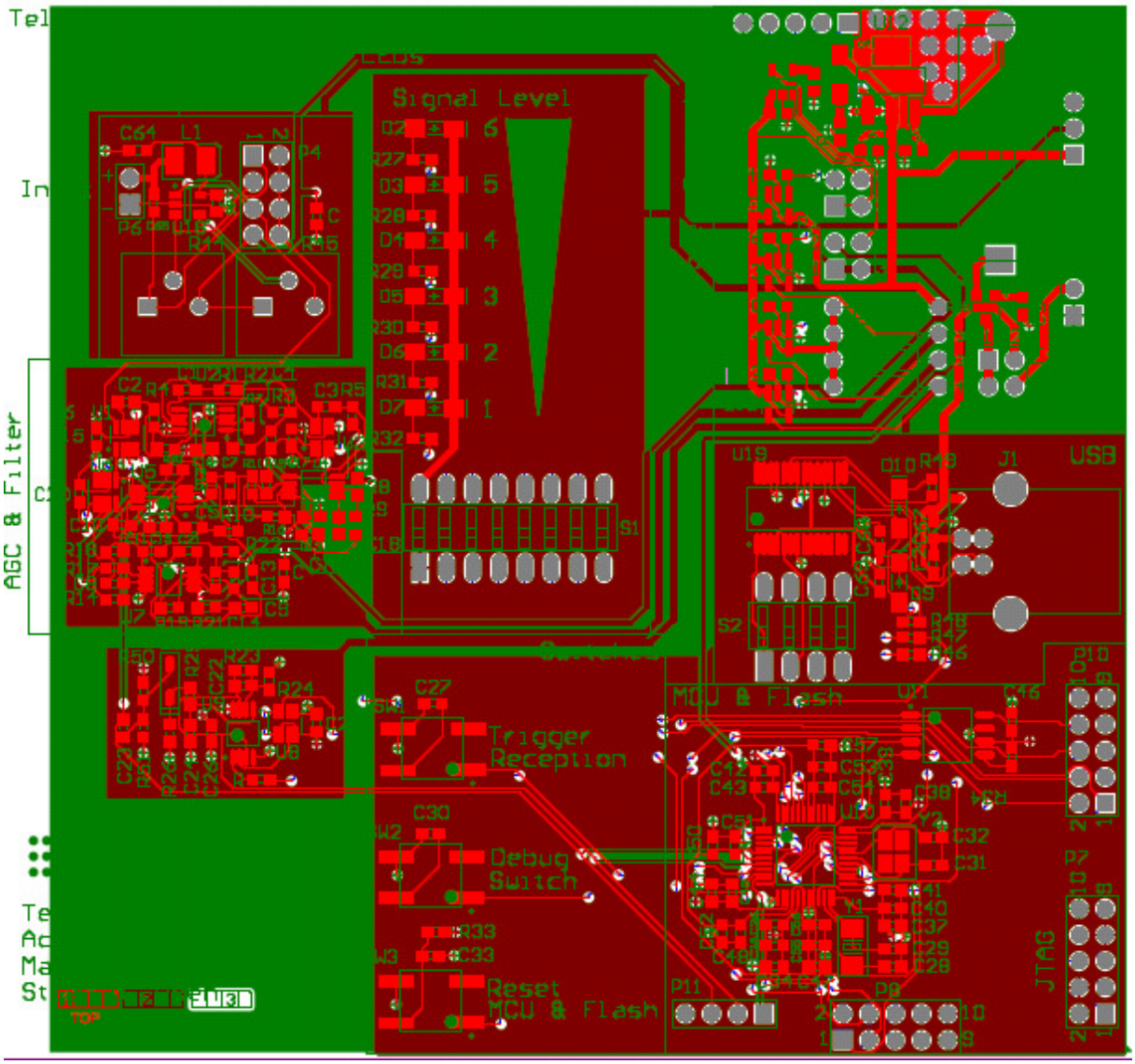


Figure 6-8 - PCB layout

The assembled prototype is depicted in Figure 6-9.

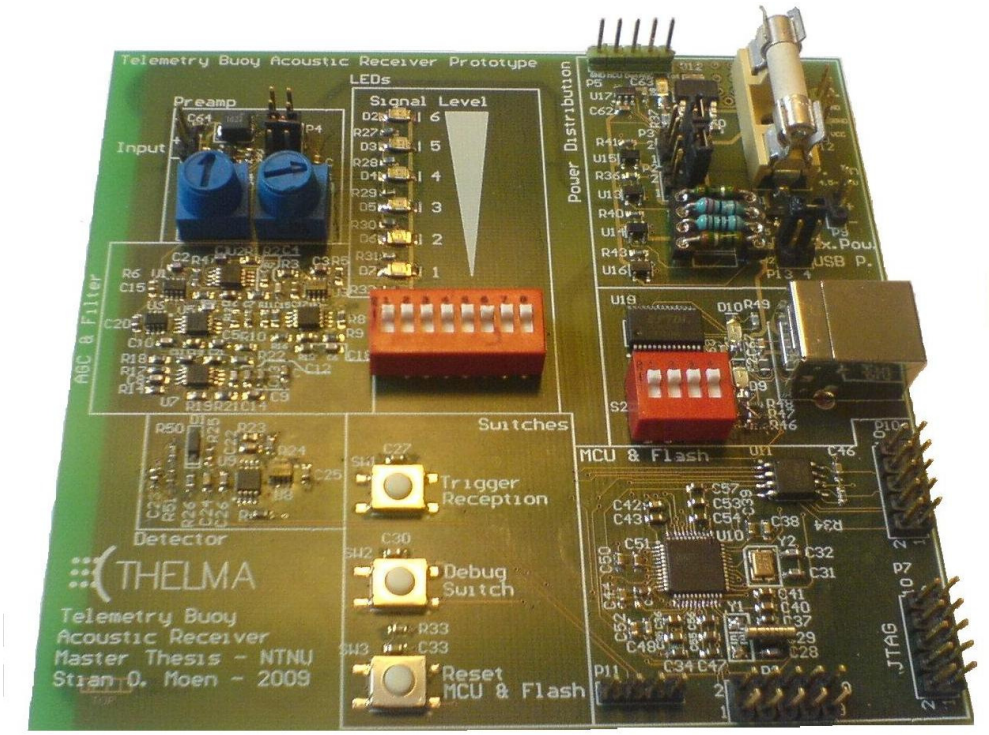


Figure 6-9 - Assembled prototype

7. Firmware

The firmware for the prototype is written in the programming language C. AVR32 Studio 2.12 was used which includes a gcc compiler ported by Atmel to work with AVR32® microcontrollers. It also includes a software framework written by Atmel. The framework includes drivers for all modules, some with examples, as well as a digital signal processing (DSP) library and more. The drivers, the examples and the DSP library [27] have been used in the design of the firmware.

The firmware for the prototype is only written for testing and measurement purposes. It is not written to be either speed or size optimal, but serves as a good basis for testing and further development. The complete code is found in appendix 8 and B. The float diagrams for the main routine and the active interrupt routines are shown in Figure 7-1, Figure 2-1, Figure 7-3 and Figure 7-4.

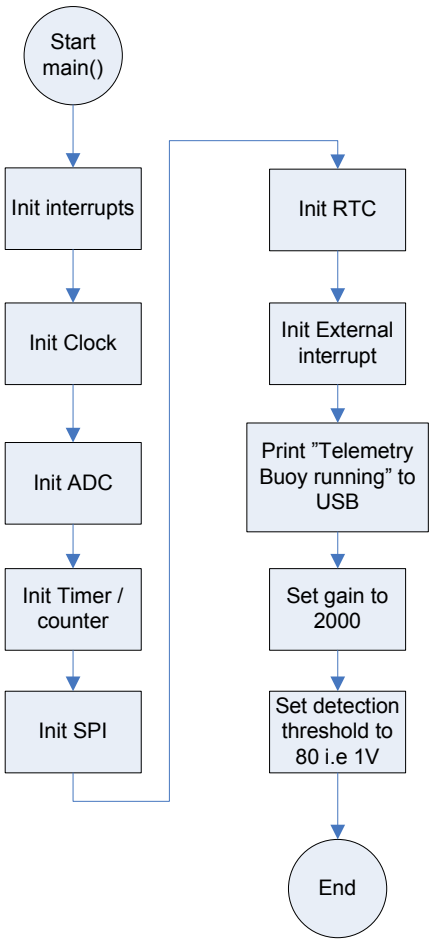


Figure 7-1 - Main routine float diagram

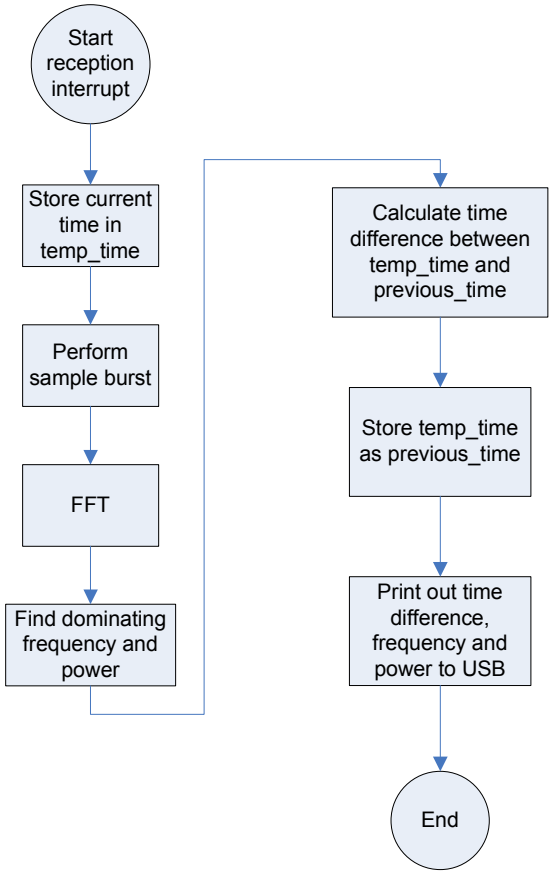


Figure 7-2 - Reception interrupt handler

Note that the reception handler uses Fast Fourier transform to check the frequency. This means that several carrier frequencies easily can be supported as well as a multiple frequency shift keying modulation scheme.

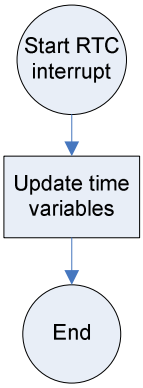


Figure 7-3 - RTC interrupt handler

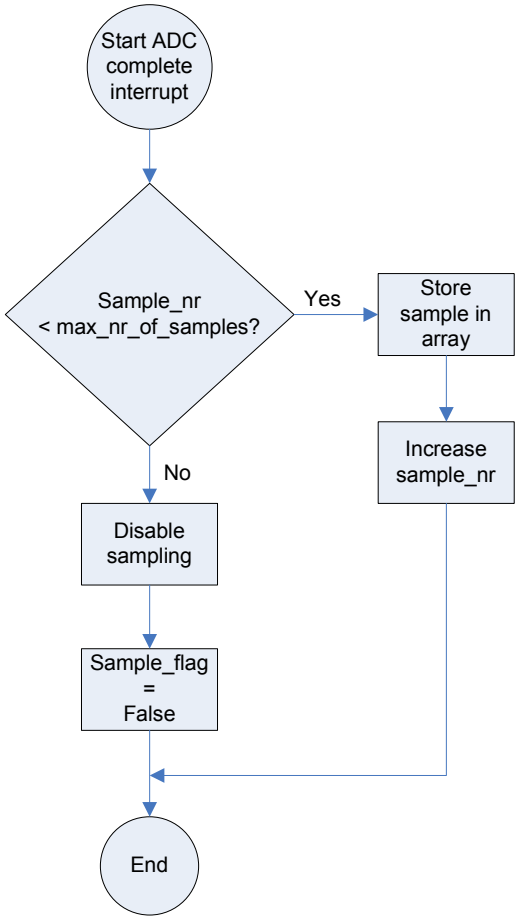


Figure 7-4 - ADC conversion complete interrupt handler

The code is structured as ”MUDULE_functions.c” and “MODULE_functions.h”. As an example the function for setting the gain in the variable gain amplifier is found in “SPI_functions.c” since the SPI module is used to set the gain. The only exception is the functions used for testing which is found in TEST_functions.c. In addition the config.h file includes configuration settings such as pin configurations, CPU speed, sample frequency and more. This way of structuring the configuration makes it easy to use the same functions for different boards and applications. The following sections will describe the main functions of each C-file to be used as reference. For additional details please refer to the complete c-code provided in appendix 8 and B.

7.1 ADC_functions

void init_ADC(void);
This function initializes the ADC to use the signal from the variable gain amplifier and filter as input. It sets the prescaler in accordance to the datasheet, enables complete ADC interrupt and enables the hardware trigger from timer counter TIOA channel 0. The timer counter can then be used to initiate an ADC conversion. This will ensure a stable and accurate sampling frequency without CPU involvement.

static void ADC_complete_irq(void);
Stores the new adc value to the sample array and increments the sample counter. If the sample was the last of the burst the sampling is disabled while signalling this with the sample_flag.

7.2 CLOCK_functions

void init_clock(void);
This function initializes the system to use either external crystal or external clock as input to the PLL. The PLL will produce a CPU frequency of 60 MHz and a peripheral frequency of 15 MHz. The function also sets the memory unit to use one wait state according to the datasheet.

7.3 DSP_functions

FreqPower_t check_signal(void);
This function removes the DC offset and performs a 16-bit 64-point complex FFT on the global array *sample* which holds the values of the most recent sample burst. It then searches for the frequency with the highest power in the positive frequency area and returns the power and the frequency in kHz. The function requires that a burst consists of 64 16-bit fixed point samples.

7.4 EIC_functions

void init_EIC(void);
This function initializes the external interrupt controller to generate a synchronous interrupt at the rising flank of the digital output of the reception detector.

static void eic_int_reception_handler(void);
This function handles the reception of a signal. It functions as shown in Figure 7-2. This is the same routine which is used during the test of the complete system as specified in section 8.1.7. The routine can easily be changed to include digital filtration, validation of the signal and more. The interrupt flag is cleared at the end of the routine to avoid several parallel instances and the possibility of stack overflow.

7.5 RTC_functions

void init_RTC(void);
Initializes the RTC timer to use the 32.000 kHz external crystal as source and divide the clock by 32 thus configuring the timer to count milliseconds. The time variables are reset to default 0. An interrupt is generated each second to update the time variables.

void rtc_irq(void);
The interrupt handler updates a simplified RTC clock, counting seconds, minutes, hours and days. The clock will always start at zero when resetting the device.

time_variable_t get_time(void);
Ensures that no update of the time variables is ongoing while reading the time. Returns a structure containing milliseconds, seconds, minutes, hours and days.

time_variable_t calculate_time_diff(time_variable_t t1, time_variable_t t2);
Takes two structures with the time as argument. Will calculate and return t1 - t2. This routine is mainly used to calculate the space between each pulse in the PPM signal received from the fish tag.

7.6 SPI_functions

void init_SPI(void);
Initializes the SPI module to use with the digital potentiometers. Initiates the manual CS for the external dataflash to high.

void set_gain(U32 gain);
Sets the gain of the variable gain amplifier by calculating the potentiometer settings and updating the potentiometers via the SPI interface if necessary. The gain is calculated so that the signal is at maximum throughout the system to achieve the best SNR.

void set_theshold(unsigned char data);
Updates the potentiometer which sets the threshold for the digital output of the reception detector. Input it the raw data sent to the potentiometer. i.e voltage = (data/255)*3.3V.

void store_data_in_dataflash(unsigned char * data, unsigned short size);
Stores a maximum of 512 bytes of data to address 0 in the dataflash. The microcontroller does not have sufficient chip selects. The dataflash is therefore connected to a gpio pin which must be set manually. To avoid that any other chip select is active while using the SPI module the selected NPCS pin in the SPI module is configured as a GPIO pin during the writing of the page. The system is reset to the previous setting at the end of the function. The function is only implemented to measure time and current consumption.

unsigned char read_data_from_dataflash(unsigned char *data, unsigned short size);
Reads a maximum of 512 bytes of data form address 0 in the dataflash. This routine is only implemented to validate that the store_data_in_dataflash routine is functioning. The chip select is manipulated in the same manner as with the store function.

7.7 TC_functions

void init_TC(void);
Initializes the timer counter used for timing the sampling of the analogue signal. The timer is initialized as stopped.

void enable_sampling(void);
Starts the timer counter, rests the sample counter variable and enables the ADC interrupt. The ADC is set to starts a conversion each time the timer reaches the value defined in the RC register.

void diable_sampling(void);
Stops the sampling timer and disables the ADC interrupt.

7.8 USART_functions

void init_USART(void);
Initializes the USART to communicate with the UART to USB bridge. The baudrate is defined in config.h

7.9 TEST_functions

void test(void);
Includes all functional and performance tests which requires the use of the MCU. Only the test of the complete system is not defined here. For details about the tests please refer to chapter 8.

7.10 main.c

The main.c file runs all init functions as well as setting the gain and threshold for the test of the complete system.

8. Test and measurement procedures

This chapter describes the tests needed to validate the prototype. Tests are divided into functional and performance tests. As an example a functional test of a digital Fourier transform routine will be that that dominant frequency of the output can be pinpointed as the input frequency. The performance tests can include the execution time, power consumption of the microcontroller and the accuracy of the output.

Functionality tests should be performed for the following modules/circuits:

- Power distribution
- Preamp
- Variable gain amplifier and filter
- USB circuit
- MCU and Flash memory
- Complete system

When all listed modules/circuits have passed the functionality tests, the performance tests can be carried out.

Performance tests should be performed to the following modules.

- Variable gain amplifier and filter
- Reception detector
- MCU and flash memory

Unless other setting is specified all tests shall be carried out under the following conditions:

- external power supply of 5.0 V connected
- Jumper on pins 4-3 on header P1
- Jumper on pins 4-3 on header P2
- Jumper on pins 4-3 on header P3
- Jumper on pins 5-6 on header P4
- 1 Ω resistor placed in socket R39
- 22 Ω resistor placed in socket R35

- 100 Ω resistor placed in socket R38
- 1 Ω resistor placed in socket R42
- All switches on S1 and S2 in OFF position

GND is reference to all voltage measurements unless other is specified

The following sections will describe the tests to be performed in detail.

8.1 Functionality tests

These tests are used to indicate that the assembly has been carried out correctly. These tests should be performed before ant performance tests to ensure that the measurements that require high accuracy are carried out under correct operating conditions.

8.1.1 Power distribution

The power distribution block tasks are to provide a stable operating voltage for the circuit and provide a means of measuring the current consumption of selected modules.

The functional tests are as follows.

Test #	1
Motive	Ensure correct operating voltages
Module	Power distribution
Tool	Oscilloscope
Min value	3.25 V
Max value	3.35 V
Procedure	Measure the voltage at P12 pin 1, U13 pin 4, U14 pin 4, U16 pin 4. Ensure that there are no short or long term voltages outside the limits
Measured value(s)	
Status	

Test #	2
Motive	Ensure that the current measurement circuits function properly
Module	Power distribution
Tool	Oscilloscope
Min value	0.5 V
Max value	3 V
Procedure	Measure the voltage at P5 pin 1,2,3,4 Ensure that there are no short or long term voltages outside the limits
Measured value(s)	
Status	

Test #	3
Motive	Ensure that the SGND regulator circuit is functioning properly
Module	Power distribution
Tool	Oscilloscope
Min value	1.45 V
Max value	1.55 V
Procedure	Measure the voltage at P12 pin 2 Ensure that there are no short or long term voltages outside the limits
Measured value(s)	
Status	

8.1.2 Preamp

The preamp is added to the prototype to be able to scale the input signal so that the prototype can be used with various types of signal sources.

The functional tests are as follows.

Test #	4
Motive	Ensure that the Preamp functions properly
Module	Preamp
Tool	Oscilloscope, signal generator
Min value	500 mVp-p
Max value	100 mVp-p
Procedure	Connect the signal generator to P6. Adjust it to 70kHz sinusoidal, 150mVp-p. Adjust R44 and R45 to centre position. Measure the signal at P4 pin 5. Ensure that the signal is not outside the specified limits. Check that the amplitude changes when turning the potentiometers R44 and R45
Measured value(s)	
Status	

8.1.3 Variable gain amplifier and filter

The variable gain amplifier and filter block scales the signal according to the C-function set_gain(). This is to provide the ADC and the detector with a signal of the correct amplitude as well as setting the sensitivity of the receiver.

The functional tests are as follows.

Test #	5
Motive	Ensure that the Variable gain and filter block functions properly
Module	Variable gain and filter
Tool	Oscilloscope, signal generator, JTAGICEmkII
Min value	-
Max value	-
Procedure	Remove jumper at P4 when performing this test. Connect signal generator to P4 pin 2. Adjust the signal generator to 10 mVp-p, 70 kHz. Connect the Oscilloscope to P11 pin 3. Use the set_gain() function with the MCU to set the gain to 1, 10, 100, 200. Observe that the output signal is a 70 kHz sinusoidal and that the amplitude changes when changing gain.
Measured value(s)	
Status	

8.1.4 Reception detector

The reception detector provides a DC which is proportional to the amplitude of the output signal of the variable gain amplifier and filter. The analogue output of the detector can be used to estimate the signals amplitude. The block also features a digital “wake-up” signal with variable threshold.
The functional tests are as follows.

Test #	6
Motive	Ensure that the Detector block functions properly
Module	Reception detector
Tool	Oscilloscope, signal generator, JTAGICE mkII
Min value	-

Max value	-
Procedure	Remove jumper at P4 when performing this test. Connect signal generator to P4 pin 2. Adjust the signal generator to 10 mVp-p, 70 kHz. Connect the Oscilloscope to P11 pin 2. Use the set_gain() function with the MCU to set the gain to 1, 10, 100, 200. Observe that the output signal changes is a DC signal that changes value when changing gain.
Measured value(s)	
Status	

Test #	7
Motive	Ensure that the Detector block digital output functions properly
Module	Reception detector
Tool	Oscilloscope, signal generator, JTAGICE mkII
Min value	-
Max value	-
Procedure	Remove jumper at P4 when performing this test. Connect signal generator to P4 pin 8. Adjust the signal generator to 10 mVp-p, 70 kHz. Connect the Oscilloscope to P8 pin 2. Use the set_gain() function with the MCU to set the gain to 100. Use the set_threshold() function to set the threshold to 128. Vary the signal amplitude and observe that the output changes.
Measured value(s)	
Status	

8.1.5 USB circuit

The USB circuit is used to output debug information and data typically to a terminal program.

The tests are as follows:

Test #	8
Motive	Ensure that the USB circuit functions properly
Module	Connectors and headers
Tool	JTAGICEmkII, Terminal program
Min value	-
Max value	-
Procedure	Set all switches on S2 to the ON position.
	Connect the USB to a PC. Open a terminal program and select the associated COM-port. Use the usart_write_line() function to send a known string. Verify that the same string have been received with the terminal program.
Measured value(s)	
Status	

8.1.6 MCU and flash memory

The MCU is the main processing unit in the system and has many tasks and C-functions that must be tested. All C-functions not described in this section has been tested during the development of the code and can therefore be assumed to work as specified in the code comments and in chapter 7. The external components such as serial flash and external crystal will be covered by the following tests.

Test #	9
Motive	Ensure that the MCU crystal oscillators functions properly
Module	MCU and flash memory
Tool	JTAGICE mkII, Terminal program
Min value	-
Max value	-
Procedure	Connect the USB to a PC. Open a terminal program and select the Correct COM-port. Initialize the MCU to use the external high speed crystal as clock source and start the RTC. Write the clock variables to the USART indefinitely with delay between each send. Observe that the clock changes value in the terminal program.
Measured value(s)	
Status	

Test #	10
Motive	Ensure that the MCU flash functions properly
Module	MCU and flash memory
Tool	JTAGICE mkII, Terminal program
Min value	-
Max value	-
Procedure	Use the store_data_in_dataflash() function to store a page where all bytes are 0x55. Use the read_data_from_dataflash() function to read the data back. Check that all the data is 0x55, write "OK" to USART if OK, otherwise write "fail". Perform the same procedure again with the number 0x11.
Measured value(s)	
Status	

8.1.7 Complete system test

This section describes the most comprehensive and important functionality test. To verify that the system design concept works the system shall be tested with a standard Thelma fish tag and a passive hydrophone.

Test #	20
Motive	Ensure that the system concept works
Module	All
Tool	JTAGICE mkII, Terminal program, Thelma fish tag, passive hydrophone
Min value	-
Max value	-
Procedure	Connect the passive hydrophone to P6 input. Set the variable gain amplifier to 2000 using the set_gain() function. Set the threshold limit of the detector to 80 i.e 1 V. Enable rising flank external interrupt for the input connected to the digital output of the reception detecor. The interrupt service routine should be as described in Figure 8-1. Lower both the fish tag and the hydrophone into water with a spacing of 1-5 meters. Verify that the interrupt triggers and that the output has plausible values. Perform the same test with gain set to 100 000. Set the threshold limit to a reasonable value.

Measured value(s)	
Status	

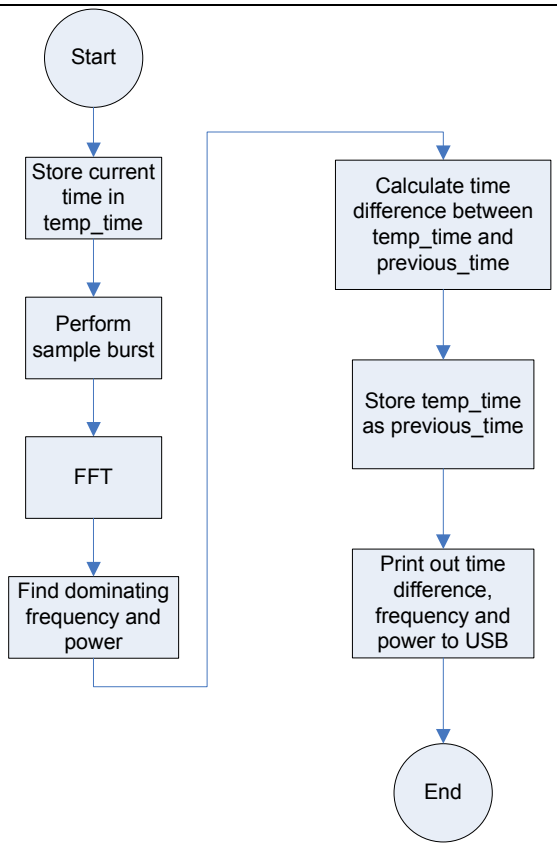


Figure 8-1 - Reception test interrupt handler float diagram

8.2 Performance tests and measurements

Performance tests will be carried out for the modules:

- Variable gain amplifier and filter
- Reception detector
- MCU and flash memory

The goal of the tests is to achieve enough data to be able to validate the prototype characteristics with respect to frequency response, low power, CPU performance, accuracy and correctness. The tests will be performed on a module by module basis to be able to pinpoint any weaknesses in the design and suggest improvements.

The following tests should be carried out after the prototype has passed all the functionality tests.

8.2.1 Variable gain amplifier and filter

The following procedures will test the variable gain amplifier and filter and provide data for further analysis.

Test #	11
Motive	Find the gain response curve of the variable gain amplifier
Module	Variable gain amplifier and filter
Tool	JTAGICEmkII, oscilloscope, signal generator
Procedure	Set the signal generator to generate a sinusoidal of 70 kHz, the amplitude should be as specified for each measurement. Remove the jumper at P4. Connect the signal generator to P4 pin 2. Connect the oscilloscope to P11 pin 3. Use the set_gain() function to set the gain as specified in the table. Set the signal generators amplitude as specified in the table. Note the output amplitude of the signal and calculate the gain.

Amplitude [mV]	**Gain**	**Measured amplitude [mV]**	**Calculated gain**
100	1		
100	2		
100	5		
100	10		
10	20		
10	50		
10	100		
1	200		
1	500		
1	1000		
0,1	2000		
0,1	5000		
0,1	10000		
0,01	20000		
0,01	50000		
0,01	100000		
0,01	140000		

Test #	12
Motive	Find the frequency response curve of the variable gain amplifier and filter
Module	Variable gain amplifier and filter
Tool	JTAGICEmkII, oscilloscope, signal generator
Procedure	Set the signal generator to generate a sinusoidal with a frequency as specified in the table. Set the amplitude to 200 mVp-p. Remove the jumper at P4. Connect the signal generator to P4 pin 2. Connect the oscilloscope to P11 pin 3. Use the set_gain() function to set the gain to 10. Note the amplitude of the output signal at the different frequencies.

Frequency [kHz]	**Output amplitude**
10	
20	
30	
40	
50	
52	
55	
58	

60
62
65
68
70
72
75
78
80
82
85
88
90
100
110
120

Test #	13
Motive	Measure internally generated noise with no input signal
Module	Variable gain amplifier and filter
Tool	JTAGICEmkII, oscilloscope, signal generator
Procedure	Remove jumper at P4. Place a jumper at P4 pin 7-8 Connect the oscilloscope to P11 pin 3. Use the set_gain() function to set the gain according to the table Measure the RMS value of the output signal at the various gain settings.

Gain	**Output [mVrms]**
1	
2	
5	
10	
20	
50	
100	
200	
500	
1000	
2000	
5000	
10000	
20000	
50000	
100000	
140000	

Test #	13
Motive	Measure internally generated noise with no input signal
Module	Variable gain amplifier and filter
Tool	JTAGICEmkII, oscilloscope, signal generator
Procedure	Remove jumper at P4. Place a jumper at P4 pin 7-8 Connect the oscilloscope to P11 pin 3. Use the set_gain() function to set the gain according to the table Measure the RMS value of the output signal at the various gain settings.

Gain	**Output [mVrms]**
1	
2	
5	
10	
20	
50	
100	
200	
500	
1000	
2000	
5000	
10000	
20000	
50000	
100000	
140000	

Test #	14
Motive	Measure the current consumption of the variable gain amplifier and filter
Module	Variable gain amplifier and filter
Tool	JTAGICEmkII, oscilloscope, signal generator
Procedure	Set the signal generator to generate a sinusoidal of 70 kHz, the amplitude should be as specified for each measurement. Remove jumper at P4. Connect the signal generator to P4 pin 2. Connect the oscilloscope to P5 pin 2, AGC current measurement output. Use the set_gain() function to set the gain as specified in the table. Set the signal generators amplitude as specified in the table. Note the output voltage of the signal and calculate the current consumption

Amplitude [mV]	**Gain**	**Measured voltage[mV]**	**Calculated current [mA]**
100	1		
100	2		
100	5		
100	10		
10	20		
10	50		
10	100		
1	200		
1	500		
1	1000		

0,1	2000		
0,1	5000		
0,1	10000		
0,01	20000		
0,01	50000		
0,01	100000		
0,01	140000		

8.2.2 Reception detector

The following procedures will test the reception detector and provide data for further analysis.

Test #	15
Motive	Find the response curve of the reception detector analogue output
Module	Reception detector
Tool	JTAGICEmkII, oscilloscope, signal generator
Procedure	Set the signal generator to generate a sinusoidal, the amplitude and frequency should be as specified for each measurement. Remove jumper at P4. Connect the signal generator to P4 pin 2. Connect CH1 on the oscilloscope to P11 pin 2. Connect CH2 to P11 pin 3. Use the set_gain() function to set the gain to 1. Note the output voltage of both the signals

Amplitude [mV]	**Frequency [kH<]**	**Measured amplitude (CH2) [mV]**	**Detector output [mV]**
0	60		
10	60		
20	60		
50	60		
100	60		
150	60		
200	60		
300	60		
500	60		
700	60		
1000	60		
1300	60		
1500	60		
1800	60		
2000	60		
2500	60		
0	70		
10	70		
20	70		
50	70		
100	70		
150	70		
200	70		
300	70		
500	70		

700	70		
1000	70		
1300	70		
1500	70		
1800	70		
2000	70		
2500	70		
0	80		
10	80		
20	80		
50	80		
100	80		
150	80		
200	80		
300	80		
500	80		
700	80		
1000	80		
1300	80		
1500	80		
1800	80		
2000	80		
2500	80		

Test #	16
Motive	Measure the current consumption of the reception detector
Module	Reception detector
Tool	JTAGICEmkII, oscilloscope, signal generator
Procedure	Set the signal generator to generate a sinusoidal of 70 kHz, the amplitude should be as specified for each measurement. Remove jumper at P4. Connect the signal generator to P4 pin 2. Connect CH1 on the oscilloscope to P5 pin 3. Connect CH2 to P11 pin 3. Use the set_gain() function to set the gain to 1. Note the output voltage of both the signals

Amplitude [mV]	**Measured amplitude (CH2) [mV]**	**Detector output [mV]**
0		
10		
20		
50		
100		
200		
500		
1000		
2000		

Test #	17
Motive	Find the step response curve for the reception detector
Module	Reception detector

Tool	JTAGICEmkII, oscilloscope, signal generator
Procedure	Set the signal generator to generate a sinusoidal of 70 kHz with an amplitude of 1000 mV. Set the scope to single trigger on CH1, input signal. Remove jumper at P4. Connect CH1 on the oscilloscope to P5 pin 3. Connect CH2 to P11 pin 3. Use the set_gain() function to set the gain to 1. Connect the signal generator to P4 pin 2 and store the response curve.

8.2.3 MCU and flash memory

The following procedures will measure the execution time and current consumption for the main functions of the MCU. The most important functions include storing data, functions used to receive data and digital signal processing. Execution times for some of the functions are already benchmark tested by Atmel in [28] and the datasheet [19] provides current consumption for active and static mode, hence the table only includes routines written especially for the prototype.

Test #	18
Motive	Find the execution time of the main CPU activities
Module	MCU and flash memory
Tool	JTAGICEmkII, oscilloscope
Procedure	Connect the oscilloscope CH2 to P8 pin 1. Use the oscilloscope to measure the execution time by setting GPIO0 before the routine and clearing the same pin after the routine: *AVR32_set_gpio_pin(GPIO0);* *set_gain(100);* *AVR32_clear_gpio_pin(GPIO0);* Perform this procedure on every routine in the table. Note the width of the positive pulse for each routine.

Action	**Execution time MCU [uS]**
get 64 samples	
get 256 samples	
Store flash page 512 byte	
set_gain()	
get_time()	
calculate_time_difference();	
find dominating frequency	

Test #	19
Motive	Find the current consumption of the MCU and flash in sleep mode with RTC
Module	MCU and flash memory
Tool	JTAGICEmkII, oscilloscope
Procedure	Connect the oscilloscope to pin 4 on header P5 Enable the RTC with external 32.000kHz crystal, enable pull-ups for all unconnected pins or pins usually set to output. Put the MCU in sleep mode deep stop. Measure the voltage with the oscilloscope and calculate the Current consumption.

9. Test results and measurement data

This chapter presents the results of the tests and measurements described in chapter 8. To provide a good basis for analysis the results are presented as graphs and figures. Please refer to appendix E for the raw data for each test. The c-code for each test is found in test_functions.c and test_functions.h in appendix 8 and B.

9.1 Accuracy of the results

The current measurements are carried out using a series resistor and a current shunt resistor amplifier with an amplification of 100. A TENMA 72-7235 150MHz oscilloscope is used for all measurements.

The tolerance of the measurements is calculated as follows:
Series resistor: 1%
Resistor in current shunt amplifier circuit: 499kΩ, 1%
Current shunt amplifier total output error: 2%.
Oscilloscope vertical accuracy: 3%

The worst case scenario error is calculated in Table 9-1.

Table 9-1 - Worst case measurement error

Device	Error	Comment
Series resistor	-1%	
Current shunt resistor	492.01/500 - 1 = -1.6%	100 gain is for 500k resistor
Current shunt amplifier output error	-2%	
Oscilloscope measurement error:	-3%	
SUM	**-7.6%**	

If we assume that noise and thermal effects are small and can be neglected, current measurements have a tolerance of 7.6%. For voltage measurements the tolerance is only dependent on the oscilloscope and is 3%. Time measurements are done by toggling a port pin. This will introduce a slight delay, but can be neglected due to the relatively long execution time of the routines/algorithms. The oscilloscope will introduce a tolerance of 0.01% for time measurements. A summary of the tolerances is given in Table 9-2.

Table 9-2 - Measurement tolerances

Measurement	Tolerance
Current	7.6%
Voltage	3%
Time	0.01%

9.2 Functionality test results

Functionality test #8 revealed one error in design; the TxD and RxD pins on the AT32UC3B where connected to the TxD and RxD pins respectively on the USB-UART bridge. Swapping these pins solves the problem and the test is passed. The schematics is updated to the correct setting.

Functionality test #20 passed. A sample of the terminal output is provided below.

```
*** Telemetry Buoy Running ***
Pulse detected:
 space: 00381 ms, 00000 s
Frequency: 00069 kHz
 Power: 06317


Pulse detected:
 space: 00460 ms, 00000 s
Frequency: 00069 kHz
 Power: 06341


Pulse detected:
 space: 00680 ms, 00000 s
Frequency: 00069 kHz
 Power: 06147


Pulse detected:
 space: 00520 ms, 00000 s
Frequency: 00069 kHz
 Power: 06244


Pulse detected:
 space: 00500 ms, 00000 s
Frequency: 00069 kHz
 Power: 06410


Pulse detected:
 space: 00460 ms, 00000 s
Frequency: 00069 kHz
 Power: 06218
```

The results are plausible. Figure 9-1 and Figure 9-2 shows oscilloscope measurement of the reception.

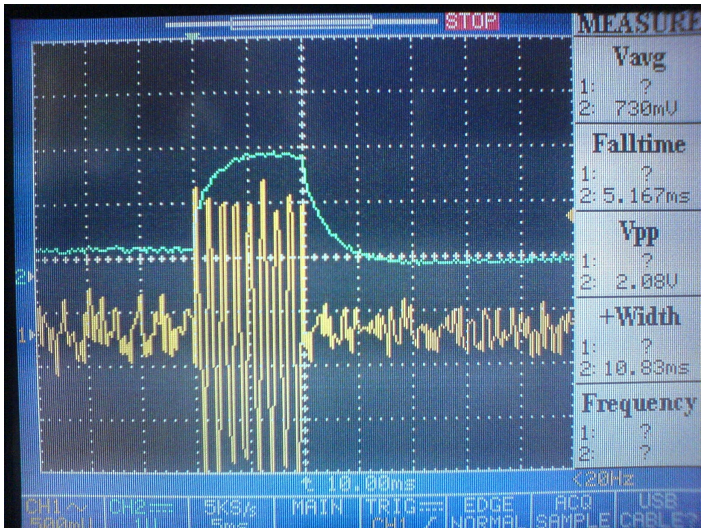


Figure 9-1 - Reception of real signal, CH1: variable gain amplifier output, CH2: Detector analogue output

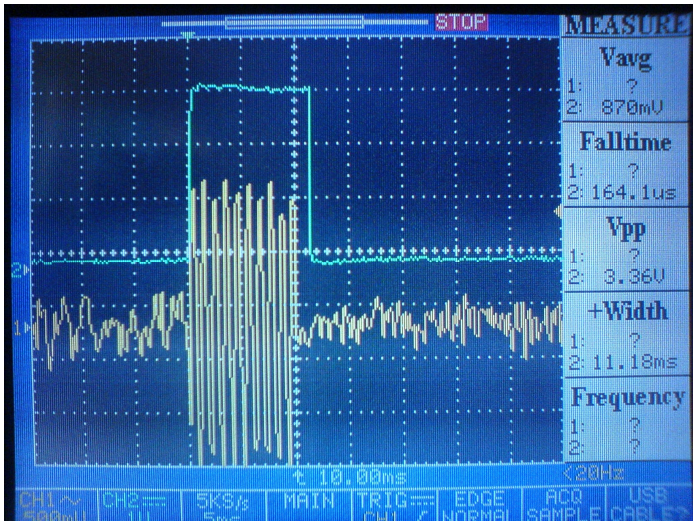


Figure 9-2 - Reception of real signal, CH1: variable gain amplifier output, CH2: Detector digital output

Increasing the amplification to 100 000 introduces much noise. The modification done to the original diode detector by inserting the extra resistor to suppress short term voltages makes the system function well even under noisy conditions. The reception works fine even with the extremely high gain. Both the timing of the pulses and the frequency measurement output is correct. The scope shot of the test with a gain of 100 000 is shown in Figure 9-3.

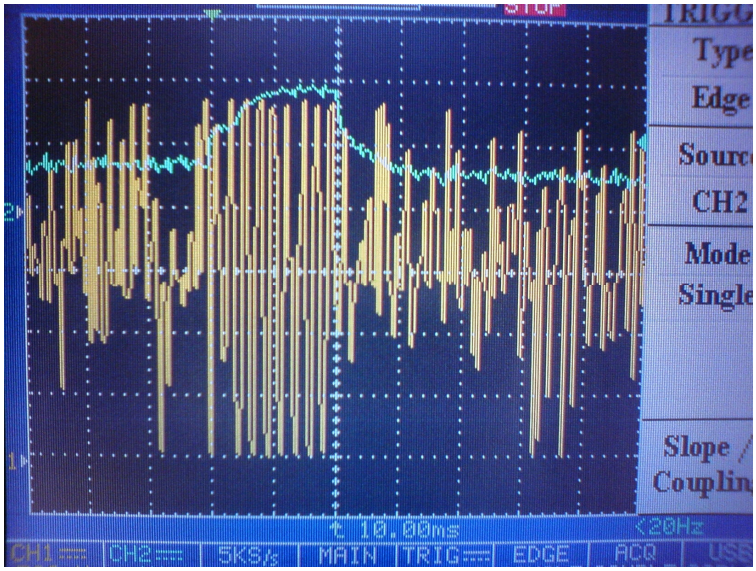


Figure 9-3 - Reception of real signal with high gain and high noise level, CH: variable gain amplifier output, CH2: Detector analogue output

The prototype passed all other tests without modification. Please refer to appendix E for details.

9.3 Performance test and measurement results

The performance tests are presented as diagrams. Please refer to appendix E for the raw data.

9.3.1 Variable gain amplifier and filter test results

Test #11: Find the gain response curve of the variable gain amplifier

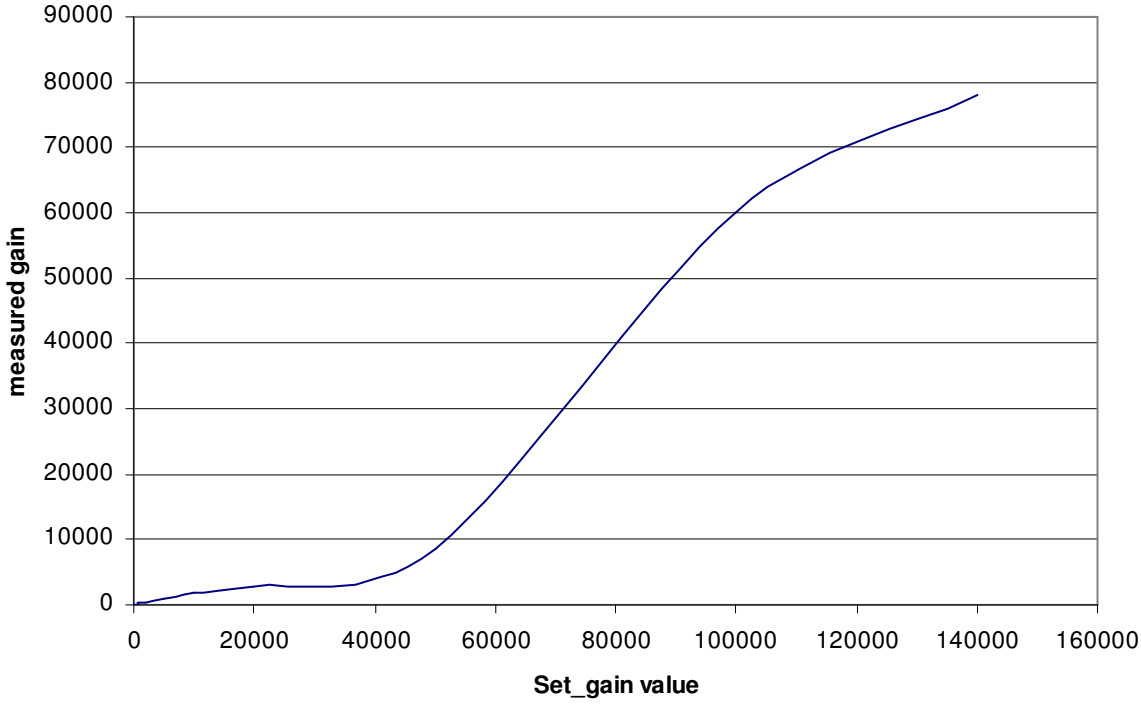


Figure 9-4- Gain response curve

Test #12: Find the frequency response curve of the variable gain amplifier and filter

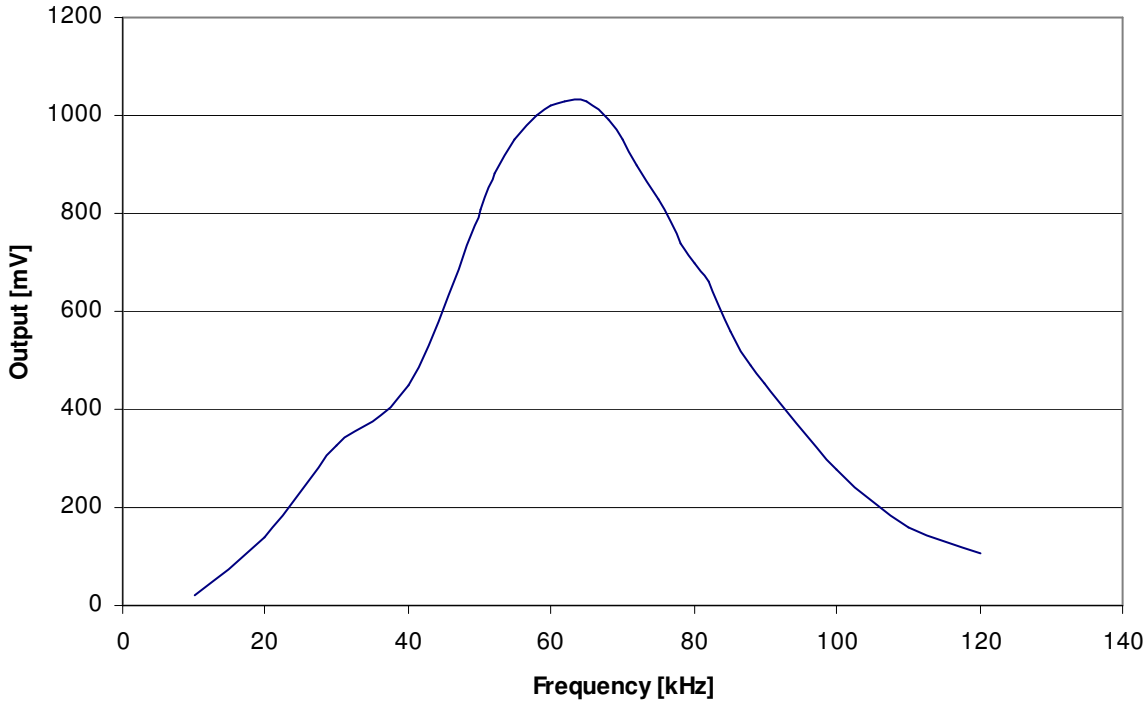


Figure 9-5 - Frequency response of the variable gain amplifier and filter

Test # 13: Measure the internal noise with no input signal

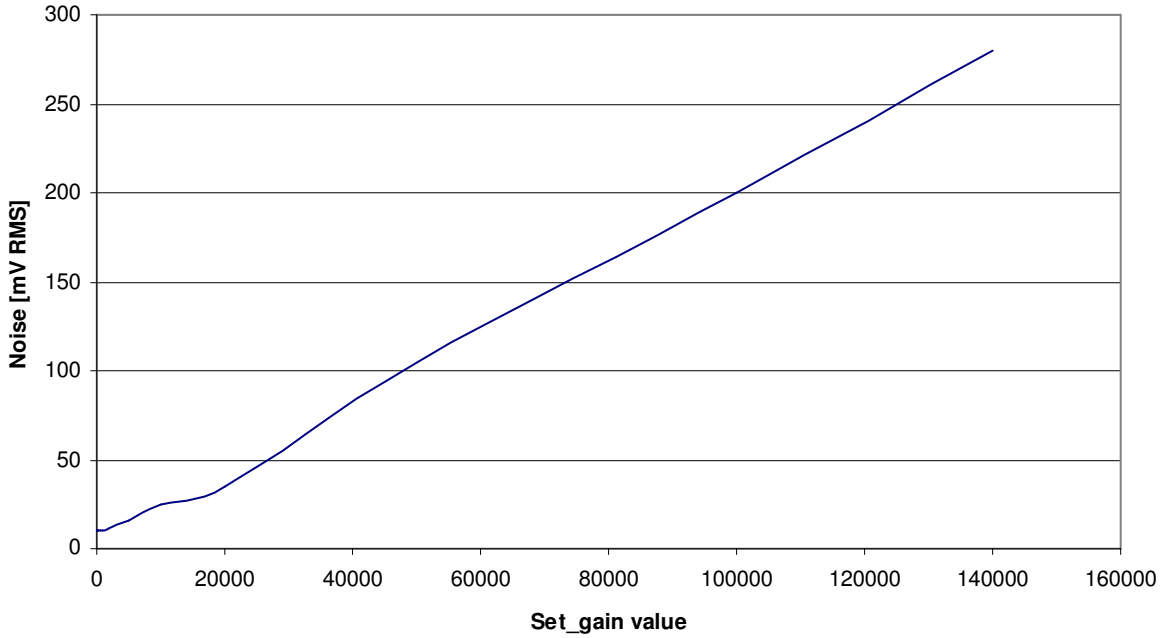


Figure 9-6 - Internal noise vs gain

Test #14: Measure the current consumption of the variable gain amplifier and filter

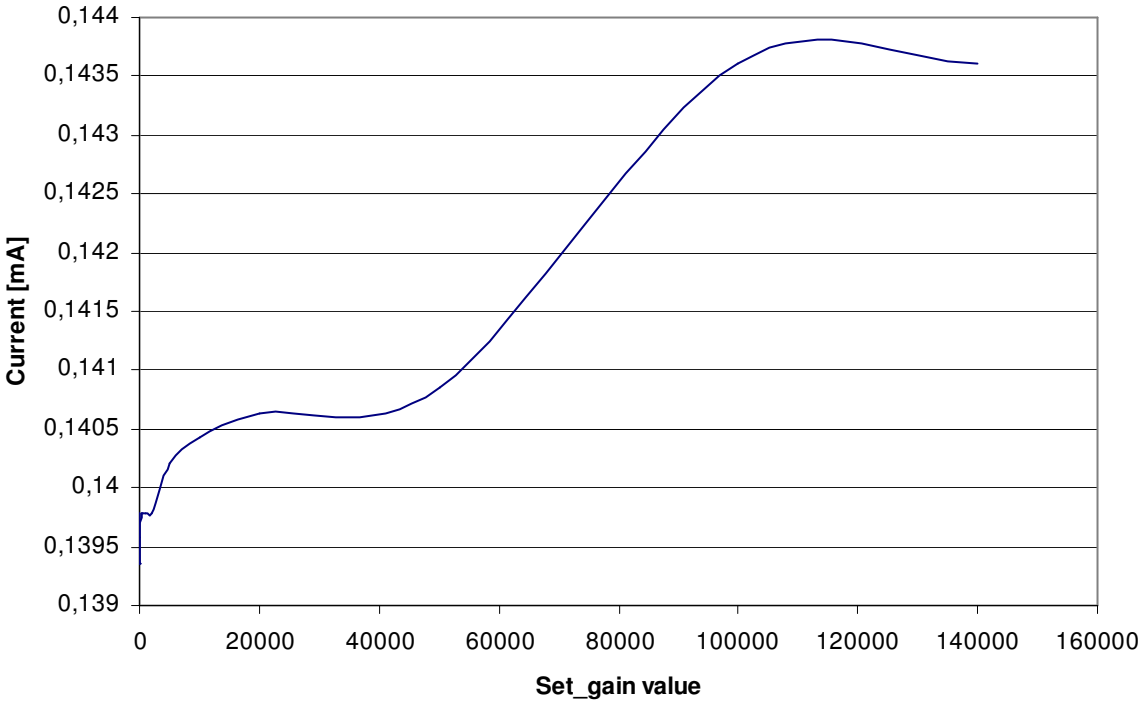


Figure 9-7 - Current consumption vs. set_gain value for the variable gain amplifier and filter

9.3.2 Reception detector test results

Test #15: Find the response curve of the analogue output of the reception detector

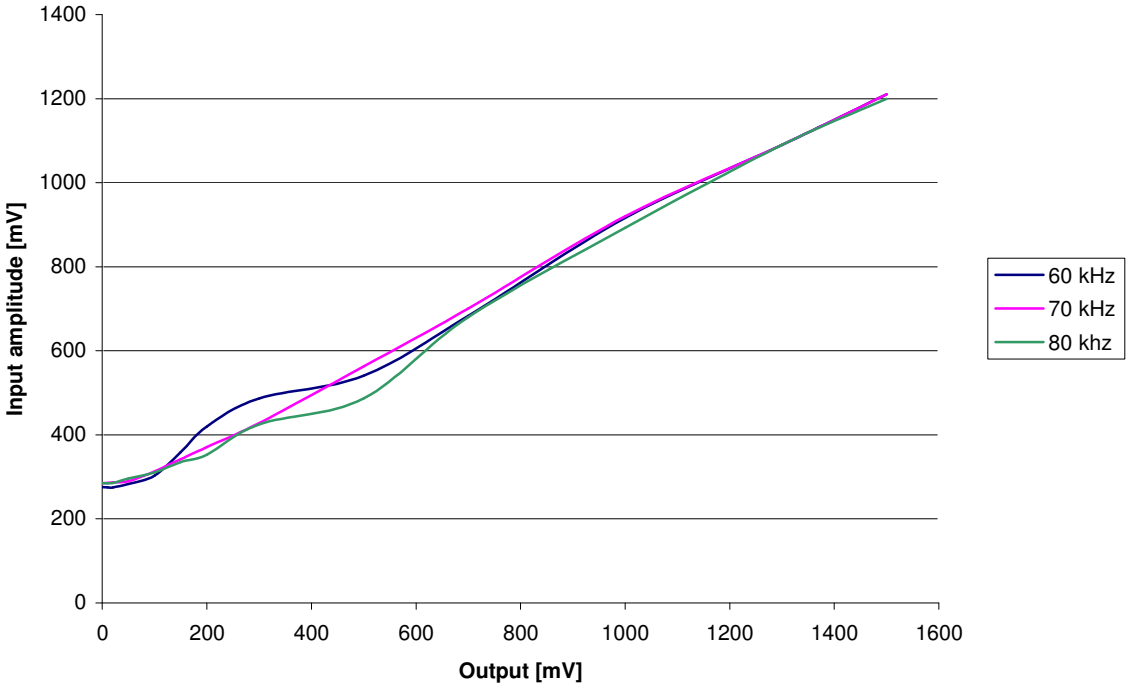


Figure 9-8 - Response curve for the reception detector for various frequencies

Test #16: Measure the current consumption of the reception detector

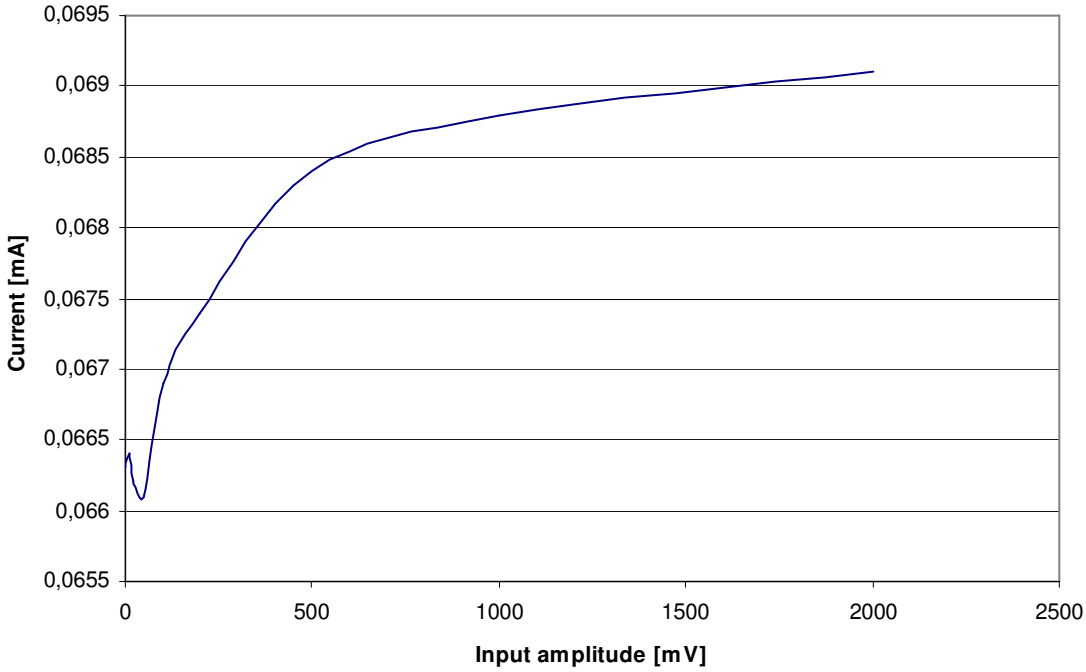


Figure 9-9 - Current consumption vs. input amplitude of the reception detector

Test #17: Find the step response curve for the reception detector

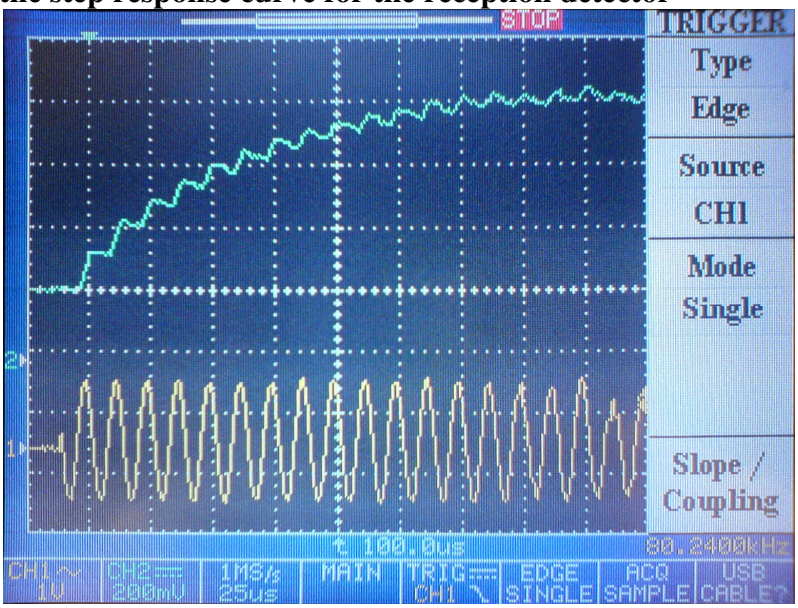


Table 9-3 - Reception detector response

9.3.3 MCU and flash memory test results

The execution time was tested with a CPU frequency of 60 Mhz.

Test #18 Find the execution time for the main MCU activities

Action	Execution time MCU [uS]
get 64 samples	1210
get 256 samples	4561
Store flash page 512 byte	1280
set_gain()	54,408
get_time()	4,9704
calculate_time_difference();	4,4936
find dominating frequency	439,92

Test #19 Find the current consumption of the MCU and flash in sleep mode with RTC

Measured voltage: 35.6 mV
Calculated current: 356 uA.

10. Discussion

This chapter discusses the measurement results and compares them to the analogue and digital design chapters. The measurements will first be discussed module by module while the last section will discuss the complete system.

10.1 Analogue design

The analogue design in general fulfils its low power demands with good margin. The functionality is somewhat as expected although there are some adjustments that should be done to provide optimal characteristics. The following sections describe this in detail for each circuit type.

10.1.1 Band pass filter

The filter acts as simulated with the correct crossover frequencies. It is though required to investigate if this filter has the characteristics needed for the telemetry buoy application.

The design of the filter is a compromise of using amplifiers with low gain-bandwidth product and hence low power consumption and filter characteristics. Using the measured frequency response we plot the first positive and the first negative alias for the chosen sampling frequency of 58 kHz. The result is shown in Figure 10-1. We clearly see that due to the non-ideal filter characteristics the result is an overlap between the first positive and the first negative alias. It follows that a high power signal with a lower frequency than the high pass crossover frequency might be mistaken for a signal inside the pass band. It is therefore important that either the filter is improved by increasing the flank of the analogue filter or by increasing the sampling frequency.

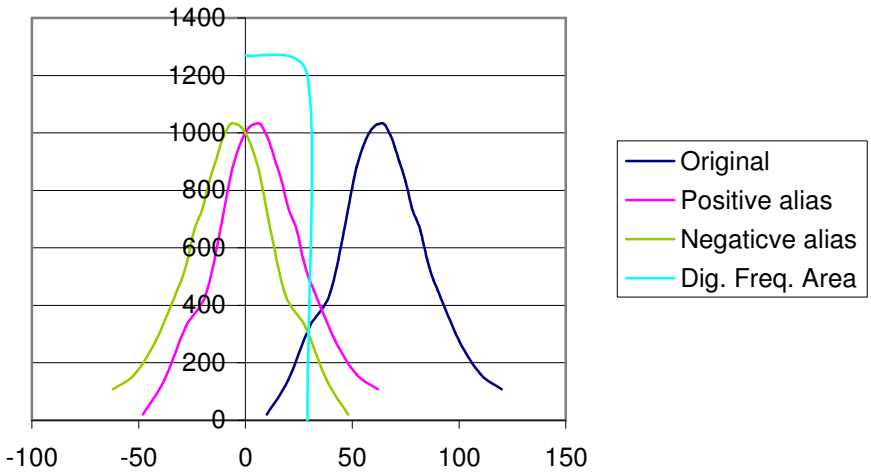


Figure 10-1 - Plot of aliases for fs = 58 kHz

The allowable sampling frequencies are listed in Table 2-1 on page 5. Choosing a sampling frequency of 90 kHz will result in less overlap as seen in Figure 10-2. It is important to notice that although there is an overlap much of these frequencies can be filtered out digitally when using a higher sampling frequency and this may not represent any problem.

The discussion of whether the improvements should be done in the analogue domain or in the digital domain is covered in section 10.3 on page 90.

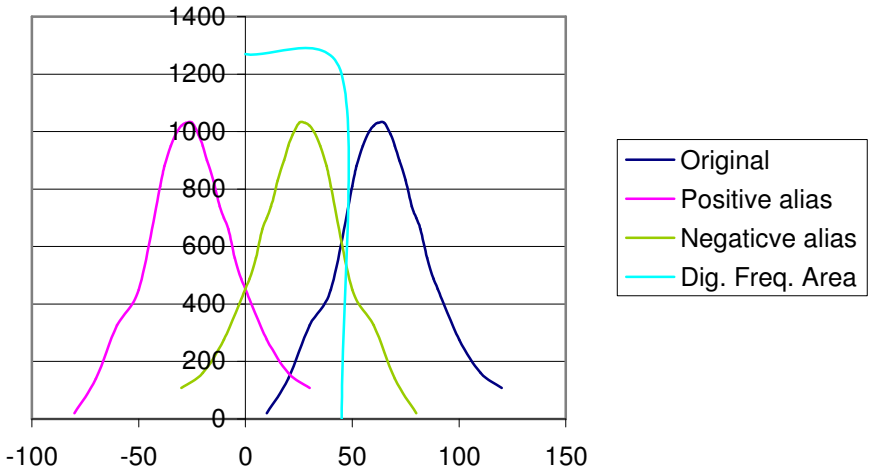


Figure 10-2 - Plot of aliases for fs = 90 kHz

10.1.2 Variable gain amplifier

The variable gain amplifier functions well. Together with the filter it only consumes a current of about 300 uA. This is exceptionally low compared to the alternative variable gain amplifier ICs which would require about three AD600 each consuming 14 mA to provide the same amplification.

Some adjustments are though needed to get optimal behaviour. We see from Figure 9-4 that the gain is non-linear. This can be compensated for in firmware, but this might require that an individual compensation table is constructed for each produced unit. It is desired to avoid this if possible to ease the production time and cost.

When testing the variable gain amplifier the high gain forces the use of very small input signals which cannot be measured with an ordinary oscilloscope. Analysis will therefore have to be based on theories not verified by measurements.

It is the author's conviction that the root of the problem is the generated signal ground. The signal ground SGND is generated as shown in Figure 10-3.

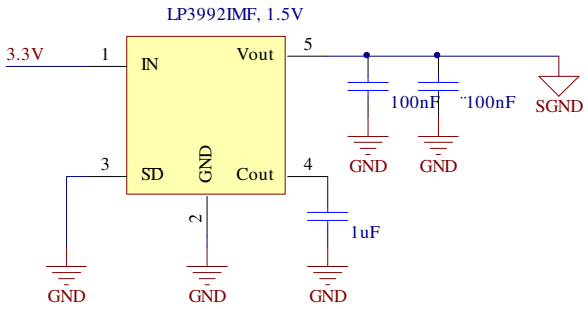


Figure 10-3 - SGND generation

Since the SGND power is not ideal both noise and signal can occur at the top of the signal ground. This can lead to unwanted feedback in the amplifier. As an example we consider the first and the last amplification segment of the variable gain amplifier as shown in Figure 10-4.

If the signal routed to SGND through resistors connected to the output of the operational amplifier and SGND, is not completely removed by the regulator and the decoupling this may have impact the variable gain amplifier performance. According to the phase of the output signal the feedback may be negative or positive. We also notice that the effect of this feedback is dependent on the wiper position of the digital potentiometer. This may be the reason for the non-linear behaviour. Note that this is only an example of one feedback. If the SGND does not suppress all signals, these feedbacks will occur in all amplifier segments, both in the filter and the variable gain amplifier.

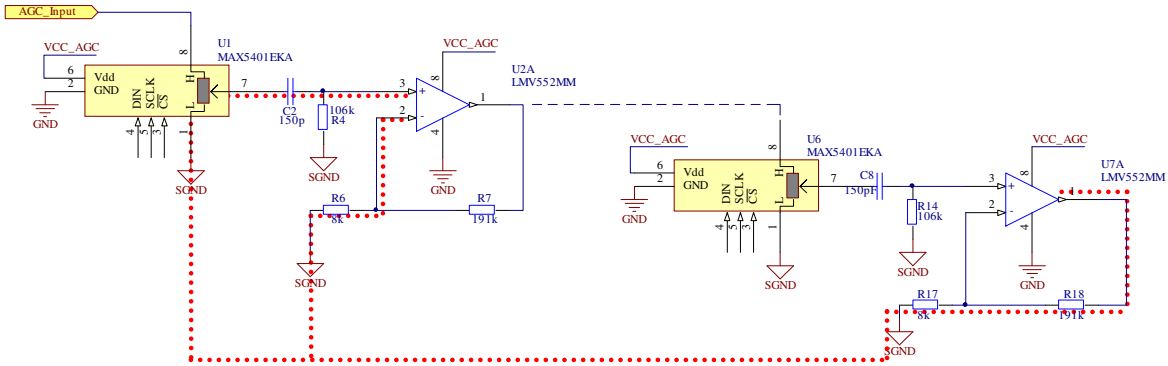


Figure 10-4 - Unwanted feedback example

To analyse this behaviour further we assume the following scenario:

- The output signal of U7A is a 70 kHz sinusoidal with an amplitude of 1V
- The voltage regulator has not the sufficient bandwidth to suppress the signal routed to SGND through R17 and R18.
- The SGND is only decoupled with 1uF
- No other feedbacks occur

The output of the last amplification segment will result in a voltage divider as in Figure 10-5 where Xc is the equivalent signal resistance of the 1 uF decoupling capacitor at 70 kHz.

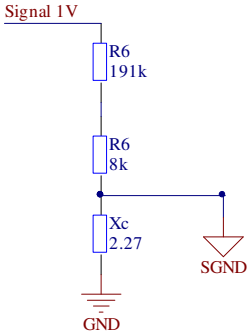


Figure 10-5 – Signal equivalent voltage divider

The result is a signal voltage on the SGND with an amplitude of 11.41 uV. The maximum gain of the amplifier is about 140 000. It follows that the worst case scenario leads to a feedback resulting in an unwanted output voltage of

$$V_{out} = 11.4 \cdot 10^{-6} \cdot 140000 = 1.59V \qquad (10.1)$$

Note that this is not accurate numbers, but only an example to emphasize the importance of the design of the SGND circuit.

Depending on the phase of the output signal may result in a negative or positive feedback. If the unwanted feedback results in a dominating positive feedback, the circuit may start to

oscillate and produce noise. The measurements of test #13 show a great deal of self-generated noise at high gain settings. It is believed that this is a result of the same effect as the latter example. We notice that the noise is increasing with the gain setting. With low gain setting the wiper Figure 10-4 will be at the low end of the potentiometer thus resulting in about the same signal at the negative and positive input of the operational amplifier. It follows that the CMRR will suppresses the noise signal. At high gain settings the wiper will be at the top position of the potentiometer thus resulting in different feedback factors for the negative and the positive input. It follows that the effect of the unwanted feedback signal is greater.

It is therefore believed that a great improvement in both noise and linearity characteristics of the variable gain amplifier can be achieved by

- Selecting a high bandwidth – low noise regulator
- Adding additional decoupling of the SGND
- Placing the regulator close to the amplifier
- Using copper planes rather than wires to reduce voltage drops in the PCB.

10.1.3 Reception detector

The reception detector provides a well functioning analogue and digital output while consuming less than 70 uA. The output voltage is somewhat lower than the simulations, but the output is as good as frequency independent and linear in the area 60 – 80 kHz.

Due to its ultra low power consumption the circuit could be duplicated and used to measure the signal amplitude with a fixed pre-amplification to provide a forward coupling in the AGC control loop and hence a faster system response. The system then becomes:

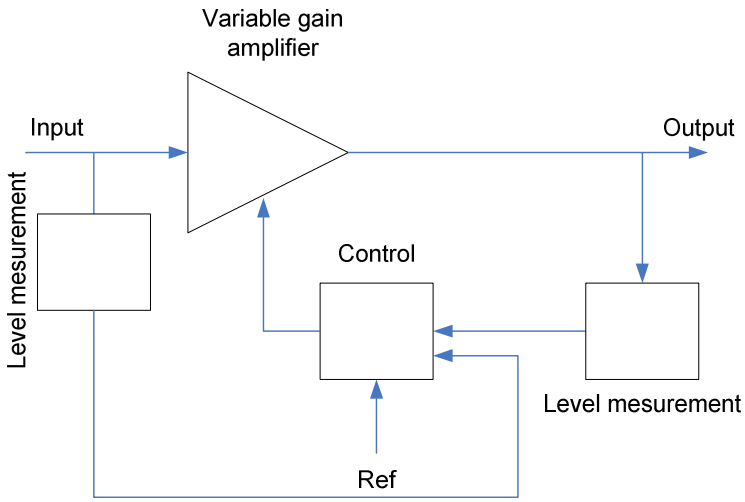


Figure 10-6 - AGC control loop with forward coupling

The complete system test given in test #20 shows that the reception detector performs remarkably well even with a relatively high level of noise. The sea will though introduce new noise sources like engine propellers, waves, bubbles, etc. It is therefore required that the circuit is tested in situ with a complete system to adjust the component values for optimal behaviour.

10.2 Digital design

The digital design of the telemetry buoy project was mainly defined in the report *Acoustic telemetry buoy* [4]. It presents calculations that support the use of digital signal processing in the application. The following section will similar calculations mainly based on real measurements rather than the assumptions made in [4].

10.2.1 MCU and Flash memory

The diagram for a typical reception is shown in Figure 10-7. It does not include all details, but provides an estimate for the execution time for a typical reception.

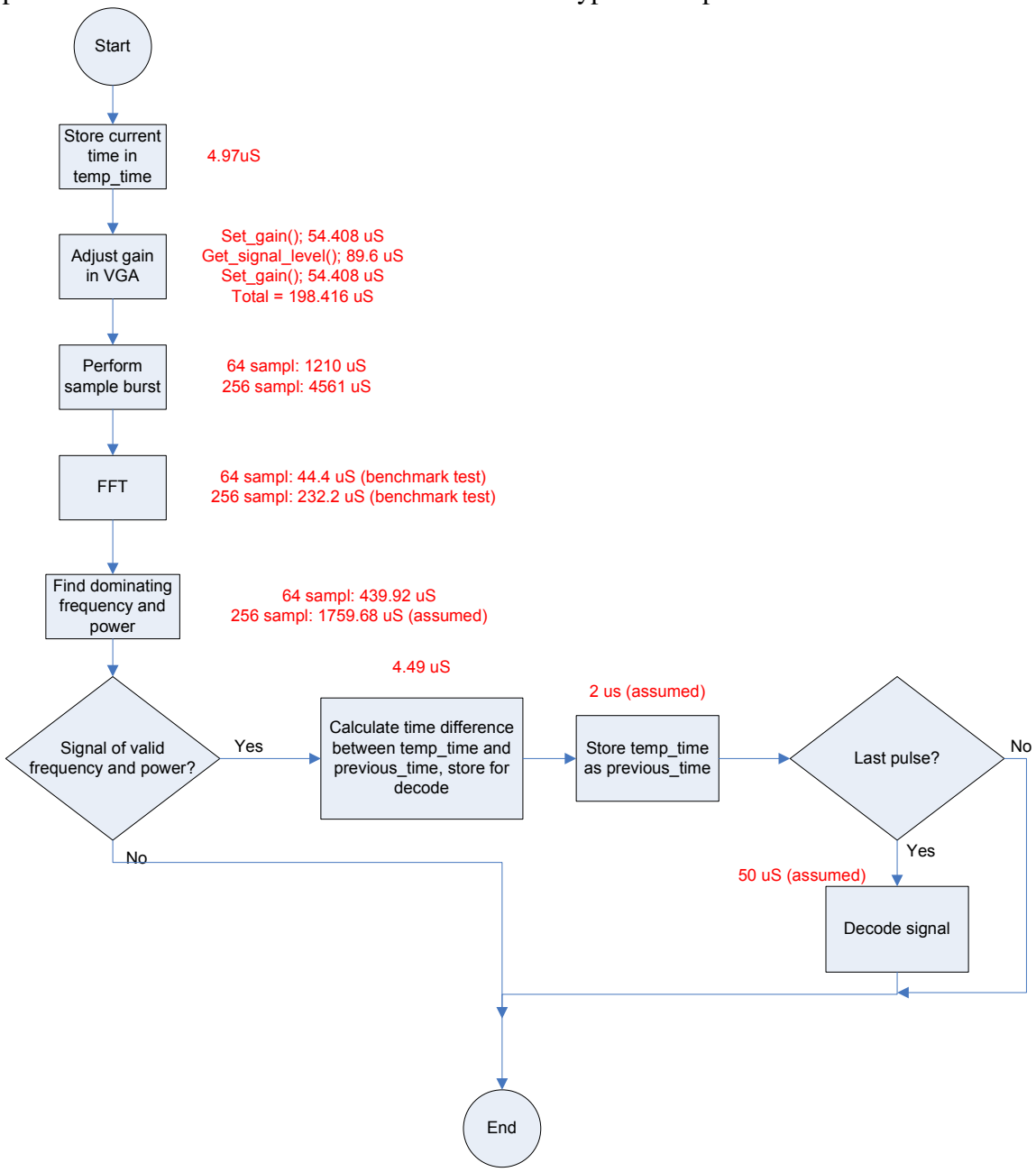


Figure 10-7 - Reception float diagram

An example of using 256 point sample bursts with a 256 point FFT is given in Table 10-1. The signal to be received is the 6 pulse example provided in section 4.1.1 on page 11.

Table 10-1 - Total receive active execution time

Pulses:	**6**		
Routine	**Execution time [uS]**	**Executions per reception**	**Total [us]**
get_time	4,97	6	29,82
set gain	54,408	12	652,896
get_signal_level	89,6	6	537,6
Do_sampling_burst	4561	6	27366
FFT	232,2	6	1393,2
Find_dominating_freq	1759,68	6	10558,08
Caclulate_time_diff	4,49	6	26,94
store temp time	2	6	12
Decode signal	50	1	50
		Total:	**40626,536**

The measured execution times are based on compilation with no optimization. With speed optimization the total execution time will decrease considerably.

Test #19 shows a current consumption of 356 uA with the MCU in deep stop sleep mode with the RTC enbled. This represents the static current consumption of both the MCU and the external dataflash when not receiving.

In addition to receiving and running the RTC the MCU will have to store the values to the external dataflash and adjust the reception sensitivity by adjusting the gain of the variable gain amplifer and the reception detector threshold for the interrupt line.

A sensitivity adjustment will have the active execution time as specified in Table 10-2.

Table 10-2 - Sensitivity adjustment active execution time

Routine	**Exection time [uS]**	**Executions per adjustment**	**Toal [us]**
set gain	54,408	1	54,408
get_signal_level	89,6	1	89,6
		Total:	**144,008**

Storing one page of data to the dataflash will result in an additional active time of 1280 uS for the microcontroller. In addition the dataflash will consume more current when storing this page from the receive buffer to the actual flash memory. The total energy consumption for the digital design per year is calculated in Table 10-3. It is assumed a 1:10 ratio between correct and false receptions due to noise. Furthermore it is assumed that storing a reception with ID, data and time stamp requires 72 bytes of data. All current numbers, except the deep stop with RTC, are collected from the datasheets [19] and [21].

Table 10-3 - Digital design energy consumption per year

Action	Active time [uS]	Executions per year	Current [mA]	Voltage	Energy per Year
Receive	40626,536	100000	23,5	3,3	315,0587867
False receive	6756,266	1000000	23,5	3,3	523,9484283
deep stop with RTC	3,1536E+13	1	0,356	3,3	37048,4928
sensitivity adjustment	144,008	31536000	23,5	3,3	352,1883841
Store page	1280	14062,5	23,5	3,3	1,3959
Flash store from buffer	6000	14062,5	17	3,3	4,7334375
				Total:	**38245,81774**

The results of the calculations will be used for further analysis in the following section.

10.3 Combined design

The prototype system design concept has been proven to work with existing tags in test #20. Besides this test the most important factor of the telemetry buoy application is low power consumption. The specifications provided in section 1.1 on page 1 lists the criteria of one year operation and a minimum of 100 000 receptions. The report *Acoustic telemetry buoy* [4] provides a calculation showing that the total battery energy is about 163800 J. For this to be correct the system must be able to operate at a voltage of 2 – 3.7 V. The following calculations are based on the assumption that a buck-boost regulator with an average power loss of 1 mW is available and used in the final product. The total energy consumption per year is shown in Table 10-4.

Table 10-4 - Total system energy consumption per year

Module	Avarage current [mA]	Voltage	Power consumption [W]	Energy per year [J]
Voltage regulator			0,001	31536
VGA and filter	0,304	3,3	0,0010032	31636,9152
Reception detector	0,07	3,3	0,000231	7284,816
MCU and Flash				38245,81774
			Total:	108703,55

The lifetime of the telemetry buoy design is therefore

$$T_{lifetime} = \frac{163800\ J}{108704\ J} = 1.51\ years \qquad (10.2)$$

This number includes 100 000 receptions per year and will increase if optimisation is applied to the compilation.

Section 10.1.1 addresses the issue that either the analogue filter should be improved or the sampling frequency should be increased to avoid unwanted aliasing. It is also important to ask the question if increased performance should be a result of analogue or digital signal processing. In any low power application an increase in performance comes with a cost of added power consumption. It is therefore important to map the systems power consumption in order to decide where the added performance comes with the lowest cost.

Figure 10-8 shows a pie chart of the energy consumption distribution. The raw data for the pie chart is found in appendix B. We clearly notice that the assumed loss in the voltage regulator, the variable gain amplifier and filter and the microcontroller sleep mode with RTC accounts for the majority of the energy consumption whilst the digital part of the reception is barely noticeable. It is therefore obvious that the complexity of the reception algorithm with

sampling and all digital signal processing can be substantially increased without any noticeable increase in overall lifetime of the telemetry buoy.

It can therefore be concluded that in order to reduce the overall power consumption the complexity of the analogue filter can be reduced while increasing the sampling frequency, applying digital filters and FFT algorithms in the microcontroller firmware. It should therefore be considered to apply the standard Nyquist sampling theorem using a sampling frequency of at least twice of the highest frequency component. Note that this assumption only yields when only sampling at reception.

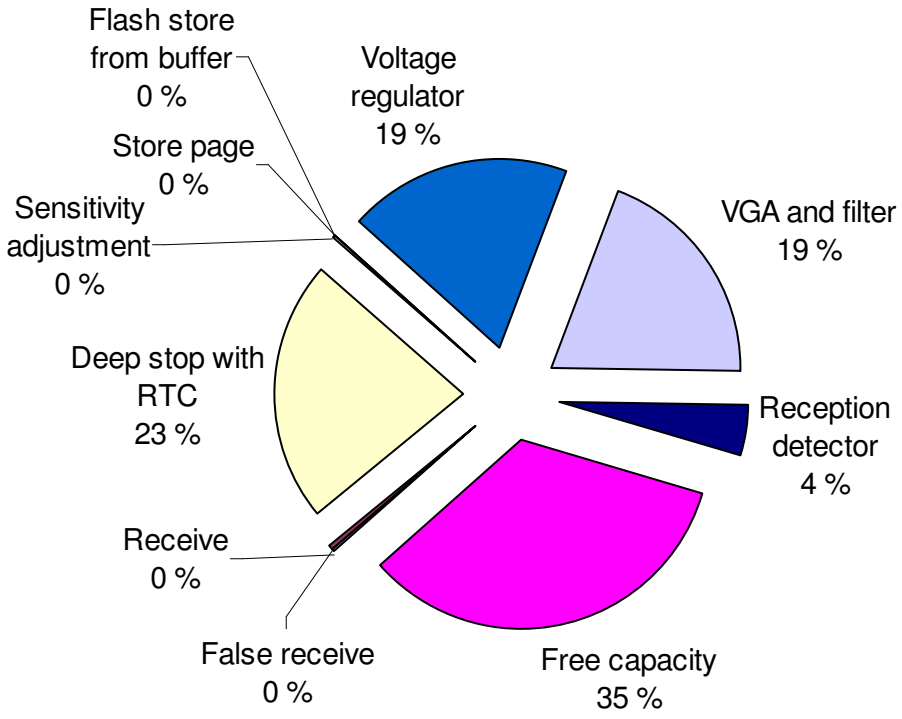


Figure 10-8 - Energy consumption distribution for one year system operation

Another important factor in a combined analogue and digital design is noise. Noise generated by the digital circuits can be a major problem in many applications. For this design it has not been identified any noise influence on the analogue circuits as a result of digital activities. The noise level was the same at the variable gain amplifier and filter output with the MCU disconnected from its power source and connected. The SPI did not introduce any noticeable noise. This is believed to be a result of the separation and space between the digital and analogue circuits as well as routing the digital tracks on one layer, analogue on a different layer with the GND plane in between acting as a shield.

11. Further development

This chapter discusses the further development of the telemetry buoy, changes that should be made, suggestion of components, added functionality and more.

11.1 The choice of microcontroller

At the end of this project Atmel launched a new UC3 microcontroller; the AT32UC3L. This device has even lower power consumption than the former UC3 microcontrollers and new power saving features such as an event system, ultralow power RTC timer and the ability to operate at voltages down to 1.6V.

If we consider Figure 10-8 we notice that one of the most power consuming tasks is having the RTC enabled when in deep stop. The datasheet for the AT32UC3L [22] specifies that having the microcontroller in Shutdown mode with the RTC timer running with an external crystal will consume only 1.5 uA. The same energy calculations is performed as in section 10.3, but using the power specifications for the AT32UC3L. It is also worth to notice that the UC3L operates at a somewhat lower system frequency of 50 MHz. The execution times have therefore been scaled from 60 MHz to 50 MHz. The resulting power estimation for the MCU activities for VCC = 3.3 V is presented in Table 11-1.

Table 11-1 - Energy consumption for the main MCU activities

Action	Active time at 50 MHz [uS]	Executions per year	Current [mA]	Voltage	Energy per Year
Receive	58502,21184	100000	15	3,3	289,5859486
False receive	8107,5192	1000000	15	3,3	401,3222004
Shutdown with RTC	3,1536E+13	1	0,015	3,3	1561,032
Sensitivity adjustment	207,37152	31536000	15	3,3	323,7135786
Store page	1536	14062,5	15	3,3	1,0692
Flash store from buffer	7200	14062,5	17	3,3	5,680125
				Total:	**2582,403053**

This is a reduction down to 6.7% of the power consumption for the AT32UC3B, mainly due to the extremely low power consumption of the RTC. The UC3L can operate at voltages down to 1.6V. This means that a lower system voltage can be used if the analogue design is adapted to this voltage. This will lead to a substantial increase in the overall battery lifetime.

The overall energy consumption of the system with the UC3L is presented in Table 11-2.

Table 11-2- Overall system energy consumption using the UC3L

Module	Avarage current [mA]	Voltage	Power consumption [W]	Energy per year [J]
Voltage regulator			0,001	31536
VGA and filter	0,304	3,3	0,0010032	31636,9152
Reception detector	0,07	3,3	0,000231	7284,816
MCU and Flash				2582,403053
			Total:	73040,13

Using the new energy consumption of the MCU the battery lifetime can be calculated for the telemetry buoy

$$T_{lifetime} = \frac{163800\ J}{73040\ J} = 2.24\ \ years \tag{11.1}$$

The energy consumption distribution chart from section 10.3 with the energy calculations for the UC3L is shown in Figure 11-1.

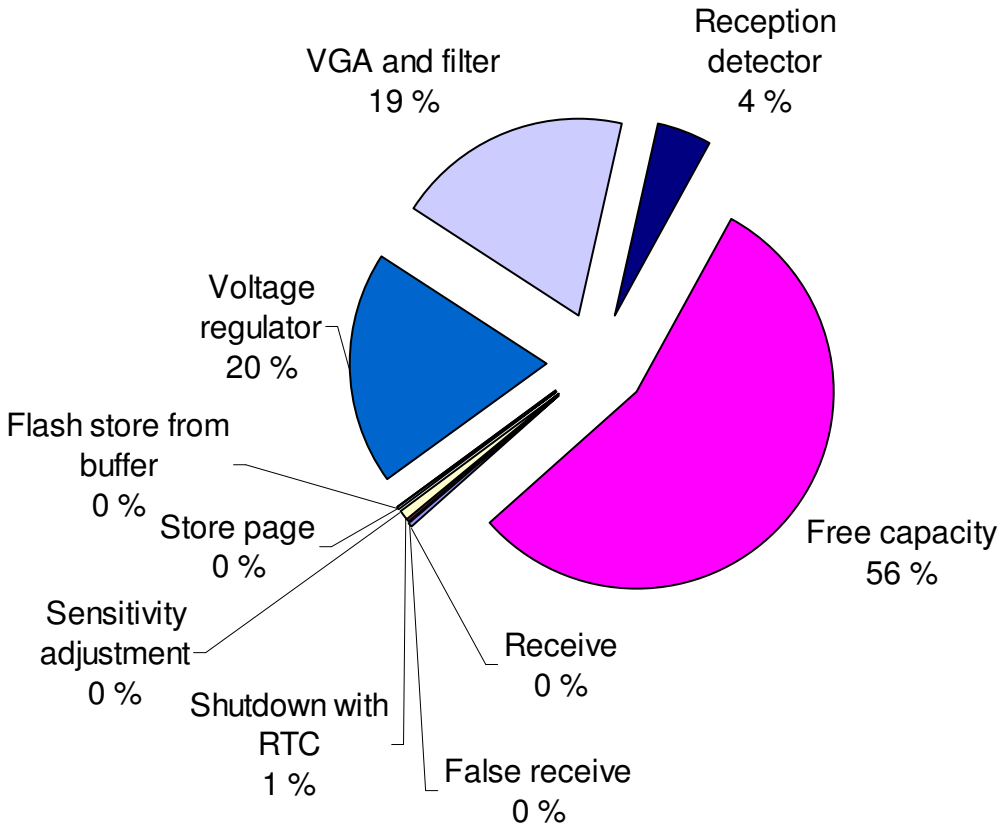


Figure 11-1 - Energy consumption distribution for one year operation using the UC3L

We see that the voltage regulator and the variable gain amplifier and filter accounts for largest energy consumption. Lowering the system voltage can help reduce the VGA and filter section. We also notice that the selection of the voltage regulator is one of the most important tasks with respect to power consumption when designing the final product.

11.2 Design improvements

Chapter 10 discusses the performance of the prototype design of the telemetry buoy. One important design factor for improving the design must be to eliminate the occurrence of aliases. In section 10.3 it is concluded that the performance improvement should mainly be done in the digital domain by increasing the sampling frequency and adding digital filtration. This will not have great impact on the power consumption, but will considerably increase the reliability and performance of the system. Using the UC3L microcontroller the designer

should also consider if the bandwidth of the filter should be increased to support a wider range of modulation schemes such as multiple frequency shift keying in a wider frequency band.

Section 10.1.2 provides a list of ways of improving the performance of the noise and gain linearity of the variable gain amplifier. These improvements to the signal ground should be considered when designing a new prototype. It has though been shown that the prototype functions well both with the gain set to 2000 and to 100 000. The noise is suppressed in the reception detector and the correctness of the gain may not be as an important factor as previously assumed. A too high gain will indeed cause saturation, but the high level of analogue filtration rebuilds the signal to a sinusoidal and the FFT algorithm is able to calculate the correct frequency of the original signal.

11.3 Additional functionality

The prototype was designed to test the design concept of the telemetry buoy. Only a USB debug interface was implemented. The final design will have to include some sort of user interface. Whether this is GSM, Bluetooth, USB or any other interface will be up to the designer and the market demand. Whichever interface is chosen, a general low level interface to the receiver should be defined to easily be able to develop new user interfaces using this low level communication layer to the receiver.

The algorithm for the automatic gain control is not implemented, but it should be a relatively easy task considering that the gain does not need to be perfectly adjusted to receive the correct signals. The algorithm for setting the threshold is also missing. The idea with the design is to make this an adaptive solution where the threshold limit is increased for every false reception and decreased for every defined period of time of no reception. This algorithm will stabilise the ratio between false and correct receptions.

The telemetry buoy is to work for at least one year without human intervention. It is therefore necessary to apply functions to handle events such as deadlock, program counter corruption or other unforeseen problems. The most effective tool for this is the use of a watchdog timer. The UC3 series of microcontrollers has this module inbuilt and it should be utilized.

The RTC is implemented as a simplified clock. A complete RTC with set functions should be added. This is an easy task to implement and Atmel provides this algorithm in the application note "*AVR134: Real-Time Clock using the Asynchronous Timer*"**.**

The AT32UC3 series offer a wide range of microcontrollers with high performance and it should not be a problem to select a microcontroller of sufficient capacity for handling the additional functionality.

11.4 Development of new modulation schemes

As described in section 1.3 it has been found that modulation schemes using pulse position and frequency are the most suitable for underwater acoustic communication. At the time of writing the dominating modulation scheme for digital fish tags is differential pulse position (DPPM). The system design is prepared for easy implementation and support for frequency modulation schemes through the powerful Atmel DSP library with optimized FFT algorithms for the UC3 series of microcontrollers.

To emphasize the advantage of using frequency as a modulation, this section will present a suggestion for a new modulation scheme using a combination of DPPM and MFSK (multiple shift keying) especially designed for fish telemetry. The new algorithm will greatly increase the throughput of the transmission. This demonstrated with an example at the end of this section.

As an example a total of twelve different frequencies are used. These are divided into two categories where eight frequencies are main frequencies used to code three bits of data and the last four are backup frequencies. We have

$$\begin{aligned} f_{1..8} &\Rightarrow main\ frequencies \\ f_{9..12} &\Rightarrow backup\ frequencies \end{aligned} \tag{11.2}$$

Furthermore we define the DPPM to have a total of four time slots per pulse coding a total of two bits per pulse. As described in section 4.1.1 on page 11 the DPPM scheme requires a guard time between pulses to ensure that the multipath distortion does not disrupt the transfer. The guard time is only needed between pulses of the same frequency. The following rule therefore applies

1. No pulse of the same frequency can be repeated until the guard time of this frequency has expired.

The frequency is though used to code a defined bit pattern, to still be able to code this pattern we define the second rule

2. During the guard period of frequency n, the free backup frequency m of the lowest order will be used as a substitute coding the bit pattern of frequency n until the guard time for frequency n has expired. The backup frequency m will not be free for this period of time.

It follows that the amount of backup frequencies will limit the amount of data that can be transferred until the number of backup frequencies exceeds the guard time divided by the time slot length in the DPPM scheme.

To be able to detect collisions a third rule is defined

3. A transfer is always initiated with frequency f_1.

The last rule also applies to the Vemco standard format. The rising flank of the pulse is a result of the shortest travel distance from the transmitter to the receiver and is not distorted by multipath.

4. All time measurements are referred to the rising edge of the pulse. All pulses except the first pulse occur in the middle of the timeslot.

The last rules are needed for formality reasons.

5. The bits coded by the DPPM schemes are calculated before the MFSK coded bits.
6. Bits are sent with MSB first.

The following presents an example where $f_1 = 000_2$, $f_2 = 001_2$, …, $f_8 = 111_2$. Timeslot $T_1 = 00_2$, $T_2 = 01_2$, $T_3 = 10$, $T_4 = 11_2$.

In the example the data 00011110_2 followed by a CRC 00000101_2 is sent. We assume the receiver knows the number of bits to be received.

The generation of the transmission details is as follows

1. First pulse is always f_1

2. Data[7:6] = 00_2 → pulse 2 in slot T_1
3. Data[5:3] = 011_2 → pulse 2 if of frequency f_4, f_9 represents 000_2
4. Data[2:1] = 11_2 → pulse 2 in slot T_4
5. Data[0] + CRC[7:6] = 000_2 → pulse 2 of frequency f_9, f_{10} represents 000_2
6. CRC[5:4] = 00_2 → pulse 3 in slot T_1
7. CRC[3:1] = 010_2 → pulse 3 of frequency f_3, f11 represents 010_2
8. CRC[0] = 1_2 → pulse 4 in slot T2, any frequency can be chosen

The example is illustrated in Figure 11-2.

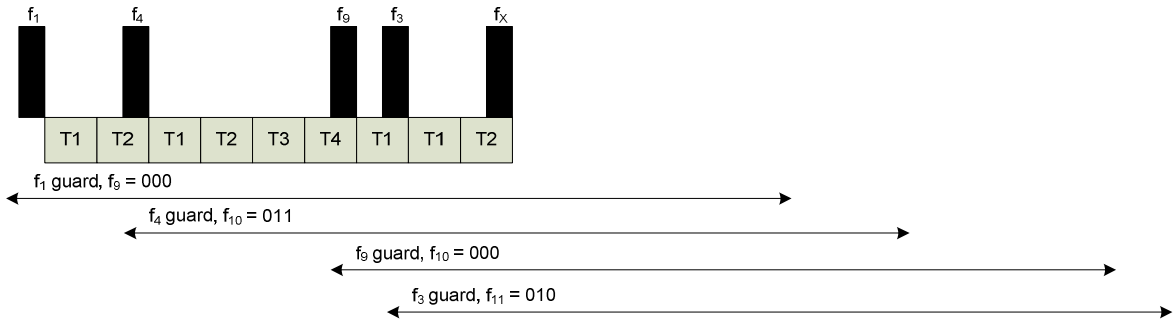


Figure 11-2 - Example transmission using a DPPM and MFSK scheme

The total time of this transfer is nine timeslots. If a slot size of 20 ms, as with the Vemco signal described in section 4.1.1 on page 11, is used the total transmission time becomes 180ms whereas with the Vemco standard the transmission would be 2280 ms. The transmission throughput is thereby increased by a factor of 12,5 for this example. Decoding this signal with the system concept described in this report will not introduce any considerable increase in complexity or execution time due to the already implemented FFT routine. It should also be possible to implement the modulation scheme on a transmitter without substantial modifications.

At the time of writing no other competitor product supports such a scheme and looking at the specifications provided in section 1.4 it is likely to believe that support for such a scheme would require the competitors to design a completely new product. Using a similar modulation scheme as the described in this section can therefore provide Thelma with a good competitive edge.

12. Conclusion

The goal of this project was to design and validate a mixed analogue and digital receiver system for an acoustic telemetry buoy fulfilling the requirements provided in section 1.1. Several design solutions have been discussed for each system function whereas one complete system have been developed and tested.

The system design concept has been proven to work by successfully receiving signals from an existing standard Thelma fish tag. Calculations based on current consumption, execution time and values provided in the components' datasheets have confirmed that the design solution will provide an expected battery lifetime of over two years, if the UC3L is selected as the main microcontroller for final design.

Some adjustments need to be made before designing the final product. These are mainly minor changes where some component values are adjusted and some microcontroller settings such as sampling frequency are changed. All suggested improvements are found in section 10 and 11. It is also vital that some additional tests are performed in situ to optimize the design.

For Thelma this telemetry buoy can lead to a good competitive advantage. There are mainly two competing products on the market; the VR2W manufactured by Vemco and the SUR manufactured by Sonotronics. If Thelma develops this prototype further to a final product, Thelma's telemetry buoy will have advantages over its competitors such as

- Extended battery lifetime
- Always listening to several frequencies
- Ability to receive at several frequencies simultaneously
- Support for new frequencies and modulation schemes can be made with the same hardware
- Special versions can be developed using the same hardware platform only developing new firmware

The final conclusion is therefore that the system design provided in this report, together with the suggested improvements and the UC3L microcontroller, provides Thelma with a good basis for developing the leading standalone ultra low power acoustic telemetry buoy on the market.

13. Appendix list

Printed appendices

1. Matlab script for calculating component values and simulating Butterworth and Chebyshev filters.
2. Matlab script for simulating the intuitive filter frequency response.
3. Matlab script for simulation Chebyshev filter with digital potentiometers
4. Screenshots of filter design with FilterPro
5. Schematics
    a. Preamp
    b. Auto gain and filter
    c. Reception detector
    d. Connectors and headers
    e. Power distribution
    f. MCU and Flash memory
6. Bill of materials
7. PCB assembly diagram
8. C-code for prototype testing

Digital appendices

A. Gerber files for PCB prototype
B. Complete C-code for the prototype
C. *Acoustic telemetry buoy* report [4] in pdf format.
D. Energy calculations for the complete system with UC3B and UC3L
E. Test results data
F. Matlab scripts for filter simulations
G. This report in pdf format

14. Bibliography

[1] Paul McCormac, National Semiconductor, May 2004:
Effects and Benefits of Undersampling in High-Speed ADC applications

[2] Wang Yiding, Wang Yunhong, Zhao Shi, IEEE - November 2007:
Errors analysis of spectrum inversion methods

[3] James H. McClellan, Ronald W. Schafer, Mark A, Yoder 2003:
Signal processing first

[4] Stian Orø Moen student project 2009:
Acoustic telemetry buoy

[5] Eamon Nash, Analog Devices Inc, Jan 2000:
Design A Logamp RF Pulse Detector

[6] Analog Devices application note - 2000:
AD737: True RMS-to-DC Converter, Rev F

[7] Jan Eyolf Bjørnsen, master thesis Jun 10th 2002:
Utvikling av intelligent hydrofonbøye for PINPOINT II

[8] Inge Mohus and Bård Holand, SINTEF report 1980:
Fish Telemetry manual

[9] Thelma brochure June 2007:
Marine related R&D

[10] Robert J. Ulrick 1983:
Principles of underwater sound, 3rd edition

[11] Maxim application note 864, Nov 28, 2001:
EPOT Applications: Gain Adjustment in Op-Amp Circuits

[12] Maxim application note 1828, Dec 26, 2002:
Audio Gain Control Using Digital Potentiometers

[13] Maxim application note 1828, Feb 27, 2004:
How to increase the bandwidth of digital potentiometers 10x to 100x

[14] Adel S. Sedra and Kenneth C. Smith, 2004:
Microelectronic Circuits, fifth edition

[15] Texas instruments application note – September 2002:
Analysis of the SAllen-Key Architecture, rev SLOA024B

[16] E. A. Faulkner and J. B. Grimbleby – IEEE report 16th July 1970:
Active filters and gain-bandwidth product

[17] Maxim application note 1762 – 28th September 2002:
A beginners guide to filter topologies

[19] Atmel AVR32 datasheet 04/08:
AT32UC3B Series Preliminary, Rev G

[20] Atmel AVR32 application note:
AVR32715: AVR32 UC3B Schematic Checklist, Rev 32095D-AVR32-12/08

[21] Atmel datasheet 08/07:
32-Mbit, 2.7 Volt DataFlash® AT45DB321D datasheet preliminary,

[22] Atmel AVR32 datasheet 06/09:
AT32UC3L064, AT32UC3L032, AT32UC3L016 Preliminary, Rev A

[23] Tadiran datasheet 01/06:
MODEL TL-5930, Rev. B

[24] Texas Instruments application report:
Choosing An Ultralow-Power MCU, SLAA207 – June 2004

[25] Atmel datasheet*:*
XMEGA Manual, 8077B-AVR-06/08

[26] Atmel AVR32 application note:
AVR32739: AVR32 UC3 Low power software design, Rev 32093B-AVR32-05/08

[27] Atmel AVR32 application note:
AVR32765: AVR32 DSPLib Reference Manual, rev A 07/09

[28] Atmel AVR32 application note:
AVR32718: AT32UC3 Series Software Framework DSPLib, Rev 32076A-AVR32-11/07

Appendices

1. Matlab script for calculating component values and simulating Butterworth and Chebyshev filters

```
%Script that gives the bode responce of chebychev or Butterworth filter
%Implemented as Sallen & Key configuration. Script will calculate all
%Component values according to the crossover frequency and gain settings.
clc
clear all

%desired lower corner frequency [Hz]
f0 = 60000;

%desired higher corner frequency [Hz]
f1 = 80000;

%desired passband gain per amplifier
k =2.000001;

%low pass filter order
n_l = 4;

%high pass filter order
n_h = 4;


%Uncomment the poles to use

%calculate general (w0 = 1) lowpass butterworth pole positions
all_poles = roots([(-1)^n_l, zeros(1,2*n_l-1),1]);
negative_real_poles = all_poles(find(real(all_poles)<0));
den = poly(negative_real_poles);

%chebychev poles with 3dB rippel
%negative_real_poles = [-0.0340675444 + 0.3785869383i
%                       -0.0340675444 - 0.3785869383i
%                        -0.0822463277 + 0.1568158444i
%                        -0.0822463277 - 0.1568158444i];

%chebychev poles with 1 dB rippel
negative_real_poles = [ -0.0558133907 + 0.3933445644i
                         -0.0558133907 - 0.3933445644i
                         -0.1347454448 + 0.1629286533i
                         -0.1347454448 - 0.1629286533i];




%calculate desired w0/Q for filter tranferfunction one and 2
w0_Q1 = -(negative_real_poles(1)+negative_real_poles(2));
w0_Q2 = -(negative_real_poles(3)+negative_real_poles(4));

%plot general low pass function
%H_butt = tf([k*k],den);
%bode(H_butt);


%plot general high pass function
%H_butt = tf([k*k 0 0 0 0],den)
%bode(H_butt)
```

```
%hold on

%%
% ******* LOW PASS CALCULATIONS *******

%choose C1=C2=C3=C4=C
C = 1;%47e-12;
C1 = C;
C2 = C;
C3 = C;
C4 = C;

%calcualte Resistor values for w0 = 1 for first sallen key circuit
w0 = 1; %do not change!
Q1  =  w0/w0_Q1;
%R1 = (-1/Q1+sqrt(1/Q1^2-4*(C1+C2*(1-k))*1/C2))/(2*(C1+C2*(1-k)))
R1 = 1/2*(w0*C2-sqrt(w0^2*C2^2-4*Q1^2*C2*C1-4*Q1^2*C2^2+4*Q1^2*C2^2*k))/(Q1*C2*(C1+C2-↙
C2*k));
R2 = 1/(R1*C1*C2);

%calcualte Resistor values for w0 = 1 for second sallen key circuit
Q2  = w0/w0_Q2;
R3 = 1/2*(w0*C4-sqrt(w0^2*C4^2-4*Q2^2*C4*C3-4*Q2^2*C3^2+4*Q2^2*C4^2*k))/(Q2*C4*(C3+C4-↙
C4*k));
R4 = 1/(R3*C3*C4);

%define lowpass filter transferfunction
nom1 = [k];
den1 = [R1*R2*C1*C2 (R1*C1+R2*C1 + R1*C2*(1-k)) 1];

nom2 = [k];
den2 = [R3*R4*C3*C4 (R3*C3+R4*C3 + R3*C4*(1-k)) 1];

%Calculate nominal low pass transfer function (i.e. w0 = 1)
H_l1 = tf(nom1,den1);
H_l2 = tf(nom2,den2);
H_l  = H_l1*H_l2

%scale values to get crossover at f0

C = C/(f1*2*pi); %scale to freqency
C_desired = 47e-12;

%scale to stadard capacitor value
factor  = C/C_desired
C = C/factor     %scale to resistors

C1 = C
C2 = C
C3 = C
C4 = C

R1 = R1*factor
R2 = R2*factor
R3 = R3*factor
R4 = R4*factor
```

```
%Calculate actual transferfunction
nom1a = [k];
den1a = [R1*R2*C1*C2 (R1*C1+R2*C1 + R1*C2*(1-k)) 1];

nom2a = [k];
den2a = [R3*R4*C3*C4 (R3*C3+R4*C3 + R3*C4*(1-k)) 1];

H_la1 = tf(nom1a,den1a);
H_la2 = tf(nom2a,den2a);
H_a  = H_la1*H_la2;



%%
%******* HIGH PASS CALCULATIONS *******

%choose C1=C2=C3=C4=C
C = 1;%47e-12;
C5 = C;
C6 = C;
C7 = C;
C8 = C;

%calcualte Resistor values for w0 = 1 for third sallen key circuit
w0 = 1; %do not change!
Q1  =  w0/w0_Q1
R6 = 1/2*(w0*C5+sqrt(w0^2*C5^2-4*Q1^2*C6*C5+4*Q1^2*C6*C5*k-4*C5^2*Q1^2+4*C5^2*Q1^2*k))/↙
(C5*Q1*(C6+C5));
R5 = 1/(R6*C5*C6);

%calcualte Resistor values for w0 = 1 for fourth sallen key circuit
Q2  = w0/w0_Q2
Q1  = w0/w0_Q1
R8 = 1/2*(w0*C7+sqrt(w0^2*C7^2-4*Q2^2*C8*C7+4*Q2^2*C8*C7*k-4*C7^2*Q2^2+4*C7^2*Q2^2*k))/↙
(C7*Q2*(C8+C7));
R7 = 1/(R8*C7*C8);


%define highpass filter transferfunction
nom3 = [k*R5*R6*C5*C6 0 0];
den3 = [R5*R6*C5*C6 (R6*C6 + R6*C5 + R5*C6*(1-k)) 1];

nom4 = [k*R7*R8*C7*C8 0 0];
den4 = [R7*R8*C7*C8 (R8*C8 + R8*C7 + R7*C8*(1-k)) 1];

%Plot nominal transfer function (i.e. w0 = 1)
H_h1 = tf(nom3,den3)
H_h2 = tf(nom4,den4)
H_h  = H_h1*H_h2


%scale values to get crossover at f0
%factor = 100e3; %component scale factor
```

```
C = C/(f0*2*pi); %scale to freqency
C_desired = 47e-12;

%scale to stadard capacitor value
factor  = C/C_desired
C = C/factor     %scale to resistors

C5 = C
C6 = C
C7 = C
C8 = C

R5 = R5*factor
R6 = R6*factor
R7 = R7*factor
R8 = R8*factor


%Calculate actual high pass transferfunction
nom3a = [k*R5*R6*C5*C6 0 0];
den3a = [R5*R6*C5*C6 (R6*C6 + R6*C5 + R5*C6*(1-k)) 1];

nom4a = [k*R7*R8*C7*C8 0 0];
den4a = [R7*R8*C7*C8 (R8*C8 + R8*C7 + R7*C8*(1-k)) 1];

H_ha1 = tf(nom3a,den3a);
H_ha2 = tf(nom4a,den4a);
H_ha   = H_ha1*H_ha2;

%calculate total band pass transferfunction

H = H_a*H_ha;

%Bodeplots
P = bodeoptions;
P.PhaseVisible = 'off';
P.FreqUnits = 'Hz';
P.Xlim = [1000 500000];


figure(1)
P.Title.String = 'Low pass transfer function';
subplot(2,2,1);
bode(H_a,P);

P.Title.String = 'High pass transfer function';
subplot(2,2,2);
bode(H_ha,P);

P.Title.String = 'Combined band pass transfer function';
figure(2);
bode(H,P);

%%
%figure(1);
%subplot(2,2,1);
```

```
%bode(Hni,P);
%P.Title.String = 'Active low pass filter with gain';
%subplot(2,2,2);
%title('Active Low Pass');
%bode(G,P);
%P.Title.String = 'Combined potentiometer, HP and LP';
%subplot(2,2,3);
%title('Combined one link');
%bode(G*Hni,P);
%bode(Hp*Hlp,P); only for testing
%subplot(2,2,4);
%title('Full AGC responce');
%bode(Hni*G*Hni*G*Hni*G*Hp*G,P);
```

2. Matlab script for simulating the intuitive filter frequency response

```
%Script that gives the bode responce of the variable gain amplifier
%and filter designed by the intuitive solution
clc
clear all
%wiper position (0 means top, full input signal)
hold on
N = 0.0;
%potentiometer end-to-end resistance
RP = 100e3;
%high pass capacitor value
C1 = 5.6e-12
%desired lower corner frequency [Hz]
f0 = 60000;
%filter resistor value
R1 = 1/(C1*2*pi*f0)

%low pass filter resistor value
R3 = 423.3e3;
R2 = 17637;
%low pass filter capacitor value
C2 = 4.7e-12;

%Wiper resistance
RW = 250;
%Wiper capacitance
CW = 25e-12;

%calculate potentimeter resistor values.
RL = RP*(1-N)
RH = RP*N


%transfer function for an ideal potentiometer and high pass filter
Hi = tf([RL*R1*C1 0],[C1*(R1*RL+RH*RL+RH*R1) RH+RL])

%transfer function for high pass filter without potentiometer
Hp = tf([R1*C1 0],[R1*C1 1])

%transfer function for an non-ideal potentiometer with HP-filter
Hni= tf([RL*R1*C1 0],[CW*C1*R1*((RL+RW)*RH+RL*RW) (((C1+CW)*RL+(RW+R1)*C1+RW*CW)*RH+↙
((RW+R1)*C1+RW*CW)*RL) RL+RH])


%Transferfunction for the Active low pass filter with gain
G = 1+tf([R3],[R2*R3*C2 R2]);
hold on

%Plot transfer functions

P = bodeoptions;
P.PhaseVisible = 'off';
P.FreqUnits = 'Hz';
P.Xlim = [1000 500000];
P.Title.String = 'Potentiometer with high pass filter';

jadda = 'jadda';
```

```
figure(1);
subplot(2,2,1);
bode(Hni,P);
P.Title.String = 'Active low pass filter with gain';
subplot(2,2,2);
title('Active Low Pass');
bode(G,P);
P.Title.String = 'Combined potentiometer, HP and LP';
subplot(2,2,3);
title('Combined one link');
bode(G*Hni,P);
bode(Hp*Hlp,P); only for testing
P.Title.String = 'Complete AGC with 4 gain stages, 3 potentiometes and 4 HP';
subplot(2,2,4);
title('Full AGC responce');
bode(Hni*G*Hni*G*Hni*G*Hp*G,P);


%Calculate poles as a function of potentiometer position.
K = 255;
for k=1:K
    RL = RP*(1-k/K);
    RH = RP*k/K;
    den = [CW*C1*R1*((RL+RW)*RH+RL*RW) (((C1+CW)*RL+(RW+R1)*C1+RW*CW)*RH+((RW+R1)↙
*C1+RW*CW)*RL) RL+RH];
    r_temp = -roots(den)/(2*pi);
    r1(k) = r_temp(1);
    r2(k) = r_temp(2);
end

%check that the upper pole does not efffect the frequency band of interesst
lowest_high_pole = min(r1)

%check the crossoverfreqeuncy variation
variation = max(r2)-min(r2)
```

3. Matlab script for simulating the Chebyshev filter with digital potentiometers

```
%Script that gives the bode responce of the chebyshev filter, potentiometer
%and the high pass potentiometer noise removal filter

clc
clear all
hold on
%desired lower corner frequency [Hz]
f0 = 60000;

%desired higher corner frequency [Hz]
f1 = 80000;

%desired passband gain per amplifier
k = 7;

%low pass filter order
n_l = 4;

%high pass filter order
n_h = 4;


%Uncomment the poles to use

%calculate general (w0 = 1) lowpass butterworth pole positions
all_poles = roots([(-1)^n_l, zeros(1,2*n_l-1),1]);
negative_real_poles = all_poles(find(real(all_poles)<0));
den = poly(negative_real_poles);

%chebychev poles with 3dB rippel
%negative_real_poles = [-0.0340675444 + 0.3785869383i
%                       -0.0340675444 - 0.3785869383i
%                        -0.0822463277 + 0.1568158444i
%                        -0.0822463277 - 0.1568158444i];

%chebychev poles with 1 dB rippel
negative_real_poles = [ -0.0558133907 + 0.3933445644i
                         -0.0558133907 - 0.3933445644i
                         -0.1347454448 + 0.1629286533i
                         -0.1347454448 - 0.1629286533i];




%calculate desired w0/Q for filter tranferfunction one and 2
w0_Q1 = -(negative_real_poles(1)+negative_real_poles(2));
w0_Q2 = -(negative_real_poles(3)+negative_real_poles(4));

%plot general low pass function
%H_butt = tf([k*k],den);
%bode(H_butt);

%plot general high pass function
H_butt = tf([k*k 0 0 0 0],den)

%bode(H_butt)
```

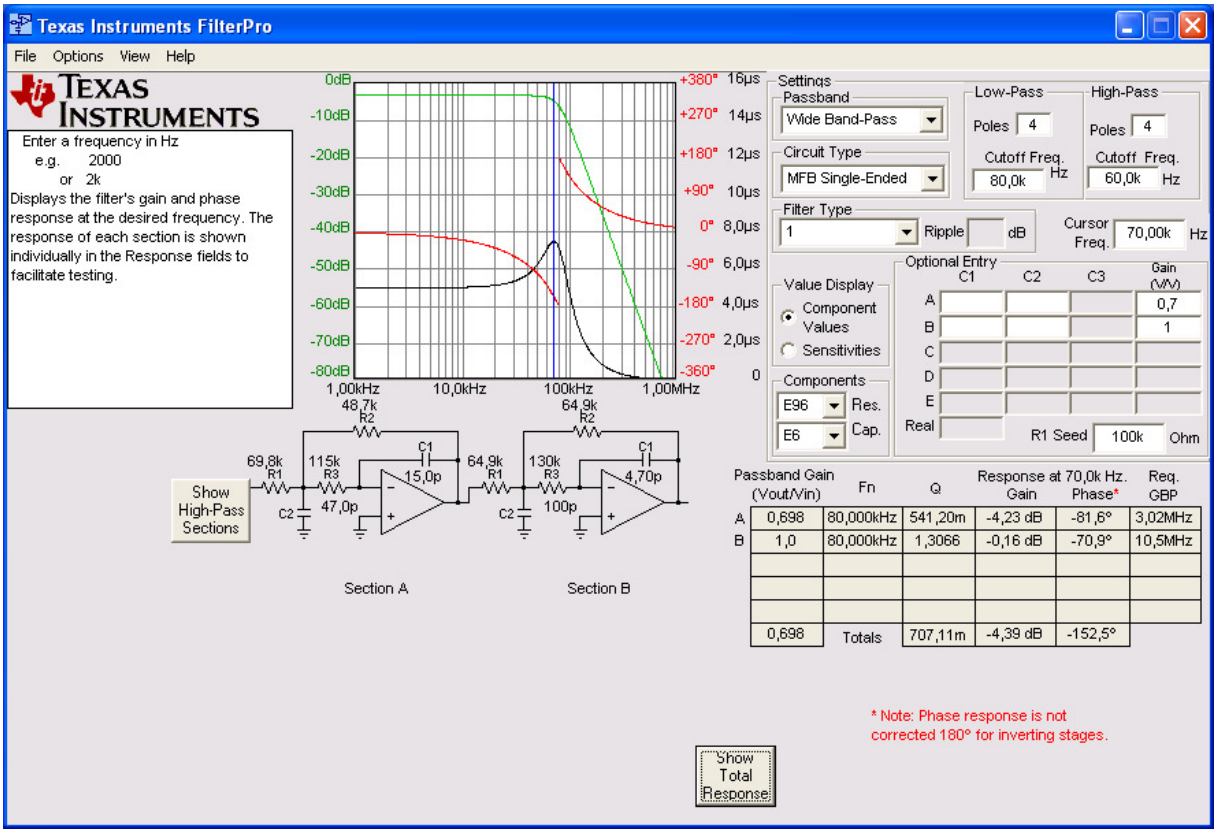
```
%hold on


% ******* LOW PASS CALCULATIONS *******

%choose C1=C2=C3=C4=C
C = 1;%47e-12;
C1 = C;
C2 = C;
C3 = C;
C4 = C;

%calcualte Resistor values for w0 = 1 for first sallen key circuit
w0 = 1; %do not change!
Q1  =  w0/w0_Q1;
%R1 = (-1/Q1+sqrt(1/Q1^2-4*(C1+C2*(1-k))*1/C2))/(2*(C1+C2*(1-k)))
R1 = 1/2*(w0*C2-sqrt(w0^2*C2^2-4*Q1^2*C2*C1-4*Q1^2*C2^2+4*Q1^2*C2^2*k))/(Q1*C2*(C1+C2-↙
C2*k));
R2 = 1/(R1*C1*C2);

%calcualte Resistor values for w0 = 1 for second sallen key circuit
Q2  = w0/w0_Q2;
R3 = 1/2*(w0*C4-sqrt(w0^2*C4^2-4*Q2^2*C4*C3-4*Q2^2*C3^2+4*Q2^2*C4^2*k))/(Q2*C4*(C3+C4-↙
C4*k));
R4 = 1/(R3*C3*C4);

%define lowpass filter transferfunction
nom1 = [k];
den1 = [R1*R2*C1*C2 (R1*C1+R2*C1 + R1*C2*(1-k)) 1];

nom2 = [k];
den2 = [R3*R4*C3*C4 (R3*C3+R4*C3 + R3*C4*(1-k)) 1];

%Plot nominal low pass transfer function (i.e. w0 = 1)
H_l1 = tf(nom1,den1);
H_l2 = tf(nom2,den2);
H_l  = H_l1*H_l2;


%scale values to get crossover at f0
factor = 100e3; %component scale factor
C = C/(f1*2*pi); %scale to freqency
C = C/factor     %scale to resistors
C1 = C
C2 = C
C3 = C
C4 = C

R1 = R1*factor
R2 = R2*factor
R3 = R3*factor
R4 = R4*factor


%Calculate actual transferfunction
nom1a = [k];
```

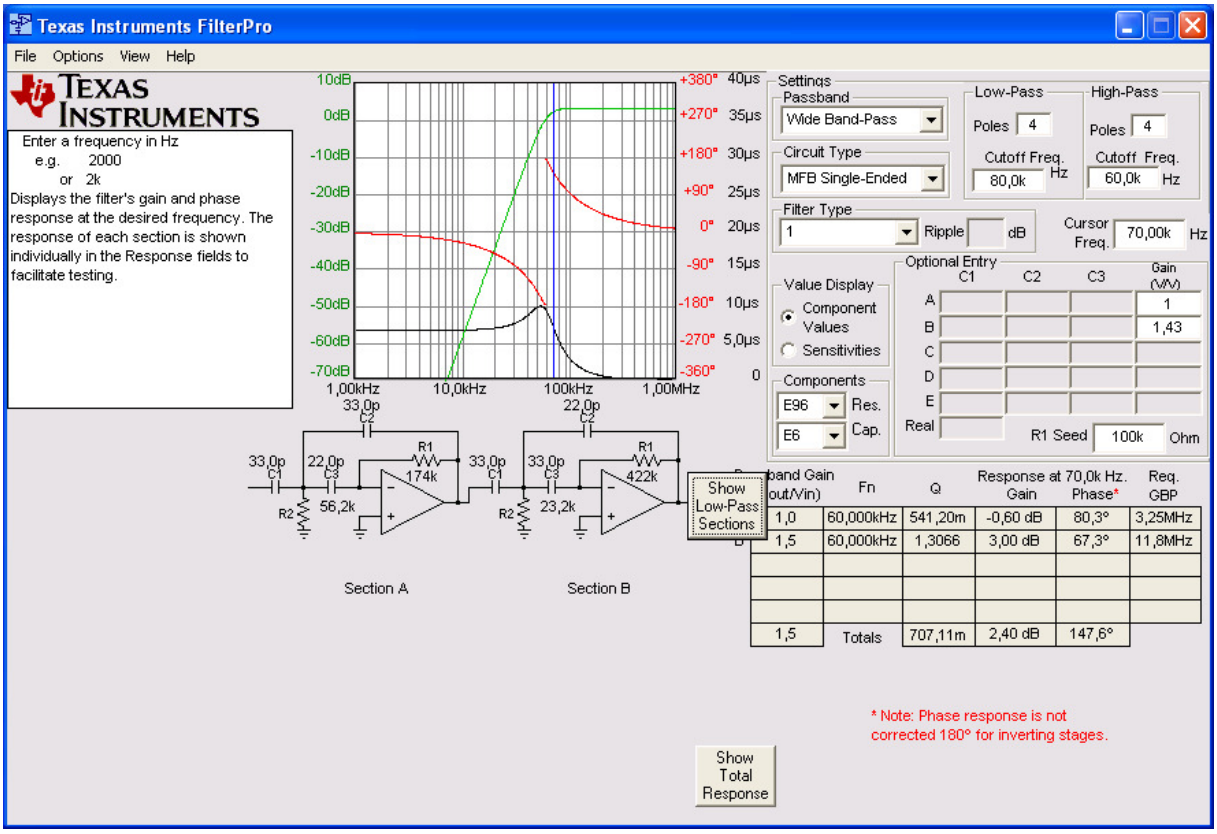
```
den1a = [R1*R2*C1*C2 (R1*C1+R2*C1 + R1*C2*(1-k)) 1];

nom2a = [k];
den2a = [R3*R4*C3*C4 (R3*C3+R4*C3 + R3*C4*(1-k)) 1];

H_la1 = tf(nom1a,den1a);
H_la2 = tf(nom2a,den2a);
H_a  = H_la1*H_la2;




%******* HIGH PASS CALCULATIONS *******

%choose C1=C2=C3=C4=C
C = 1;%47e-12;
C5 = C;
C6 = C;
C7 = C;
C8 = C;

%calcualte Resistor values for w0 = 1 for third sallen key circuit
w0 = 1; %do not change!
Q1  =  w0/w0_Q1
R6 = 1/2*(w0*C5+sqrt(w0^2*C5^2-4*Q1^2*C6*C5+4*Q1^2*C6*C5*k-4*C5^2*Q1^2+4*C5^2*Q1^2*k))/↙
(C5*Q1*(C6+C5));
R5 = 1/(R6*C5*C6);

%calcualte Resistor values for w0 = 1 for fourth sallen key circuit
Q2  = w0/w0_Q2
Q1  = w0/w0_Q1
R8 = 1/2*(w0*C7+sqrt(w0^2*C7^2-4*Q2^2*C8*C7+4*Q2^2*C8*C7*k-4*C7^2*Q2^2+4*C7^2*Q2^2*k))/↙
(C7*Q2*(C8+C7));
R7 = 1/(R8*C7*C8);


%define highpass filter transferfunction
nom3 = [k*R5*R6*C5*C6 0 0];
den3 = [R5*R6*C5*C6 (R6*C6 + R6*C5 + R5*C6*(1-k)) 1];

nom4 = [k*R7*R8*C7*C8 0 0];
den4 = [R7*R8*C7*C8 (R8*C8 + R8*C7 + R7*C8*(1-k)) 1];

%Plot nominal transfer function (i.e. w0 = 1)
H_h1 = tf(nom3,den3)
H_h2 = tf(nom4,den4)
H_h  = H_h1*H_h2


%scale values to get crossover at f0
factor = 100e3; %component scale factor
C = C/(f0*2*pi); %scale to freqency
C = C/factor     %scale to resistors
C5 = C
C6 = C
C7 = C
```

```
C8 = C

R5 = R5*factor
R6 = R6*factor
R7 = R7*factor
R8 = R8*factor


%Calculate actual high pass transferfunction
nom3a = [k*R5*R6*C5*C6 0 0];
den3a = [R5*R6*C5*C6 (R6*C6 + R6*C5 + R5*C6*(1-k)) 1];

nom4a = [k*R7*R8*C7*C8 0 0];
den4a = [R7*R8*C7*C8 (R8*C8 + R8*C7 + R7*C8*(1-k)) 1];

H_ha1 = tf(nom3a,den3a);
H_ha2 = tf(nom4a,den4a);
H_ha   = H_ha1*H_ha2;

%calculate total band pass transferfunction

H = H_a*H_ha;

%Bodeplots
P = bodeoptions;
P.PhaseVisible = 'off';
P.FreqUnits = 'Hz';
P.Xlim = [1000 500000];

%P.Title.String = 'Variable gain amp and ';
figure(1);
bode(H,P);

hold on

%******* Potentiometer with high pass filter transfer function ****

%wiper position (0 means top, full input signal)
N = 0.0;
%potentiometer end-to-end resistance
RP = 100e3;
%high pass capacitor value
C1 = 150e-12;
%filter resistor value
R1 = 106.1e3;

%Wiper resistance
RW = 250;
%Wiper capacitance
CW = 25e-12;

%calculate potentimeter resistor values.
RL = RP*(1-N)
RH = RP*N

%total transfer function for an non-ideal potentiometer with high pass
```

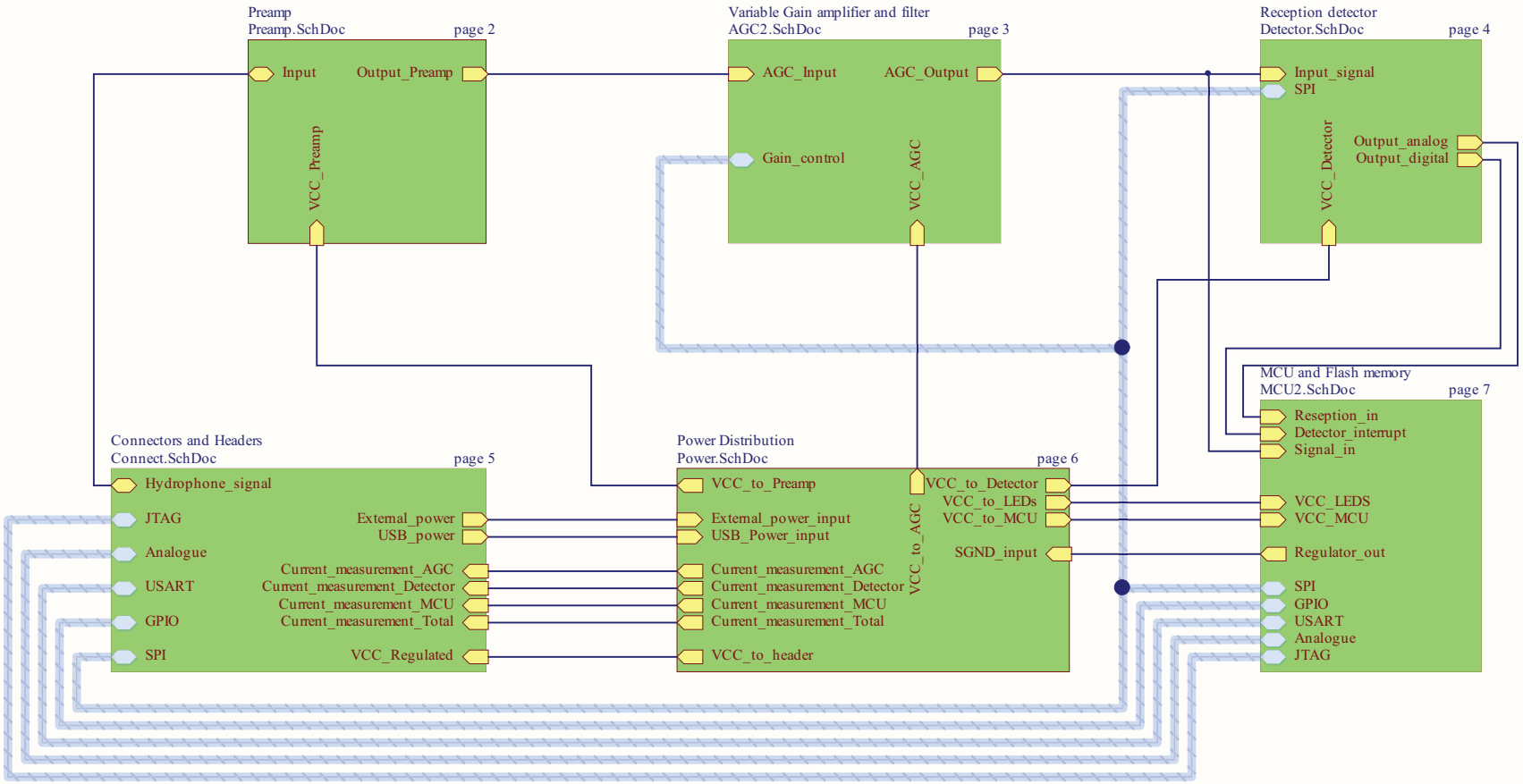
```
%filter

Hni= tf([RL*R1*C1 0],[CW*C1*R1*((RL+RW)*RH+RL*RW) (((C1+CW)*RL+(RW+R1)*C1+RW*CW)*RH+↙
((RW+R1)*C1+RW*CW)*RL) RL+RH])

%The complete transferfunction for low pass chebyshev, high pass Chebyshev,
%and potentiomters with filter

TF_tot = Hni*Hni*Hni*H;
bode(TF_tot,P);

pass_band_gain = evalfr(TF_tot,69000*2*pi)
gain = 10^(pass_band_gain/20)

Hni_fixed=Hni;
K = 256;
step_size = 1;
for k=1:step_size:K-1
    RL = RP*(1-k/K);
    RH = RP*k/K;
    Hni= tf([RL*R1*C1 0],[CW*C1*R1*((RL+RW)*RH+RL*RW) (((C1+CW)*RL+(RW+R1)*C1+RW*CW)↙
*RH+((RW+R1)*C1+RW*CW)*RL) RL+RH]);
    TF_tot = Hni*Hni_fixed*Hni_fixed*H;
   % bode(TF_tot,P);
    test((k-1)/step_size+1) = freqresp(TF_tot,69e3*2*pi);
    test((k-1)/step_size+1) = sqrt(real(test((k-1)/step_size+1))^2 + imag(test((k-1)↙
/step_size+1))^2);

    %for p=1:size(temp)
    %    bode_diagrams(k,p) = temp(p);
    %end


end
figure(2);
plot(test);
```

4. Screenshots of filter design with FilterPro

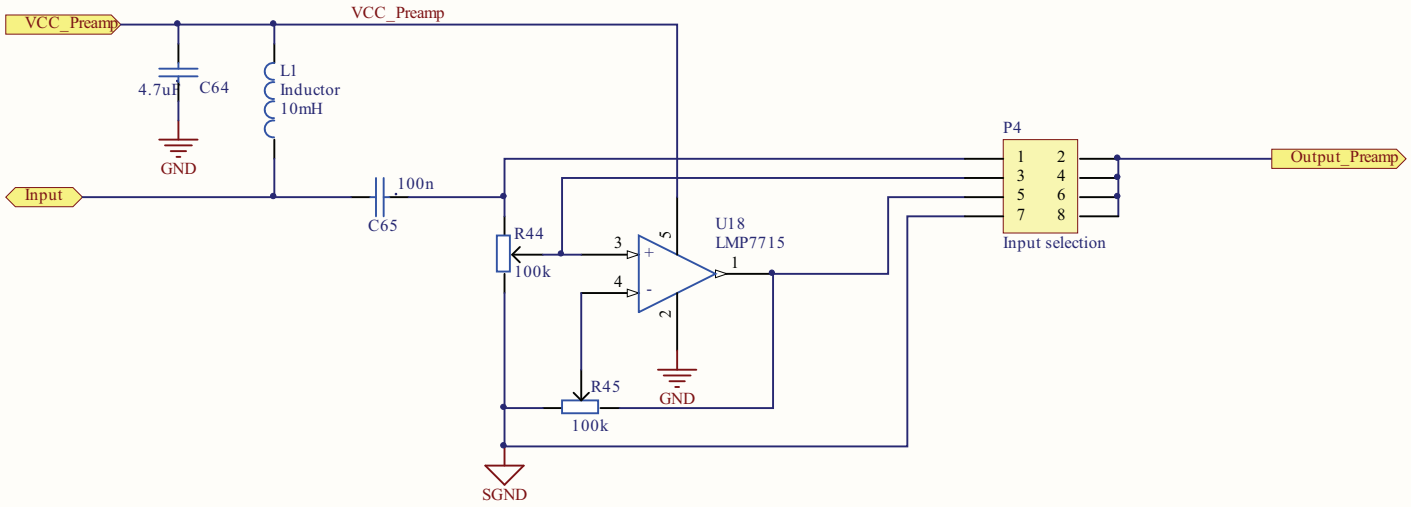

Texas Instruments FilterPro
File Options View Help
TEXAS INSTRUMENTS
Enter a frequency in Hz
e.g. 2000
or 2k
Displays the filter's gain and phase response at the desired frequency. The response of each section is shown individually in the Response fields to facilitate testing.
0dB -10dB -20dB -30dB -40dB -50dB -60dB -70dB -80dB
+380° +270° +180° +90° 0° -90° -180° -270° -360°
16µs 14µs 12µs 10µs 8,0µs 6,0µs 4,0µs 2,0µs 0
1,00kHz 10,0kHz 100kHz 1,00MHz
Settings
Passband
Wide Band-Pass
Low-Pass
Poles 4
Cutoff Freq. 80,0k Hz
High-Pass
Poles 4
Cutoff Freq. 60,0k Hz
Circuit Type
MFB Single-Ended
Filter Type
1
Ripple dB
Cursor Freq. 70,00k Hz
Value Display
Component Values
Sensitivities
Components
E96 Res.
E6 Cap.
Optional Entry
C1 C2 C3 Gain (V/V)
A 0,7
B 1
C
D
E
Real
R1 Seed 100k Ohm
48,7k R2
69,8k R1 115k R3 C1 15,0p C2 47,0p
64,9k R2
64,9k R1 130k R3 C1 4,70p C2 100p
Show High-Pass Sections
Section A
Section B
Passband Gain (Vout/Vin) Fn Q Response at 70,0k Hz. Gain Phase* Req. GBP
A 0,698 80,000kHz 541,20m -4,23 dB -81,6° 3,02MHz
B 1,0 80,000kHz 1,3066 -0,16 dB -70,9° 10,5MHz
0,698 Totals 707,11m -4,39 dB -152,5°
* Note: Phase response is not corrected 180° for inverting stages.
Show Total Response

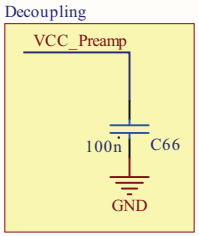

Texas Instruments FilterPro
File Options View Help
TEXAS INSTRUMENTS
Enter a frequency in Hz
e.g. 2000
or 2k
Displays the filter's gain and phase response at the desired frequency. The response of each section is shown individually in the Response fields to facilitate testing.
10dB 0dB -10dB -20dB -30dB -40dB -50dB -60dB -70dB
+380° +270° +180° +90° 0° -90° -180° -270° -360°
40µs 35µs 30µs 25µs 20µs 15µs 10µs 5,0µs 0
1,00kHz 10,0kHz 100kHz 1,00MHz
Settings
Passband
Wide Band-Pass
Low-Pass
Poles 4
Cutoff Freq. 80,0k Hz
High-Pass
Poles 4
Cutoff Freq. 60,0k Hz
Circuit Type
MFB Single-Ended
Filter Type
1
Ripple dB
Cursor Freq. 70,00k Hz
Value Display
Component Values
Sensitivities
Components
E96 Res.
E6 Cap.
Optional Entry
C1 C2 C3 Gain (V/V)
A 1
B 1,43
C
D
E
Real
R1 Seed 100k Ohm
33,0p C2
33,0p C1 22,0p C3 R1 174k R2 56,2k
33,0p C2
33,0p C1 33,0p C3 R1 422k R2 23,2k
Show Low-Pass Sections
Section A
Section B
band Gain (out/Vin) Fn Q Response at 70,0k Hz. Gain Phase* Req. GBP
1,0 60,000kHz 541,20m -0,60 dB 80,3° 3,25MHz
1,5 60,000kHz 1,3066 3,00 dB 67,3° 11,8MHz
1,5 Totals 707,11m 2,40 dB 147,6°
* Note: Phase response is not corrected 180° for inverting stages.
Show Total Response

5. Schematics

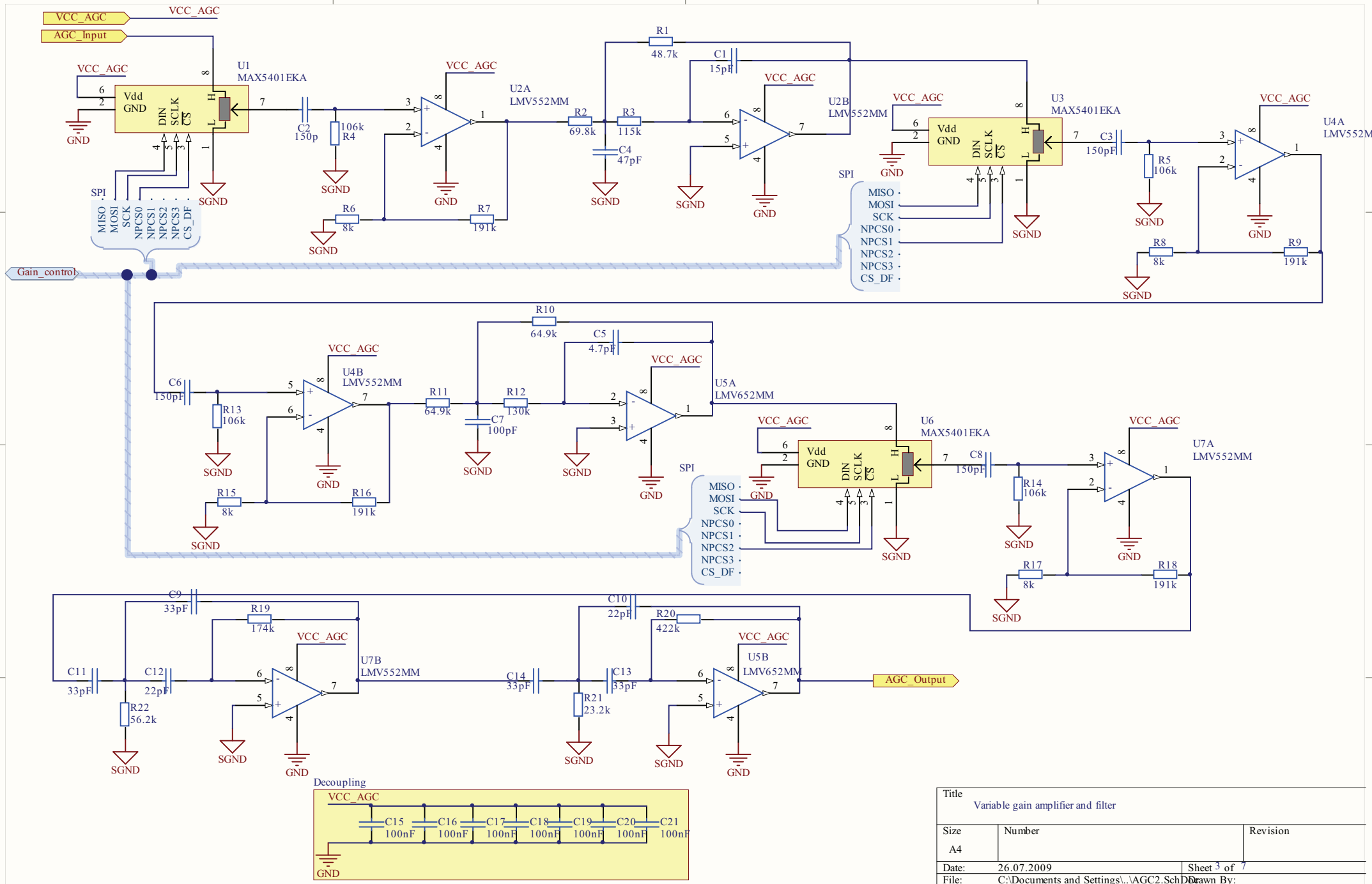


Title: Main schematic		
Size: A4	Number:	Revision:
Date: 26.07.2009	Sheet 1 of 7	
File: C:\Documents and Settings\...\MasterSchematic.SchDoc		

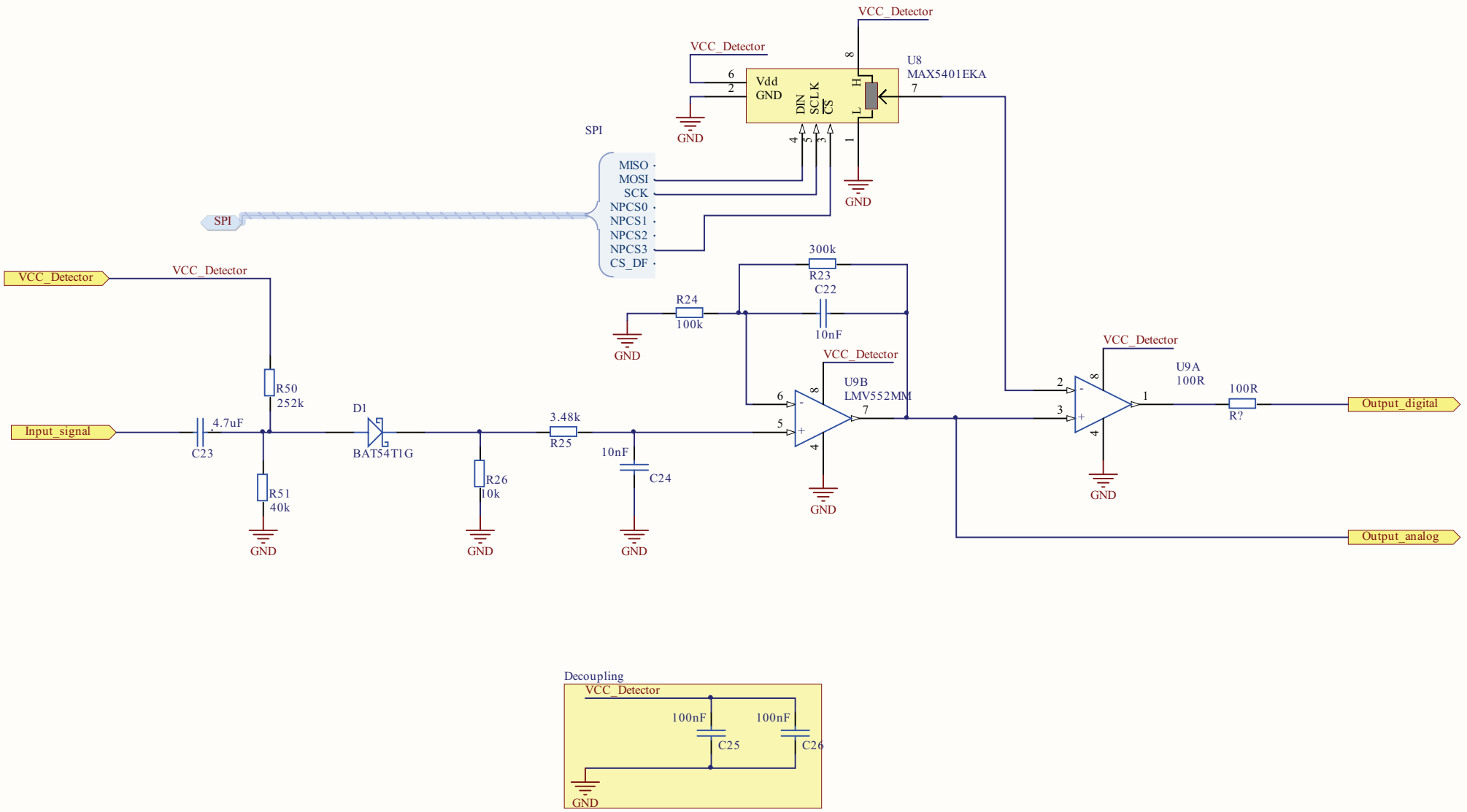




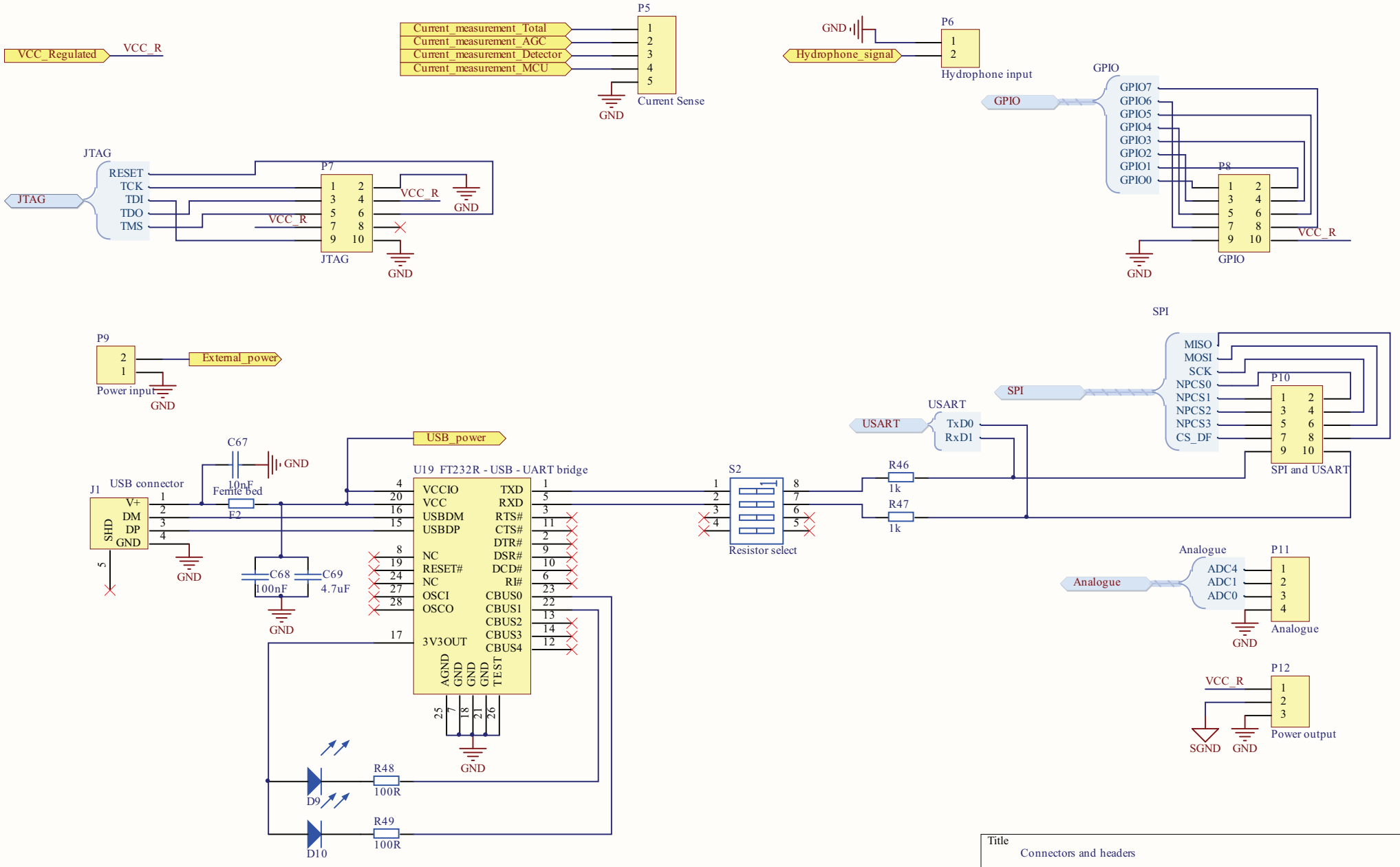
Title: Preamp		
Size: A4	Number:	Revision:
Date: 26.07.2009	Sheet 2 of 7	
File: C:\Documents and Settings\..\Preamp.SchDoc	Drawn By:	



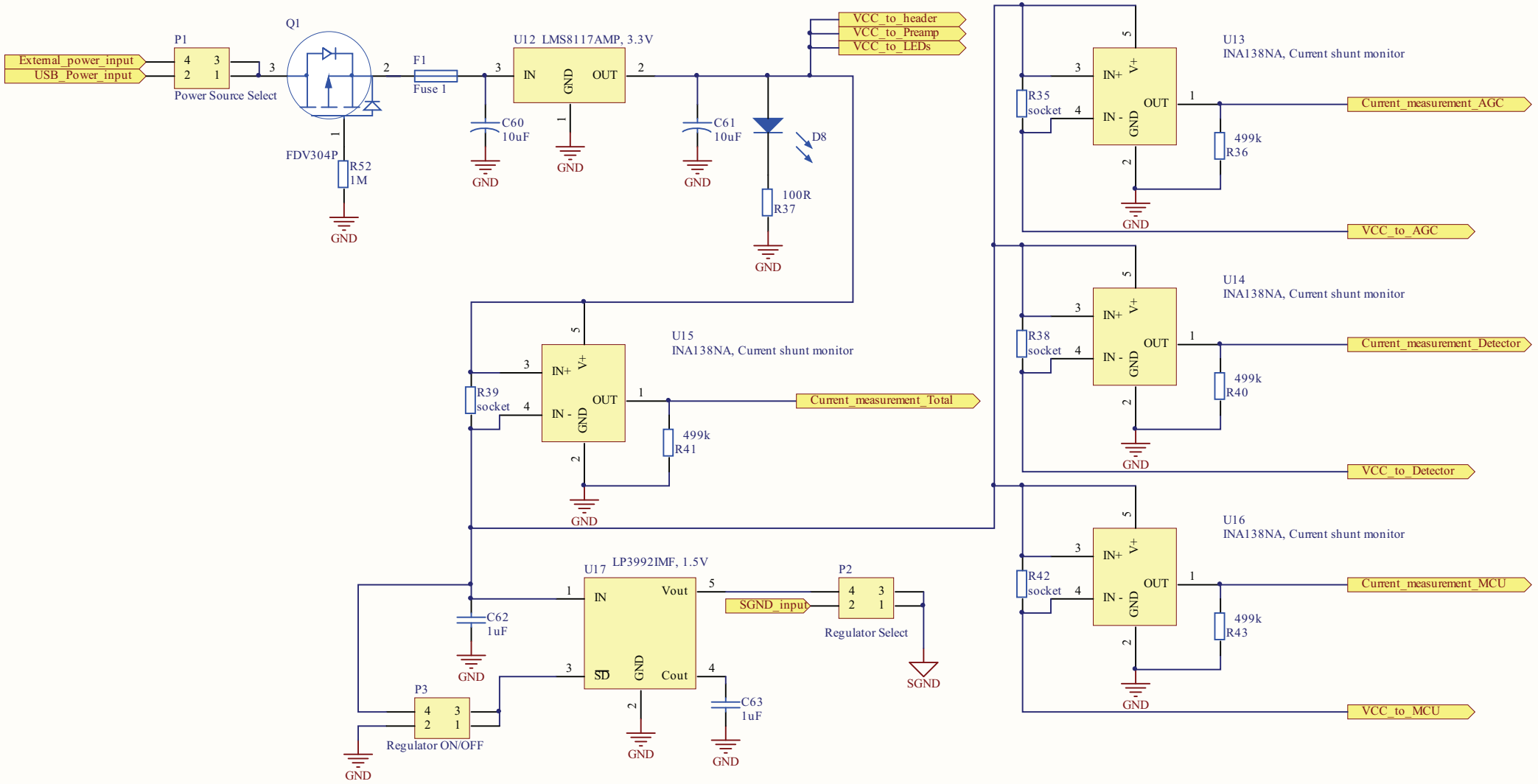
Title: Variable gain amplifier and filter		
Size: A4	Number:	Revision:
Date: 26.07.2009	Sheet 3 of 7	
File: C:\Documents and Settings\..\AGC2.SchDoc	Drawn By:	



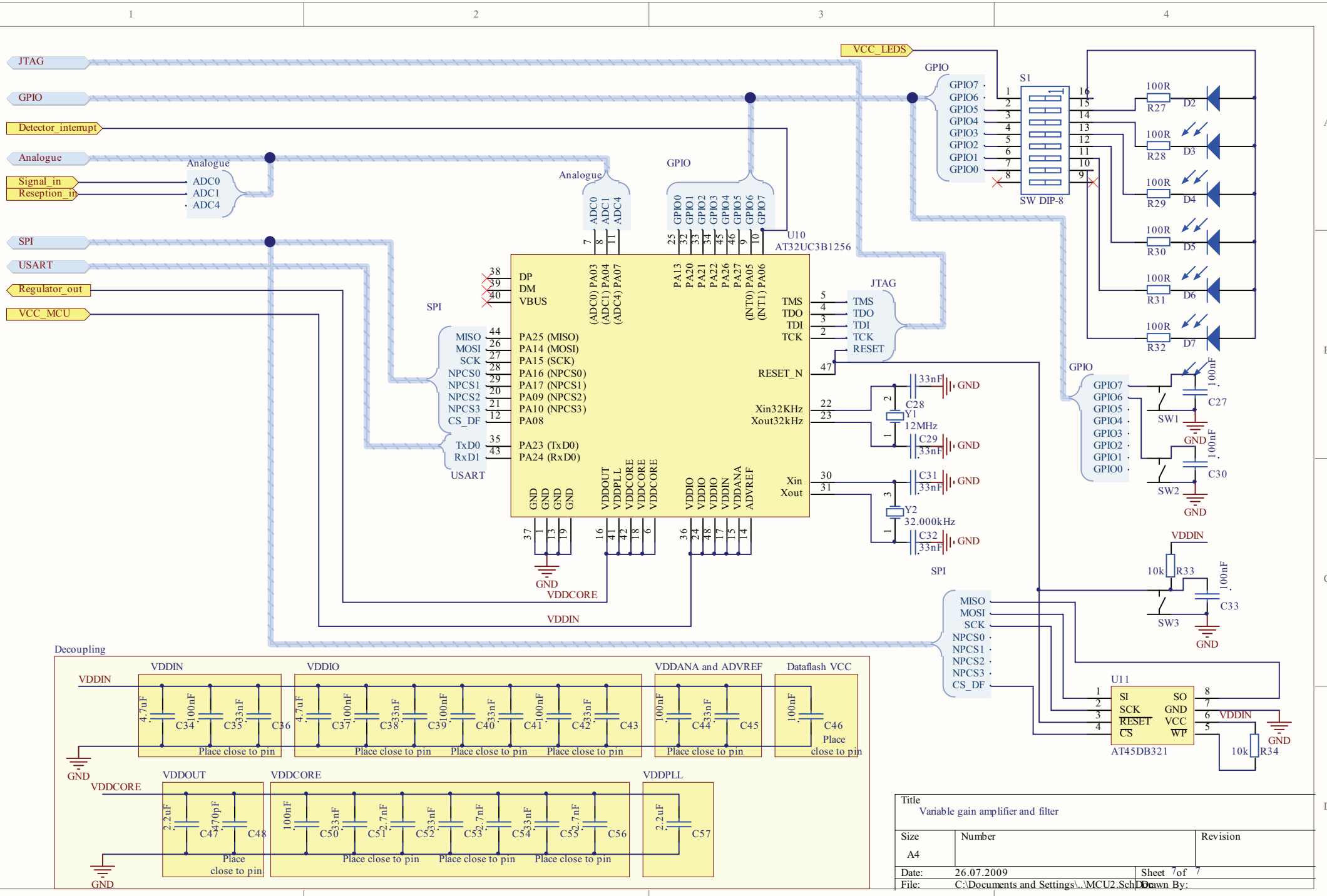
Title: Reception Detector		
Size: A4	Number:	Revision:
Date: 26.07.2009	Sheet 4 of 7	
File: C:\Documents and Settings\..\Detector.SchDoc	Drawn By:	



Title: Connectors and headers		
Size: A4	Number:	Revision:
Date: 26.07.2009	Sheet 5 of 7	
File: C:\Documents and Settings\..\Connect.SchDoc	Drawn By:	



Title: Power Distribution		
Size: A4	Number:	Revision:
Date: 26.07.2009	Sheet 6 of 7	
File: C:\Documents and Settings\..\Power.SchDoc	Drawn By:	

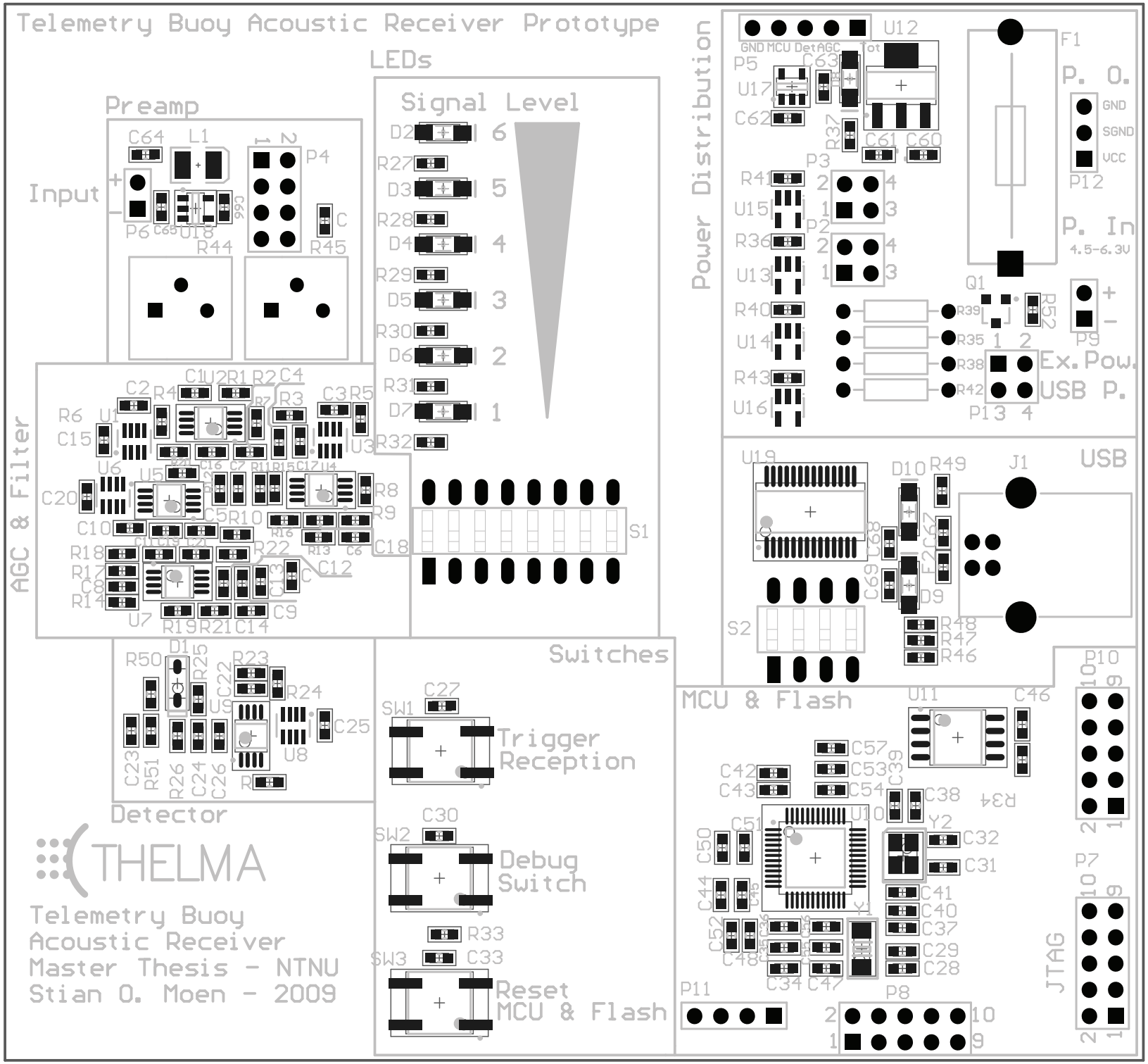


Title: Variable gain amplifier and filter		
Size: A4	Number:	Revision:
Date: 26.07.2009	Sheet 7 of 7	
File: C:\Documents and Settings\...\MCU2.SchDoc	Drawn By:	

6. Bill of materials

LibRef	Designator	Quantity	Value	Comment
Capacitor	C1	1	15pF	0603 capactitor
Capacitor	C2	1	150p	0603 capactitor
Capacitor	C3, C6, C8	3	150pF	0603 capactitor
Capacitor	C4	1	47pF	0603 capactitor
Capacitor	C5	1	4.7pF	0603 capactitor
Capacitor	C7	1	100pF	0603 capactitor
Capacitor	C9, C11, C13, C14	4	33pF	0603 capactitor
Capacitor	C10, C12	2	22pF	0603 capactitor
Capacitor	C15, C16, C17, C18, C	20	100nF	0603 capactitor
Capacitor	C22, C24, C67	3	10nF	0603 capactitor
Capacitor	C23, C34, C37, C64, C	5	4.7uF	0603 capactitor
Capacitor	C28, C29, C31, C32, C	12	33nF	0603 capactitor
Capacitor	C47, C57	2	2.2uF	0603 capactitor
Capacitor	C48	1	470pF	0603 capactitor
Capacitor	C52, C54, C56	3	2.7nF	0603 capactitor
C_pol	C60, C61	2	10uF	C_pol
Capacitor	C62, C63	2	1uF	0603 capactitor
Capacitor	C65, C66	2	100n	0603 capactitor
BAT54T1G	D1	1		BAT54T1G
LED	D2, D3, D4, D5, D6, D	9		SMD LED
Fuse 1	F1	1		Fuse 1
Resistor	F2	1		Ferrite bed
USB connector	J1	1		USB connector
Inductor	L1	1	10mH	Inductor
Header 2X2	P1, P2, P3	3		Power Source Select, Regulator ON/OFF, Regulator Select
Header 4X2	P4	1		Input selection
Header 5	P5	1		Current Sense
Header 2	P6, P9	2		Hydrophone input, Power input
Header 5X2	P7, P8, P10	3		GPIO, JTAG, SPI and USART
Header 4	P11	1		Analogue
Header 3	P12	1		Power output
FDV304P	Q1	1		FDV304P
Resistor	R1	1	48.7k	0603 Resistor
Resistor	R2	1	69.8k	0603 Resistor
Resistor	R3	1	115k	0603 Resistor
Resistor	R4, R5, R13, R14	4	106k	0603 Resistor
Resistor	R6, R8, R15, R17	4	8k	0603 Resistor
Resistor	R7, R9, R16, R18	4	191k	0603 Resistor
Resistor	R10, R11	2	64.9k	0603 Resistor
Resistor	R12	1	130k	0603 Resistor
Resistor	R19	1	174k	0603 Resistor
Resistor	R20	1	422k	0603 Resistor
Resistor	R21	1	23.2k	0603 Resistor
Resistor	R22	1	56.2k	0603 Resistor
Resistor	R23	1	300k	0603 Resistor
Resistor	R24	1	100k	0603 Resistor
Resistor	R25	1	3.48k	0603 Resistor
Resistor	R26, R33, R34	3	10k	0603 Resistor
Resistor	R27, R28, R29, R30, R	10	100R	0603 Resistor
Res2	R35, R38, R39, R42	4	socket	Res2
Resistor	R36, R40, R41, R43	4	499k	0603 Resistor
Potentiometer	R44, R45	2	100k	Pot
Resistor	R46, R47	2	1k	0603 Resistor
Resistor	R50	1	252k	0603 Resistor
Resistor	R51	1	40k	0603 Resistor
Resistor	R52	1	1M	0603 Resistor
SW DIP-8	S1	1		SW DIP-8
SW DIP-4	S2	1		Resistor select
Button	SW1, SW2, SW3	3		Push button
MAX5401EKA	U1, U3, U6, U8	4		MAX5401EKA
LMV552MM	U2, U4, U7, U9	4		100R, LMV552MM
LMV652MM	U5	1		LMV652MM
AT32UC3B1256_n	U10	1		AT32UC3B1256
AT45DB321-SU	U11	1		AT45DB321
LMS8117AMP	U12	1		LMS8117AMP, 3.3V
INA138NA, Current sh	U13, U14, U15, U16	4		INA138NA, Current shunt monitor
LP3992IMF	U17	1		LP3992IMF, 1.5V
LMP7715	U18	1		LMP7715
FT232R - USB - UART	U19	1		FT232R - USB - UART bridge
XTAL	Y1, Y2	2		12MHz, 32.000kHz

7. PCB assembly diagram


Telemetry Buoy Acoustic Receiver Prototype
LEDs
Signal Level
Preamp
Input
P4
P6
R44
R45
AGC & Filter
Power Distribution
U12
GND MCU Det AGC
P5
U17
C62
C63
C61
C60
R37
P3
P2
R41
U15
R36
U13
R40
U14
R43
U16
R39
R35
R38
R42
Q1
F1
P. O.
GND
SGND
VCC
P12
P. In
4.5-6.3V
P9
Ex. Pow.
USB P.
P13 4
U19
D10
R49
J1
USB
S2
R48
R47
R46
Switches
SW1
C27
Trigger
Reception
SW2
C30
Debug
Switch
SW3
R33
C33
Reset
MCU & Flash
Detector
MCU & Flash
U11
C46
P10
C57
C53
C54
C39
C38
C42
C43
U10
Y2
C32
C31
C51
C50
C44
C41
C40
C37
C29
C28
C52
C48
C34
C47
P11
P8
P7
JTAG
THELMA
Telemetry Buoy
Acoustic Receiver
Master Thesis - NTNU
Stian O. Moen - 2009

8. C-code for prototype testing

1) main.c
2) config.h
3) ADC_functions.c
4) ADC_functions.h
5) CLOCK_functions.c
6) CLOCK_functions.h
7) DSP_functions.c
8) DSP_functions.h
9) EIC_functions.c
10) EIC_functions.h
11) RTC_functions.c
12) RTC_functions.h
13) SPI_functions.c
14) SPI_functions.h
15) TC_functions.c
16) TC_functions.h
17) TEST_functions.c
18) TEST_functions.h
19) USART_fucntions.c
20) USART_fucntions.h

```
/*
FILE:           main.c
PROJECT:        Telemetry Buoy project
DESCRIPTION:    Main file for the project
DATE:           17.06.09
AUTHOR:         Stian O. Moen
*/

#include <avr32\io.h>
#include "gpio.h"
#include "usart.h"
#include "pm.h"
#include "USART_functions.h"
#include "config.h"
#include "CLOCK_functions.h"
#include "stdio.h"
#include "tc.h"
#include "TC_functions.h"
#include "adc.h"
#include "ADC_functions.h"
#include "dsp.h"
#include "DSP_functions.h"
#include "intc.h"
#include "dsp_debug.h"
#include "SPI_functions.h"
#include "spi.h"
#include "rtc.h"
#include "RTC_functions.h"
#include "eic.h"
#include "EIC_functions.h"
#include "TEST_functions.h"

void display_level(unsigned short level);

int main(void) {

    //Initialize modules
    INTC_init_interrupts();
    init_clock();
    init_USART();
    init_ADC();
    init_TC();
    init_SPI();
    init_RTC();
    init_EIC();


    usart_write_line(USB_USART, "\n\n *** Telemetry Buoy Running ***\n");

    //set reception detector threshold to 1 V.
    set_threshold(80);

    //set variable gain amplifier gain to 2000, or actually 380 due to unlinearity
    set_gain(2000);

    while(TRUE);


}


//function outputs the signal level on the LEDS as a bar graph
void display_level(unsigned short level)
{
    char leds = (level/1024 + 0.5);

    gpio_set_gpio_pin(LED6);
    gpio_set_gpio_pin(LED5);
    gpio_set_gpio_pin(LED4);
    gpio_set_gpio_pin(LED3);
    gpio_set_gpio_pin(LED2);
    gpio_set_gpio_pin(LED1);

    if (leds >= 1)
        gpio_clr_gpio_pin(LED1);
    if (leds >= 2)
```

```
        gpio_clr_gpio_pin(LED2);
    if (leds >= 3)
        gpio_clr_gpio_pin(LED3);
    if (leds >= 4)
        gpio_clr_gpio_pin(LED4);
    if (leds >= 5)
        gpio_clr_gpio_pin(LED5);
    if (leds >= 6)
        gpio_clr_gpio_pin(LED6);

}
```

```
/*
FILE:           config.h
PROJECT:        Telemetry Buoy project
DESCRIPTION:    Includes configuration parameters
DATE:           17.06.09
AUTHOR:         Stian O. Moen
*/

#ifndef config_h
#define config_h

#include "usart.h"

//Specify the base operating frequency
#define F_CPU   60000000

//Specify the clock speed of the timer/counter used for sampling
#define F_TIM   15000000

//Specify the sampling frequency
#define F_SMP   57692

//specify the number of samples per burst
#define NUMBER_OF_SAMPLES   64  //must be 6*n due to filter optimization

//specify maximum analogue gain
#define MAX_GAIN    178828

//specify the gain of the cascaded stages
#define GAIN1   25
#define GAIN2   625
#define GAIN3   25

//specify the number of steps for the digital potentiometers.
#define POT_RES     256


//Uncomment to select clock source
#define EXTERNAL_CRYSTAL    1
//#define EXTERNAL_CLOCK        1

//The RTC timer prescaler
#define RTC_PRESCALER   4   //will provide the RTC counter to count milliseconds

//GPIO MAP DEFINES
//ADC signal input
#define SIGNAL_IN_PIN           AVR32_ADC_AD_0_PIN
#define SINGAL_IN_ADC_CHANNEL   0
#define SIGNAL_IN_ADC_FUNCTION  AVR32_ADC_AD_0_FUNCTION

//ADC signal level
#define SIGNAL_LEVEL_PIN        AVR32_ADC_AD_1_PIN
#define SIGNAL_LEVEL_CHANNEL    1
#define SIGNAL_LVEL_FUNCTION    AVR32_ADC_AD_1_FUNCTION

//ADC spare input
#define ADC_EXTRA_PIN           AVR32_ADC_AD_4_PIN
#define ADC_EXTRA_CHANNEL       4
#define ADC_EXTRA_FUNCTION      AVR32_ADC_AD_4_FUNCTION

//Reception detecor interrupt pin
#define EXT_INT_RECEPTION_PIN               AVR32_EIC_EXTINT_1_PIN
#define EXT_INT_RECEPTION_FUNCTION          AVR32_EIC_EXTINT_1_FUNCTION
#define EXT_INT_RECEPTION_LINE              EXT_INT1
#define EXT_INT_RECEPTION_IRQ               AVR32_EIC_IRQ_1

//SPI pin definitions
#  define SPI_MOSI_PIN          AVR32_SPI_MOSI_0_0_PIN
#  define SPI_MOSI_FUNCTION     AVR32_SPI_MOSI_0_0_FUNCTION
#  define SPI_MISO_PIN          AVR32_SPI_MISO_0_0_PIN
#  define SPI_MISO_FUNCTION     AVR32_SPI_MISO_0_0_FUNCTION
#  define SPI_SCK_PIN           AVR32_SPI_SCK_0_0_PIN
#  define SPI_SCK_FUNCTION      AVR32_SPI_SCK_0_0_FUNCTION
#  define SPI_NPCS0_PIN         AVR32_SPI_NPCS_0_0_PIN
#  define SPI_NPCS0_FUNCTION    AVR32_SPI_NPCS_0_0_FUNCTION
#  define SPI_NPCS1_PIN         AVR32_SPI_NPCS_1_0_PIN
```

```
#  define SPI_NPCS1_FUNCTION   AVR32_SPI_NPCS_1_0_FUNCTION
#  define SPI_NPCS2_PIN        AVR32_SPI_NPCS_2_0_PIN
#  define SPI_NPCS2_FUNCTION   AVR32_SPI_NPCS_2_0_FUNCTION
#  define SPI_NPCS3_PIN        AVR32_SPI_NPCS_3_0_PIN
#  define SPI_NPCS3_FUNCTION   AVR32_SPI_NPCS_3_0_FUNCTION
#  define SPI_DATAFLASH_CS     AVR32_PIN_PA08

//USB <--> USART definitions
#  define USB_USART               (&AVR32_USART1)
#  define USB_USART_RX_PIN        AVR32_USART1_RXD_0_0_PIN
#  define USB_USART_RX_FUNCTION   AVR32_USART1_RXD_0_0_FUNCTION
#  define USB_USART_TX_PIN        AVR32_USART1_TXD_0_0_PIN
#  define USB_USART_TX_FUNCTION   AVR32_USART1_TXD_0_0_FUNCTION

//PIN definitions
#define LED6    AVR32_PIN_PA27
#define LED5    AVR32_PIN_PA26
#define LED4    AVR32_PIN_PA22
#define LED3    AVR32_PIN_PA21
#define LED2    AVR32_PIN_PA20
#define LED1    AVR32_PIN_PA13

// USART options.
static const usart_options_t USART_OPTIONS =
  {
    .baudrate     = 9600,
    .charlength   = 8,
    .paritytype   = USART_NO_PARITY,
    .stopbits     = USART_1_STOPBIT,
    .channelmode  = USART_NORMAL_CHMODE
  };


#endif
```

```c
/*
 * ADC_functions.c
 *
 *  Created on: 26.jun.2009
 *      Author: Stian O. Moen
 *
 */


#include "ADC_functions.h"
#include "gpio.h"
#include "config.h"
#include "adc.h"
#include "intc.h"
#include "USART_functions.h"
#include "TC_functions.h"
#include "dsp.h"
#include "DSP_functions.h"


volatile signed int sample_nr;

A_ALIGNED dsp16_t sample[NUMBER_OF_SAMPLES];

volatile char sample_flag;

__attribute__((__interrupt__))
static void ADC_complete_irq(void)
{
    //gpio_clr_gpio_pin(AVR32_PIN_PA06);
    volatile avr32_adc_t *adc = &AVR32_ADC;

    if (sample_nr < NUMBER_OF_SAMPLES)
    {
        sample[sample_nr] = (adc->LCDR.ldata - dc_offset)*32;
        sample_nr++;
    }
    else
    {
        disable_sampling();
        // Clear the interrupt flag by reading the ADC_LCDR register.
        volatile unsigned short dummy = adc->LCDR.ldata;
    }

}

void init_ADC(void)
{
    // GPIO pin/adc-function map.
    static const gpio_map_t ADC_GPIO_MAP =
      {
        {AVR32_ADC_AD_0_PIN, AVR32_ADC_AD_0_FUNCTION},
        {AVR32_ADC_AD_1_PIN, AVR32_ADC_AD_1_FUNCTION},
      };

    // ADC IP registers address
    volatile avr32_adc_t *adc = &AVR32_ADC;

    // Assign and enable GPIO pins to the ADC function.-
    gpio_enable_module(ADC_GPIO_MAP, sizeof(ADC_GPIO_MAP) / sizeof(ADC_GPIO_MAP[0]));

    // configure ADC
    adc_configure(adc);

    // Enable the ADC channels.
    adc_enable(adc,SINGAL_IN_ADC_CHANNEL);
    //adc_enable(adc,SIGNAL_LEVEL_CHANNEL);

    //change prescaler settings to satisfy the meximum ADC frequency limitation
    adc->MR.prescal = 3; //prescaler = div16

    //enable hardware trigger from timer counter TIOA channel 0
    adc->MR.trgen = 1;
    adc->MR.trgsel = 0;

    Disable_global_interrupt();
```

```
    // Initialize interrupt vectors.

    // Register the ADC complete interrupt handler to the interrupt controller.
    INTC_register_interrupt(&ADC_complete_irq, AVR32_ADC_IRQ, AVR32_INTC_INT1);

    Enable_global_interrupt();


}

//This function will take one sample of the signal output of the variable gain
//amplifier and filter. It is not used.
signed short get_signal_sample(void)
{
    volatile avr32_adc_t *adc = &AVR32_ADC;
    adc_start(adc);
    return adc_get_value(adc, SINGAL_IN_ADC_CHANNEL);
}


//performs one ADC conversion for the reception detector analogue output.
signed short get_signal_level(void)
{
    volatile avr32_adc_t *adc = &AVR32_ADC;
    adc_enable(adc,SIGNAL_LEVEL_CHANNEL);
    DISABLE_ADC_INTERRUPT();
    adc_start(adc);
    return adc_get_value(adc, SIGNAL_LEVEL_CHANNEL);
}
```

```
/*
 * ADC_functions.h
 *
 *  Created on: 26.jun.2009
 *      Author: Stian O. Moen
 */

#ifndef ADC_FUNCTIONS_H_
#define ADC_FUNCTIONS_H_

#include "config.h"
#include "dsp.h"

#define ENABLE_ADC_INTERRUPT()  adc->IER.drdy = 1;
#define DISABLE_ADC_INTERRUPT() adc->IER.drdy = 0;


//Prototypes
void init_ADC(void);
signed short get_signal_sample(void);
signed short get_signal_level(void);

extern A_ALIGNED dsp16_t sample[];
extern volatile signed int sample_nr;
extern volatile char sample_flag;

#endif /* ADC_FUNCTIONS_H_ */
```

```
/*
FILE:           clock.c
PROJECT:        Telemetry Buoy project
DESCRIPTION:    Functions for changing system or timer clock source and prescaling
DATE:           17.06.09
AUTHOR:         Stian O. Moen
*/

#include "CLOCK_functions.h"

//Init_clock will set the CPU to use the PLL as main oscillator, PLL is
//setup to provide 60Mhz from the external 12MHz crystal

void init_clock( void )
{
#ifdef EXTERNAL_CRYSTAL
    // Switch main clock to external oscillator 0 (crystal).
    pm_switch_to_osc0(&AVR32_PM, FOSC0, AVR32_PM_OSCCTRL0_STARTUP_4096_RCOSC);
#endif
#ifdef EXTERNAL_CLOCK
    //switch to external clock
    pm_enable_osc0_ext_clock(&AVR32_PM);
    pm_enable_clk0(&AVR32_PM, 0);
    pm_switch_to_clock(&AVR32_PM, AVR32_PM_MCSEL_OSC0);
#endif

       volatile avr32_pm_t* pm = &AVR32_PM;

       /* start PLL0 and switch main clock to PLL0 output */
       local_start_pll0(pm);

}

/* Start PLL0, enable a generic clock with PLL0 output then switch main clock to PLL0
output.
   All calculations in this function suppose that the Osc0 frequency is 12MHz. */
void local_start_pll0(volatile avr32_pm_t* pm)
{
//  pm_switch_to_osc0(pm, FOSC0, OSC0_STARTUP);  // Switch main clock to Osc0.

  /* Setup PLL0 on Osc0, mul=9 ,no divisor, lockcount=16, ie. 12Mhzx10 = 120MHz output */
  /*void pm_pll_setup(volatile avr32_pm_t* pm,
                    unsigned int pll,
                    unsigned int mul,
                    unsigned int div,
                    unsigned int osc,
                    unsigned int lockcount) {
   */
  pm_pll_setup(pm,
               0,   // use PLL0
               9,   // MUL=7 in the formula
               1,   // DIV=1 in the formula
               0,   // Sel Osc0/PLL0 or Osc1/PLL1
               16); // lockcount in main clock for the PLL wait lock

  /*
   This function will set a PLL option.
   *pm Base address of the Power Manager (i.e. &AVR32_PM)
   pll PLL number 0
   pll_freq Set to 1 for VCO frequency range 80-180MHz, set to 0 for VCO frequency
range 160-240Mhz.
   pll_div2 Divide the PLL output frequency by 2 (this settings does not change the
FVCO value)
   pll_wbwdisable 1 Disable the Wide-Bandith Mode (Wide-Bandwith mode allow a faster
startup time and out-of-lock time). 0 to enable the Wide-Bandith Mode.
  */
  /* PLL output VCO frequency is 120MHz. We divide it by 2 with the pll_div2=1. This
enable to get later main clock to 60MHz */
  pm_pll_set_option(pm, 0, 1, 1, 0);

  /* Enable PLL0 */
  pm_pll_enable(pm,0);

  /* Wait for PLL0 locked */
  pm_wait_for_pll0_locked(pm) ;
```

```
  //setup PLL
  pm_gc_setup(pm,
              EXAMPLE_GCLK_ID,
              1,  // Use Osc (=0) or PLL (=1), here PLL
              0,  // Sel Osc0/PLL0 or Osc1/PLL1
              0,  // disable divisor
              0); // no divisor

  /* Enable Generic clock */
  pm_gc_enable(pm, EXAMPLE_GCLK_ID);

  pm_cksel(pm, 1, 0, 0, 0, 0, 0);

  // Set one wait-state (WS) for flash controller. 0 WS access is up to 30MHz for
HSB/CPU clock.
  // As we want to have 60MHz on HSB/CPU clock, we need to set 1 WS on flash controller.
  flashc_set_wait_state(1);

  pm_switch_to_clock(pm, AVR32_PM_MCSEL_PLL0); /* Switch main clock to 60MHz */
}
```

```
/*
FILE:           clock.h
PROJECT:        Telemetry Buoy project
DESCRIPTION:    Functions for changing system or timer clock source and prescaling
DATE:           17.06.09
AUTHOR:         Stian O. Moen
*/


#ifndef clock_h
#define clock_h

#include "config.h"
#include "pm.h"
#include "gpio.h"
#include "flashc.h"

#  define EXAMPLE_GCLK_ID             2
#  define EXAMPLE_GCLK_PIN            AVR32_PM_GCLK_2_PIN
#  define EXAMPLE_GCLK_FUNCTION       AVR32_PM_GCLK_2_FUNCTION
#  define FOSC0 12000000

void init_clock( void );
void local_start_pll0(volatile avr32_pm_t* pm);
#endif
```

```c
/*
 * DSP_functions.c
 *
 *  Created on: 28.jun.2009
 *      Author: Stian O. Moen
 */

#include "dsp.h"
#include "DSP_functions.h"
#include "config.h"
#include "ADC_functions.h"
#include "TC_functions.h"


//The DC level of the signal for the 1.5V regualtor
A_ALIGNED dsp16_t dc_offset = 465;

//will reuturn a struct containing the frequency of the highest power with
//the assisated power. The rreturn frequency is scaled to acount for the
//undersampling.
FreqPower_t find_dominating_frequency(dsp16_complex_t *complex_vector)
{
    dsp16_t vector_abs[NUMBER_OF_SAMPLES];
    dsp16_vect_complex_abs(vector_abs, complex_vector, NUMBER_OF_SAMPLES);
    //dsp16_debug_print_vect(vector_abs, NUMBER_OF_SAMPLES);
    //find highest value
    dsp16_t max_value = dsp16_vect_max(vector_abs, NUMBER_OF_SAMPLES);
    //find the location of the highest value

    int i = 0;
    char exit = FALSE;
    while (exit == FALSE)
    {
        i++;
        if (max_value == vector_abs[i] || i >= NUMBER_OF_SAMPLES)
            exit = TRUE;
    }
    FreqPower_t return_value;
    //calculate the alias frequency in kHz for 64 point two sided FFT
    return_value.freq = (i*(F_SMP/2)/32.0)/1000.0+F_SMP/1000.0;

    return_value.power = max_value;
    return return_value;
}

//The function will perform a sample burst, remove the DC offset and
//find the dominating frequency and assosiated power. This frequency
//and power is returned
FreqPower_t check_signal(void)
{
    A_ALIGNED dsp16_complex_t vect1[NUMBER_OF_SAMPLES];
    enable_sampling();
    while(sample_flag == TRUE);
    //remove DC
    int i;
    for (i = 0; i < NUMBER_OF_SAMPLES; i++)
    {
        sample[i] -= dc_offset;
    }


    dsp16_trans_realcomplexfft(vect1, sample, NLOG);
    return find_dominating_frequency(vect1);
}
```

```
/*
 * DSP_functions.h
 *
 *  Created on: 28.jun.2009
 *      Author: Stian O. Moen
 */

#ifndef DSP_FUNCTIONS_H_
#define DSP_FUNCTIONS_H_

#include "dsp.h"

//needed for FFT routine
#define NLOG 6

#define SIZE    NUMBER_OF_SAMPLES

//The struct is used to specify a frequency with an assosiated power
typedef struct FreqPower{
    unsigned short freq;
    unsigned short power;
} FreqPower_t;


extern A_ALIGNED dsp16_t dc_offset;


FreqPower_t find_dominating_frequency(dsp16_complex_t *complex_vector);
FreqPower_t check_signal(void);


#endif /* DSP_FUNCTIONS_H_ */
```

```c
/*
 * EIC.functions.c
 *
 *  Created on: 02.jul.2009
 *      Author: Stian O. Moen
 */

#include "compiler.h"
#include "EIC_functions.h"
#include "eic.h"
#include "intc.h"
#include "gpio.h"
#include "RTC_functions.h"
#include "USART_functions.h"
#include "usart.h"
#include "DSP_functions.h"
#include "config.h"


//variables used by the external intertupt handler
time_variable_t current_pulse, previous_pulse;



//debug function: converts unsigned short to string.
void short_to_char(char *string, unsigned short number)
{
    string[0] = number/10000 + 0x30;
    number -= (number/10000)*10000;
    string[1] = number/1000 + 0x30;
    number -=(number/1000)*1000;
    string[2] = number/100  + 0x30;
    number -= (number/100)*100;
    string[3] = number/10   + 0x30;
    number -= (number/10)*10;
    string[4] = number  + 0x30;
    string[5] = '\0';
}


//Interrupt handler of the External interrupt reception detector
__attribute__((__interrupt__))
static void eic_int_reception_detector(void)
{
    char temp_string[6];
    gpio_clr_gpio_pin(LED6);

    //Save vurrent time in temporary variable
    current_pulse = get_time();
    //adjust gain here.

    //Perform sample_burst, FFT an find dominating frequency and power
    FreqPower_t signal = check_signal();

    //assume valid pulse.calculate time and print data
    time_variable_t difference = calculate_time_diff(current_pulse, previous_pulse);
    if (difference.millisec < 0)
        difference = calculate_time_diff(current_pulse, previous_pulse);
    previous_pulse = current_pulse;

    //output the difference
    usart_write_line(USB_USART, "Pulse detected:\n space: ");
    short_to_char(temp_string, difference.millisec);
    usart_write_line(USB_USART, temp_string);
    usart_write_line(USB_USART, " ms, ");
    short_to_char(temp_string, difference.sec);
    usart_write_line(USB_USART, temp_string);
    usart_write_line(USB_USART, " s\n");

    //output the frequency and power
    usart_write_line(USB_USART, "Frequency: ");
    short_to_char(temp_string, signal.freq);
    usart_write_line(USB_USART, temp_string);
    usart_write_line(USB_USART, " kHz\n");
    usart_write_line(USB_USART, " Power: ");
    short_to_char(temp_string, signal.power);
```

```
    usart_write_line(USB_USART, temp_string);
    usart_write_line(USB_USART, "\n\n\n");

    gpio_set_gpio_pin(LED6);
    eic_clear_interrupt_line(&AVR32_EIC, EXT_INT_RECEPTION_LINE);
}



//Intitializes external interrupts
void init_EIC(void)
{
    // Structure holding the configuration parameters of the EIC module.
    eic_options_t eic_options_reception;

      // Enable edge-triggered interrupt.
    eic_options_reception.eic_mode   = EIC_MODE_EDGE_TRIGGERED;
      // Interrupt will trigger on rising edge.
    eic_options_reception.eic_edge  = EIC_EDGE_RISING_EDGE;
      // Initialize in synchronous mode : interrupt is synchronized to the clock
    eic_options_reception.eic_async  = EIC_SYNCH_MODE;
      // Set the interrupt line number.
    eic_options_reception.eic_line   = EXT_INT_RECEPTION_LINE;

    // Map the interrupt lines to the GPIO pins with the right peripheral functions.
    gpio_enable_module_pin(EXT_INT_RECEPTION_PIN, EXT_INT_RECEPTION_FUNCTION);

    Disable_global_interrupt();

    //register interrupt handler
    INTC_register_interrupt(&eic_int_reception_detector, EXT_INT_RECEPTION_IRQ,
AVR32_INTC_INT0);

    // Init the EIC controller with the options
    eic_init(&AVR32_EIC, &eic_options_reception, 1);

    // Enable the chosen lines and their corresponding interrupt feature.
    eic_enable_line(&AVR32_EIC, eic_options_reception.eic_line);
    eic_enable_interrupt_line(&AVR32_EIC, eic_options_reception.eic_line);

    Enable_global_interrupt();
}
```

```
/*
 * EIC_functions.h
 *
 *  Created on: 02.jul.2009
 *      Author: Stian O. Moen
 */

#ifndef EIC_FUNCTIONS_H_
#define EIC_FUNCTIONS_H_

#include "eic.h"
#include "compiler.h"
#include "RTC_functions.h"

//prototypes
void init_EIC(void);

//global variables
extern time_variable_t current_pulse, previous_pulse;

#endif /* EIC_FUNCTIONS_H_ */
```

```c
/*
 * RTC_functions.c
 *
 *  Created on: 02.jul.2009
 *      Author: Stian O. Moen
 */



#include "rtc.h"
#include "config.h"
#include "compiler.h"
#include "intc.h"
#include "RTC_functions.h"
#include "gpio.h"

//global variable containing the
time_variable_t time;


//RTC interrupt routine, updates a simplified real time clock.
//A clock with date and year can easily be implemented but is not
//needed for testing purpose.
__attribute__((__interrupt__))
void rtc_irq(void)
{
    //gpio_tgl_gpio_pin(AVR32_PIN_PA07);
  // Increment the seconds counter and handle other variables
  time.sec++;
  if (time.sec >= 60)
  {
      time.sec = 0;
      time.min++;
      if (time.min >= 60)
      {
          time.min = 0;
          time.hour++;
          if (time.hour >= 24)
          {
              time.hour = 0;
              time.day++;
          }
      }
  }

  // clear the interrupt flag
  rtc_clear_interrupt(&AVR32_RTC);
}

//the function will initialize the RTC module to use the external 32.000
//crystal as source, devide by 32 to count milliseconds and generate an
//interrupt every 1000ms to count seconds as well as setting this as the top
//for the timer.
void init_RTC(void)
{
    //initialize time to zero, this is only for testing. A real RTC should
    //always run and have functions to set the time. This can easily be
    //implemented when the user interface is defined.
    time.sec = 0;
    time.min = 0;
    time.hour = 0;
    time.day = 0;

    Disable_global_interrupt();

    //register RTC interrupt routine
    INTC_register_interrupt(&rtc_irq, AVR32_RTC_IRQ, AVR32_INTC_INT0);

    //initalize RTC to use 32kHz source
    rtc_init(&AVR32_RTC,/*0*/ RTC_OSC_32KHZ, RTC_PRESCALER);

    //set top to 1000 to generate interrupt each second.
    rtc_set_top_value(&AVR32_RTC, 1000);

    //Enable interrupt
    rtc_enable_interrupt(&AVR32_RTC);
```

```
    // Enable the RTC
    rtc_enable(&AVR32_RTC);

    Enable_global_interrupt();
}

//function returns time; intead of reading the global variable directly a
//this function assures that no writing is ongoing or will start during the
//read. This will prevent inconsistance between the time variables.
time_variable_t get_time(void)
{
    while(rtc_is_interrupt(&AVR32_RTC)); //wait until RTC interrupt is finished
    Disable_global_interrupt();
    time_variable_t temp = time;
    temp.millisec = (unsigned short)rtc_get_value(&AVR32_RTC);
    Enable_global_interrupt();
    return temp;
}

//calculates t1-t2. t1 must be larger than t2 to avoid negative time.
time_variable_t calculate_time_diff(time_variable_t t1, time_variable_t t2)
{
    time_variable_t temp;

    //calculate days
    temp.day = t1.day - t2.day;

    //calculate hours
    if(t1.hour - t2.hour >= 0)
    {
        temp.hour = t1.hour - t2.hour;
    }
    else
    {
        temp.hour = t1.hour + 24 - t2.hour;
        temp.day--;
    }

    //calculate minutes
    if(t1.min - t2.min >= 0)
    {
        temp.min = t1.min - t2.min;
    }
    else
    {
        temp.min = t1.min + 60 - t2.min;
        temp.hour--;
    }

    //calculate seconds
    if(t1.sec - t2.sec >= 0)
    {
        temp.sec = t1.sec - t2.sec;
    }
    else
    {
        temp.sec = t1.sec + 60 - t2.sec;
        temp.min--;
    }

    //calculate milliseconds
    if (t1.millisec - t2.millisec >= 0)
        temp.millisec = t1.millisec - t2.millisec;
    else
    {
        temp.millisec = t1.millisec + 1000 - t2.millisec;
        temp.sec--;
    }

    return temp;
}
```

```
/*
 * RTC_functions.h
 *
 *  Created on: 02.jul.2009
 *      Author: Stian O. Moen
 */

#ifndef RTC_FUNCTIONS_H_
#define RTC_FUNCTIONS_H_

#include "compiler.h"

typedef struct time_variable {
volatile short millisec;
volatile U8 sec;
volatile U8 min;
volatile U8 hour;
volatile short day;
} time_variable_t;


//prototypes
void init_RTC(void);
time_variable_t get_time(void);
time_variable_t calculate_time_diff(time_variable_t t1, time_variable_t t2);

#endif /* RTC_FUNCTIONS_H_ */
```

```
/*
 * SPI_functions.c
 *
 *  Created on: 30.jun.2009
 *      Author: Stian O. Moen
 */

#include "spi.h"
#include "compiler.h"
#include "config.h"
#include "SPI_functions.h"
#include "gpio.h"

volatile avr32_spi_t *spi = &AVR32_SPI;

unsigned char pot1_setting = 0;
unsigned char pot2_setting = 0;
unsigned char pot3_setting = 0;

//SPI options for setup with the dataflash
static const spi_options_t DATAFLASH_options =
{
    .reg = 3,
    .baudrate = 10000000,
    .bits = 8,
    .spck_delay = 0,
    .trans_delay = 0,
    .stay_act = 1,
    .spi_mode = 0,
    .modfdis = 1,
};

//SPI options for potentiometer updates.
static spi_options_t POTENTIOMETER_options =
{
    .reg = 0,
    .baudrate = 10000000,
    .bits = 8,
    .spck_delay = 5,
    .trans_delay = 5,
    .stay_act = FALSE,
    .spi_mode = 0,
    .modfdis = TRUE
};
//initialise to use potentiometer settings and set as master.
//The CS for the dataflash is set to high manually by the function
void init_SPI(void)
{
    static const gpio_map_t SPI_GPIO_MAP =
      {
        {SPI_SCK_PIN, SPI_SCK_FUNCTION},
        {SPI_MISO_PIN, SPI_MISO_FUNCTION},
        {SPI_MOSI_PIN, SPI_MOSI_FUNCTION},
        {SPI_NPCS0_PIN, SPI_NPCS0_FUNCTION},
        {SPI_NPCS1_PIN, SPI_NPCS1_FUNCTION},
        {SPI_NPCS2_PIN, SPI_NPCS2_FUNCTION},
        {SPI_NPCS3_PIN, SPI_NPCS3_FUNCTION}
      };

      // Assign GPIO to SPI
      gpio_enable_module(SPI_GPIO_MAP,
                         sizeof(SPI_GPIO_MAP) / sizeof(SPI_GPIO_MAP[0]));

      //initialize as master
      spi_initMaster(spi, &POTENTIOMETER_options);

      // Set selection mode: variable_ps, pcs_decode, delay.
      spi_selectionMode(spi, 0, 0, 0);

      // Enable SPI.
      spi_enable(spi);

      //set manual chip select for dataflash to default high
      gpio_set_gpio_pin(SPI_DATAFLASH_CS);

      //initialize all potentiometers
```

```
      short i;
      for (i = 0; i < 4; i++)
      {
          POTENTIOMETER_options.reg = i;
          spi_setupChipReg(spi, &POTENTIOMETER_options, F_CPU/2);
      }

}

//Funtion sets the gain using the digital potentiometers
//This function will provide a maximum level throughout the
//cicuit by prioritizing the first gain stages.
void set_gain(U32 gain)
{
    unsigned short n1,n2,n3;    //potentiometer settings
    float g1, g2, g3;           //gain of the induvidual stages

    //calculate gain for each gain stage
    if (gain <= GAIN1) //gain is only needed in stage 1
    {
        g1 = gain;
        g2 = 1;
        g3 = 1;
    }
    else if(gain <= GAIN1*GAIN2)
    {
        g1 = GAIN1;
        g2 = gain / GAIN1;
        g3 = 1;
    }
    else
    {
        g1 = GAIN1;
        g2 = GAIN2;
        g3 = (gain / GAIN1) / GAIN2;
    }

    //calculate potentiometer settings and ensure that overflow does not occur
    n1 = (g1 * POT_RES / GAIN1 + 0.5);
    if (n1 > 255) n1 = 255;
    n2 = (g2 * POT_RES / GAIN2 + 0.5);
    if (n2 > 255) n2 = 255;
    else if (n2 < 1) n2 = 1;
    n3 = (g3 * POT_RES / GAIN3 + 0.5);
    if (n3 > 255) n3 = 255;

    //set potentiometer gain only if different from last value (to reduce noise)
    if (n1 != pot1_setting)
        update_potentiometer(0, (char) n1);
    if (n2 != pot2_setting)
        update_potentiometer(1, (char) n2);
    if (n3 != pot3_setting)
        update_potentiometer(2, (char) n3);
}

//updates the given potentiometer with the raw input data
void update_potentiometer(unsigned char potentiometer, char data)
{
    spi_selectChip(spi, potentiometer);
    while(spi_writeEndCheck(spi) == 0);
    spi_write(spi, data);
    spi_unselectChip(spi, potentiometer);
}

//sets the threshold for the reception detector interrup with the raw input data
void set_threshold(unsigned char data)
{
    update_potentiometer(3, data);
}

//writes up-to one page to address zero. Different addresses can easily be implemented
//but is not needed for current measurement testing.
unsigned char store_data_in_dataflash(unsigned char *data, unsigned short size)
{
    //perform sanity check
    if(size > DATAFLASH_PAGE_SIZE)
```

```
        return 0;
    spi_setupChipReg(spi, &DATAFLASH_options, F_CPU/2);
    //The dataflash is not connected to a NPCS pin. Instead a gpio pin is set
    //manually to provide the chip select. To avoid that the SPI module
    //selects another device while programmeing the flash the NPCS3 pin is
    //rearranged to a gpio pin, thereby removing NPCS3 control for the SPI module
    //And the NPCS3 is selected when communicating with the serial flash.
    gpio_enable_gpio_pin(SPI_NPCS3_PIN);

    //set high to not select device before intentional select
    gpio_set_gpio_pin(SPI_NPCS3_PIN);
    spi_selectChip(spi, 3);
    //select dataflash
    gpio_clr_gpio_pin(SPI_DATAFLASH_CS);

    //check that SPI is not busy
    while(spi_writeEndCheck(spi) == 0);

    //write command "buffer1 write"
    spi_write(spi, CMD_WRITE_BUFFER_1);
    while(spi_writeEndCheck(spi) == 0);

    //write dont cares and buffer start index = 0
    spi_write(spi, 0);
    while(spi_writeEndCheck(spi) == 0);
    spi_write(spi, 0);
    while(spi_writeEndCheck(spi) == 0)
    ;spi_write(spi, 0);
    while(spi_writeEndCheck(spi) == 0);

    //write data to buffer
    int i;
    for(i = 0; i < size; i++)
    {
        spi_write(spi, data[i]);
        while(spi_writeEndCheck(spi) == 0);
    }
    //signal end of command by setting CS high
    gpio_set_gpio_pin(SPI_DATAFLASH_CS);

    //send command "Buffer 1 to Main Memory Page Program with Built-in Erase"
    gpio_clr_gpio_pin(SPI_DATAFLASH_CS);
    spi_write(spi, CMD_BUFFER1_TO_MEM_WITH_ERASE);
    while(spi_writeEndCheck(spi) == 0);

    //write address bits and don't care. for testing the address is always 0
    spi_write(spi, 0);
    while(spi_writeEndCheck(spi) == 0);
    spi_write(spi, 0);
    while(spi_writeEndCheck(spi) == 0)
    ;spi_write(spi, 0);
    while(spi_writeEndCheck(spi) == 0);

    //deselect serial flash
    gpio_set_gpio_pin(SPI_DATAFLASH_CS);

    //set options back to potentiometer settings
    POTENTIOMETER_options.reg = 3;
    spi_setupChipReg(spi, &POTENTIOMETER_options, F_CPU/2);

    //arrange NPCS3 to SPI module
    gpio_enable_module_pin(SPI_NPCS3_PIN, SPI_NPCS3_FUNCTION);

    return 1;
}

//reads up-to one page to address zero. Different addresses can easily be implemented
//but is not needed debug testing. This rutine is only used to test the correctness
//of the write routine.
unsigned char read_data_from_dataflash(unsigned char *data, unsigned short size)
{
    //perform sanity check
    if(size > DATAFLASH_PAGE_SIZE)
        return 0;
    spi_setupChipReg(spi, &DATAFLASH_options, F_CPU/2);
    //The dataflash is not connected to a NPCS pin. Instead a gpio pin is set
```

```
    //manually to provide the chip select. To avoid that the SPI module
    //selects another device while programmeing the flash the NPCS3 pin is
    //rearranged to a gpio pin, thereby removing NPCS3 control for the SPI module
    //And the NPCS3 is selected when communicating with the serial flash.
    gpio_enable_gpio_pin(SPI_NPCS3_PIN);

    //set high to not select device before intentional select
    gpio_set_gpio_pin(SPI_NPCS3_PIN);
    spi_selectChip(spi, 3);
    //select dataflash
    gpio_clr_gpio_pin(SPI_DATAFLASH_CS);

    //check that SPI is not busy
    while(spi_writeEndCheck(spi) == 0);

    //write command "Read page from main memory"
    spi_write(spi, CMD_MAIN_PAGE_READ);
    while(spi_writeEndCheck(spi) == 0);

    //write three address bytes and four dummy bytes according to datasheet
    unsigned short i;
    for(i=0; i < 7; i++)
    {
        spi_write(spi, 0);
        while(spi_writeEndCheck(spi) == 0);
    }

    //read data
    for(i = 0; i < size; i++)
    {
        //write dummy to receive data
        spi_write(spi,0);
        static unsigned short temp;
        spi_read(spi, &temp);
        data[i] = (unsigned char) temp;
    }


    //deselect serial flash
    gpio_set_gpio_pin(SPI_DATAFLASH_CS);

    //set options back to potentiometer settings
    POTENTIOMETER_options.reg = 3;
    spi_setupChipReg(spi, &POTENTIOMETER_options, F_CPU/2);

    //arrange NPCS3 to SPI module
    gpio_enable_module_pin(SPI_NPCS3_PIN, SPI_NPCS3_FUNCTION);
    return 1;
}
```

```
/*
 * SPI_functions.h
 *
 *  Created on: 30.jun.2009
 *      Author: Stian O. Moen
 */

#ifndef SPI_FUNCTIONS_H_
#define SPI_FUNCTIONS_H_


//Specifies chip select order
#define DIGPOT1_CS  0
#define DIGPOT2_CS  1
#define DIGPOT3_CS  2
#define DIGPOT4_CS  3
#define FLASH_CS    0



//dataflash defines
#define CMD_WRITE_BUFFER_1              0x84
#define CMD_BUFFER1_TO_MEM_WITH_ERASE   0x83
#define CMD_MAIN_PAGE_READ              0xD2
#define DATAFLASH_PAGE_SIZE             528

//Prototypes
void update_potentiometer(unsigned char potentiometer, char data);
void set_gain(U32 gain);
void set_threshold(unsigned char data);
void init_SPI(void);
unsigned char store_data_in_dataflash(unsigned char *data, unsigned short size);
unsigned char read_data_from_dataflash(unsigned char *data, unsigned short size);

#endif /* SPI_FUNCTIONS_H_ */
```

```c
/*
 * TC_functions.c
 *
 *  Created on: 22.jun.2009
 *      Author: Stian O. Moen
 */



#include "tc.h"
#include "TC_functions.h"
#include "compiler.h"
#include "gpio.h"
#include "intc.h"
#include "USART_functions.h"
#include "ADC_functions.h"
#include <AVR32/io.h>
#include "ADC_functions.h"
#include "adc.h"

//Global variables
volatile avr32_tc_t *tc = &AVR32_TC;


//initializes the timer counter to the options set in WAVEFORM_OPT.
//the timer is used to generate a hardware signal on timer/counter
//channel 0, triggering an ADC conversion.
void init_TC( void )
{
      // Initialize the timer/counter waveform.
      tc_init_waveform(tc, &WAVEFORM_OPT);
      tc_write_ra(tc, TC_CHANNEL, 10); //assign a low value to reset TIOA
      //Set ADC samplig frequency
      tc_write_rc(tc, TC_CHANNEL, (F_TIM / 4) / F_SMP);
      //ensure that the timer is stopped and reset so the sampling does not start
      tc_stop(tc, TC_CHANNEL);
}

//This function will enable timer counter channel 0 to trigger an ADC conversion
// by hardware through TIOA
void enable_sampling(void)
{
    //include reset of timer
    sample_nr = 0;
    sample_flag = TRUE;
    //enable ADC complete interrupt
    volatile avr32_adc_t *adc = &AVR32_ADC;
    ENABLE_ADC_INTERRUPT();
    tc_start(tc, TC_CHANNEL);                    // Start the timer/counter.
}

//this function will stop the timer and disable the ADC interrupt
void disable_sampling(void)
{
    tc_stop(tc, TC_CHANNEL);                    // Stop the timer/counter and reset
counter
    sample_flag = FALSE;
    volatile avr32_adc_t *adc = &AVR32_ADC;
    DISABLE_ADC_INTERRUPT();
}
```

```
/*
 * TC_functions.h
 *
 *  Created on: 22.jun.2009
 *      Author: Stian O. Moen
 */

#ifndef TC_FUNCTIONS_H_
#define TC_FUNCTIONS_H_

#include "config.h"
#include "tc.h"

//specify the frequency of FPBA
#define FPBA    F_CPU
//specify timer/counter channel to be used to trigger ADC conversion
#define TC_CHANNEL    0


//Prototypes

  void init_TC( void );
  void enable_sampling(void);
  void disable_sampling(void);

  extern volatile U32 tc_tick;
  extern volatile U16 print_sec;
  extern volatile avr32_tc_t *tc;

 // Options for waveform generation setting up the hardware signal to trigger
 // ADC conversions
    static const tc_waveform_opt_t WAVEFORM_OPT =
    {
      .channel  = TC_CHANNEL,                         // Channel selection.

      .bswtrg   = TC_EVT_EFFECT_NOOP,                 // Software trigger effect on TIOB.
      .beevt    = TC_EVT_EFFECT_NOOP,                 // External event effect on TIOB.
      .bcpc     = TC_EVT_EFFECT_NOOP,                 // RC compare effect on TIOB.
      .bcpb     = TC_EVT_EFFECT_NOOP,                 // RB compare effect on TIOB.

      .aswtrg   = TC_EVT_EFFECT_NOOP,                 // Software trigger effect on TIOA.
      .aeevt    = TC_EVT_EFFECT_NOOP,                 // External event effect on TIOA.
      .acpc     = TC_EVT_EFFECT_CLEAR,                 // RC compare effect on TIOA:
clear. this signal is used to trigger ADC conversion
      .acpa     = TC_EVT_EFFECT_SET,                // RA compare effect on TIOA: set.

      .wavsel   = TC_WAVEFORM_SEL_UP_MODE_RC_TRIGGER,// Waveform selection: Up mode
with automatic trigger(reset) on RC compare.
      .enetrg   = FALSE,                              // External event trigger enable.
      .eevt     = 0,                                  // External event selection.
      .eevtedg  = TC_SEL_NO_EDGE,                     // External event edge selection.
      .cpcdis   = FALSE,                              // Counter disable when RC compare.
      .cpcstop  = FALSE,                              // Counter clock stopped with RC
compare.

      .burst    = FALSE,                              // Burst signal selection.
      .clki     = FALSE,                              // Clock inversion.
      .tcclks   = TC_CLOCK_SOURCE_TC3                // Internal source clock 3,
connected to fPBA / 8.
    };
#endif /* TC_FUNCTIONS_H_ */
```

```
/*
 * test_functions.c
 *
 *  Created on: 16.jul.2009
 *      Author: Stian O. Moen
 */

#include <avr32\io.h>
#include "gpio.h"
#include "usart.h"
#include "pm.h"
#include "USART_functions.h"
#include "config.h"
#include "CLOCK_functions.h"
#include "stdio.h"
#include "tc.h"
#include "TC_functions.h"
#include "adc.h"
#include "ADC_functions.h"
#include "dsp.h"
#include "DSP_functions.h"
#include "intc.h"
#include "dsp_debug.h"
#include "SPI_functions.h"
#include "spi.h"
#include "rtc.h"
#include "RTC_functions.h"
#include "eic.h"
#include "EIC_functions.h"
#include "TEST_functions.h"


//this function includes the code needed to provide the functionality and performance
//tests. To prevent interference between tests comment out the tests that are not
//to be performed only leaving one test. The complete system test is implemented
//to the system using the standard setup.
void test(void)
{
/*
//test #5
    set_gain(1);
    set_gain(10);
    set_gain(100);
    set_gain(200);

//test #6
    set_gain(1);
    set_gain(10);
    set_gain(100);
    set_gain(200);

//test #7
    set_gain(100);
    set_threshold(70);

//test #8
    usart_write_line(USB_USART, "Test successful\n");

//test #9
    int i;
    for(i =0; i < 20; i++)
    {
        volatile time_variable_t time = get_time();
        usart_putchar(USB_USART, time.millisec);
    }

//test #10
    //create data set and store to flash
    unsigned char flash_store_data[512];
    for (i = 0; i < 512; i++)
    {
        flash_store_data[i] = 0x55;
    }
    store_data_in_dataflash(flash_store_data, sizeof(flash_store_data));

    //read data black from flash
```

```
    unsigned char flash_read_data[512];
    for(i = 0; i < 512; i++)
    {
        flash_read_data[i] = 0;
    }
    read_data_from_dataflash(flash_read_data, sizeof(flash_read_data));

    //validate data
    char flag = 0;
    for(i = 0; i < 512; i++)
    {
        if (flash_read_data[i] != flash_store_data[i])
            flag = 1;
    }
    if (flag == 1)
        usart_write_line(USB_USART, "Flash FAIL with 0x55\n");
    else
        usart_write_line(USB_USART, "Flash OK with 0x55\n");

    //Perform same test with data = 0x11
    for (i = 0; i < 512; i++)
    {
        flash_store_data[i] = 0x11;
    }
    store_data_in_dataflash(flash_store_data, sizeof(flash_store_data));

    //read data black from flash
    read_data_from_dataflash(flash_read_data, sizeof(flash_read_data));

    //validate data
    flag = 0;
    for(i = 0; i < 512; i++)
    {
        if (flash_read_data[i] != flash_store_data[i])
            flag = 1;
    }
    if (flag == 1)
        usart_write_line(USB_USART, "Flash FAIL with 0x11\n");
    else
        usart_write_line(USB_USART, "Flash OK with 0x11\n");

//test #11
    set_gain(1);
    set_gain(2);
    set_gain(5);
    set_gain(10);
    set_gain(20);
    set_gain(50);
    set_gain(100);
    set_gain(200);
    set_gain(500);
    set_gain(1000);
    set_gain(2000);
    set_gain(5000);
    set_gain(10000);
    set_gain(20000);
    set_gain(50000);
    set_gain(100000);
    set_gain(140000);

//test #12
    set_gain(10);
    //SLEEP(AVR32_PM_SMODE_STATIC); //uncomment when performing test 12.

//test #13
    set_gain(1);
    set_gain(2);
    set_gain(5);
    set_gain(10);
    set_gain(20);
    set_gain(50);
    set_gain(100);
    set_gain(200);
    set_gain(500);
    set_gain(1000);
    set_gain(2000);
```

```
    set_gain(5000);
    set_gain(10000);
    set_gain(20000);
    set_gain(50000);
    set_gain(100000);
    set_gain(140000);

//test #14
    set_gain(1);
    set_gain(2);
    set_gain(5);
    set_gain(10);
    set_gain(20);
    set_gain(50);
    set_gain(100);
    set_gain(200);
    set_gain(500);
    set_gain(1000);
    set_gain(2000);
    set_gain(5000);
    set_gain(10000);
    set_gain(20000);
    set_gain(50000);
    set_gain(100000);
    set_gain(140000);

//test #15
    set_gain(1);

//test #16
    set_gain(1);

//test #17
    set_gain(1);

//test #18
    gpio_set_gpio_pin(LED1);
    enable_sampling();
    while(sample_flag == TRUE);
    gpio_clr_gpio_pin(LED1);

    unsigned char test[512];
    gpio_set_gpio_pin(LED1);
    store_data_in_dataflash(test, sizeof(test));
    gpio_clr_gpio_pin(LED1);

    gpio_set_gpio_pin(LED1);
    set_gain(12000);
    gpio_clr_gpio_pin(LED1);


    gpio_set_gpio_pin(LED1);
    time_variable_t time = get_time();
    gpio_clr_gpio_pin(LED1);

    time_variable_t time2 = get_time();
    gpio_set_gpio_pin(LED1);
    calculate_time_diff(time2, time);
    gpio_clr_gpio_pin(LED1);

    A_ALIGNED dsp16_complex_t vect1[NUMBER_OF_SAMPLES];
    gpio_set_gpio_pin(LED1);
    FreqPower_t test_freq =  find_dominating_frequency(vect1);
    gpio_clr_gpio_pin(LED1);

    gpio_set_gpio_pin(LED1);
    signed short temp_level  = get_signal_level();
    gpio_clr_gpio_pin(LED1);
*/
//test #19; NB! should be performed without any initialization in main()
    enable_all_pullups();
    init_RTC();
    SLEEP(AVR32_PM_SMODE_DEEP_STOP);
}

void enable_all_pullups(void)
```

```
{
    //gpio_enable_pin_pull_up(AVR32_PIN_PA03);
    //gpio_enable_pin_pull_up(AVR32_PIN_PA04);
    gpio_enable_pin_pull_up(AVR32_PIN_PA07);
    gpio_enable_pin_pull_up(AVR32_PIN_PA13);
    gpio_enable_pin_pull_up(AVR32_PIN_PA20);
    gpio_enable_pin_pull_up(AVR32_PIN_PA21);
    gpio_enable_pin_pull_up(AVR32_PIN_PA22);
    gpio_enable_pin_pull_up(AVR32_PIN_PA26);
    gpio_enable_pin_pull_up(AVR32_PIN_PA27);
    gpio_enable_pin_pull_up(AVR32_PIN_PA05);
    //gpio_enable_pin_pull_up(AVR32_PIN_PA06);
    gpio_enable_pin_pull_up(AVR32_PIN_PA25);
    gpio_enable_pin_pull_up(AVR32_PIN_PA14);
    gpio_enable_pin_pull_up(AVR32_PIN_PA15);
    gpio_enable_pin_pull_up(AVR32_PIN_PA16);
    gpio_enable_pin_pull_up(AVR32_PIN_PA17);
    gpio_enable_pin_pull_up(AVR32_PIN_PA09);
    gpio_enable_pin_pull_up(AVR32_PIN_PA10);
    gpio_enable_pin_pull_up(AVR32_PIN_PA08);
    gpio_enable_pin_pull_up(AVR32_PIN_PA23);
    gpio_enable_pin_pull_up(AVR32_PIN_PA24);

    //oscillator pins
    gpio_enable_pin_pull_up(AVR32_PIN_PA18);
    gpio_enable_pin_pull_up(AVR32_PIN_PA19);
    gpio_enable_pin_pull_up(AVR32_PIN_PA11);
    gpio_enable_pin_pull_up(AVR32_PIN_PA12);


}
```

```
/*
 * test_functions.h
 *
 *  Created on: 16.jul.2009
 *      Author: Stian O. Moen
 */

#ifndef TEST_FUNCTIONS_H_
#define TEST_FUNCTIONS_H_

#define LED0    AVR32_PIN_PA13

void test(void);
void enable_all_pullups(void);


#endif /* TEST_FUNCTIONS_H_ */
```

```
/*
FILE:           USART_functions.c
PROJECT:        Telemetry Buoy project
DESCRIPTION:    Includes functions for debug-output as well as output of data
DATE:           17.06.09
AUTHOR:         Stian O. Moen
*/


#include "USART_functions.h"
#include "config.h"

//This function initializes the USART to communicate with the USB-UART bridge
//Requires that the clock source is initialized
void init_USART(void)
{
      static const gpio_map_t USART_GPIO_MAP =
      {
        {USB_USART_RX_PIN, USB_USART_RX_FUNCTION},
        {USB_USART_TX_PIN, USB_USART_TX_FUNCTION}
      };

      // Assign GPIO to USART.
      gpio_enable_module(USART_GPIO_MAP,
                         sizeof(USART_GPIO_MAP) / sizeof(USART_GPIO_MAP[0]));

      // Initialize USART in RS232 mode. Must be changed if PLL setup is altered
      usart_init_rs232(USB_USART, &USART_OPTIONS, F_CPU/2);
}
```

```
/*
FILE:           USART_functions.h
PROJECT:        Telemetry Buoy project
DESCRIPTION:    Includes functions for debug-output as well as output of data
DATE:           17.06.09
AUTHOR:         Stian O. Moen
*/

#ifndef USART_funtions_h
#define USART_funtions_h


#include <avr32/io.h>
#include "compiler.h"
#include "pm.h"
#include "gpio.h"
#include "usart.h"
#include "config.h"
#include "stdio.h"


//Prototypes
void init_USART(void);
void write_character( char c );

#endif
```