Improving the analogue section of IO extenders for a hydraulic cylinder controller

Mattijn Spitteler (m.spitteler@student.utwente.nl)

Committee chair & daily supervisor	Amir Yousefzadeh PhD
External daily supervisor	Rudi Koskamp
Committee member	dr. ir. Marco Ottavi
External committee member	dr. ir. Mark Oude Alink

Faculty of EEMCS — University of Twente

BRUNELCO
ELECTRONIC INNOVATORS

CONTENTS

I. Introduction

A. Application

The SCC (Smart Cylinder Control) Compact is a product designed by Brunelco Electronic Innovators [1]. It is used as an industrial controller for hydraulic cylinders in heavy lifting applications. The SCC Compact itself is basically an interface between the analogue sensors and actuators and a digital control system. The SCC Compact has analogue and digital inputs and outputs, the specifications and uses of these will be discussed in a later section.

B. SCC Compact IO extenders (existing solution)

There also exist extenders for the SCC Compact system. These extenders have either 8 analogue or digital inputs or outputs and can be connected to an SCC Compact via a CAN bus.

C. Problems with IO extenders

The current extender designs have proven to work in the field since their creation almost 15 years ago. There are however some improvements possible in the context of robustness and the current solutions also don't entirely conform to the international standards that apply, such as [2]. A desirable feature that is lacking in the current designs is software configurability of the type of analogue input or output (current or voltage). If a redesign is to take place, it is useful to investigate whether this feature can be added. There are some recently developed integrated circuits, such as the NAFE family from NXP [3], known as industrial analogue front-ends, which could integrate a lot of the functionality required in an SCC IO extender. Besides these problems which apply to both analogue input and output extenders, the analogue output extenders also could benefit from improved performance. Relevant performance metrics are settle time, noise and distortion in this case.

D. Research questions

- What improvements regarding versatility and robustness can be made to an analogue input section for an industrial controller, with the use of an integrated analogue front-end (AFE)?
- What is the feasibility of alternatives to low-pass filtered PWM as high resolution analogue outputs, optimizing for low settle time and low noise, in the context of an industrial controller?

E. Solution

An analysis of all types of the existing extenders is presented and preliminary solutions are proposed with a prototype for the analogue input and output extenders. Measurements and tests will be done with this prototype and the existing SCC Compact extenders. Comparison between these measurements will yield a recommendation for a possible future redesign cycle.

II. Background

A. IEC 61131-2 standard and additional requirements

The IEC 61131-2 standard provides voltage and current ratings of different types of industrial inputs and outputs. For the SCC Compact extenders it is only relevant to look at sections "6.4.4 Digital inputs (positive logic, current sinking)", "6.4.6 Digital outputs for direct current (current sourcing)", "6.5.2 Analog inputs", "6.5.3 Analog outputs" and "6.5.6 Verification of analog I/Os" in [2].

1) Digital inputs: DC digital inputs on the SCC extenders use a rated voltage of 24V [4] and need to be compatible with both type 1 (suitable for mechanical contact switching devices) and type 3 (suitable for solid state switching devices). This means that the upper two rows of Table 24 in [2] are of interest. These two rows can be seen repeated in TABLE I.

	Type 1		Type 3	
	min	max	min	max
U_L	-3V	5V/15V	-3V	5V/11V
I_L	ND	15mA	ND	15mA
U_T	5V	15V	5V	11V
I_T	0.5mA	15mA	1.5mA	15mA
U_H	15V	30V	11V	30V
I_H	2mA	15mA	2mA	15mA

TABLE I: Digital input types
Source: adapted from [2]

In [2] it is stated that "It is necessary to exceed both U_T min and I_T min to leave the "off region", and to exceed both I_H min and U_H min to enter the "on region".". This means that if a purely resistive load is connected on a digital input for type 1 it ideally needs to have resistance $R = \min\left(\frac{U_H\text{min}}{I_H\text{min}}, \frac{U_T\text{min}}{I_T\text{min}}\right) = \min\left(7.5 \cdot 10^3, 1.0 \cdot 10^4\right) = 7.5\text{k}\Omega$ for minimal power dissipation and 3.3kΩ for type 3. Power dissipation can however be significantly reduced if above some voltage (for example U_T min) the digital input sinks a constant current that is higher than I_H min.

Since the ranges for U_T are so large, it is possible and maybe beneficial for interference rejection to have a hysteresis present on the transition from low to high and visa-versa, which may occupy a part of (or the whole of) the U_T range.

2) Digital outputs: The digital outputs on the SCC extenders are specified for an operating voltage of 24V [4]. There are 5 types of digital outputs standardized in [2]; type 0,1, 0,25, 0,5, 1 and 2 rated for a sourcing current of 100mA, 250mA, 500mA, 1A and 2A respectively. Each type has to be implemented as a high-side switch, which can handle a continuous current of 1.2 times the rated current. Only type 2 outputs are of interest for the SCC extenders [4], which have as additional requirements that the leakage current shall be less than 1mA and the voltage drop under load less than 3V.

3) Analogue inputs: The only analogue input types of interest are $0V - 10V$, which needs to have an input impedance of $\geq 10k\Omega$ and $4mA - 20mA$ which needs to have a input impedance of $\leq 300\Omega$ [2], [5]. The resolution must be at least 16-bit [4], although the effective number of bits (ENOB) only has to be at least 10-bit. Software configurability of current or voltage input is a desirable feature.

4) Analogue outputs: The only analogue output types of interest are $0V - 10V$, which needs to work as specified with an impedance of $\geq 1k\Omega$ and $4mA - 20mA$ which needs to work as specified with an impedance of $\leq 600\Omega$ [2], [5]. All analogue outputs should be able to withstand any resistive load from short circuit to open circuit [2]. The resolution again must be at least 16-bit [4], and ENOB at least 10-bit.

B. Interface between SCC compact and extenders

An isolation barrier between the CAN bus and inputs/outputs of extenders is required to prevent ground loops. The isolation voltage must be at least 2.5kVrms.

1) CAN bus: The CAN bus has a bitrate of 1Mbit/s and the SCC compact and extenders use a proprietary higher level protocol on top of the CAN hardware layer also developed by Brunelco, so this will have to be taken into account to maintain compatibility.

C. IEC 61000-4-2 and IEC 61000-4-5 standards

The IEC 61000-4 standards provide testing and measurement techniques for electromagnetic compatibility (EMC). Part 4-2 provides standards for testing electrostatic discharge (ESD) immunity [6] and 4-5 provides surge immunity standards [7]. The product falls under the "industrial" category for CE certification [8]. The European Committee for Electrotechnical Standardization CLC Guide 34 lists the standards which the product needs to conform to. These include the IEC 61000-6-2 standard for immunity in industrial environments and IEC 61000-6-4 for emission in industrial environments [8]. The standard for immunity is the only relevant one when selecting an ADC type of device. Analogue inputs and outputs fall under signal/control port [9]. In Table 2 in [9] it is stated that a surge according to [7] of $\pm 1kV$ must be survived by the device. When doing a redesign it is useful to look at whether a chosen integrated circuit has already been tested to withstand certain standardized surges and ESD.

D. Requirements summary

A summary of the requirements on the analogue section for a new design set by Brunelco (drawn partly from existing designs) and the international standards that apply can be seen in TABLE II

	Analogue inputs	Analogue outputs
Input/output voltage range	$0V - 10V$	$0V - 10V$
Input/output current range	$4mA - 20mA$	$4mA - 20mA$
Voltage mode input impedance/output load impedance	$\geq 10k\Omega$	$\geq 1k\Omega$
Current mode input impedance/output load impedance	$\leq 300\Omega$	$\leq 600\Omega$
Resolution	16-bit	16-bit
Maximum noise and distortion (voltage mode)	$5mV_{RMS}$	$5mV_{RMS}$
Configurability of current/voltage mode	In software	In software
IEC 61000-4-5 surge immunity	$\pm 1kV$	$\pm 1kV$

TABLE II: Requirements summary

III. Current designs

A. Shared

The existing SCC extender designs are all have a common part, which is the CAN interface and the microcontroller. The microcontroller is an STM32F103 on all 4 designs. Its most important feature in this case is that it has a CAN interface built-in [10], which removes the need for an external interface [5], [11]–[13]. The update period of the inputs and outputs on the CAN bus is 10ms, which means that 50Hz is the theoretical highest frequency which can be generated, or measured without aliasing.

1) Shortcomings of the current design: All CAN transceivers are implemented with the TJA1042T [5], [11]–[13], which has supply voltage range of 4.5V - 5.5V [14], this means the TX and RX pins connected to the microcontroller are also 5V logic IO, which is not compatible with the STM32F103 3.3V IO [10].

Input TX needs to be at least $0.7\times$ the supply for it to be seen as a HIGH level, see Table 7 in [14], which is 3.5V for nominal 5V, this requirement is not fulfilled with a 3.3V GPIO output from the microcontroller. RX from the CAN interface IC will be 5V, which can damage the microcontroller input if it is not 5V tolerant.

In practice it does work in the current design because most of the STM32F103 GPIO internal buffers are 5V tolerant [10], and 3.3V is still close enough to 3.5V for the CAN interface to detect it as a high level most likely (it's just not within specifications).

When a redesign is to take place, the microcontroller family of choice is the STM32G0, which uses an ARM Cortex® M0+ core, since it contains a memory protection unit (MPU) and hardware based encryption (AES) [15]. These are all security related features, making the STM32G0 family a good choice for applications where security and reliability are important aspects. This microcontroller family is also PSA certified [16].

B. Digital inputs

The digital input extender has 8 inputs which conform to all types defined in [2], and are electrically isolated from the rest of the design. There are also indicator LEDs for each channel which turn on when a high level is detected, and off otherwise.

1) Shortcomings of the current design: On the input side there is basically a constant current sink in series with the LED of an optocoupler for each channel [12]. This constant current sink is constructed with a BF556A J-FET with its V_{GS} always zero, which results in an I_D which converges to a constant 4.5mA typically [17]. An additional resistive divider is placed on the input to prevent the LED in the optocoupler turning on with a small leakage current below I_T from TABLE I.

There is nothing intrinsically wrong with this design, and it even has the nice feature of not dissipating an unnecessarily high amount of power at higher input voltages as would be the case with a purely resistive input. The BF556A is however no longer being manufactured and J-FETs are going obsolete left and right, so making a similar redesign with another J-FET is not very future-proof.

C. Digital outputs

The digital output extender has 8 output channels which conform to type 2 defined in [2]. They are not electrically isolated, and again there are indicator LEDs for each channel that are on when a high level is outputted, and off otherwise.

1) Shortcomings of the current design: The high-side switches are implemented with an integrated solution; the BTS5210L [18]. Which already take the IEC61131-2 standard into account. The device contains a protection circuit for inductive loads that will turn on the built-in MOSFETs slightly when the voltage across the load exceeds a certain threshold effectively damping the negative inductive spike. This clamping voltage is about 47V [18]. There are however also 30V metal oxide varistors (MOV) on the output which will break down before the built-in protection has any chance to dampen the spike [13]. Since a MOV degrades every time a surge occurs, and are really only designed to operate in exceptional cases it is not a very robust design.

The indicator LEDs are connected directly to the outputs in series with 6.8kΩ resistors, which will yield a maximum power dissipation in all of these resistors above the rated 0.1W for the 0603 package when the common voltage is the maximum specified 40V. Besides this technicality which could easily be solved by using larger resistors, it is kind of wasteful to dissipate around 1.7W when the supply is 40V. Directly powering the LEDs from the output also has as a negative side effect that when a negative inductive spike occurs, it could easily exceed the reverse breakdown voltage of the LEDs (which is about 5V typically, so the MOVs and protection built-in to the BTS5210L also don't help).

The digital outputs are not electrically isolated from the CAN interface, which should be the case in a new design.

2) Specification irregularities: In the specification document it is stated that the digital output is rated for 2.4A, but this is not one of the standardized types, so it should really state 2A, which also should be able to handle a continuous current of $1.2 \cdot 2\text{A} = 2.4\text{A}$ [2]. A maximum load inductance of 30mH is specified. This is probably obtained by looking at the graph "Maximum allowable load inductance for a single switch off" in [18] at $I_L = 2.4$A, but the graph is only valid for $V_{bb} = 12$V which is not clarified in the extender specifications, and one would assume that this 30mH is for the nominal common voltage of 24V [13]

D. Analogue inputs

1) Shortcomings of the current design: The analog inputs are implemented with a simple voltage divider with a ratio of about $2.65 : 1$ and an LTC1867 SAR ADC with built-in analogue multiplexer and voltage reference [5]. This yields an input voltage range of $0\text{V} - 3.77\text{V}$ to the ADC. The internal 2.5V reference is used in combination with the 1.638V/V reference amplifier, which means the actual reference voltage for the ADC is 4.095V [19], so the design doesn't use the full dynamic range of the ADC.

To enable current inputs instead of voltage inputs, mechanical switches are used to wire in an effective shunt resistance of 495Ω [5], which is above the upper limit of 300Ω that the standard requires [2].

The specified resolution is 16-bit, which is also what the ADC specifies, but the signal to noise and distortion ratio (SINAD) is 88dB meaning an ENOB of 14.3-bit [19]. However, there is an RC low-pass filter on the input with a cut-off frequency of $f_c = \frac{1}{2\pi RC} = \frac{1}{2\pi \cdot 12452.8 \cdot 10 \cdot 10^{-6}} \approx 1.28$Hz [5], so a very cheap improvement would be to oversample a lot (since the analogue bandwidth is extremely low). This way an ENOB of almost 16-bit could be achieved.

The analogue inputs are not electrically isolated from the CAN interface, which should be the case in a new design.

2) Specification irregularities: The specified input current range is $0\text{mA} - 20\text{mA}$, which is no longer recommended for new designs [2], [20]. This is because a $4\text{mA} - 20\text{mA}$ input can be used to detect an open circuit (when the current is below 4mA).

E. Analogue outputs

1) Shortcomings of the current design: The analogue output extenders can only implement voltage output, but software configurable current output may be implemented in a new design. The analogue voltage outputs are essentially 1st order low pass filtered PWM channels directly from the microcontroller fed into some LM258 operational amplifiers configured with fixed amplification using negative feedback. These low-pass filters have a time constant of $\tau = 0.1$, and therefore a cut-off frequency of around 1.59Hz [11]. This means the analogue outputs have a settle time of around $t = -\tau \ln \frac{1}{2^{10}} \approx 0.7$s when assuming a desired ENOB of 10-bit.

The cut-off frequency was chosen so low because the minimum obtainable PWM period is approximately 1ms. This is limited by the maximum clock frequency of 64MHz and minimum resolution of 16-bit, yielding 65536 codes. Since a low-pass filter is used with a -20dB/dec slope, this design will have around -60dB attenuation of the PWM AC component. This will make it theoretically possible to obtain an ENOB of at least 9.7-bit. The total SNR will depend heavily on the PWM compare value/analogue output value.

The analogue outputs are not electrically isolated from the CAN interface, which should be the case in a new design.

IV. Literature review for analogue outputs

A. DAC devices

The safest solution to improve the analogue output section would be to use 8 separate DAC devices or one 8 channel DAC device or anything in between. This would be quite expensive however, if you would set 16-bit resolution as a requirement. There don't exist any STM32 microcontrollers with 16-bit DACs, let alone ones with 8 output channels, so selecting a microcontroller with an internal DAC and using this is not really an option. Using high resolution DACs would also be a little bit overkill, since the settle time doesn't have to be very low (< 10ms doesn't make any sense because the update period is 10ms), and the analogue outputs can be considered closer to a precise voltage/current source than a DAC. There are also no heavy requirements in frequency domain; the only relevant aspects of the transient response to a step are that there is no over- and undershoot.

B. Implementation in microcontroller

1) PWM: The current design uses PWM since it is easy and reliable because there are enough built-in peripherals (timers with compare) for this. This solution does result in a lot of noise after filtering though, because high resolution causes a large PWM period. It would be possible to use a 2nd order analogue filter. This could increase the cut-off frequency by a factor $10\sqrt{10}$, yielding a settle time of around 22ms, which is already more than fast enough.

2) Sigma-Delta modulation: An idea is to implement Sigma-Delta modulation (SDM) in software on the microcontroller, and 1st order SDM would already be a large improvement, since the noise shaping characteristic results in noise mostly being shifted towards the high frequencies (with a slope of 20dB/dec), so when applying the same 1st order analogue filter, the noise power will be much lower. When the input to 1st order SDM is rational number b/a normalized to quantizer step, the period of the limit cycle will be a regardless of the initial condition [21]. The input is always this rational number b/a for a digital to digital converter (DDC) or DAC, as b is an integer and a is a power of 2. This can intuitively be understood by observing that the integrated error (Sigma) will always be equal to 0 again after a time steps. One can therefore concatenate one period indefinitely yielding exactly the same bitstream (when assuming a constant input, which is valid, as the update period ($= 10$ms) is significantly higher than the limit cycle period (≈ 1ms assuming 64MHz clock)). Knowing this, it is possible to implement the computation in software only when the input value is updated and circularly write out a 2^{16} bit buffer using direct memory access (DMA). When the input is a constant value, 1st order SDM can also be understood in the time domain as pulses spaced apart by a period which is inversely proportional to the input. This can be understood intuitively since an SDM essentially integrates the difference between the input and the output (illustrated in Fig. 1), where this output is a 1-bit signal. As said before the error Δ will be 0 when averaged over one limit cycle, so the total number of pulses in one period has to be equal to the N-bit input. The distance between these pulses will be constant since the output of the integrator Σ linearly increases over time until a pulse occurs (so it will have a sawtooth shape).

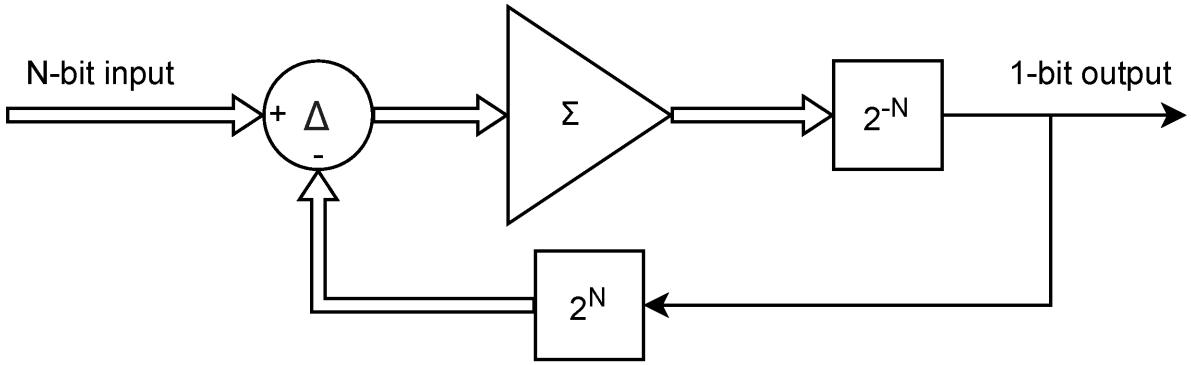


Fig. 1: Digital 1st order SDM block diagram

It can be shown that a voltage controlled oscillator ADC (VCO-ADC) not only produces a signal with the same frequency spectrum on its output, but also generates exactly the same bit pattern (time-domain signal) as a 1st order SDM [22]. This can also be understood intuitively since in a VCO-ADC, the VCO will output a squarewave whose period is inversely proportional to the input voltage. This is then 1-bit differentiated in a sense (two inputs of a XOR gate connected to the same signal, but one via a single timestep delay). The output of this XOR gate will therefore be pulses spaced apart by the inverse of the input voltage. The fact that the bit pattern has the aforementioned property is even clearer from the core concept of a VCO-ADC than from an SDM. A block diagram of such a VCO-ADC is shown in Fig. 2.

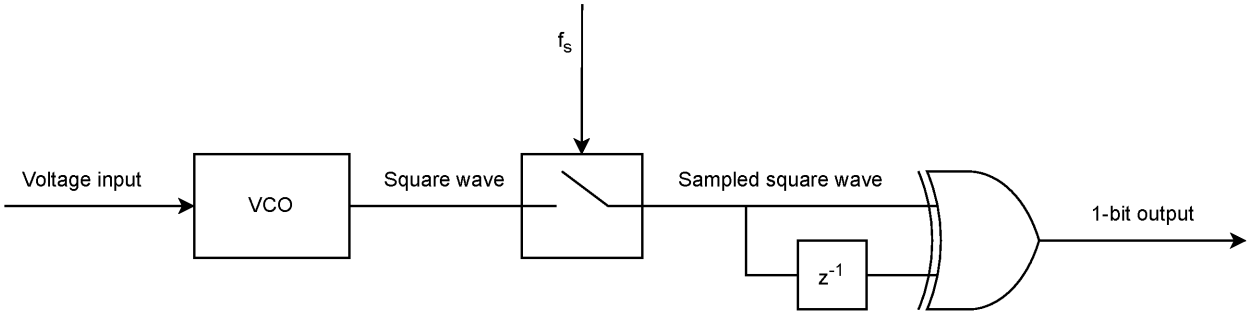


Fig. 2: VCO-ADC block diagram
Source: adapted from [23]

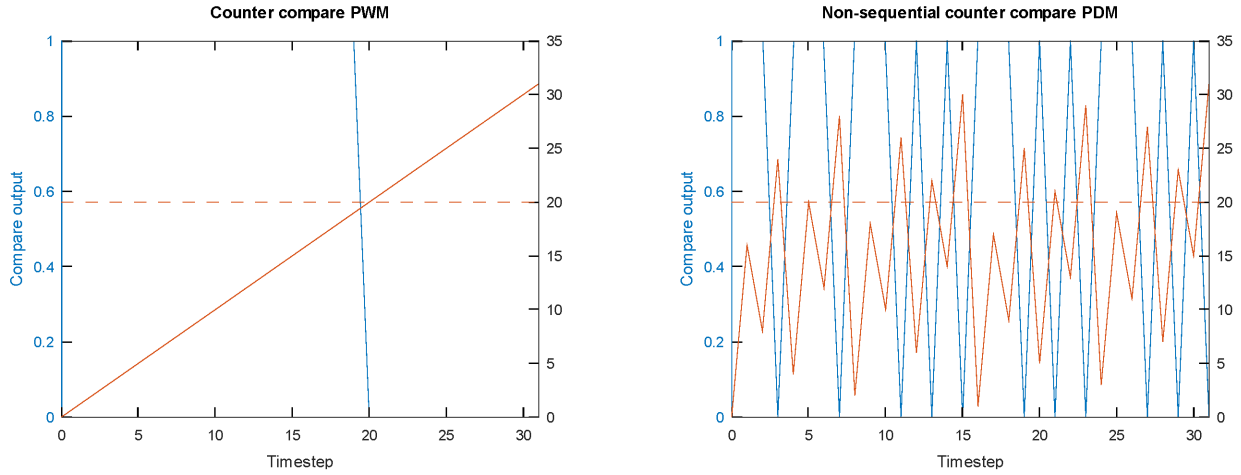
The whole point of this whole elaborate comparison is that there is a major advantage when implementing an algorithm based on a VCO-ADC in software (and not in real-time) over implementing SDM in software; there is no feedback. This means that for an SDM, the state (from which the output value follows) of the system has to be calculated at every timestep of the algorithm sequentially, while for a VCO-ADC based algorithm, the output value of the system can theoretically be calculated for all times simultaneously.

It may sound a bit strange to implement an ADC in software, but is is mostly just (a slightly modified version of) the core concept which can be implemented in software. This core concept is also sometimes called pulse frequency modulation (PFM) or pulse density modulation (PDM), I will use the latter going forward.

3) Alternatives: There are some other methods to generate something close or identical to PDM from a VCO-ADC or SDM. The first which could be interesting to consider is the method presented in [24], the working principle is very close to that of PWM, the only difference being that the counter is non-sequential. This essentially distributes the pulses over the entire counter period. In hardware the easiest way to make a counter non-sequential would be to reverse the bit order. The difference in operation can be seen in Fig. 3.

The downside of this method is that the pulses are not equally distributed over the period. This means that the frequency spectrum of the output will also not have the characteristic 20dB/dec slope. A comparison of the frequency spectra for a resolution of 16-bit and a constant input of 40961 (which is a prime number to prevent a limit cycle shorter than $a = 2^{16} = 65536$) can be seen in Fig. 4

The same signals were filtered with a 1st order low-pass filter with a cut-off of $\Omega = 1.561 \cdot 10^{-7}$rad, which corresponds to a cut-off frequency of 1.59Hz when the sample frequency is 64MHz. Their frequency spectra can be seen in Fig. 5



(a) 5-bit PWM with compare level equal to 20

(b) 5-bit PDM with compare level equal to 20

Fig. 3: PWM and non-sequential counter PDM comparison. Dashed orange line is compare level (reference input), uninterrupted orange line is counter level.

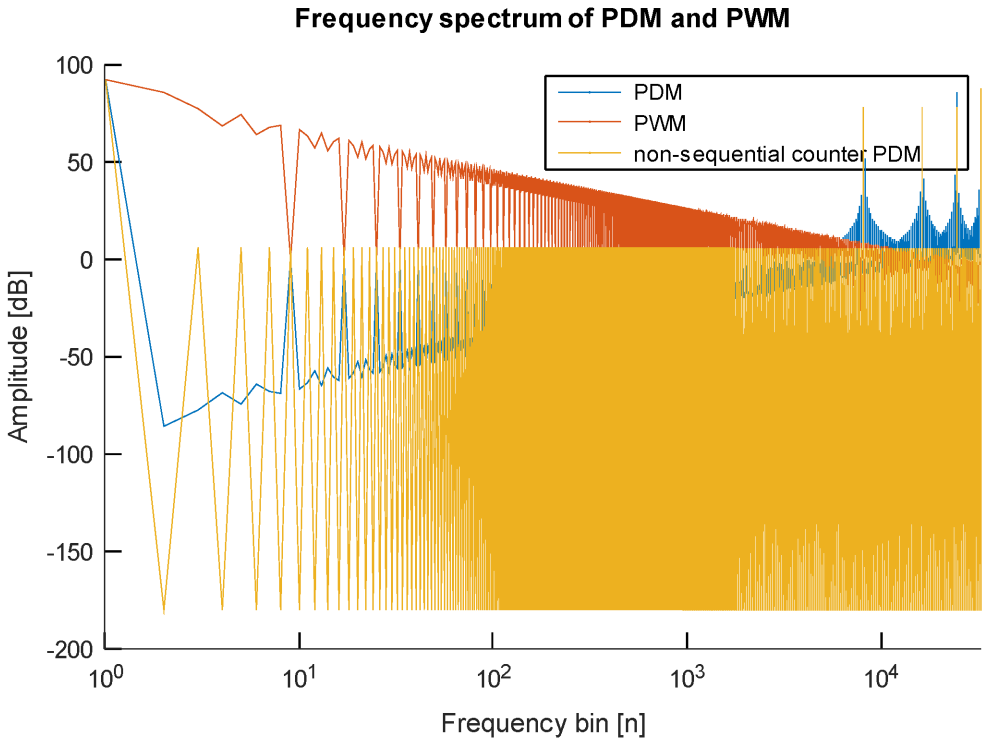


Fig. 4: Frequency spectrum comparison of PWM, PDM generated with a non-sequential counter and PDM generated by a SDM or VCO-ADC.

This method doesn't have any advantages when implementing it in software over a method based on a VCO-ADC, and it may be hard or impossible to find a method for transforming the counter sequence in hardware to a non-sequential one which has an better shaped frequency spectrum for all compare values, but it could be interesting to see if there are more optimal transformations than reversing the bit order which could be implemented in software. When this method would be implemented using digital hardware, it would be an interesting alternative or addition to regular timer/counter peripherals, since it is doesn't require more hardware (so less than a 1st order SDM too). Such a hardware peripheral is very uncommon however.

When a simple implementation in hardware is desirable (so that it may be used as a peripheral in a microcontroller for example), the method presented in [25] may also be more interesting when 1st order noise shaping is a must. This method uses only adders and is very similar to a 1st order SDM. It uses an integrator where the overflow/carry generates an output

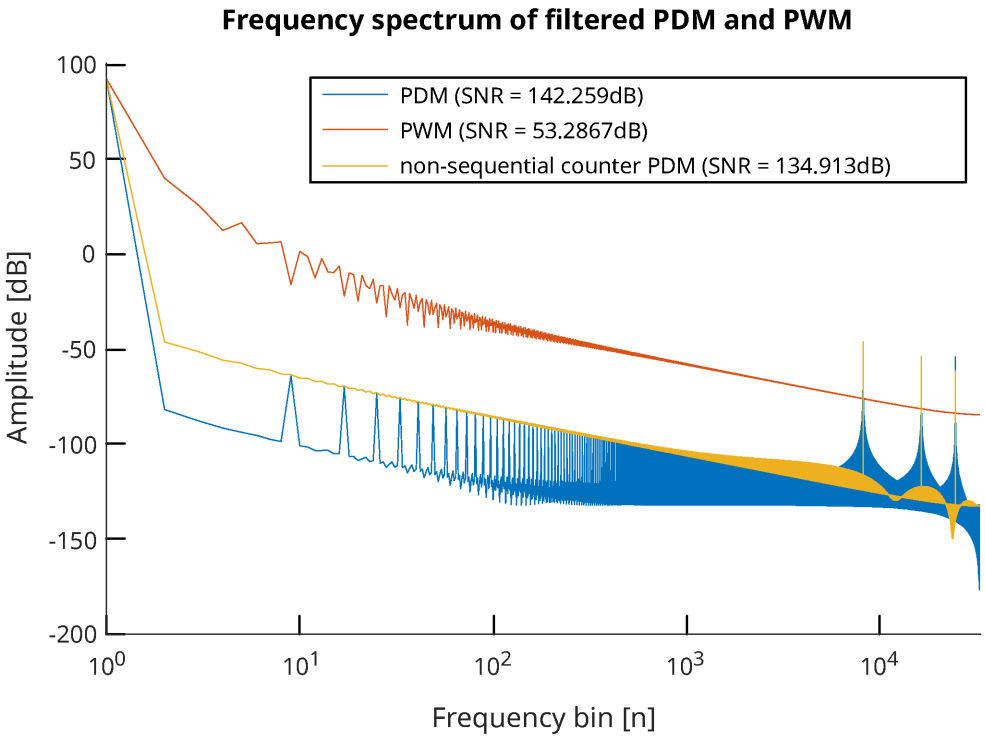


Fig. 5: Frequency spectrum comparison of filtered PWM, PDM generated with a non-sequential counter and PDM generated by a SDM or VCO-ADC.

pulse.

Since the microcontroller family which would be used in a redesign is a given, an implementation in software is worked out in the next section.

V. PDM IMPLEMENTATION

To implement something efficiently on a microcontroller which generates PDM, it is useful to first look at what would happen if one would just replace the VCO and sampler with a function that generates a square wave (which will be inherently sampled) with period $\frac{a}{b}$, where b is the reference input and a is the number of quantization steps. The output will then have $2b$ pulses per a samples after it is 1-bit differentiated, since a square wave has 2 edges per period. When this squarewave would be swapped out to a sawtooth wave, and do regular differentiation, the output will have b negative pulses per a samples, since a sawtooth wave has 1 falling edge per period. There will also be an offset equal to $A\frac{b}{a}$, since the sawtooth wave has a constant gradient of $\frac{\Delta y}{\Delta x} = A\frac{b}{a}$, where A is the amplitude of the sawtooth wave. Let's set A to 1 for now, so that the output offset is equal to $\frac{b}{a}$, which is exactly the same as the reference input normalized to quantization step. Essentially what has been constructed now is an SDM inside out (sort of). The sawtooth wave can be seen as the output of the integrator. If we differentiate this (which is what happens next in the VCO-ADC), we can see this as going in the reverse direction in the SDM (so the input of the integrator). The input of the integrator in the SDM is indeed the same as the output of the differentiator in the VCO-ADC. Since they are both equal to the sum of the constant input and b negative pulses per a samples.

A. Implementation in MATLAB

An implementation in MATLAB as close as possible to the description above was tested to have exactly the same bit pattern for $a = 2^{16}$ and every possible value of b except 0. This implementation can be seen in Listing 1, the SDM implementation in MATLAB, matched as closely as possible to the one in Fig 1 can be seen in Listing 2.

Listing 1: Algorithm based on VCO-ADC

```
% Algorithm which is the most directly based on a VCO-ADC.
function y = vco_adc(ref, N)
    % Generate sawtooth with period N / ref and amplitude 1.
    sawtooth = mod((0:N)', N / ref) * ref / N;
    y = -(diff(sawtooth) - ref / N);
end
```

Listing 2: MATLAB implementation of Fig. 1

```
% First order classic sigma delta.
function y = sigma_delta(ref, N)
    sigma = 0;
    y = zeros(N, 1);
    for i = 0:N
        out = floor(sigma / N);
        delta = ref - out * N;
        sigma = sigma + delta;

        if (i > 0)
            y(i, 1) = out;
        end
    end
end
```

For the actual implementation on a microcontroller, this algorithm can be simplified a lot still, since instead of doing differentiation, one can just detect the falling edge by comparing the sawtooth in Listing 1 to $\frac{b}{a}$ and emitting a pulse every time the sawtooth is smaller. The comparison involves $\frac{b}{a}$ because of the fractional modulo. $\frac{b}{a}\left(n \mod \frac{a}{b}\right) < \frac{b}{a} \to n \mod \frac{a}{b} < 1$ where n is the timestep, is true exactly once per period of the sawtooth because the lowest point of the sawtooth will always be where modulo is lower than the timestep increment, which is 1 by definition.

An implementation according to this simplification can be seen in Listing 3.

Listing 3: Simplified algorithm based on VCO-ADC

```
% Stateless pulse density calculation algorithm.
% Theoretically you can calculate all output values simultaneously.
function y = pulse_density(ref, N)
    high_pulse_period = N / ref;
    y = double(mod((1:N)', high_pulse_period) <= (N - 1)/N);
end
```

This implementation uses a comparison $\leq \frac{a-1}{a}$ instead of < 1 because of floating point rounding errors in the `mod` MATLAB function. The exact same code, but then with the < 1 comparison was tested in GNU Octave, which did yield the expected results. The output was verified to have exactly the same bit pattern for $a = 2^{16}$ and every possible value of b except 0, as the previous two implementations.

Another approach which is still according to the same principle is possible, but instead of using a fractional modulo which would be prohibitively expensive to implement on a microcontroller, it just uses a single division to calculate the fractional period/distance between pulses and only sets bits which need to be set.

We can define a set of indices Y at which a pulse occurs over one period (always with length a). Let's define $N := a$ for the period to improve readability:

$$\begin{aligned} Y &= \left\{ n : n \mod \frac{N}{b} < 1 \middle| n \in \{0, \ldots, N-1\} \right\} \\ &= \left\{ n : n - \frac{N}{b} \left\lfloor \frac{b}{N} n \right\rfloor < 1 \middle| n \in \{0, \ldots, N-1\} \right\} \\ &= \left\{ \left\lceil \frac{N}{b} n \right\rceil : \frac{N}{b} n - \frac{N}{b} \left\lfloor \frac{bN}{Nb} n \right\rfloor < 1 \middle| n \in \{0, \ldots, b-1\} \right\} \\ &= \left\{ \left\lceil \frac{N}{b} n \right\rceil \middle| n \in \{0, \ldots, b-1\} \right\} \end{aligned} \tag{1}$$

An implementation according to this more efficient definition can be seen in Listing 4

Listing 4: Efficient pulse density algorithm

```
% Only assign ones. Very efficient for lowest levels. As a possible optimization
% you could transitions to low_pulse_period once you go past the middle level.
% This would also make it very efficient for highest levels.
function y = efficient_pulse_density(ref, N)
    high_pulse_period = N / ref;
    y = zeros(N, 1);
    y(ceil(high_pulse_period * (1:ref) - 1/N)) = 1;
end
```

The output was verified to have exactly the same bit pattern for $a = 2^{16}$ and every possible value of b except 0, as the previous three implementations.

B. Implementation in C

An implementation in C can be found in Listing 5 under the name `NON_OPTIMIZED_ALGORITHM`. Some things have to be taken into account in this low level rewrite. The most important one is that the output bits are packed most significant bit first, in a buffer with 8- or 16-bit elements. Another is that this buffer has to be written out via DMA continuously, so asynchronously to what is happening on the main microcontroller core. This means that from the perspective of the DMA controller, during a change of the analogue output value we can't make any assumptions about which part of the buffer has been overwritten, and which part hasn't. The only assumption we can make is that the buffer will be overwritten contiguously from start to end.

From this, and only this assumption we can derive that there cannot be any over- or undershoot. This can also be understood intuitively by the fact that the pulses will be distributed more or less equally over an entire period, regardless of what the reference input value b is.

C. Derivation of limited under- and overshoot

For input b and period N, the number of pulses y is:

$$y = \sum_{n=0}^{N-1} \left\lceil n \mod \frac{N}{b} < 1 \right\rceil = b \tag{2}$$

For some arbitrary transition number (representing the point where we are in overwriting the old buffer with the new buffer) k between 0 and N and arbitrary inputs b_1 and b_2

$$\begin{aligned} y_t &= \sum_{n=0}^{k-1} \left\lceil n \mod \frac{N}{b_1} < 1 \right\rceil + \sum_{n=k}^{N-1} \left\lceil n \mod \frac{N}{b_2} < 1 \right\rceil \\ &=? \qquad\qquad\qquad +? \end{aligned} \tag{3}$$

We can use y to obtain a set of indices Y at which a pulse occurs, derivation shown in Equation 1:

$$Y = \left\{ \left\lceil \frac{N}{b} n \right\rceil \,\middle|\, n \in \{0, \ldots, b-1\} \right\} \tag{4}$$

Let's define the set of all possible indices as $V = \{0, \ldots, N-1\}$. Since the pulse indices can only be between 0 and $N-1$ (formally $Y \subseteq V$) we know that $Y = Y \cap V$. If we define two partitions of V, split at the same k defined before, $T = \{0, \ldots, k-1\}$ and $W = \{k, \ldots, N-1\}$, we know from the definition of set partitions that $V = T \cup W$. Lastly if we define $A = Y \cap T$ and $B = Y \cap W$, the following will hold:

$$Y = Y \cap V = Y \cap (T \cup W) = (Y \cap T) \cup (Y \cap W) = A \cup B \tag{5}$$

Next we define Y_1 and Y_2 which are the sets of indices corresponding to inputs b_1 and b_2.

$$\begin{aligned} A_1 &= Y_1 \cap \{0, \ldots, k-1\} \\ &= \left\{ \left\lceil \frac{N}{b_1} n \right\rceil \,\middle|\, n \in \left\{ 0, \ldots, \left\lfloor b_1 \frac{k-1}{N} \right\rfloor \right\} \right\} \\ B_2 &= Y_2 \cap \{k, \ldots, N-1\} \\ &= \left\{ \left\lceil \frac{N}{b_2} n \right\rceil \,\middle|\, n \in \left\{ \left\lceil b_2 \frac{k-1}{N} + \epsilon \right\rceil, \ldots, b_2 - 1 \right\} \right\} \end{aligned} \tag{6}$$

It can easily be seen that A has $\lfloor b_1 \frac{k-1}{N} \rfloor + 1$ elements and B has $b_2 - \lceil b_2 \frac{k-1}{N} + \epsilon \rceil$ elements. As a sanity check we can observe from $\lfloor x \rfloor - \lceil x + \epsilon \rceil = -1$ that the number of elements of $Y = A \cup B$ adds up to b again when $b_1 = b_2 = b$. We can also observe that A and B are partitions of Y ($A \cap B = \emptyset$ and $A \cup B = Y$), since B can be rewritten as:

$$B_2 = \left\{ \left\lceil \frac{N}{b_2} n \right\rceil \,\middle|\, n \in \left\{ \left\lfloor b_2 \frac{k-1}{N} \right\rfloor + 1, \ldots, b_2 - 1 \right\} \right\} \tag{7}$$

When instead we don't assume this equality of $b_1 = b_2 = b$, we will obtain:

$$\begin{aligned} y_t &= \left\lfloor b_1 \frac{k-1}{N} \right\rfloor + 1 + b_2 - \left\lfloor b_2 \frac{k-1}{N} \right\rfloor - 1 \\ &= b_2 + \left\lfloor b_1 \frac{k-1}{N} \right\rfloor - \left\lfloor b_2 \frac{k-1}{N} \right\rfloor \\ y_t - 1 &< b_2 + b_1 \frac{k-1}{N} - b_2 \frac{k-1}{N} < y_t + 1 \\ y_t - 1 &< \frac{N-k+1}{N} b_2 + \frac{k-1}{N} b_1 < y_t + 1 \end{aligned} \tag{8}$$

From this we can see that the number of pulses can never go above or below the weighted average of b_1 and b_2 with weights $\frac{k-1}{N}$ and $\frac{N-k+1}{N}$ respectively by more than $1 - \epsilon$.

One extra thing has to be taken into account when we realize that no memory copying implementation does this one bit at a time, since that would be incredibly inefficient. A real implementation will realistically overwrite any multiple of 8 bits at a time. If the buffer contains 16-bit elements a problem may arise when we overwrite 8 bits at a time (but not when overwriting 16 bits or more at a time), since we overwrite from start to end, and the 16-bit elements are ordered most significant bit first.

In the context of this subsection this boils down to 2 transitions between the old buffer and the new buffer (instead of the 1 which has been assumed). To make absolutely sure that there will only be 1 transition, the memory copying implementation will have to copy 16 or 32 bits at a time for example.

D. Emulation in QEMU

To be able to get a feel for how efficient different implementations are, a simulation in an emulator would be nice.

This would of course also be possible by just measuring time on the real microcontroller between breakpoints set in a debugger, but this will not be very accurate because there is no cycle count register [15] (there is only a program counter sampling register as required by the ARMv6-M specifications, see section D3.7 in [26] and section 40.8.2 in reference manual RM0444 [15]). Therefore the timing is required to be measured on the computer which hosts the debugger, which will introduce a lot of jitter and latency.

Another possibility would be to toggle a GPIO pin before and after the execution of the algorithm and measure the on-time with an oscilloscope, this will involve a lot of manual data processing however.

A third option would be to use a hardware timer to keep track of elapsed time between two points and print this out via UART for example, so that the data processing can be automated more easily. The downside of this is that when you still want sufficient time resolution at cycle counts in the order of magnitude 10^2 or 10^3, while also being able to measure cycle counts in the order of magnitude 10^6, the timer interrupt has to fire very often and this may significantly reduce the amount of clock cycles left for the algorithm itself. This will skew the time at least to some degree.

Profiling the different implementations in an emulator will also make it faster to iterate and test modifications. There are a few different cycle-accurate emulators for Cortex® M cores, but they are all very expensive [27], [28]. Some non-cycle-accurate emulators are freely available or even open source [29], [30]. Renode has some coarse profiling features which require you to specify how many instructions per second the microcontroller typically can execute (MIPS) [31]. QEMU allows you to keep track of the number of instructions executed [32], which is also a good indication of how long something would take on the real core. It is also deterministic and reproducible (down to the single instruction) contrary to if one would just measure the time the emulation takes.

Counting the instructions instead of counting clock-cycles is good enough in this case, since most instructions are single-cycle on a Cortex® M0+ core, branches take two cycles (and one if not-taken for a conditional branch), loads and stores are two-cycle if to/from the main bus matrix and single-cycle if to GPIO. All six three-cycle instructions will not be used in a simple algorithm (they are all hardware (memory barrier and special register) related instructions, and a branch with link) [33].

The multiplier is single-cycle in the STM32G0 family according to datasheet DS13560 [15].

This means that the absolute maximum error range will be $\times 2$ to $\times 0.5$ if one algorithm would purely consist of two-cycle instructions and the other purely of single-cycle instructions.

Four different implementations were profiled in QEMU:

1) A naive implementation of a 1st order SDM under the name `SIGMA_DELTA_ALGORITHM` in Listing 5.
2) The non-optimized efficient algorithm from Listing 4 under the name `NON_OPTIMIZED_ALGORITHM`.
3) An optimized algorithm that outputs the same bit pattern, but works in reverse (clearing bits instead of setting them) when the input is more than half of the total range, under the name `OPTIMIZED_ALGORITHM`.
4) An algorithm with the principle from implementation 3 taken further to 8 different base levels under the name `TRADEOFF_ALGORITHM`.

Implementation 4 is a lot faster since there is no longer the need to set or clear individual bits in integer elements of the buffer, because the number of base levels is 8, so it is easier to just alternate between base level patterns for each element in the buffer. The downside is that the bit pattern will not be exactly equal to that of a 1st order SDM.

Essentially how this can be understood is a kind of a hard-coded 3-bit PDM within a 13-bit PDM algorithm, which are both according to what a 1st order SDM of those two resolutions would generate.

The target machine in QEMU was chosen to be the LM3S6965EVB, since this target could be configured in QEMU to have a Cortex® M0 core, which is binary compatible with the Cortex® M0+ core, the only difference between the CPUs in these cores is that some instructions take less clock cycles in the Cortex® M0+ core [34], but this is not relevant when only counting instructions. The Luminary Micro Stellaris LM3S6965EVB has 64KiB of SRAM [35]. This cannot be changed with QEMU command-line options, but it is enough in this case.

The instruction count differences in QEMU were measured by setting breakpoints in GDB before and after the algorithm and reading out the absolute instruction counts at these points. These can be seen in Figure 6 plotted for all possible reference input values.

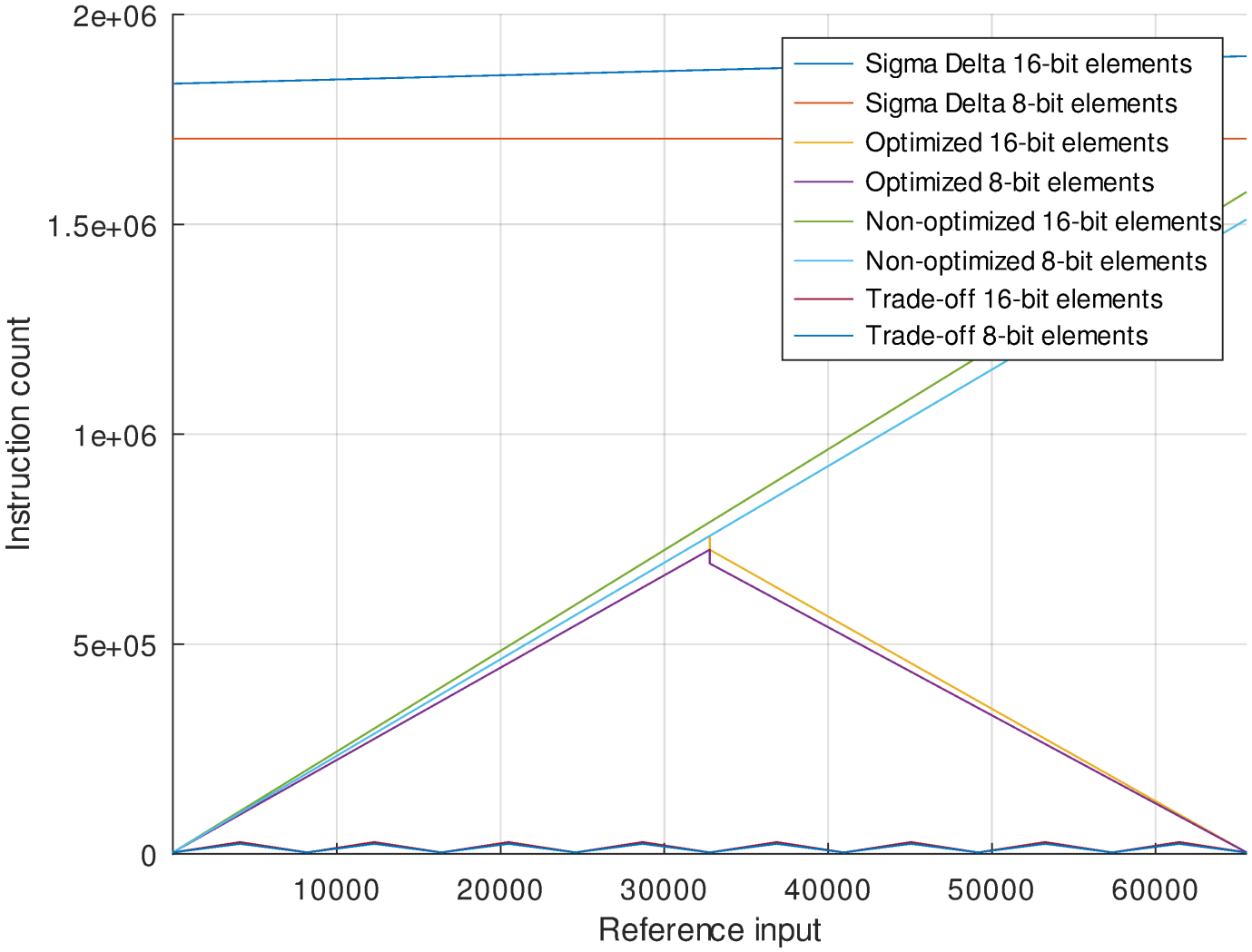


Fig. 6: Instruction count

The scripts used for profiling in QEMU with the help of GDB can be seen in Listing 6, Listing 7 and Listing 8.

VI. Devices survey

Since there is an enormous selection of devices with some of the features of an industrial AFE available it would be silly to compare all of these, so a pre-selection has to be made. There are only a few manufacturers of ADCs (which seems to be a superset of manufacturers of AFEs for industrial applications). These are:

- Texas Instruments/National Semiconductor/Burr Brown
- Renesas/Intersil
- NXP
- Microchip

- Analog Devices/Maxim Integrated/Linear Technology

To minimize the amount of devices needed, a requirement on the amount of multiplexed channels is set to 4 or 8 for the ADC. This still leaves a lot of options to select one from, so a limitation of the price is set to below $25 at 100 units. The device is required to have at least 16-bit resolution, and input voltage span above 10V to eliminate the need for an external voltage divider which needs to be calibrated. Some devices which have one or more DACs are included in the selection too even if they have a resolution of less than 16-bit, or have less than 4 channels, since they are not very common, and it might be interesting to compare these devices still if later during a redesign cycle it is decided that the price may be higher or that it is okay if more than 2 devices are required to bring the total to 8 channels. First a comparison will be presented of devices with built-in DACs.

For the sake of increasing robustness, the EMC ratings will have a high priority in the selection of a suitable device.

A. MAX22000

The MAX22000 is a relatively new device created by Analog Devices [36]. It is designed to be used (with a few external components) as an industrial input/output with 3 regular analogue inputs and 1 analogue input which can also be configured as an output. A summary of the most relevant features can be seen listed below:

- $28.70 at 1k units.
- 18-bit DAC and 24-bit ADC.
- 3 regular analogue inputs can be used as voltage inputs only, without significant external hardware (current shunts and analogue switches).
- 1 analogue input/output can be configured for current/voltage input/output.
- 1 extra input for thermocouple sensing is available.
- The gain on 2 out of 3 regular inputs can be programmed.
- No automatic channel sequencing.
- Automatic application of one input calibration offset/gain pair and output offset/gain pair.
- No factory calibration.
- 8-bit CRC on SPI bus.
- Up to $\pm$36V over-voltage protection on analogue input/output pins.
- 1kV human body model (HBM) protection on all pins.
- IEC 61000-4-5, 1.2/50μs 1kV pulse protection with 4.75kΩ series MELF resistor on all analogue input/output pins.

[36]

A design using 2 of these devices can be made which would result in an input extender with 8 inputs, 2 of which can also be configured as outputs. There are some drawbacks however. This device is quite expensive, which would mean at least a contribution of $57.40 by this device to the total bill of materials price for an SCC Compact extender. Another drawback is that the only regular analogue input that is not connected to a programmable gain amplifier (PGA) will not utilize the full range of the ADC when connected to a current shunt and used as a current input, since at the maximum current of 20mA and maximum current shunt resistance of 300Ω, the input voltage will be 6V, while the input voltage range is $\pm$12.5V.

The HBM (standardized in Method 3015.9 Electrostatic discharge sensitivity classification in [37]) defines an ESD pulse generated by a charged 100pF capacitor through a 1500Ω resistor, which yields a peak current of 0.67A when the capacitor is charged to 1kV.

The protection on all analogue input/output pins is more resilient than on the other pins, since additionally it is able to withstand a surge standardized by IEC 61000-4-5. Even though such a surge has a longer rise time ($\approx$ 720ns) due to the added inductor [7] (which will give the protection semiconductors more time to turn on) and the 4.75kΩ series resistance is also higher than the 1500Ω for the HBM, the total surge duration will also be orders of magnitude longer ($= 50\mu$s), than in case of an ESD pulse. This means that the protection semiconductors are tested more for their energy absorbing capabilities than for fast response by this specification.

No mention is made of compliance with the IEC 61000-4-2 standard, which defines an ESD pulse generated by a charged 150pF capacitor through a 330Ω resistor [6]. This results in a much faster rise time 0.8ns vs $<$ 10ns (and a higher peak current) [37].

B. NAFE33352

The NAFE33352 is a similar device developed by NXP [38]. A summary of the most relevant features can be seen listed below:

- $6.85 at 100 units.
- 18-bit DAC and 24-bit ADC.
- 2 regular analogue inputs can be used as voltage inputs only, without significant external hardware (current shunts and analogue switches).

- 1 analogue input/output can be configured for current/voltage input/output.
- A PGA is connected on the 2 regular analogue input channels.
- An automatic channel sequencer is built-in to allow for per-channel configuration.
- Automatic application of corresponding offset/gain calibration pairs.
- Offset/gain calibration pairs loaded with factory calibration on reset.
- 8-bit CRC on SPI bus.
- Up to $\pm$36V over-voltage protection on analogue input/output pins.
- Analogue input pins are connected via protection diodes to the high voltage supplies to limit power dissipation compared to zener diode/TVS protection to ground.
- 7.5kV HBM protection on all pins.
- IEC61004-2 ESD protected on analogue input pins with the condition that they have a $\geq$ 3kΩ resistor connected in series.
- IEC61004-5, 1.2/50μs 2kV protected on analogue input pins with the condition that they have a $\geq$ 3kΩ resistor connected in series.

[38]
A design using 3 of these devices can be made which would result in an input extender with 9 inputs, 3 of which can also be configured as outputs.

This device focuses a lot on robustness, evident from the fact that it specifies that it has been tested with the procedures defined in [6], [7], [37]. It is also relatively cheap, but a downside is that 3 of these devices would be required to fulfil the amount of analogue inputs needed. External analogue switches and current shunts are also still required. The PGA also only has two different gain settings: 1 and 16, so either a very low resistance current shunt has to be used (putting heavy requirements on the analogue switch on resistance) or a part of the ADC dynamic range is wasted.

Next, a comparison is made between some interesting devices which only incorporate an ADC. They all have 8 input channels.

C. AD7606

A summary of the most relevant features can be seen listed below:

- \$20.16 at 1k units.
- 16-bit ADC.
- Only bipolar input ranges $\pm$10V and $\pm$5V (for each pin configurable), so effectively resolution is reduced to 15 bit.
- Only one 5V supply is needed (no high voltage bipolar supplies).
- Protection of input using zener diodes $\pm 16.5V$ (so only one external resistor is needed to implement protection against a wrongly connected 24V line).
- Anti-aliasing filters built-in (15kHz or 23kHz cut-off depending on range).
- Single-ended inputs only.
- Contains an optional digital first-order SINC filter.
- 2kV ESD protected (all pins except analogue inputs).
- 7kV ESD protected on the analogue input pins.

[39]
This device doesn't have any CRC implementation for the SPI bus, it also doesn't state according to what standard the 7kV and 2kV ESD ratings were obtained [39], so presumably with the HBM from [37]. The fact that it has simultaneous sampling ADCs (as opposed to one ADC with some kind of multiplexer in front of it) also makes it relatively expensive.

D. ADS9815

A summary of the most relevant features can be seen listed below:

- \$23.80 at 100 units.
- Dual 18-bit ADC.
- Simultaneous sampling: 4 channels per ADC.
- Only one 5V supply is needed (no high voltage bipolar supplies).
- Programmable anti-aliasing filters built-in ($>$ 100kHz or $>$ 21kHz cut-off).
- Protection of input using zener diodes $\pm$18V (so only one external resistor is needed).
- No digital filter built-in.
- One PGA per channel.
- 2kV HBM protection on all pins.

[40]
This device doesn't have any CRC implementation for the SPI bus. It also doesn't have any additional ESD or surge protection on the analogue input pins.

E. MAX1300

A summary of the most relevant features can be seen listed below:

- $10.21 at 1k units.
- 16-bit ADC.
- Also has unipolar ranges (configurable for each pin).
- Only one 5V supply is needed (no high voltage bipolar supplies).
- Protection of input using zener diodes ± 16.5V.
- No analogue or digital filters built in.
- No ESD or surge rating.

[41]

This device has a very low input impedance (typically a purely resistive part of 17kΩ), so adding external resistors to allow for 24V tolerant inputs will significantly effect the gain error [41]. Adding external 150Ω series resistors would limit the current through the clamping zener diodes to the rated 50mA at 24V, but it would also create a voltage divider with a ratio of 1.0088:1. It also doesn't have CRC hardware for the SPI communication.

F. NAFE13388

A summary of the most relevant features can be seen listed below:

- $20.07 at 100 units.
- 24-bit ADC.
- Eight single-ended or four differential inputs.
- PGA with range 0.2 to 16 (only placed after a multiplexer, but a sequencer is built-in to automatically change gain to the configured value for each channel).
- Automatic application of corresponding offset/gain calibration pairs.
- Offset/gain calibration pairs loaded with factory calibration on reset.
- 8-bit CRC on SPI bus.
- Up to ± 36V over-voltage protection on analogue input/output pins.
- Input clamp is connected directly on input pins with diodes to differential high voltage supplies.
- High voltage supplies are clamped w.r.t. ground to +29V and -29V (probably with zener diodes).
- 7.5kV HBM protection on all pins.
- IEC61004-5, 1.2/50μs 2kV protected on analogue input pins with the condition that they have a ≥ 2.5kΩ resistor connected in series.

[42]

The NAFE13388 also has quite a unique feature in that it has an excitation source built in (a low resolution, but high precision DAC), that can be multiplexed to any of the analogue input pins, which can be used to measure resistances. This may be useful if a Wheatstone bridge or thermistor is connected as a sensor for example.

G. Isolation

Since the existing designs for both the analogue input extender and the analogue output extender don't have any electrical isolation, it is good to consider where one might place the isolation barrier in a redesign. All devices in the AFE selection presented in the above subsections need at least one low voltage supply (3.3V or 5V for example), so a power supply is needed to convert the input supply of 24V nominal to this lower voltage. It is therefore not convenient to place a digital isolator between the microcontroller and the AFE, since the microcontroller will also need a low voltage supply (so you would need two power supplies when placing it here). If one would place the isolator between the microcontroller and the CAN transceiver, you would only need one power supply. Another benefit of placing the isolation barrier between the microcontroller and CAN bus is that only 2 digital lines need to be isolated instead of at least 3 when isolating a SPI bus. We also know the CAN bus to have a speed of 1Mbit/s already, while the SPI bus might operate at higher speeds depending on the chosen sample rate.

There are isolated CAN transceivers (physical layer), which have integrated isolated DC-DC converters so that no power supply is needed on the bus-side [43].

Even if it's too expensive to use an isolated CAN transceiver (with integrated power supply), it is still possible to have a digital isolator in combination with a CAN transceiver, which will both need less power than either the AFE or microcontroller, so it can have a smaller power supply. Which will result in the design requiring one higher power power supply and one low power (smaller) power supply instead of two higher power power supplies if the isolation lies between the microcontroller and the AFE.

The most optimal solution is then to have the isolation barrier between CAN bus and microcontroller.

An example of an isolated CAN transceiver without integrated power supply is the ISO1042, which is designed for use with 24V buses [44]. A similar device already used in other designs by Brunelco is the ISO1050, which would also be usable in this case even though it is designed for use with 12V buses, meaning that it allows a maximum voltage range on the bus of -27V to 40V [45].

VII. Methodology

A. Device selection

A prototype is made with which measurements and tests can be performed. For this a final selection is made for the microcontroller and industrial AFE.

1) Microcontroller: A microcontroller in the STM32G0 family has to be chosen. The choice is not very difficult since the STM32G0B1 and STM32G0C1 microcontrollers are the only two which incorporate a CAN interface [15]. All variants have 144KiB of SRAM which will turn out to be useful later. The only difference between the STM32G0B1 and STM32G0C1 microcontroller variants is that the former doesn't have AES hardware encryption while the latter does [15]. This is not relevant for the SCC Compact (extender).

A NUCLEO-G0B1RE development board [46] is used in the prototype to speed up development.

2) Industrial AFE: The NAFE13388 is chosen partly because of built-in channel sequencer, high voltage clamped inputs, digital filters and factory calibrated offsets and gains. Another important reason for choosing this device is its robustness with regard to ESD and surge protection.

For the prototype, the NAFE13388-UIM development board was chosen [47], since it can easily be plugged in to the NUCLEO-G0B1RE with the Arduino® Uno V3 connectivity. This development board also has isolated analogue switches and 250Ω current shunts for 4 out of 8 channels on board [47].

Since the NAFE13388 doesn't enable a design which includes some additional analogue outputs, an extender with 8 analogue outputs is all the more relevant. Such an extender can use (one of) the methods described in section V to generate 8 analogue signals with the microcontroller. These analogue signals still have to be converted to the required type by additional analogue circuitry.

3) Analogue output driver: An example of devices which are specifically designed to do this are the XTR300 [48] and its (lower cost and precision) pin compatible sibling; the XTR305 [49]. These devices are drivers which on their high-impedance input accept a low voltage analogue signal, and convert this to an analogue output which complies with the standard described in [2]. The XTR305 is also used in other designs by Brunelco already.

The thing that makes such a device special compared to just an operational amplifier with some external components is that these drivers allow selecting if the output is a current source/sink or voltage source/sink without changing the output circuitry.

They are also able to detect output over-range, over-temperature and fault conditions in the load and pass this on to the microcontroller [48], [49]. All pins are also able to withstand an HBM ESD pulse of 2kV.

The XTR300EVM was chosen as a development board for the prototype [50], since it has a voltage reference on board, which allows for setting the input offset [48], [49].

The NAFE13388-UIM has a built-in charge pump on board to generate the negative supply it needs, which can also be used to supply the XTR300EVM with power.

4) CAN transceiver: It is outside the scope of this project to implement and test a CAN transceiver according to the recommendations made in section VI-G and isolation is not a requirement for the prototype, so a prototyping board is made with the TJA1042 [14], which is also used in the existing extender designs. This prototyping board can be seen in Fig. 8. A low forward voltage diode is placed in series with the 5V supply to bring it down to approximately 4.7V, fulfilling the requirement that a logic high level needs to be at least $0.7\times$ the supply as mentioned in section III-A. Other than this, the application diagram from Figure 7 in section 12 in [14] is followed.

B. Prototype setup

A block diagram of how the development boards were connected can be seen in Fig. 7.

1) Development board modifications: On the NAFE13388-UIM development board, R236, R237, R239 and R241 are 3.3kΩ [47], which yields a current through the LEDs in the VO1400AEFTR isolated analogue switches of around $I = \frac{V_{CC}-V_F}{R} = \frac{3.3-1.3}{3.3\cdot10^3} \approx 600\mu$A [51] when they are driven by the NAFE13388 GPIO, which is the only option on the NAFE13388-UIM. The LED forward current at which the switch typically turns on is 800μA [51]. These resistor values may therefore result in the switch having a very high on-resistance, or not turning on at all. The former was observed while testing. For the final test setup, R236, R237, R239 and R241 were replaced with 100Ω resistors, resulting in a theoretical LED current of 18mA. A typical R_{on} of 2.3Ω is specified for a forward current of 10mA [51].

On the XTR300 development board, a resistance of 2.7kΩ was chosen for R_{SET} (in parallel with the 2.49K resistor already on the evaluation board) and 15kΩ chosen for R_{OS}. R_G was kept as is. This results in the following driver output ranges with a $0\text{V} - 3.3\text{V}$ input range:

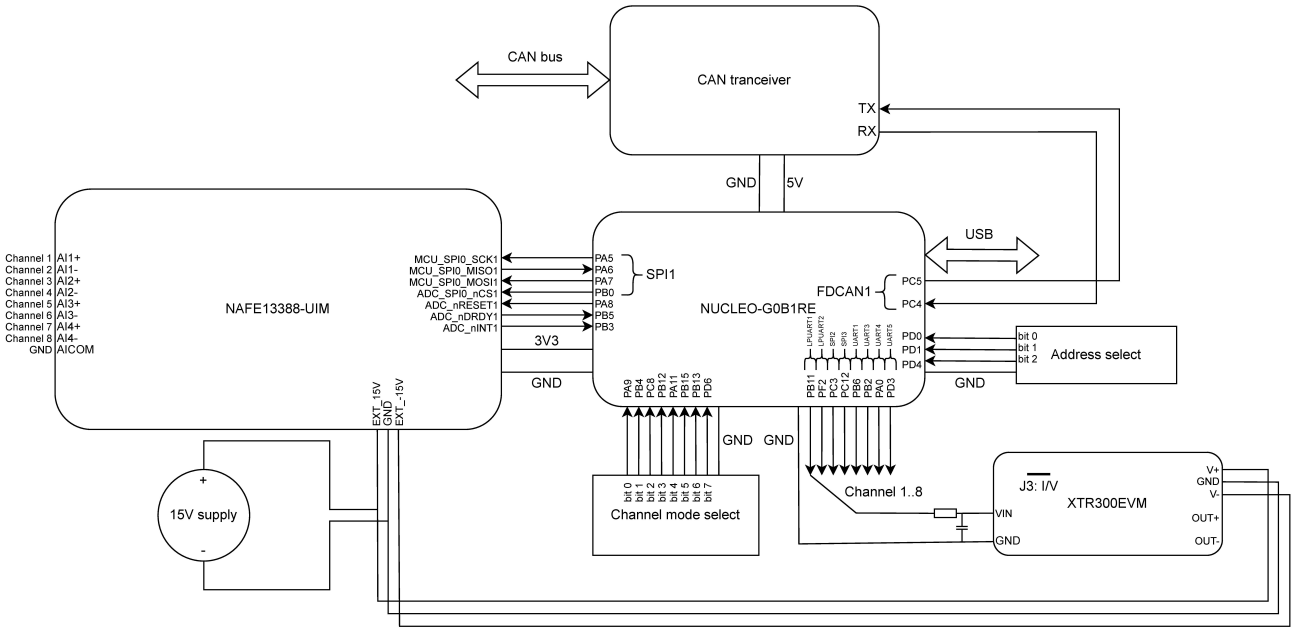
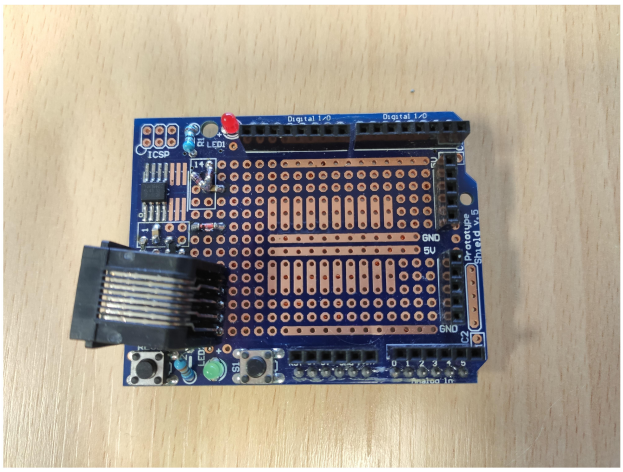
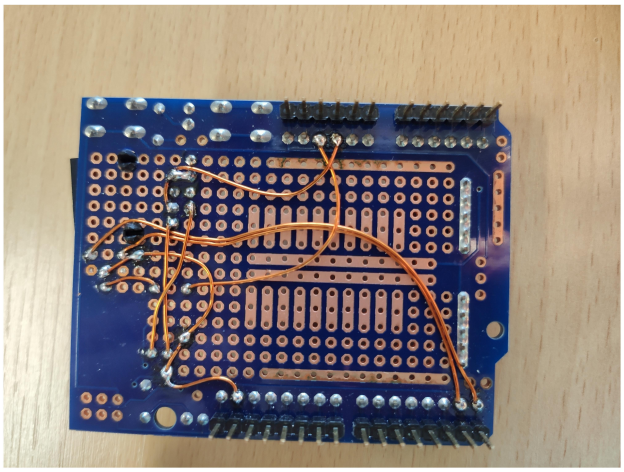


Fig. 7: Prototype block diagram



(a) CAN transceiver top



(b) CAN transceiver bottom

Fig. 8: Photos of CAN transceiver prototyping board.

$$
\begin{aligned}
V_{OUT} &= \frac{R_G}{2}\left(\frac{V_{IN}}{R_{SET}} + \frac{V_{IN} - V_{REF}}{R_{OS}}\right) \\
V_{OUT1} &= \frac{R_G}{2}\left(\frac{V_{IN1}}{R_{SET}} + \frac{V_{IN1} - V_{REF}}{R_{OS}}\right) \\
&= \frac{10 \cdot 10^3}{2}\left(\frac{0}{1.295 \cdot 10^3} + \frac{0-5}{15 \cdot 10^3}\right) \\
&= -1\frac{2}{3}\text{V} \\
V_{OUT2} &= \frac{R_G}{2}\left(\frac{V_{IN2}}{R_{SET}} + \frac{V_{IN2} - V_{REF}}{R_{OS}}\right) \\
&= \frac{10 \cdot 10^3}{2}\left(\frac{3.3}{1.295 \cdot 10^3} + \frac{3.3-5}{15 \cdot 10^3}\right) \\
&\approx 12.17\text{V}
\end{aligned}
\tag{9}
$$

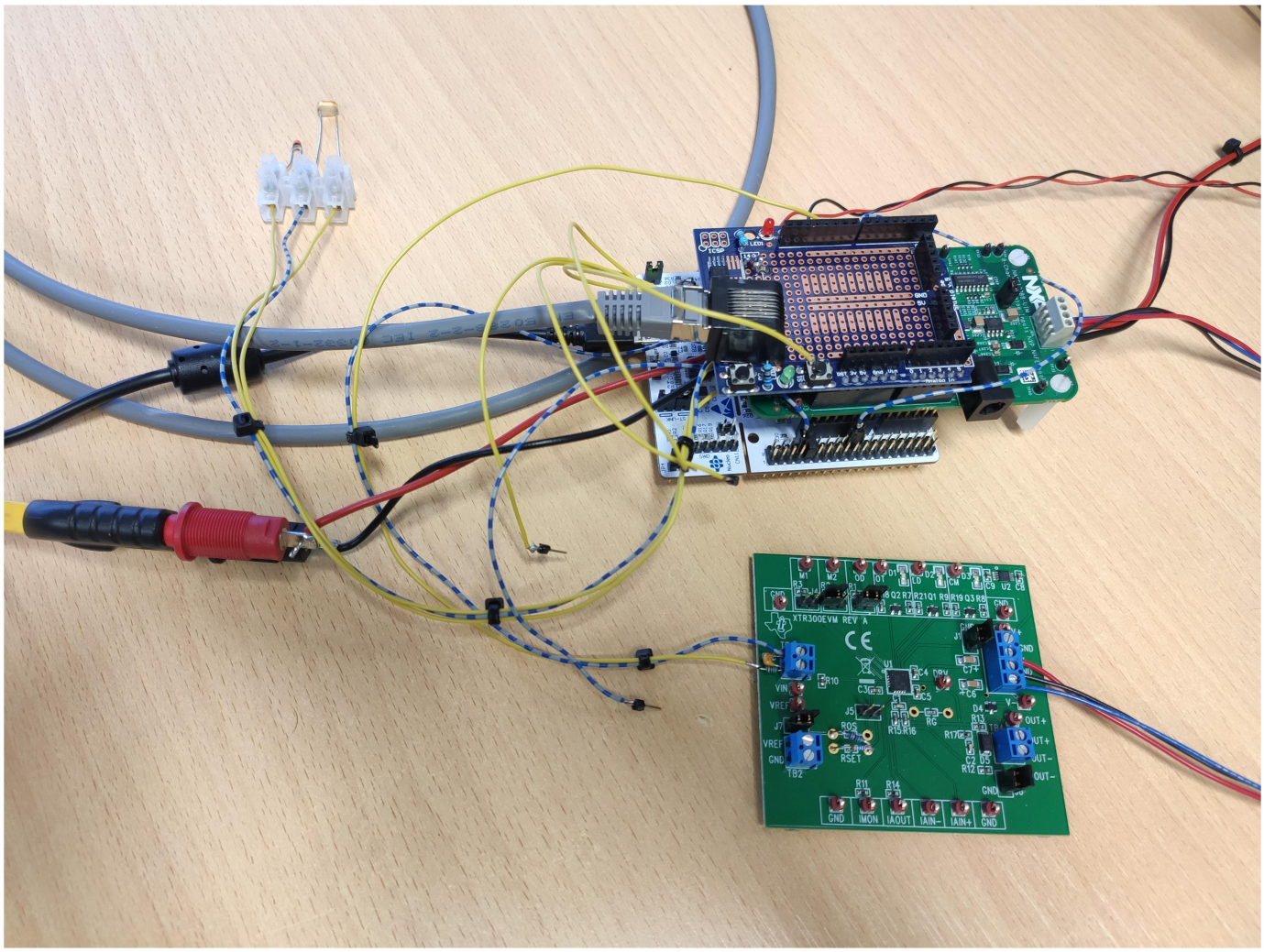


Fig. 9: Photo of prototype setup

[48]

This means that the output voltage range will be approximately $-1.67\text{V} - 12.17\text{V}$. The current range is the same scaled by a factor $\frac{20}{R_G}$, which is $-3.33\text{mA} - 24.35\text{mA}$.

C. Software implementation

1) CAN interface: An implementation of the SCC protocol is ported to the STM32G0B1 to allow the prototype to communicate with an SCC Compact. Both analogue output extender and analogue input extender functionality is implemented. A more elaborate description of how the prototype and existing extenders are connected to an SCC Compact can be found in section Appendix-B.

2) Analog inputs: The NAFE13388 is connected to the microcontroller via SPI. On the microcontroller it is configured for a speed of 1Mbit/s. The hardware CRC peripheral on the STM32G0B1 is used to validate the received data and to calculate the CRC accompanying the transmitted data. The NAFE13388 channel sequencer is configured in multichannel, continuous reading mode (MCCR). There are 16 built-in logical channels. 8 of these are configured with $0.2\times$ gain, and the other with $0.4\times$ gain. These gains are chosen because the 250Ω current shunts will generate a voltage range of $0 - 5\text{V}$ with a $0 - 20\text{mA}$ current range. The voltage range corresponding to $0.2\times$ gain is $\pm 12.5\text{V}$ and the range corresponding to $0.4\times$ gain is then $\pm 6.25\text{V}$ of course [42]. Setting up the logical channels like this allows easily changing the type for physical channels dynamically by just enabling or disabling logical channels for the sequencer. The analogue switches can be turned on and off via the same SPI bus since they are controlled via GPIO integrated into the NAFE13388. Because the shunts are controlled by the STM32G0B1, it is possible to disconnect them when an overload situation is detected. On the NAFE13388 it is possible to set over- and under-range thresholds. When these thresholds are exceeded, a bit in a status byte prepended to each conversion transmission

gets set. Other bits in this same status byte indicate whether the PGA or ADC is overloaded. It can therefore be quite reliably be detected if the current shunts are overloaded.

A flow diagram of the software interacting with the NAFE13388 can be seen in Fig. 10.

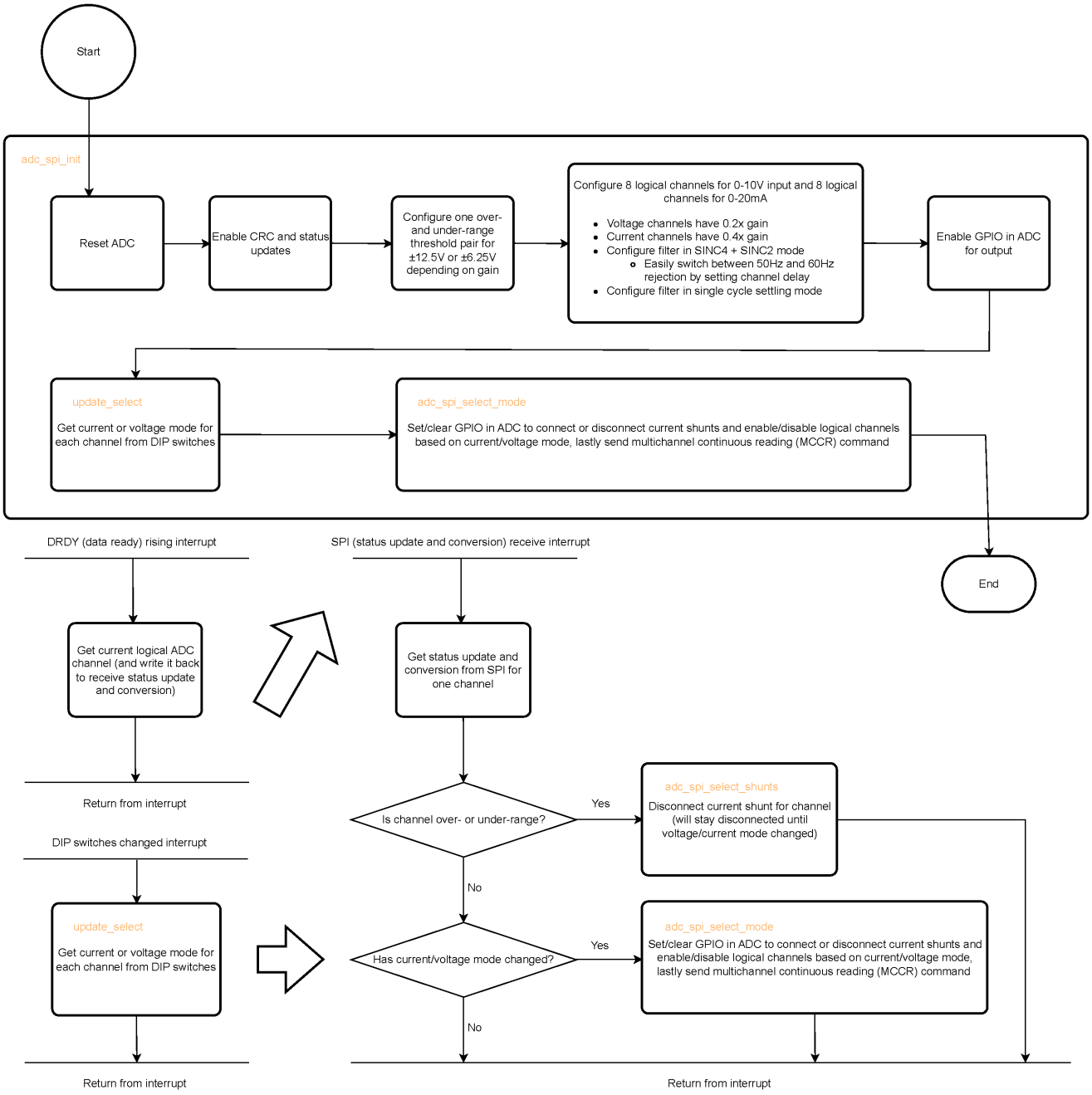


Fig. 10: Flow chart for software interacting with the ADC

The NAFE13388 has factory calibration gain and offset coefficients for all possible gain settings. The correct pairs are applied automatically by the channel sequencer to the logical channels configured for $0.2\times$ and $0.4\times$ gain. Table 12 in [42] shows which calibration registers belong to which gain settings, pointers 0 and 1 (`CH_CAL_GAIN_OFFSET`) are the relevant ones here.

Normally when measuring external signals `TCC_OFF` should be 0 for a logical channel configuration, when using the excitation source it should be set to 1. This is because it turns off temperature compensation for the built-in voltage reference. This temperature dependence is cancelled out when resistance is measured because the excitation source uses the same voltage reference [42].

For 50/60Hz rejection, the built-in SINC filters can be configured to have a frequency response that can be seen in Fig. 11.

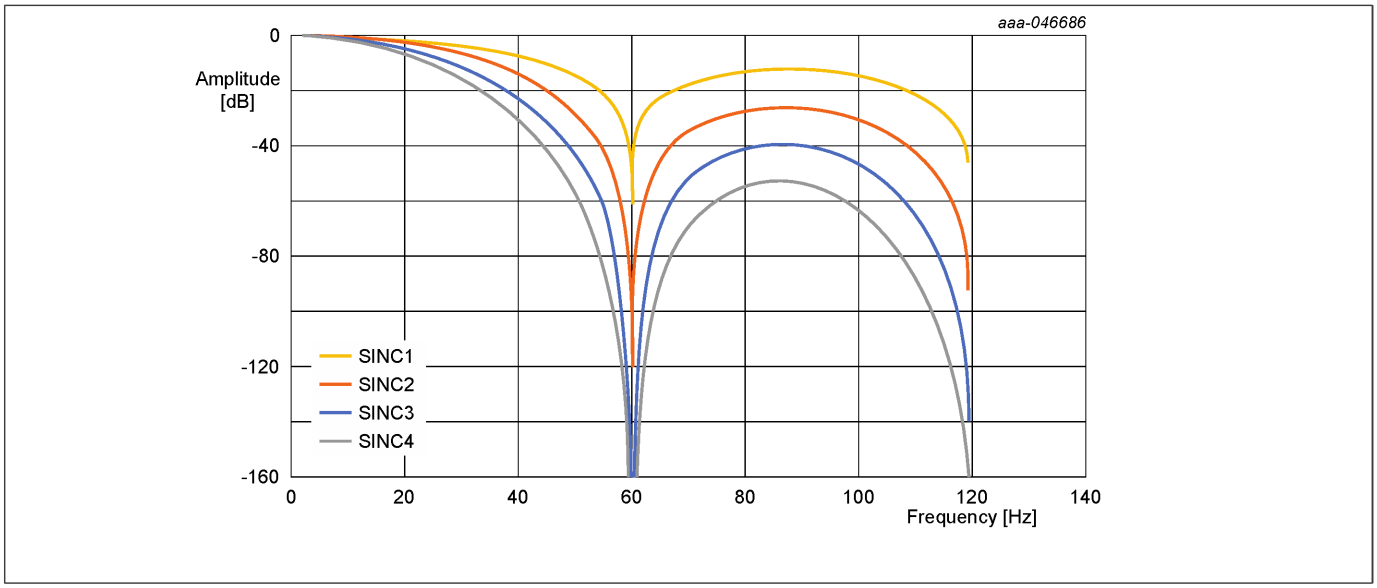


Fig. 11: SINC filter transfer function
Source: [42]

The SINC filter is a moving average filter which has a number of passes equal to the number n in SINCn. Such a filter is mostly useful for smoothing a time domain encoded signal, but not so much to filter specific frequency bands like a low pass filter [52]. There is however one frequency domain property of a SINC filter which is very useful in this case: it has transmission zeros (infinite attenuation) at points $\frac{nf_s}{M}, 1 \leq n \leq \frac{M}{2}$, where M is the number of samples averaged in the SINC filter [52]. The transfer function of this filter can be seen in Equation 10.

$$|H[\Omega]| = \left| \frac{\sin\left(\frac{1}{2}\Omega M\right)}{M \sin\left(\frac{1}{2}\Omega\right)} \right| \tag{10}$$

[52]

It is not stated clearly in [42], but from Figure 7, Figure 8 and Table 7 in [42] one can deduce that each pass/order averages 16 samples. The signal is subsequently down-sampled by a factor 16, so that the first transmission zero is at the sample frequency, as can be seen in Figure 7 in [42]. This means that aliasing will occur in the pass band of the SINC filter, so the only use for the digital filters is to reject noise of a very specific frequency.

For a multichannel measurement, single-cycle settling mode has to be used to avoid settling error, which essentially down-samples by a factor equal to the filter order plus one to allow the SINC filter to settle between channel switching. The transmission zeros stay at the same frequencies then of course, so one has to look at the normal settling data rate column in Table 7 in [42] to determine the frequency which is rejected for a specific DRO code. So for example DRO code 21 for 60Hz rejection always (independent of SINC filter order) and DRO code 22 for 50Hz rejection [42]. This yields maximum single-cycle data rates of 30Sps for 60Hz rejection and 25Sps for 50Hz rejection.

The effective resolution for the desirable ranges is 22-bit, because the ADC itself needs to be able to handle a 50V full scale range in differential bipolar mode at $0.2\times$ gain, but if you use a channel in unipolar single-ended mode for the same gain, it will only be a 12.5V range [42]. Luckily the ENOB is always above 18-bit (subtract 2 to compensate for only using $\frac{1}{4}$th of the total range) for data rates $\leq$ 48kSps [42], which is nowhere near as high as we need.

3) Analog outputs: As mentioned before in section V, the internal DACs of the STM32G0B1 can't be used: it only has two converters each having a resolution of 12 bits, which is not enough.

A version of the method described in section V has to be implemented taking the hardware constraints of the STM32G0B1 into account. If 8 outputs are needed, at least 64KiB of memory has to be available, since the buffer for each channel will be 2^{16} bits long.

The most elegant solution would be to use DMA to write out the buffer to a single GPIO port, in which each bit represents one GPIO pin, although this would mean that updating one channel while leaving the others unmodified would be a lot less efficient. However, the GPIO peripheral is not connected to bus matrix on the STM32G0B1. It is connected directly to the

M0+ core to enable single cycle GPIO access [15]. This makes it impossible to write out with DMA. An architecture block diagram can be seen in Fig. 12.

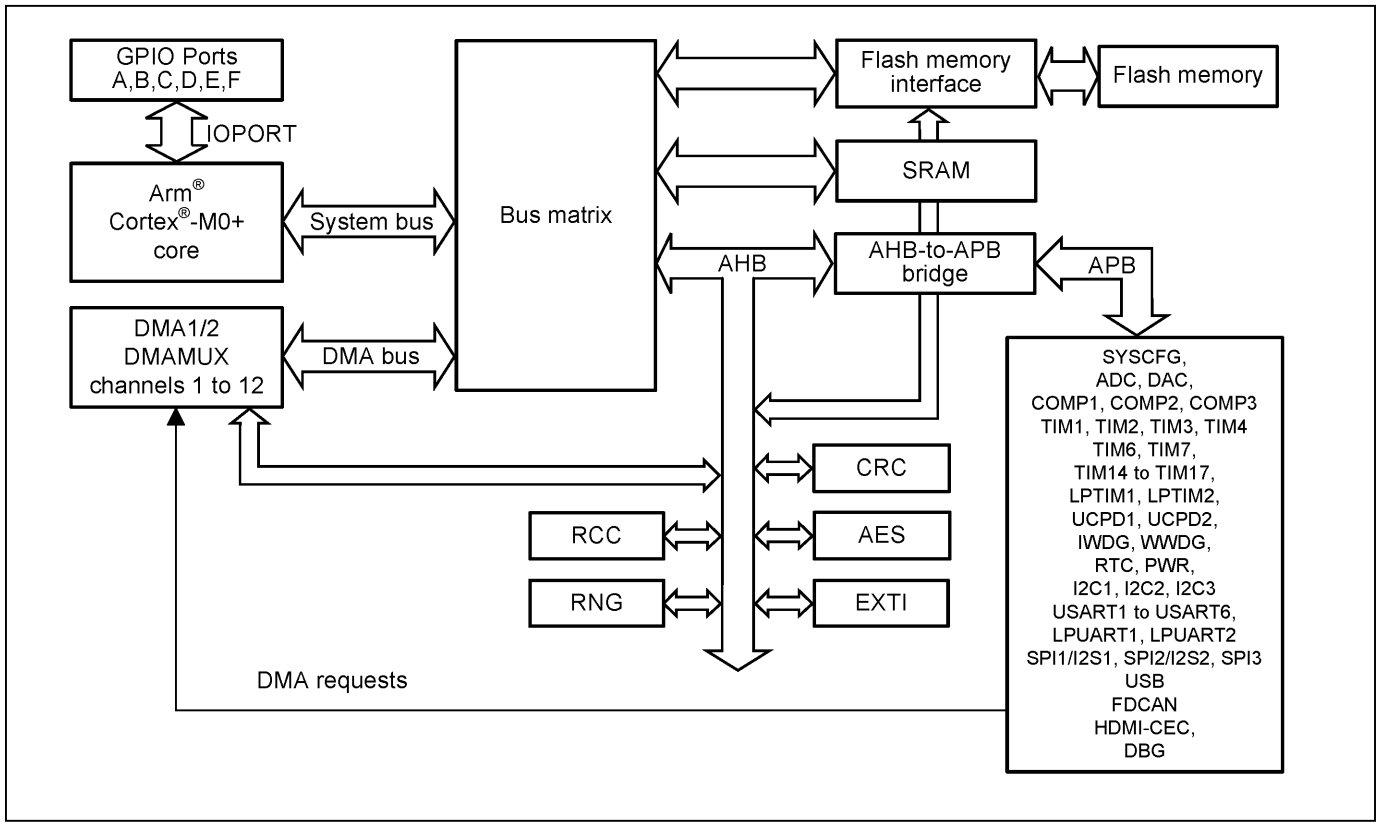


Fig. 12: STM32G0B1 architecture
Source: [15]

Instead it is possible to use timer channels in forced output mode, however this is not very memory efficient since the CCxP (where x is the channel number) bits to force the output high or low are in registers together with other bits (1 CCxP bit per 4 bits), so it is not possible to have 1:1 memory to peripheral output [15].

It is also possible to use the MOSI line of an SPI peripheral or the TX line of a UART peripheral. There are 3 SPI peripherals and 8 UART peripherals in the STM32G0B1 [15], so it is technically possible to have 8 outputs while still having UART2 left for debugging.

SPI is perfect for this application, but UART has start and stop bits, which essentially means the full range is between $\frac{1}{19}$ and $\frac{17}{19}$ when using 0.5 stop bits, since the start bit always goes low and the stop bit always goes high. The full range is between $\frac{1}{10}$ and $\frac{9}{10}$ when using 1 stop bit. The extra margin both in the positive and negative direction chosen for the XTR300 driver, is enough for both the limited range of the LPUART peripherals, which only allow 1 stop bit (and not 0.5 stop bits) and the UART peripherals which allow 0.5 stop bits [15]. The theoretical ranges are $-0.28\text{V} - 10.79\text{V}$ for LPUART and $-0.94\text{V} - 10.71\text{V}$ for UART.

An estimation of the maximum used DMA bandwidth is useful to have. Both SPI peripherals are configured with a 32Mbit/s bitrate, both LPUART peripherals with a bitrate of 21.33Mbit/s and 4 UART peripherals with a bitrate of 8Mbit/s. These all are the maximum speeds at which the peripherals can operate with a 64MHz clock frequency. The data size for SPI can be configured to a maximum of 16-bit, which will reduce the amount of transfers per second required. These configurations will bring the total required bandwidth to $13.33 \cdot 10^6$ transfers per second.

The DMA controller uses a round-robin arbitration mechanism to allow for equal bus access times granted to each DMA channel, if the priorities are the same [53]. The bus access time is 3 AHB clock cycles (so not including request arbitration, address computation and acknowledgment) in case of a AHB to AHB transfer. The SPI and UART peripherals are connected to the APB bus, which means that one extra AHB cycle gets added to the bus access time for bridge synchronization if the APB frequency is lower than the AHB frequency [53]. This is however not the case when maximum SPI and UART speeds are desirable; then the APB clock frequency has to be 64MHz too.

We know now that each transfer will keep the bus occupied for 3 clock cycles, which means DMA can do a theoretical maximum of $21.33 \cdot 10^6$ transfers per second. Application note AN2548 recommends leaving one-third of the total bus capacity

in reserve [53]. The configurations listed above comply with this.

The DMA controller in the STM32G0B1 can do circular transfers. This allows the output generation without any intervention from the CPU. It would also be possible to use double buffering and swap out the buffer when a buffer with an updated output value has been constructed, but this would have a higher chance of going wrong than just setting up the DMA buffer after the microcontroller is reset and never swapping it, but just overwriting in place.

It was proven in section V that any combination of the old buffer contents and the new buffer contents (with only one transition) will not be more than one code off compared to the weighted average of the old and the new reference. So we will even have a nice linear transition from the old to the new level as the old buffer "slowly" gets overwritten with the new buffer.

D. Analogue input measurements and tests

A measurement will be done to determine factory calibrated accuracy for both voltage and current input for both the prototype and the existing design. The frequency response of the digital filter in the prototype with the NAFE13388 will be measured and compared to the theoretical response from Fig. 10. The SNR and distortion will be determined for different input signals on both the existing design and the prototype. Measurements will be done to determine excitation source performance when measured with the ADC but also with a calibrated multimeter. The switching between current/voltage mode and current shunt overload protection on each physical channel will have to be tested.

E. Analogue output measurements and tests

The SNR is measured on the existing design and on the prototype with different 1st order analogue low-pass filters and different implementations on the microcontroller. Two SPI peripherals, two LPUART peripherals and four UART peripherals are used for outputting the desired digital signal. Additionally, distortion is measured on the prototype. Emulation of optimized algorithm and trade-off algorithm in QEMU will be verified by toggling a GPIO pin before and after algorithm execution on the physical microcontroller. This is done for five strategically chosen reference values and 8 bit elements only. It is very important here to verify that peripherals don't stall intermittently due to DMA controller bus arbitration. This would be the case when the bus bandwidth is too low. This can be verified very easily by measuring if the period of the circular transfer is never higher than expected on all SPI, LPUART and UART peripherals.

VIII. Results

A. Analog inputs

In Fig. 13, the frequency response of the digital filter built-in to the NAFE13388 can be seen with DRO code 21 and SINC1 configuration, to obtain 60Hz normal mode rejection (NMR) [42]. This was obtained by sweeping a Wave Factory WF1974 function generator from 0.01Hz to 100Hz logarithmically with constant amplitude 2.5V and offset 5V in 9999s. A MATLAB script to plot the frequency response using the ADC output data can be seen in Listing 11. This MATLAB script uses the RMS voltage only for the frequency range which can be expected in the filter output. This is always the same as the input frequency range for an LTI system, which the FIR filter is. This eliminates some noise and distortion introduced by the ADC, especially if the frequency range in the sweep slice over which the FFT is calculated is very small. Even though some non-linear effects introduced by the ADC can be eliminated in Fig. 13a, it still seems to have a less smooth frequency response than the one you can visually observe from the envelope in Fig. 13b. The 60Hz rejection should be -60dB according to Fig. 11, but the measured attenuation was only -35dB. This is because the sweep slice for 60Hz contains a range of frequencies, and not only the one specific frequency. From the envelope an amplitude of approximately 6mV can be seen, which is an attenuation of $20\log 0.006 - 20\log 2.5 \approx -52.4$dB.

The exact same measurement was repeated for DRO code 21, resulting in 50Hz (and 100Hz) rejection. This can be seen in Fig. 14.

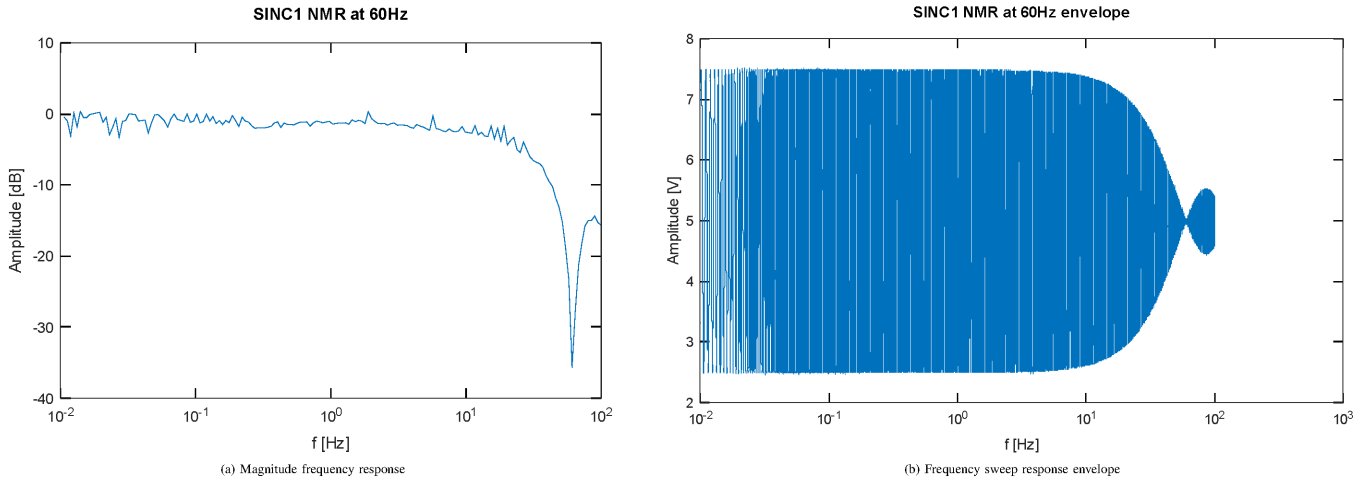
The SINC1 filter was the only filter chosen to be characterized because in single-cycle settling mode, this filter has the highest data rate. Although the data rate is still only 25Sps for 50Hz and 30Sps for 60Hz rejection, meaning an update rate for all 8 channels of 3.125Hz and 3.75Hz respectively. This is also why the frequency sweep was chosen to be so long.

All additional measurements with the prototype were done with a data rate of 900Sps (with the SINC4 filter enabled, now having 4.5kHz rejection, so as to influence the frequency range of interest (mostly up to 50Hz) as little as possible). This results in an update rate for all channels of 112.5Hz.

The noise and distortion of the NAFE13388 and SCC Compact were measured when presented with a sinewave generated by the same function generator. The low pass behaviour of the SCC Compact extender can be seen clearly in Fig. 15b and Fig. 15c.

Noise was also measured with different DC input voltages generated by the same function generator. These can be seen in TABLE III.

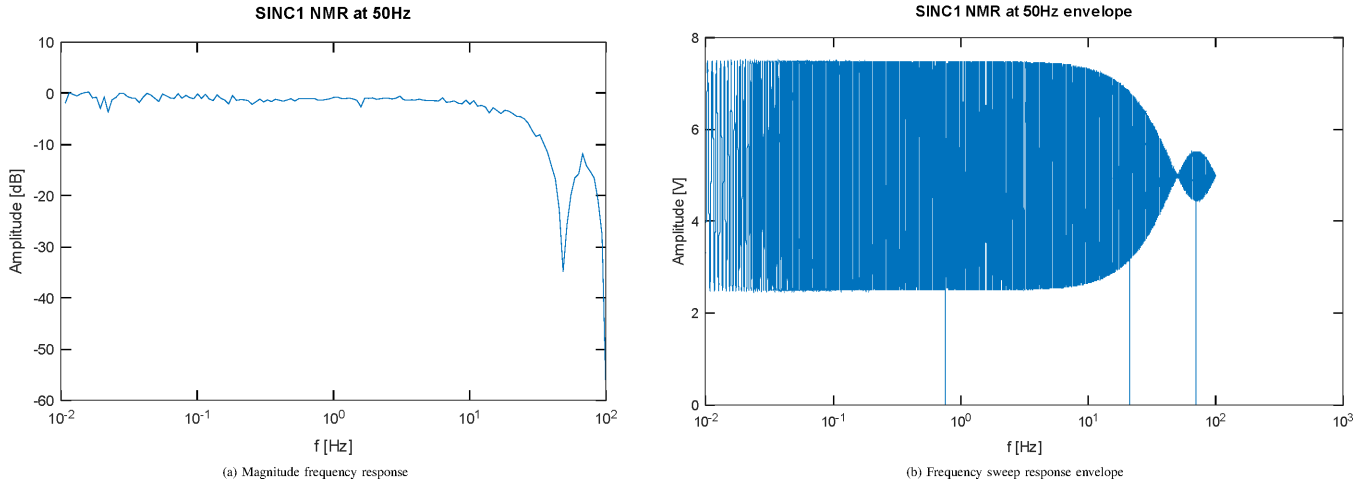
The fact that the measured noise was exactly 0mV_{RMS} for the prototype when supplied with 0V input from the function generator is because it is slightly below 0V, so it is clamped digitally. Other than this, it can be seen that the noise is about



(a) Magnitude frequency response

(b) Frequency sweep response envelope

Fig. 13: Built-in digital SINC1 filter 60Hz NMR.



(a) Magnitude frequency response

(b) Frequency sweep response envelope

Fig. 14: Built-in digital SINC1 filter 50Hz NMR.

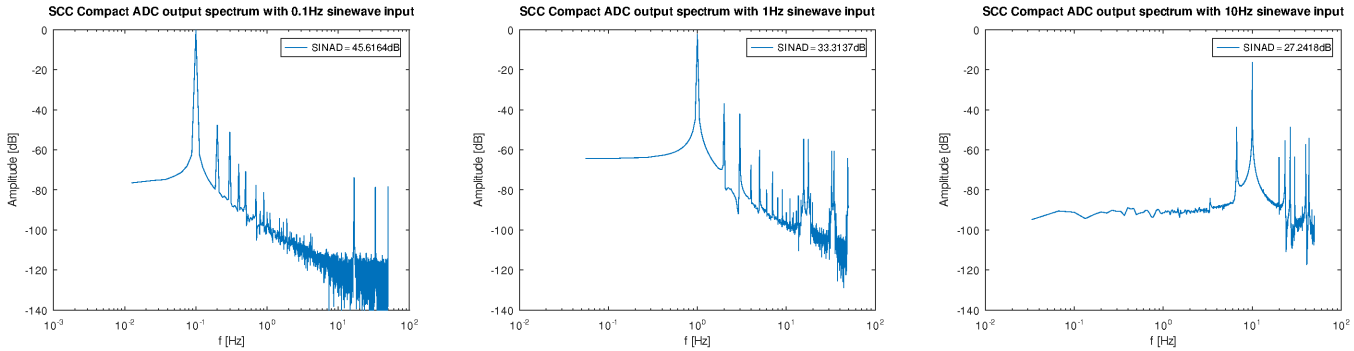
equal for both the SCC Compact and the prototype with NAFE13388. Not much can be said about the ADCs itself though, because...

A measurement to determine the precision of the SCC Compact and the prototype was done with a calibrated Fluke 175 multimeter and 5V low noise power supply. This time two channels of the NAFE13388 were tested.

For the voltage measurement, the multimeter displayed 5.039V, the prototype read out code 26397 out of 65535 (where 65535 is 12.5V), so 5.035V, meaning -0.08% error, which is within specification for the NAFE13388 (maximum $\pm 1526\mu$V offset and $\pm 0.15\%$ gain error after factory calibration), see Table 41 in [42]. This measurement was performed on channel 1 of the prototype. For the current measurement, the tolerance and on-resistance of the analogue switch will play a large role. The multimeter displayed a current of 19.80mA, the prototype read out code 52321 out of 65535 (where 65535 is 25mA), so 19.98mA, meaning 0.9% error. This is not within specification, but when we factor in the typical on-resistance of the analogue switches of 2.3Ω and rated resistance for the current shunts of 250Ω, this would come out to almost exactly this measured error: $\frac{2.3}{250} = 0.92\%$. Another measurement on channel 2 was done to be more sure of this hypothesis. The voltage measured with the prototype is exactly the same, and the ADC code in current mode was read out to be 52319.

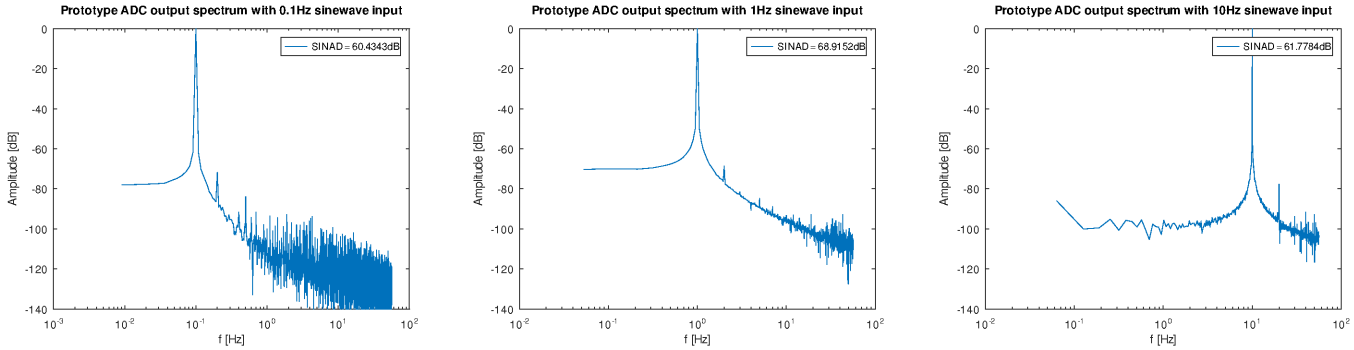
For the measurements with the SCC Compact extender, it is only useful to do current measurements, since the SCC Compact can only be calibrated for current and it uses these same calibration pairs in voltage mode, which means that the inverse of errors caused by current shunt tolerances get applied in voltage mode. The multimeter displayed 10.15mA, while the SCC Compact extender read out 10.17mA, which is an error of 0.20%.

The turn-off time for the analogue switches in case of an over- or under-range situation in current mode was not measured with an oscilloscope, since it can never be more than $\frac{1}{112.5}$s, but with indicator LEDs connected to each analogue switch input, no noticeable delay between applying 15V or -15V to the input and the respective LED for that channel turning off could be



(a) 0.1Hz frequency spectrum (b) 1Hz frequency spectrum (c) 10Hz frequency spectrum

Fig. 15: SCC Compact ADC output spectrum for 3 different sinewave input frequencies.



(a) 0.1Hz frequency spectrum (b) 1Hz frequency spectrum (c) 10Hz frequency spectrum

Fig. 16: Prototype ADC output spectrum for 3 different sinewave input frequencies.

seen. Toggling the current/voltage mode selection pin twice for that channel, while the channel was still over- or under-range resulted in the LED turning on very briefly.

1) Excitation sources: The excitation source and its multiplexer are tested to see if they can be used in a future redesign when resistive (passive) sensors may be supported. Only the current mode is tested because this is the most interesting. Because the excitation sources and the ADC can be connected to a physical channel simultaneously, this is the most interesting to test because it reduces the amount of pins taken up by such a resistive measurement. To maintain conformity to the EMC standards specified in section VI-F, the 2.5kΩ resistors have to be kept in series with the pins, meaning a resistive offset of this 2.5kΩ when using the excitation source in current mode, and a high impedance output when using it in voltage mode.

To determine the precision of a resistive measurement with the NAFE13388, a measurement of a potentiometer was first done with the same Fluke 175 also used before: R_{ref}, and then with the excitation sources configured for two different currents I_{ex}: 1mA and 62.5μA. The former can be seen in TABLE IV, the latter can be seen in TABLE V. The third column is the calculated resistance from the known excitation current and measured voltage. The fourth column is the resistance with the 2.5kΩ offset removed. The last column is the calculated error.

There is a large error of around 2% to 3%, while the specified precision of the excitation source is a lot better, see for example Table 52 in [42]. The error with 62.5μA excitation also goes down to negative values as the resistance increases. This is because 1nF capacitors are placed on all channels, and multiplexing the channels at the chosen sample rate of 900Hz would indeed cause a charge voltage about 86% the final value.

With the Thevenin equivalent we can see that the series resistance would be equal to $R_{Th} \approx 90.3\text{k}\Omega$ and the supply voltage equal to $V_{Th} \approx 5.64\text{V}$. With the capacitance, we get a time constant of $\tau \approx 90.3 \cdot 10^{-6}$, and a charge time to reach 86% of 178μs. The time it takes for the ADC to start a conversion after switching channels with the excitation source enabled isn't stated in [42], but when the sampling time for each channel is approximately 1.1ms, 178μs sounds quite reasonable.

To trace back the cause for the large 2% to 3% error, a measurement with the excitation source always multiplexed to a single channel was done (to eliminate dynamic behaviour from capacitances and make measurements with a multimeter easy). The measured currents were 780μA instead of 750μA according to the multimeter, 1.03mA instead of specified 1mA, 1.55mA instead of 1.5mA and 2.07mA instead of 2mA. The multimeter current measurement mode was verified to still be within calibration by measuring the voltage (at least in respect to the voltage mode) of the 5V low noise supply with a 0.1% 10kΩ

Input DC voltage	Measured noise SCC Compact	Measured noise Prototype
short	0.090mVrms	0.035mVrms
0V	0mVrms	0.041mVrms
2V	0.152mVrms	0.081mVrms
5V	0.212mVrms	0.192mVrms
10V	0.234mVrms	0.384mVrms

TABLE III: Noise measurements with DC input from function generator.

R_{ref}	code	$R = \frac{12.5 \cdot \text{code}}{65536 \cdot I_{ex}}$	$R - R_{short}$	$\frac{(R-R_{short})-R_{ref}}{R_{ref}}$
short	12961	2.472k	ND	ND
92.8	13460	2.567k	95	2.4%
284.6	14492	2.764k	292	2.6%
374.2	14957	2.853k	381	1.8%
813	17321	3.304k	832	2.3%
2.920k	28546	5.445k	2.973k	1.8%
5.067k	39920	7.614k	5.142k	1.5%

TABLE IV: Resistive measurements with 1mA excitation.

resistor across it, and subsequently measuring the current through it with the same multimeter.

B. Analogue outputs

1) Optimized algorithm: First a measurement was done to verify that the serial peripherals weren't stalling occasionally due to memory bandwidth limitations. The period is measured with an Analog Discovery 2 oscilloscope for all maximum data rates by setting half of buffer with all ones and other half with all zeros, so that a square wave (aside from start and stop bits) with period equal to the circular DMA period can be easily observed. This measurement can be seen in TABLE VI.

Emulation of optimized algorithm in QEMU is verified by toggling a GPIO pin before and after executing the algorithm. This is done for five strategically chosen reference values and 8 bit elements only. This measurement can be seen in TABLE VII. The times correspond quite well (well within the $2\times$ maximum error), apart from a 1ms offset, which is discussed later.

Measurements were done to determine the integral non-linearity (INL) by sweeping the analogue outputs from code 0 to 65535 (a sawtooth shape). The RC filter was chosen with $R = 10\text{k}\Omega$ and $C = 10\mu\text{F}$ to keep to the values in the SCC Compact for now. This measurement can be seen in Fig. 17. An additional measurement with PWM to compare can be seen in Fig. 21b.

Aside from the non-linear shape which is ridiculously large at the highest speed for SPI, there is also a squarewave fluctuation visible on each plot. This is because there was an LED connected on the prototype blinking with a frequency of 1Hz. Going forward, the speeds chosen for SPI, LPUART and UART are 4Mbit/s, 8Mbit/s and 8Mbit/s respectively. This also drastically reduces the load put on the memory bus. It is now $6.5 \cdot 10^6$ transfers per second, which is less than one third of the available bandwidth.

Measurements of noise with the input of the Analog Discovery 2 set to AC coupling can be seen for different output codes in Fig. 18. Both the SCC Compact and the prototype are measured. AC coupling was used to be able to use a very low voltage range, reducing the amount of noise introduced by the measuring equipment drastically.

The noise is however still only less than 1mV_{RMS} for all output codes on the prototype. This is why another measurement was done with UART1 at 4kbit/s, which should cause 60dB more noise (compared to SPI at 4Mbit/s). We can see in Fig. 19 that the noise is now approximately what it is on the SCC Compact. We could therefore increase the cut-off frequency of the RC-filter by a factor 500 before going above the desired 5mV_{RMS} noise/below the 10-bit ENOB.

2) Trade-off algorithm: The same measurements done in the previous section VIII-B1 are repeated with the trade-off algorithm mentioned in section V. Emulation of optimized algorithm in QEMU is again verified by toggling a GPIO pin before and after executing the algorithm. This is done for five strategically chosen reference values and 8 bit elements only. This measurement can be seen in TABLE VIII.

The same 1ms offset can be seen here. It is caused by the memset function which fills the buffer with the correct value beforehand. When compiling for the emulation, a version of newlib is used which was configured for optimization focusing on speed instead of size, while the toolchain used for the STM32G0 has newlib configured for size. In [54], the C source code can be seen which is optimized for speed. It writes 16 bytes per loop iteration if the alignment allows for it (which it does in this case). The size "optimized" version writes out only 1 byte per loop iteration. The disassembly from the actual build for the STM32G0 with optimization -O2 enabled can be seen in Listing 15.

After execution of the algorithm, the temporary buffer has to be copied over the DMA buffer. This is done by the memcpy function, which is also optimized for size instead of speed (Listing 16), which means that it copies 8-bits at a time, which it shouldn't do to fulfil the condition mentioned in section V-C.

Measurements were done to determine the noise with the trade-off algorithm. The RC filter was chosen with $R = 10\text{k}\Omega$ and $C = 100\text{nF}$ this time, so that a cut-off frequency of 159Hz is obtained. Measurements with SPI2 can be seen in TABLE IX.

R_{ref}	code	$R = \frac{12.5 \cdot \text{code}}{65536 \cdot I_{ex}}$	$R - R_{short}$	$\frac{(R-R_{short})-R_{ref}}{R_{ref}}$
short	820	2.502k	ND	ND
2.660k	1720	5.249k	2.747k	3.3%
5.70k	2740	8.362k	5.860k	2.8%
11.63k	4691	14.32k	11.82k	1.6%
20.02k	7385	22.537k	20.04k	0.1%
50.32k	16345	49.88k	47.38k	-5.8%
87.8k	25630	78.22k	75.72k	-14%

TABLE V: Resistive measurements with 62.5μA excitation.

Peripheral	Measured period	Calculated bitrate
SPI2	2.045ms	32.047Mbit/s
SPI3	2.045ms	32.047Mbit/s
LPUART1	3.835ms	21.361Mbit/s
LPUART2	3.835ms	21.361Mbit/s
UART1	9.726ms	8.001Mbit/s
UART3	9.721ms	8.005Mbit/s
UART4	9.724ms	8.003Mbit/s
UART5	9.712ms	8.013Mbit/s

TABLE VI: DMA period measurements.

Another measurement was done to determine the INL can be seen in Fig. 21a, which is expected to be about the same as before (Fig. 17c).

IX. Conclusions

With the NAFE13388 it is possible to redesign the SCC Compact analogue input extenders with robustness in mind. It allows the omission of expensive external protection components because it has robust input protection built-in. This input protection is made to survive standardized input surges and spikes which the final controller will need to conform to. With the programmable gain amplifier it requires minimal external hardware to make an input which can be configured (by a microcontroller) to measure either current or voltage: only a current shunt and analogue switch need to be added. The analogue switch can be turned on or off in software as well, making a fully configurable industrial analogue input. The performance in terms of noise and distortion is better with a prototype containing the NAFE13388 compared to the existing design. Factory calibration of the device also allows for its use in the SCC Compact without additional calibration, on the condition that current shunts are chosen with 0.1% tolerance and analogue switches with suitably low on-resistance. To protect the current shunts, the user-programmable over- and under-range detection can be used to let the microcontroller know that it should turn off the analogue switches. The built-in digital filters are not of much use when used in multichannel reading mode and simultaneously a relatively high data output rate and 50Hz/60Hz rejection are desired.

Using pulse density modulation (PDM) it is possible to obtain orders of magnitude lower settle time than with PWM, while also having lower noise, when used as analogue outputs for an industrial controller. The desired maximum noise RMS voltage is currently exceeded by the existing SCC Compact analogue output extender design. It is possible to obtain noise amplitudes at least 10 times lower than in the existing design, while also having a settle time 100 times lower than what it is currently. A first order analogue filter is kept from the existing design. Drawbacks of PDM compared to PWM on the chosen microcontroller are increased INL and increased CPU usage. Two simple but efficient algorithms are tested on the microcontroller, which shows that even without hardware related optimization, it is possible to obtain reasonable computation times.

A. Discussion

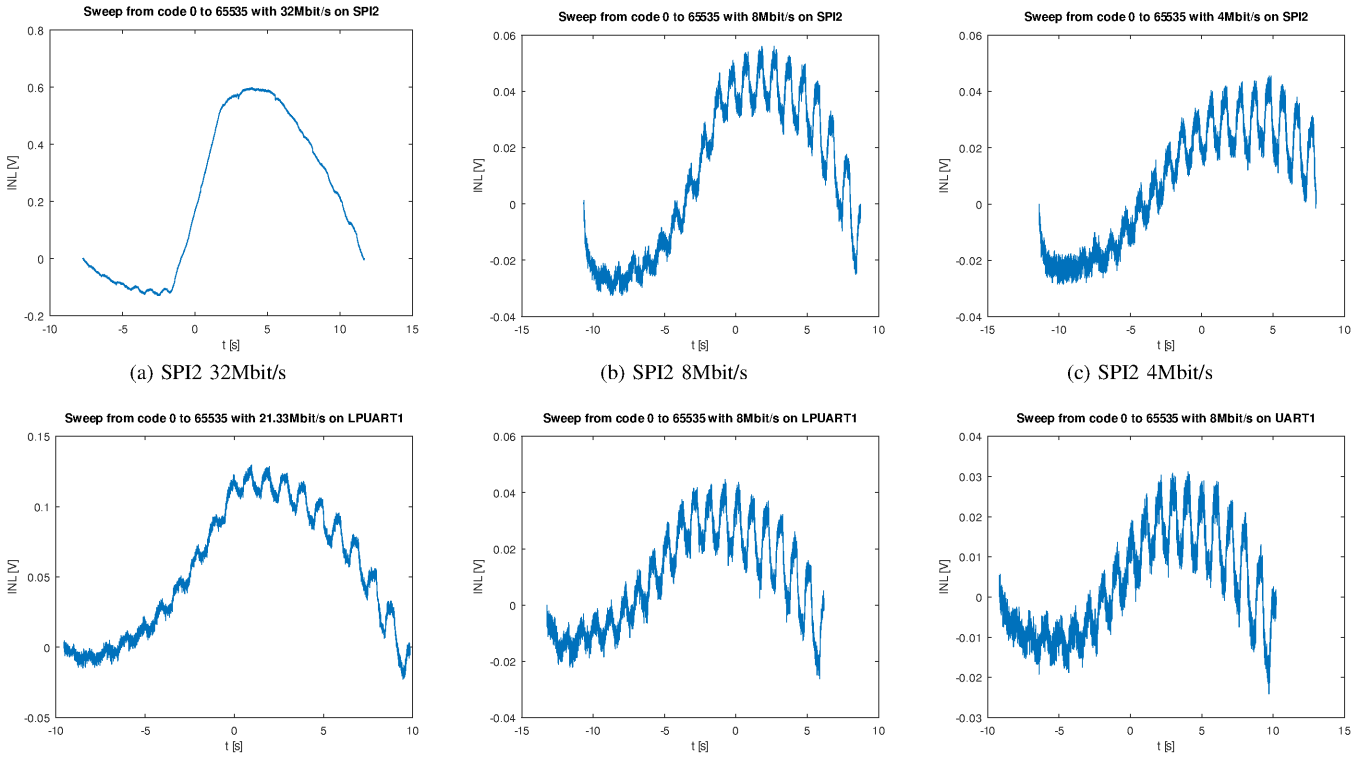
Measurements with the NAFE13388 analogue inputs revealed that the excitation source had a larger error than expected. No explanation was found for why this is the case. It may be that something was configured wrong in one of the hardware registers, such as the applied factory calibration pairs, but the error should not come anywhere near 2% to 3%, even without factory calibration. The following things were made sure however:

1) The measurement equipment produced accurate measurements (see section VIII-A1).
2) Temperature coefficient compensation was turned off because the excitation source and ADC use the same voltage reference.
3) The excitation source had an insignificant AC component when multiplexed to a single pin.
4) The power supply was ± 15V, which leaves enough headroom as long as the voltage out of the excitation source doesn't exceed ± 12V, which it can't and didn't [42].

The excitation source is also less useful in a 2 wire measurement setup, because of the required 2.5kΩ series resistors. If these resistors are chosen with 0.1% tolerance, the resulting measurements may be accurate enough. When a 3 or 4 wire

Reference input	QEMU instruction count	Real STM32 time
1	3638 ($\approx 57\mu$s)	1.09ms
16384	364495 (≈ 5.7ms)	9.06ms
32767	724912 (≈ 11ms)	17.11ms
32768	692163 (≈ 11ms)	16.07ms
65535	4193 ($\approx 66\mu$s)	1.07ms

TABLE VII: Verification of emulation in QEMU.



(a) SPI2 32Mbit/s (b) SPI2 8Mbit/s (c) SPI2 4Mbit/s

(d) LPUART1 21.33Mbit/s (e) LPUART1 8Mbit/s (f) UART1 8Mbit/s

Fig. 17: INL obtained with sawtooth of full range.

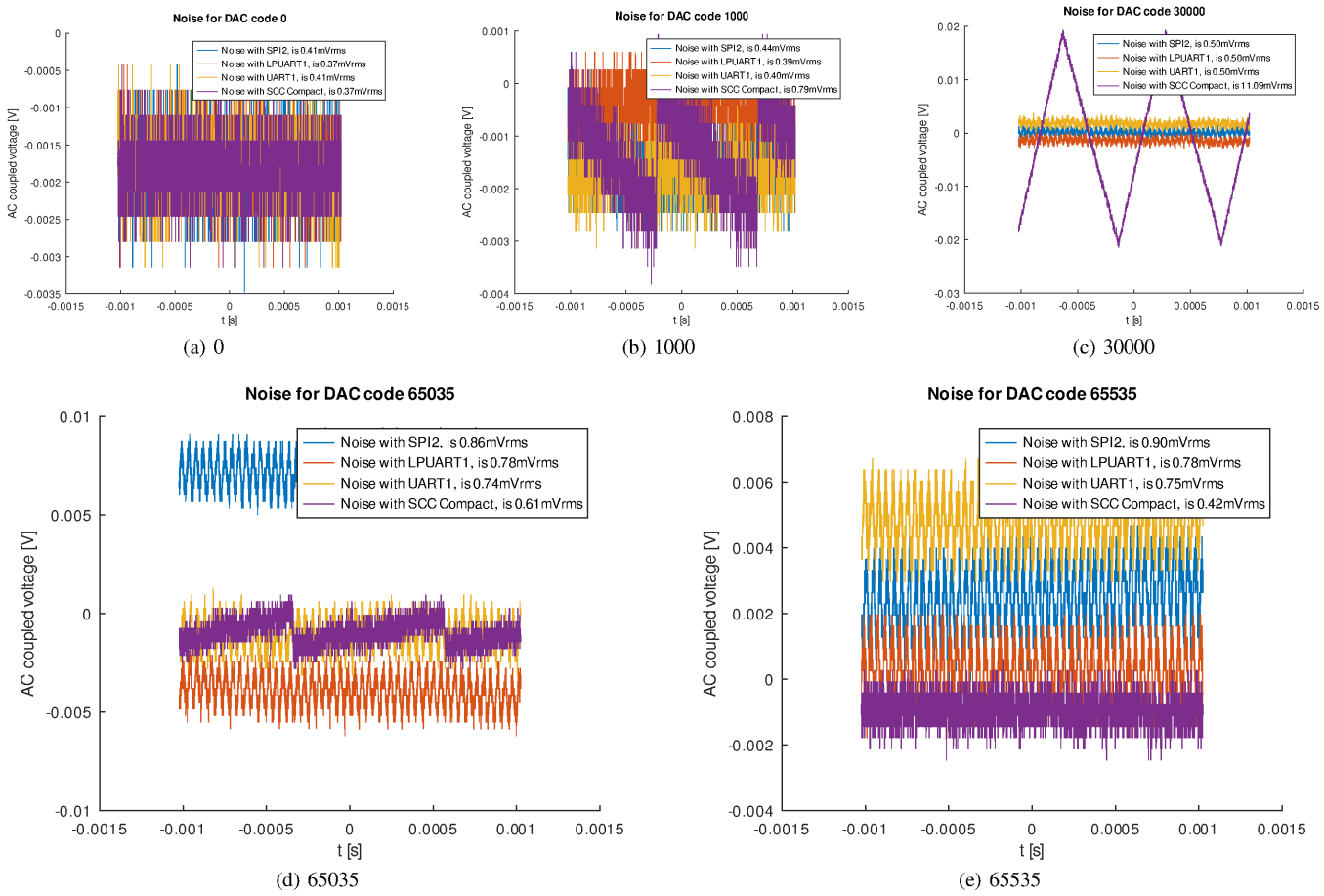
measurement setup is made, the resistance of the series resistors doesn't matter in current excitation mode. The hardware for these different setups is the same, only the software configuration of the NAFE13388 needs to be different in each case.

Besides the aforementioned points, the device performed as specified by its datasheet.

Regarding the measurements done to characterize the analogue outputs, no analysis for the frequency spectrum of the trade-off algorithm is done, like for the other algorithms. Even though this is the only algorithm which produces a different bit pattern. This could have been done for completeness, but since the byte pattern in case of only 2 base levels ('11111111' and '00000000') would be the same as when the other algorithms only have 13-bit resolution, it can be assumed that it is at least lower noise than a 13-bit 1st order SDM. From the measurements in section VIII-B2 it can be seen that the amount of noise is low enough for application in an industrial controller.

During the INL measurements it could be seen that the blinking of an LED on the same supply rail as the microcontroller caused the analogue output voltage to fluctuate. This can be explained by the fact that the voltage regulator didn't provide a super stable 3.3V. This effect was a lot less noticeable on the SCC Compact extenders, even though it also directly drives a low-pass filter from a GPIO pin, so it probably has a more stable supply. This observation does bring a major downside of driving the RC filter directly with a GPIO pin: it transfers all the power supply low frequency noise and drift over time and temperature directly to the analogue output, so calibration cannot solve everything in this case.

The fact that the INL is so much higher for PDM than for PWM is probably because the PMOS devices in the GPIO push-pull driver have higher capacitance than the NMOS devices, which causes the PMOS to conduct more current when outputting a zero, than the NMOS conducts when outputting a one, when switching at very high frequencies. This causes the average output level to be slightly higher than the time-average of all the ones and zeroes.



(a) 0 (b) 1000 (c) 30000

(d) 65035 (e) 65535

Fig. 18: Noise for different DAC codes.

Reference input	QEMU instruction count	Real STM32 time
1	3652 ($\approx 57\mu$s)	1.05ms
2048	14014 (≈ 0.22ms)	1.35ms
4095	24251 (≈ 0.38ms)	1.57ms
4096	24257 (≈ 0.38ms)	1.55ms
65535	3836 ($\approx 60\mu$s)	1.02ms

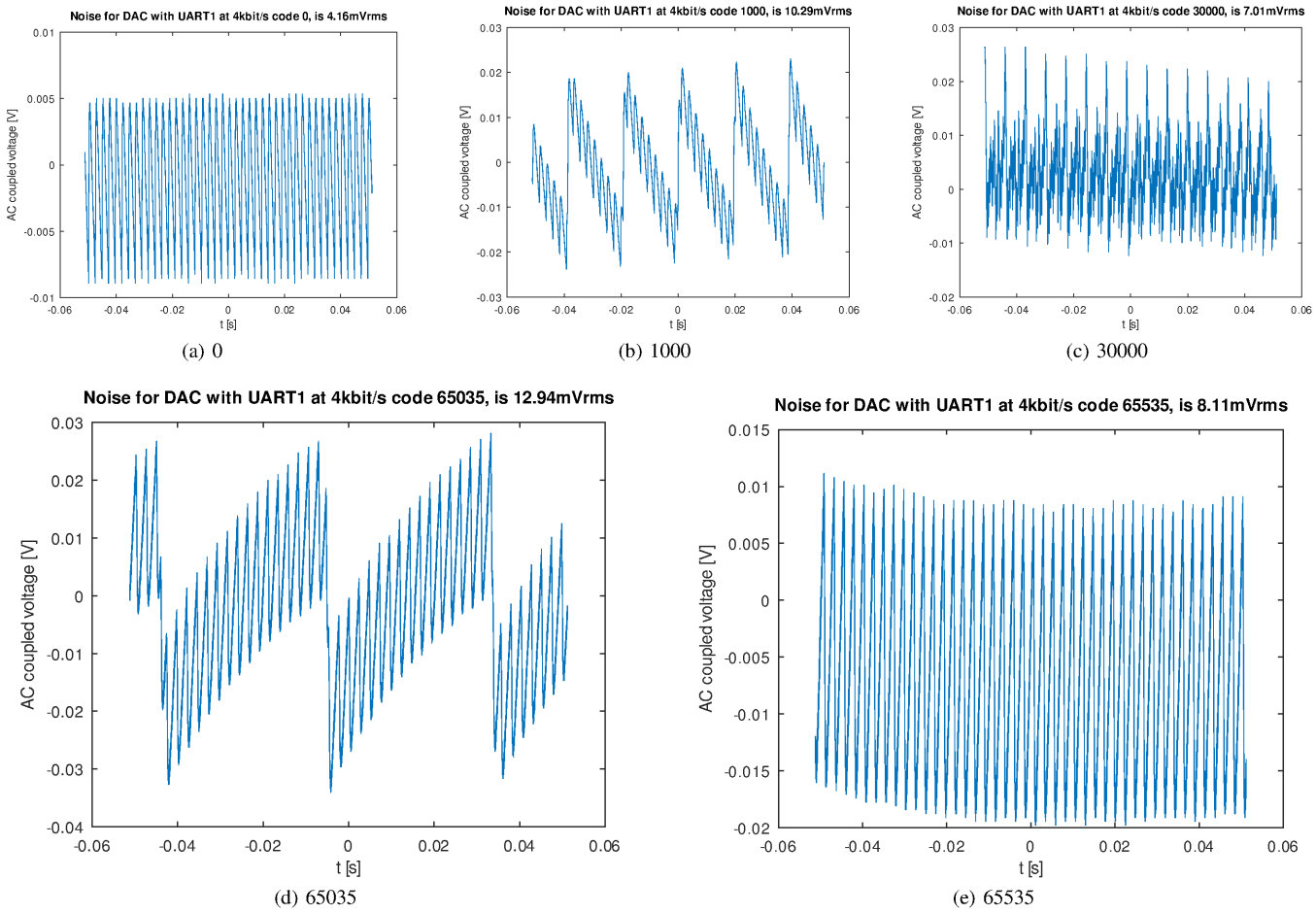
TABLE VIII: Verification of the trade-off algorithm emulation in QEMU.

B. Recommendation

In the prototype, the status byte prepended to each conversion was used by the microcontroller to check if there was an over- or under-range situation, but it is possible to use the hardware interrupt pin of the NAFE13388 to switch off the current shunts directly in hardware, by using some logical and-gates so that the microcontroller can turn the shunts on individually via one input per and-gate, and that the other input of each and-gate is connected together to the interrupt output INTB (which is active low). The interrupt output is asserted as long as the global alarm is not cleared by the microcontroller [42]. The global alarm can be configured to include a selection of the available alarms, among which are the over- and under-range alarms, but also over- and under-load, over-voltage, supply voltage and over-temperature alarms. See Table 31 in [42]. Implementing the protection in hardware makes it more reliable, since it will still work when the SPI communication fails for some reason, or the microcontroller locks up. Other than this, and electrical isolation, the way in which the NAFE13388 is connected on the NAFE13388-UIM development board [47] is also what would be the best when included in a redesign.

The NAFE13388 was chosen as a result of a comparison made between different devices, but when an extender has to be made which combines analogue inputs and outputs on the same software configurable pins, it may be a better choice to go for the NAFE33352, since it has one software configurable analogue input/output, which saves external circuitry which has to switch between modes. It is a lot cheaper than the MAX22000, and also better protected. The only downside is that it has 2 regular inputs instead of 3.

For the analogue outputs, PDM seems to be a good alternative, or a higher order analogue filter as suggested in section IV-B1. With the trade-off algorithm implementation, the maximum time the chosen microcontroller has to spend on updating



(a) 0 (b) 1000 (c) 30000

(d) 65035 (e) 65535

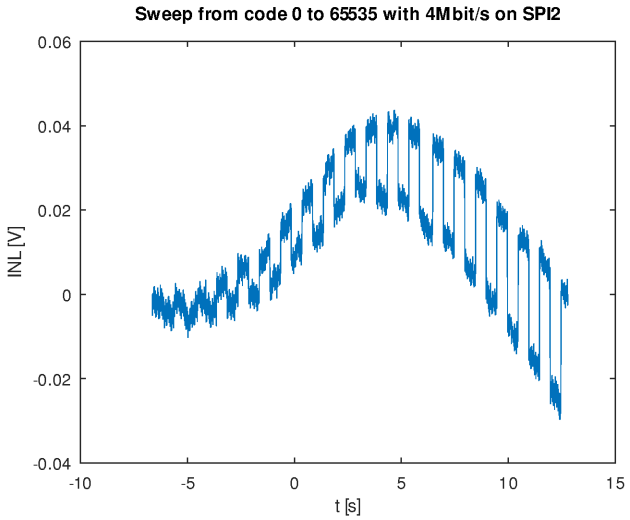
Fig. 19: Noise for different DAC codes at very low speed.

DAC code	Noise
0	0.42mVrms
100	0.49mVrms
200	0.50mVrms
300	0.47mVrms
400	0.48mVrms
500	0.47mVrms
600	0.48mVrms
700	0.49mVrms
800	0.49mVrms
900	0.51mVrms
1000	0.48mVrms
1100	0.46mVrms
1200	0.47mVrms
1300	0.44mVrms
1400	0.48mVrms
1500	0.50mVrms
1600	0.49mVrms
1700	0.47mVrms
1800	0.49mVrms
1900	0.50mVrms
2000	0.49mVrms
2100	0.49mVrms
2200	0.48mVrms
2300	0.46mVrms
2400	0.47mVrms
2500	0.46mVrms
2600	0.47mVrms
2700	0.47mVrms
2800	0.48mVrms
2900	0.47mVrms
3000	0.44mVrms
3100	0.44mVrms
3200	0.44mVrms
3300	0.45mVrms
3400	0.43mVrms
3500	0.45mVrms
3600	0.43mVrms
3700	0.44mVrms
3800	0.44mVrms
3900	0.45mVrms
4000	0.42mVrms
4100	0.45mVrms
4200	0.52mVrms
4300	0.43mVrms
4400	0.42mVrms
4500	0.42mVrms
30000	0.51mVrms
65035	0.89mVrms
65535	0.89mVrms

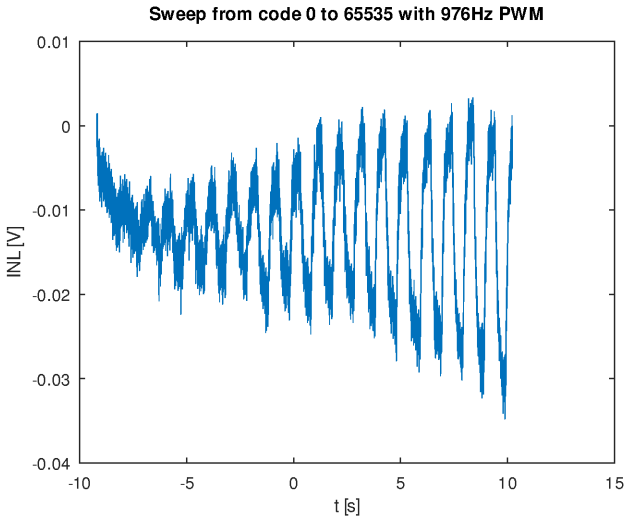
Fig. 20: TABLE IX: Measured noise at different output codes on SPI2.

the output value is 1.57ms (excluding the time copying the buffer over to the DMA buffer using memcpy takes, which is also approximately 1ms). It is however possible without much effort, to get the total time well under 1ms, simply by using the newlib memset and memcpy implementations which are optimized for speed instead of size. This can be done for example by copying these functions over under different names and compiling them in a project. Doing this may be preferable over writing custom optimized memory copying and setting functions as they then need to be thoroughly tested additionally to the used algorithm. Using optimized memset and memcpy functions has as an added benefit that the limited under- and overshoot derivation in section V-C applies as is. From measurements it could be seen that there is still no measurable amount of over- and undershoot even when copying 8 bits at a time when the element size in the buffer is 16-bit, so even though it would be nice to copy more than 8 bits at a time, which the speed optimized version for memcpy does do [55], for this aspect it doesn't matter that much.

APPENDIX



(a) SPI2 4Mbit/s



(b) PWM 976Hz

Fig. 21: INL for trade-off and PWM obtained with sawtooth sweep of full range.

A. Code

Listing 5: pdm.c: Different PDM algorithms in C

```c
#include <stdio.h>
#include <stdbool.h>
#include <stdint.h>
#include <string.h>
#include <assert.h>

#define LOW_PULSE_LOOP(buff, elem_size_shift, elem_size_mask, low_pulse_period) \
    do { \
        for (uint64_t i = 0; i <= (uint64_t)UINT16_MAX << 16; i += low_pulse_period) { \
            uint32_t floor_i = i >> 16; \
            buff[floor_i >> elem_size_shift] &= ~(1 << (elem_size_mask - (floor_i & elem_size_mask))); \
        } \
    } while (false)

#define HIGH_PULSE_LOOP(buff, elem_size_shift, elem_size_mask, high_pulse_period) \
    do { \
        for (uint64_t i = 0; i <= (uint64_t)UINT16_MAX << 16; i += high_pulse_period) { \
            uint32_t floor_i = i >> 16; \
            buff[floor_i >> elem_size_shift] |= 1 << (elem_size_mask - (floor_i & elem_size_mask)); \
        } \
    } while (false)

#define SIGMA_DELTA_LOOP(buff, elem_size_shift, elem_size_mask, ref, sigma) \
    do { \
        for (uint32_t i = 0; i <= UINT32_C(1) << 16; i++) { \
            bool out = sigma >> 16; \
            uint32_t delta = ref - ((uint32_t)out << 16); \
            sigma += delta; \
 \
            if (i > 0) { \
                if (out) \
                    buff[(i - 1) >> elem_size_shift] |= 1 << (elem_size_mask - ((i - 1) & elem_size_mask)); \
                else \
                    buff[(i - 1) >> elem_size_shift] &= ~(1 << (elem_size_mask - ((i - 1) & elem_size_mask))); \
            } \
        } \
    } while (false)

union {
    uint8_t buffer8[(1 << 16) / 8];
    uint16_t buffer16[(1 << 16) / 16];
} buff;

void __attribute__ ((noinline)) breakpoint_before() {
    asm volatile("");
}

void __attribute__ ((noinline)) breakpoint_after() {
    asm volatile("");
}

int reverse_bits(int val, int size) {
    int r = 0;

    for (int i = 0; i < size; i++) {
        r = (r << 1) | (val & 1);
        val >>= 1;
    }
    return r;
}

int main(void) {
    int elem_size = ELEM_SIZE;
    for (int ref = 0; ref <= UINT16_MAX; ref++) {
        breakpoint_before();
#ifdef TRADEOFF_ALGORITHM
        uint8_t base_levels[] = {
            0x00, // 0 out of 8 bits: 00000000.
            0x80, // 1 out of 8 bits: 10000000.
            0x88, // 2 out of 8 bits: 10001000.
            0x94, // 3 out of 8 bits: 10010100.
            0xaa, // 4 out of 8 bits: 10101010.
            0xb6, // 5 out of 8 bits: 10110110.
```

```
            0xee, // 6 out of 8 bits: 11101110.
            0xfe, // 7 out of 8 bits: 11111110.
            0xff, // 8 out of 8 bits: 11111111.
        };
        int base_on_bits = (ref + (1 << 12)) & (0x0f << 13); // Round to nearest base level by adding 0.5 to ref.
        memset(buff.buffer8, base_levels[base_on_bits >> 13], sizeof(buff.buffer8));

        int rel_ref = ref > base_on_bits ? ref - base_on_bits : base_on_bits - ref; // Turn on/off extra bits.
        if (rel_ref != 0) {
            uint32_t period = (UINT32_C(1) << 29) / rel_ref; // 13.16 fixed point.
            if (ref > base_on_bits)
                for (uint32_t i = 0; i <= ((1 << 13) - 1) << 16; i += period)
                    buff.buffer8[(i >> 16) ^ (elem_size == 16)] = base_levels[(base_on_bits >> 13) + 1];
            else
                for (uint32_t i = 0; i <= ((1 << 13) - 1) << 16; i += period)
                    buff.buffer8[(i >> 16) ^ (elem_size == 16)] = base_levels[(base_on_bits >> 13) - 1];
        }
#elif defined(OPTIMIZED_ALGORITHM)
        if (ref > UINT16_MAX / 2) {
            memset(buff.buffer8, 0xff, sizeof(buff.buffer8));
            uint64_t low_pulse_period = (UINT64_C(1) << 32) / ((UINT32_C(1) << 16) - ref); // 16.16 fixed point.
            if (elem_size == 16)
                LOW_PULSE_LOOP(buff.buffer16, 4, 15, low_pulse_period);
            else
                LOW_PULSE_LOOP(buff.buffer8, 3, 7, low_pulse_period);
        } else {
            memset(buff.buffer8, 0, sizeof(buff.buffer8));
            if (ref != 0) {
                uint64_t high_pulse_period = (UINT64_C(1) << 32) / ref; // 16.16 fixed point.
                if (elem_size == 16)
                    HIGH_PULSE_LOOP(buff.buffer16, 4, 15, high_pulse_period);
                else
                    HIGH_PULSE_LOOP(buff.buffer8, 3, 7, high_pulse_period);
            }
        }
#elif defined(NON_OPTIMIZED_ALGORITHM)
        memset(buff.buffer8, 0, sizeof(buff.buffer8));
        if (ref != 0) {
            uint64_t high_pulse_period = (UINT64_C(1) << 32) / ref; // 16.16 fixed point.
            if (elem_size == 16)
                HIGH_PULSE_LOOP(buff.buffer16, 4, 15, high_pulse_period);
            else
                HIGH_PULSE_LOOP(buff.buffer8, 3, 7, high_pulse_period);
        }

#elif defined(SIGMA_DELTA_ALGORITHM)
        uint32_t sigma = 0;
        if (elem_size == 16)
            SIGMA_DELTA_LOOP(buff.buffer16, 4, 15, ref, sigma);
        else
            SIGMA_DELTA_LOOP(buff.buffer8, 3, 7, ref, sigma);
#endif
        breakpoint_after();

#define CONCATENATE(x, y) x##y
#define EVALUATE(x, y) CONCATENATE(x, y)
#define BUFFER_EVALUATE(buffer, ELEM_SIZE)
        int pulse_pos = 0, last_pulse_pos = 0, sum = 0, max_pulse_dist = 0, min_pulse_dist = UINT16_MAX;
        // FILE *f = fopen("tmp", "w");
        for (int i = 0; i < (int)(sizeof(buff.BUFFER) / sizeof(*buff.BUFFER)); i++) {
            for (int j = 0; j < ELEM_SIZE; j++, pulse_pos++) {
                if (reverse_bits(buff.BUFFER[i], ELEM_SIZE) & (1 << j)) {
                    // fprintf(f, "%d\n", pulse_pos - last_pulse_pos);
                    if (pulse_pos - last_pulse_pos != 0) {
                        if (pulse_pos - last_pulse_pos > max_pulse_dist)
                            max_pulse_dist = pulse_pos - last_pulse_pos;
                        if (pulse_pos - last_pulse_pos < min_pulse_dist)
                            min_pulse_dist = pulse_pos - last_pulse_pos;
                    }
                    last_pulse_pos = pulse_pos;
                    sum++;
                }
            }
        }
        // fclose(f);
        printf("sum = %d, ref = %d, min = %d, max = %d\n", sum, ref, min_pulse_dist, max_pulse_dist);
        assert(max_pulse_dist - min_pulse_dist <=
```

```
#ifdef TRADEOFF_ALGORITHM
            8
#else
            1
#endif
        );
        assert(sum == ref);
    }
}
```

Listing 6: pdm.sh: Shell script to automatically generate CSV file with profiling results

```
elem_size=16
algorithm=OPTIMIZED_ALGORITHM

arm-none-eabi-gcc -Wall -Wextra -Wpedantic -O2 -ffreestanding -mcpu=cortex-m0plus \
    -mfloat-abi=soft -mthumb -D__NO_SYSTEM_INIT -DELEM_SIZE=$elem_size -D$algorithm \
    -ICMSIS_4/Device/ARM/ARMCM0plus/Include/ CMSIS_4/Device/ARM/ARMCM0plus/Source/GCC/startup_ARMCM0plus.c \
    pdm.c -T CMSIS_4/Device/ARM/ARMCM0plus/Source/GCC/gcc_arm.ld --specs=rdimon.specs -o pdm.elf
arm-none-eabi-objdump -D pdm.elf > pdm.list

qemu-system-arm -s -S -machine lm3s6965evb -cpu cortex-m0 -semihosting \
    --semihosting-config enable=on,target=native -nographic -serial mon:stdio \
    -monitor tcp:127.0.0.1:55555,server,nowait -kernel pdm.elf \
    -icount shift=auto,rr=record,rrfile=replay.bin &
> qemu_output_before.txt
> qemu_output_after.txt
arm-none-eabi-gdb -x commands.gdb --batch --args pdm.elf

while IFS= read -r before && IFS= read -r after <&3; do
    echo "$((${after#instruction count = } - ${before#instruction count = })),"
done < qemu_output_before.txt 3< qemu_output_after.txt > instruction_count.csv

rm qemu_output_before.txt qemu_output_after.txt
```

Listing 7: get_icount.sh: Shell script called from GDB script

```
( echo "info replay" | nc -N 127.0.0.1 55555 & ) | grep -Eom 1 "instruction count = [0-9]+" >> qemu_output_$1.txt
# kill 0
```

Listing 8: commands.gdb: GDB script to automate break point handling and getting the instruction count

```
set pagination off
set logging file gdb_output.txt
set logging enabled on

target remote localhost:1234

break breakpoint_before
    command 1
    shell bash get_icount.sh before
    continue
end

break breakpoint_after
    command 2
    shell bash get_icount.sh after
    continue
end

continue

set logging enabled off
quit
```

Listing 9: Script to plot AC measurements with analog input

```
freqs = [ 0.1, 1, 10 ];

for dir = { 'Prototype', 'SCC Compact' }
  % x_offset = dlmread(sprintf('%s/5V.csv', dir{1}));
  % x_offset = mean(x_offset(:, 1));
  if strcmp(dir{1}, 'Prototype')
    x_offset = 2^23 / 25 * 5;
    x_amplitude = 2^23 / 25 * 5;
    fs = 900 / 8 * 1.002; % Correct for clock frequency being slightly off.
  elseif strcmp(dir{1}, 'SCC Compact')
    x_offset = 2^16 / (4.096 * (20e3 + 33e3) / 20e3) * 5;
    x_amplitude = 2^16 / (4.096 * (20e3 + 33e3) / 20e3) * 5;
    fs = 100;
  end

  for f = freqs
    x = dlmread(sprintf('%s/%gHz.csv', dir{1}, f));
    x = x(:, 1) - mean(x(:, 1));

    % figure();
    % plot([x(end-100:end); x(1:100)]);

    y = fft(x);
    % y_sig = or(y == max(y(1:floor(end/2))), y == max(y((floor(end/2)+1):end)));
    % k = find(y_sig == 1);
    % y_sig(k(1) - 5:k(1) + 5) = 1;
    % y_sig(k(2) - 5:k(2) + 5) = 1;
    % y_sig(1) = 1;
    % y_sig = y_sig .* y;
    y_data = 20*log10(abs(y) / (length(x)/2) / x_amplitude);

    figure();
    semilogx(0:fs/length(y_data):(fs/2 - fs/length(y_data)), y_data(1:end/2));
    xlabel('f [Hz]');
    ylabel('Amplitude [dB]');
    ylim([-140, 0]);
    title(sprintf("%s ADC output spectrum with %gHz sinewave input", dir{1}, f));

    func = @(x, p) p(1) * sin(p(2) * x + p(3));
    cost = @(p) sum((func((0:(length(x) - 1))', p) - x).^2);

    theta = asin(x(1) / x_amplitude);
    if x(1) > x(2)
      theta = pi - theta;
    end
    p0 = [ x_amplitude, 2 * pi * f / fs, theta ];
    p = fminsearch(cost, p0, optimset('MaxFunEvals', 1000, 'MaxIter', 1000));

    x_sig = func((0:(length(x) - 1))', p);

    % x_sig = real(ifft(y_sig));
    x_noise = x - x_sig;
    sndr = 20*log10(rms(x_sig)/rms(x_noise));
    legend(sprintf('SINAD = %gdB', sndr));
  end
end
```

Listing 10: Script to plot DC measurements with analog input

```
voltages = [ 0, 2, 5, 10 ];

for dir = { 'Prototype', 'SCC Compact' }
  if strcmp(dir{1}, 'Prototype')
    x_codes_per_v = 2^23 / 25;
    fs = 900 / 8;
  elseif strcmp(dir{1}, 'SCC Compact')
    x_codes_per_v = 2^16 / (4.096 * (20e3 + 33e3) / 20e3);
    fs = 100;
  end

  for v = [ nan, voltages ] % NaN is short.
    if isnan(v)
      x = dlmread(sprintf('%s/short.csv', dir{1}));
    else
      x = dlmread(sprintf('%s/%gV.csv', dir{1}, v));
    end

    x = x(:, 1);

    figure();
    plot(0:1/fs:(length(x) - 1)/fs, x / x_codes_per_v);
    xlabel('t [s]');
    ylabel('ADC output code normalized to voltage [V]');
    title(sprintf("%s ADC output with %gV input", dir{1}, v));
    printf('%s %gV noise Vrms = %gmV\n', dir{1}, v, rms(x - mean(x)) / x_codes_per_v * 1000);
  end
end
```

Listing 11: Script to plot SINC frequency response with analog input

```
DRO = 21;

x = dlmread(sprintf('Prototype/DRO%d.csv', DRO));

x_codes_per_v = 2^23 / 25;
x_offset = x_codes_per_v * 5;
x_amplitude = x_codes_per_v * 2.5;

x = x(:, 1);

if DRO == 22
  fs = 25 / 8;
elseif DRO == 21
  fs = 30 / 8;
end
slice_length = 225; % Not so small that FFT will be very small but not so large that frequency step becomes
    too large.
sweep_duration = 9999; % In seconds.
sweep_decades = 4; % From 0.01Hz to 100Hz.
sweep_start_decade = -2; % From 0.01Hz.

y = zeros(1, ceil(length(x) / slice_length));
for i = 1:slice_length:length(x)
  x_slice = x(i:min((i + slice_length), end)) - x_offset;
  y_slice = abs(fft(x_slice)) / (length(x_slice) / 2) / x_amplitude;
  f = 10 .^(((i:(i + length(x_slice))) / fs / sweep_duration) * sweep_decades + sweep_start_decade);
  alias_f = fs/2 - abs(((floor(f / fs) * 2) + 1) * fs/2 - f);
  x_alias_f = alias_f / fs * length(x_slice);
  %figure();
  %hold('on');
  x_plot_data = 0:(fs / length(x_slice)):(fs/2 - fs / length(x_slice));
  y_plot_data = 20*log10(y_slice(1:end/2));
  %plot(y_plot_data);
  %plot(x_alias_f, zeros(1, length(x_alias_f)), 'r*');

  y_slice_indices = unique(round(x_alias_f) + 1);
  y((i - 1) / slice_length + 1) = sum(y_slice(y_slice_indices)) / sqrt(length(y_slice_indices));
end

x_data = 10 .^(((1:length(y)) / length(y)) * sweep_decades + sweep_start_decade);
figure();
semilogx(x_data, 20*log10(y));
xlabel('f [Hz]');
ylabel('Amplitude [dB]');
if DRO == 22
  title('SINC1 NMR at 50Hz');
elseif DRO == 21
  title('SINC1 NMR at 60Hz');
end

figure();
semilogx(10 .^(((2:length(x)) / (length(x)-1)) * sweep_decades + sweep_start_decade), x(2:end) /
    x_codes_per_v);
xlabel('f [Hz]');
ylabel('Amplitude [V]');
if DRO == 22
  title('SINC1 NMR at 50Hz envelope');
elseif DRO == 21
  title('SINC1 NMR at 60Hz envelope');
end
```

Listing 12: Script to plot measurements with analog output

```
%% Plot INL obtained with sweeps.
v_sweep_starts = [ 1900, 1100, 900, 1400, 400, 1500 ];
titles = { '32Mbit/s on SPI2', '8Mbit/s on SPI2', '4Mbit/s on SPI2', '21.33Mbit/s on LPUART1', '8Mbit/s on
    LPUART1', '8Mbit/s on UART1' };

for i = 1:6
  v_sweep = dlmread(sprintf('acq%04d.csv', 100 + i), ',');
  v_sweep_t = v_sweep(:, 1);
  v_sweep = v_sweep(:, 2);

  v_sweep_start = v_sweep_starts(i);
  v_sweep_length = 5300;
  v_sweep = v_sweep(v_sweep_start:v_sweep_start + v_sweep_length)';
  v_sweep_t = v_sweep_t(v_sweep_start:v_sweep_start + v_sweep_length)';
  figure();
  plot(v_sweep_t, v_sweep - (v_sweep(1):((v_sweep(end) - v_sweep(1)) / 5300):v_sweep(end)));
  title_ = titles(i);
  xlabel('t [s]');
  ylabel('INL [V]');
  title(sprintf('Sweep from code 0 to 65535 with %s', title_{1}));
end

v_sweep = dlmread('acq0402.csv', ',');
v_sweep_t = v_sweep(:, 1);
v_sweep = v_sweep(:, 2);

v_sweep_start = 1500;
v_sweep = v_sweep(v_sweep_start:v_sweep_start + v_sweep_length)';
v_sweep_t = v_sweep_t(v_sweep_start:v_sweep_start + v_sweep_length)';
figure();
plot(v_sweep_t, v_sweep - (v_sweep(1):((v_sweep(end) - v_sweep(1)) / 5300):v_sweep(end)));
xlabel('t [s]');
ylabel('INL [V]');
title('Sweep from code 0 to 65535 with 976Hz PWM');

%% Plot noise.
dac_codes = [ 0, 1000, 30000, 65035, 65535 ];
for i = 1:5
  v_spi2 = dlmread(sprintf('acq%04d.csv', i), ',');
  v_lpuart1 = dlmread(sprintf('acq%04d.csv', i + 5), ',');
  v_uart1 = dlmread(sprintf('acq%04d.csv', i + 10), ',');
  v_scc = dlmread(sprintf('acq%04d.csv', i + 15), ',');
  v_spi2 = v_spi2(11:end, :);
  v_lpuart1 = v_lpuart1(11:end, :);
  v_uart1 = v_uart1(11:end, :);
  v_scc = v_scc(11:end, :);

  figure();
  hold('on');
  plot(v_spi2(:, 1), v_spi2(:, 2));
  plot(v_lpuart1(:, 1), v_lpuart1(:, 2));
  plot(v_uart1(:, 1), v_uart1(:, 2));
  plot(v_scc(:, 1), v_scc(:, 2));
  mv_spi2_rms = 1000 * rms(v_spi2(:, 2) - mean(v_spi2(:, 2)));
  mv_lpuart1_rms = 1000 * rms(v_lpuart1(:, 2) - mean(v_lpuart1(:, 2)));
  mv_uart1_rms = 1000 * rms(v_uart1(:, 2) - mean(v_uart1(:, 2)));
  mv_scc_rms = 1000 * rms(v_scc(:, 2) - mean(v_scc(:, 2)));
  legend(sprintf('Noise with SPI2, is %.2fmVrms', mv_spi2_rms), sprintf('Noise with LPUART1, is %.2fmVrms',
      mv_lpuart1_rms), ...
    sprintf('Noise with UART1, is %.2fmVrms', mv_uart1_rms), sprintf('Noise with SCC Compact, is %.2fmVrms'
        , mv_scc_rms));
  xlabel('t [s]');
  ylabel('AC coupled voltage [V]');
  title(sprintf('Noise for DAC code %d', dac_codes(i)));

  v_uart1_4k = dlmread(sprintf('acq%04d.csv', 200 + i), ',');
  v_uart1_4k = v_uart1_4k(11:end, :);

  figure();
  plot(v_uart1_4k(:, 1), v_uart1_4k(:, 2));
  xlabel('t [s]');
  ylabel('AC coupled voltage [V]');
  mv_uart1_4k_rms = 1000 * rms(v_uart1_4k(:, 2) - mean(v_uart1_4k(:, 2)));
  title(sprintf('Noise for DAC with UART1 at 4kbit/s code %d, is %.2fmVrms', dac_codes(i), mv_uart1_4k_rms)
      );
```

```
end

%% Plot sinewave and its FFT.
v_uart1 = dlmread('acq0301.csv', ',');
v_uart1 = v_uart1(11:end, :);

v_sine_start = 1500;
v_sine_length = 4224;
t_sine = v_uart1(v_sine_start:v_sine_start + v_sine_length, 1);
v_sine = v_uart1(v_sine_start:v_sine_start + v_sine_length, 2);

figure();
plot(t_sine, v_sine);
xlabel('t [s]');
ylabel('v [V]');
title('Full range sinewave with 8Mbit/s on UART1');

figure();
y_sine = fft((v_sine - mean(v_sine)));
y_data = 20*log10(abs(y_sine / (length(y_sine)/2)));
semilogx(0:(1/(t_sine(end) - t_sine(1))):((length(y_sine) - 1)/(mean(diff(t_sine)) * length(y_sine))/2),
    y_data(1:end/2));
ylim([-140 20]);

% Set fundamental to 0.
y_sine(2) = 0;
y_sine(end) = 0;
v_noise = real(ifft(y_sine));
sndr = 20*log10(rms(v_sine - v_noise) / rms(v_noise));
legend(sprintf('SINAD = %gdB', sndr));
title('Full range sinewave spectrum with 8Mbit/s on UART1');
xlabel('f [Hz]');
ylabel('Amplitude [dB]');
```

Listing 13: Script to plot measurements with analog output and tradeoff algorithm

```
%% Plot INL obtained with sweep.
v_sweep = dlmread('acq0603.csv', ',');
v_sweep_t = v_sweep(:, 1);
v_sweep = v_sweep(:, 2);

v_sweep_start = 2200;
v_sweep_length = 5300;
v_sweep = v_sweep(v_sweep_start:v_sweep_start + v_sweep_length)';
v_sweep_t = v_sweep_t(v_sweep_start:v_sweep_start + v_sweep_length)';
figure();
plot(v_sweep_t, v_sweep - (v_sweep(1):((v_sweep(end) - v_sweep(1)) / 5300):v_sweep(end)));
xlabel('t [s]');
ylabel('INL [V]');
title(sprintf('Sweep from code 0 to 65535 with 4Mbit/s on SPI2'));

%% Plot noise.
dac_codes = [ 0:100:4500, 30000, 65035, 65535 ];
for i = 1:49
  continue
  v_spi2 = dlmread(sprintf('acq%04d.csv', 500 + i), ',');
  v_spi2 = v_spi2(11:end, :);

  figure();
  hold('on');
  plot(v_spi2(:, 1), v_spi2(:, 2));
  xlabel('t [s]');
  ylabel('AC coupled voltage [V]');
  mv_spi2_rms = 1000 * rms(v_spi2(:, 2) - mean(v_spi2(:, 2)));
  title(sprintf('Noise for DAC code %d with 4Mbit/s on SPI2, is %.2fmVrms', dac_codes(i), mv_spi2_rms));
end

%% Plot sinewave and its FFT.
v_uart1 = dlmread('acq0801.csv', ',');
v_uart1 = v_uart1(11:end, :);

v_sine_start = 1500;
v_sine_length = 4219;
t_sine = v_uart1(v_sine_start:v_sine_start + v_sine_length, 1);
v_sine = v_uart1(v_sine_start:v_sine_start + v_sine_length, 2);

figure();
plot(t_sine, v_sine);
xlabel('t [s]');
ylabel('v [V]');
title('Full range sinewave with 4Mbit/s on SPI2');

figure();
y_sine = fft((v_sine - mean(v_sine)));
y_data = 20*log10(abs(y_sine / (length(y_sine)/2)));
semilogx(0:(1/(t_sine(end) - t_sine(1))):((length(y_sine) - 1)/(mean(diff(t_sine)) * length(y_sine))/2),
    y_data(1:end/2));
ylim([-140 20]);

% Set fundamental to 0.
y_sine(2) = 0;
y_sine(end) = 0;
v_noise = real(ifft(y_sine));
sndr = 20*log10(rms(v_sine - v_noise) / rms(v_noise));
legend(sprintf('SINAD = %gdB', sndr));
title('Full range sinewave spectrum with 4Mbit/s on SPI2');
xlabel('f [Hz]');
ylabel('Amplitude [dB]');
```

Listing 14: Script to test and plot 2nd order PDM

```
% Test for different methods to generate PDM when assuming that the reference
% is constant over time.
N = 2^16;

% First order classic sigma delta.
x_sdm = sigma_delta(ref, N);
p_sdm = diff([0; find(x_sdm == 1)]);

% Pulse density modulation.
x_pdm = pulse_density(ref, N);
p_pdm = diff([0; find(x_pdm == 1)]);

% Second order classic sigma delta.
sigma = 0;
sigma2 = 0;
x_sdm2 = zeros(N, 1);

for i = 0:N
    out = (ref + 2 * sigma + sigma2) > N / 2;
    delta = ref - out * N;
    sigma = sigma + delta;
    sigma2 = sigma2 + sigma;

    if (i > 0)
        x_sdm2(i, 1) = out;
    end
end

p_sdm2 = diff([0; find(x_sdm2 == 1)]);

idx_incs_sdm2 = diff([0; find(p_sdm2 == ceil(N / ref))]);
idcs_sdm2 = cumsum(idx_incs_sdm2);

% Stateless pulse density calculation algorithm.
% Theoretically you can calculate all output values simultaneously.
order = 2;
%x_pdm2 = diff((mod(1:N + order, N/ref) * ref/N*2 - 1).^order, order);

p_pdm2 = zeros(ref, 1);
characteristic_pulse = diff([ones(order, 1); zeros(order, 1)], order);

closest = N / floor(N / ref);
remainder = closest - ref;

idx_incs = abs(mod(1:ref, ref / (remainder * 2)) - ref / (remainder * 4)) / (ref / (remainder * 2));
clear('idcs');
idcs(ceil(cumsum(idx_incs))) = 1:length(idx_incs);

pulse_flips = mod(idcs, ref / (remainder * 2)) > ref / (remainder * 4);
idx_ranges = (0:length(characteristic_pulse) - 1)' + idcs - ~pulse_flips;
pulses = characteristic_pulse * (1 - 2 * pulse_flips);
p_pdm2(idx_ranges) = pulses;
p_pdm2 = p_pdm2 + N / closest;

x_pdm2 = zeros(N, 1);
x_pdm2(cumsum(p_pdm2)) = ones(length(p_pdm2), 1);

%% Assert stuff

assert(all(x_pdm == x_sdm));
assert(sum(x_pdm) == ref);
assert(length(x_sdm2) == N);
assert(sum(x_sdm2) == ref);
assert(length(x_pdm2) == N);
assert(sum(x_pdm2) == ref);

%% Plot stuff

figure();
hold('on');
plot(idcs_sdm2);
plot(idcs);
legend('2nd order SDM', sprintf('%dth order PDM', order));

figure();
```

```
hold('on');
semilogx(max(20*log10(abs(fft(x_pdm))), -180));
semilogx(max(20*log10(abs(fft(x_pdm2))), -180));
semilogx(max(20*log10(abs(fft(x_sdm2))), -180));
xlim([1, N / 2]);
legend('PDM', sprintf('%dth order PDM', order), '2nd order SDM');
title(sprintf('Frequency spectra of 1-bit outputs for reference input = %d', ref));
xlabel('Frequency bin [n]');
ylabel('Amplitude [dB]');

figure();
hold('on');
semilogx(max(20*log10(abs(fft(p_sdm))), -180));
semilogx(max(20*log10(abs(fft(p_pdm2))), -180));
semilogx(max(20*log10(abs(fft(p_sdm2))), -180));
xlim([1, ref / 2]);
legend('1st order SDM', sprintf('%dth order PDM', order), '2nd order SDM');
title('Frequency spectra of p');
```

Listing 15: Dissassembly of memset function, it can be clearly seen that 1 byte gets set at a time (using the strb instruction).

```
08009930 <memset>:
 8009930:    0003            movs    r3, r0
 8009932:    1882            adds    r2, r0, r2
 8009934:    4293            cmp     r3, r2
 8009936:    d100            bne.n   800993a <memset+0xa>
 8009938:    4770            bx      lr
 800993a:    7019            strb    r1, [r3, #0]
 800993c:    3301            adds    r3, #1
 800993e:    e7f9            b.n     8009934 <memset+0x4>
```

Listing 16: Dissassembly of memcpy function, it can be clearly seen that 1 byte gets copied at a time (using the ldrb and strb instructions).

```
08009a36 <memcpy>:
 8009a36:    2300            movs    r3, #0
 8009a38:    b510            push    {r4, lr}
 8009a3a:    429a            cmp     r2, r3
 8009a3c:    d100            bne.n   8009a40 <memcpy+0xa>
 8009a3e:    bd10            pop     {r4, pc}
 8009a40:    5ccc            ldrb    r4, [r1, r3]
 8009a42:    54c4            strb    r4, [r0, r3]
 8009a44:    3301            adds    r3, #1
 8009a46:    e7f8            b.n     8009a3a <memcpy+0x4>
```

All MATLAB code was tested for compatibility with GNU Octave.

B. Prototype and SCC Compact setup

For most measurements, data was extracted from the prototype via the CAN bus using the SCC protocol. The SCC protocol, as it is used normally, is mostly documented internally, but when it is connected to a PC application called SCC Afregeltool (used for calibrating and testing the SCC Compact and extenders), there were some things not entirely clear from the documentation, so they are explained here.

There is a special calibration setup made by Brunelco used for calibrating and testing. This test setup contains a prototyping board based on a STM32 (OLIMEXINO-STM32). This prototyping board is programmed with a slight modification (SCC Emulator) of the code that normally runs on the SCC Compact. This modification allows for passing through RS485 commands from the SCC Afregeltool to the CAN bus on which the SCC extenders are connected. The main differences between the SCC Emulator code and the SCC Compact are that the SCC Emulator always has fixed address 1, while in the SCC Compact the address depends on a unique hardware register and that the SCC Emulator supports extra RS485 commands for passing through data directly to and from the SCC Compact extenders so that they can be calibrated, the software version read out and the dip switches tested. The SCC Emulator also supports a maximum of 5 connected extenders on the CAN bus instead of a maximum of 3 supported by the SCC Compact. This is purely a software configured limit, and there are no software/hardware differences to accommodate for more devices on the same bus.

To allow for direct communication between the SCC Afregeltool and the prototype (which presents itself on the CAN bus as a regular analogue input or output extender), a device had to be made similar to the SCC Emulator. It was possible to use a prototyping board and flash the SCC Emulator code on it, but it was easier to modify the regular SCC Compact code to also support the extra RS485 commands and have address 1, essentially merging the SCC Emulator and SCC Compact code, and flashing it on a regular SCC Compact. This special SCC Compact will be called SCC Compact with Extender Pass-through going forward.

On the calibration setup, there are two "golden" extenders, which are used as a reference to calibrate the extender under test with. These extenders have to have addresses 0 and 1, but their type doesn't matter for SCC Afregeltool, and since it is not supported by the SCC Compact to attach extenders with non-contiguous addresses, two kind of dummy "golden" extenders are present in the prototype setup to let the prototype have address 2. The prototype is then seen as an extender under test by SCC Afregeltool because it has address 2 (or higher).

For demonstration purposes, the prototype was also set up to be able to do measurements with thermocouple. For this the highest gain setting of $16\times$ was used. It was shown to have low enough noise to be able to measure temperature with around $0.5°$C precision.

The full prototype setup including the SCC Compact, SCC Compact with Extender Pass-through and three extenders (two of which golden) can be seen in Fig. 22.

On the analogue outputs, a low frequency sinewave was generated to be able to do a more proper distortion measurement. This sinewave was measured on the output of the XTR300EVM development board with the Analog Discovery 2. This can be seen for both the optimized algorithm and the trade-off algorithm in Fig. 23 and Fig. 24 respectively. Large peaks can be seen at 1Hz, 3Hz, 5Hz etc. caused by the LED flashing. In these measurements an ENOB of 10-bit is not yet obtained, but this could be obtained, when the effects of the LED flashing are gone. The ENOB can also be increased even further by reducing the INL, which could be done by reducing the bitrates even more. Although at some point the noise level will become significant.

In Fig. 25, the fluctuation of the output voltage when it is approximately $\frac{1}{4}$th of the maximum (code 16384 in both cases) can be seen more clearly (than in section VIII-B1) for both the SCC Compact extender (where the effect is minimal) and the prototype. Like mentioned in section IX-A, the amplitude of the fluctuation is much larger for the prototype because the supply has better load regulation in the SCC Compact.

C. Higher order noise shaping

It might be interesting to see if higher order noise shaping can be obtained in a similar way as in section V (so basically without feedback like in a SDM). Like shown in [23], the 1st order noise shaping of a VCO-ADC is a result of the sampling (time discretization) of the pulses generated by the VCO. If the pulses would be continuous time, the frequency spectrum up until the lowest frequency component (which is the pulse frequency) would be completely empty (this knowledge will come in useful later). The equivalent operation in the algorithm shown in section V is the differentiation of the sawtooth generated with a modulo operation. If we would apply differentiation with arbitrary order to this sawtooth wave, the resulting frequency spectrum will have the noise shaping characteristic of that same order. This is illustrated in Fig. 26. This results in a completely useless time domain output however, since we want the output to be 1-bit, and as soon as you differentiate more than once, there will be more than 2 possible output values. This is not the biggest problem yet though, as the shape of the 1st order pulses in the time domain are asymmetrical in the y-direction, while pulses resulting from higher order differentiation are symmetrical, thus preventing us from building up an offset to some constant level with them.

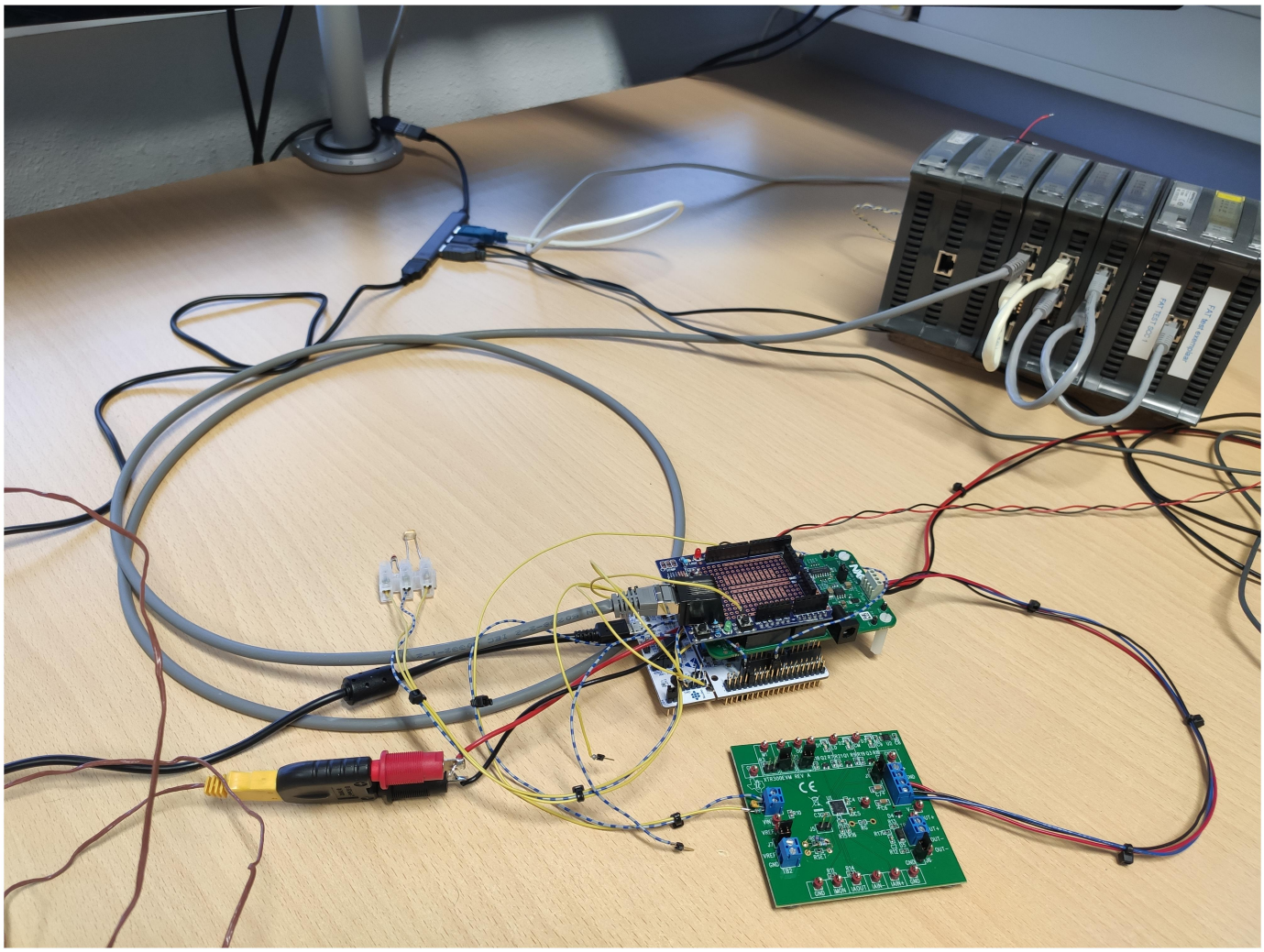


Fig. 22: Prototype setup with SCC Compact and extenders.

Extended noise shaping can be obtained in an ADC using a pulse shaping filter after the VCO (so still in continuous time) [56]. This is not very useful in our case, since a pulse shaping filter would not have a 1-bit output.

For the next part, it is useful to define p as the pulse period over time (so not the average pulse period). By plotting it for the algorithms from section V, we can see that p consists of 1st order pulses so that the mean of p is equal to the average pulse period (such as in Fig. 27.

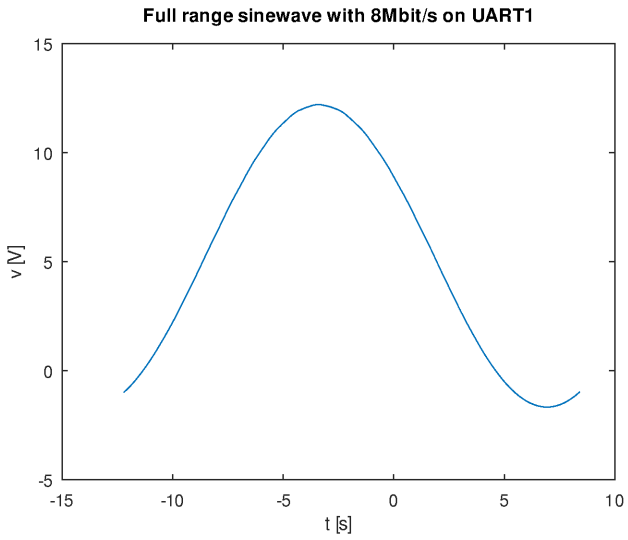
If we would make p so that it is purely constructed from "higher order pulses", the final output x might also have noise shaping of this same order, because if p would not contain any pulses, and would be able to take on some fractional value, the frequency spectrum would be completely empty up until the pulse frequency (this is kind of a rephrasing of the first paragraph of this section). These higher order pulses are essentially the result of differentiating a discrete unit step function. For illustration purposes, higher order pulses can be seen in Fig. 28

The mean of all higher order pulses is 0, since they are derivatives (removing any constant offset) of a 1st order pulse, which has a mean of 1. This means that no offset can be built up with them, like with 1st order pulses (as mentioned before, see Fig. 27 for example).

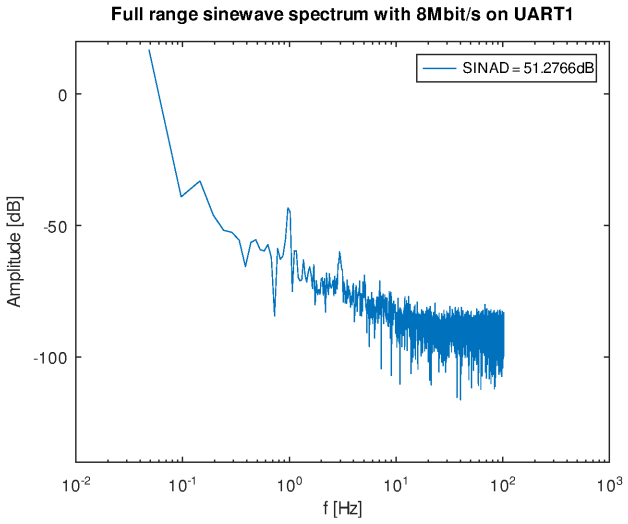
Looking at p for a 2nd order SDM, we can see that (at least for reference values very close to $\frac{N}{2} = 32768$) the offset is built up by shifting the phase of the pulses in p and "snooping" away half a pulse exactly as often as is required. This can best be seen in Fig. 29.

This idea can be applied to a kind of 2nd order PDM generation without feedback. An implementation in MATLAB used for testing can be seen in Listing 14.

In Fig. 30, it is clearly visible that 2nd order noise shaping (with 40dB/dec) can be obtained, but only for reference input values slightly below powers of 2. This algorithm without feedback even has the advantage of working well at the lower limits, when a higher order SDM would become unstable (such as in Fig. 30i. Its performance degrades quickly when going too far below a power of 2 though (such as in Fig. 30j, Fig. 30k and Fig. 30l). When going even further below a power of 2 (or going

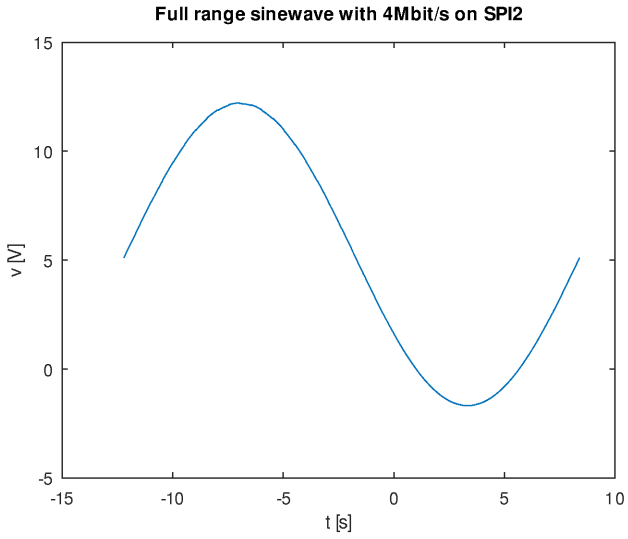


(a) Time domain plot

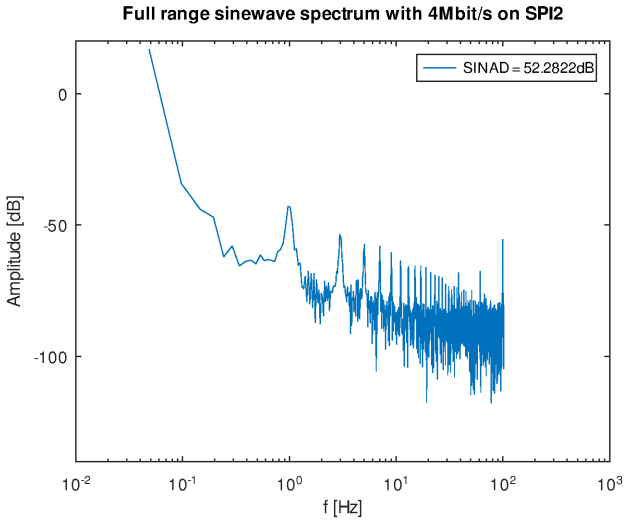


(b) Frequency spectrum

Fig. 23: Low frequency sinewave output when using the optimized algorithm on the UART1 peripheral.



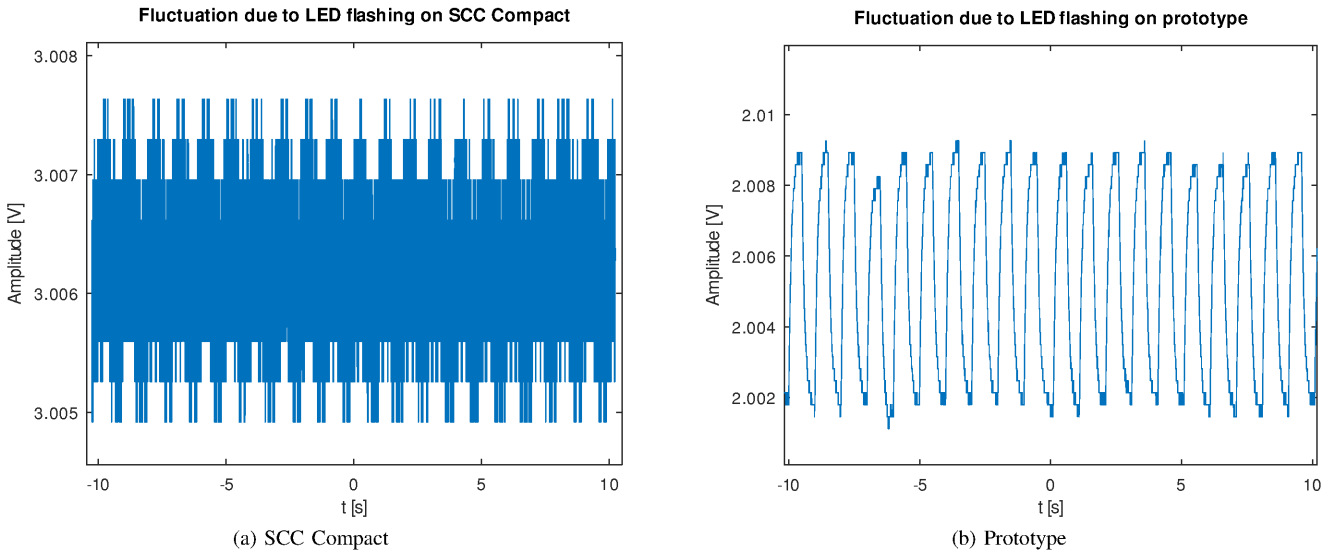
(a) Time domain plot



(b) Frequency spectrum

Fig. 24: Low frequency sinewave output when using the trade-off algorithm on the SPI2 peripheral.

slightly above powers of 2), the algorithm doesn't work at all; the output bit pattern sum doesn't equal the reference input. It would also be quite a computationally heavy algorithm for a microcontroller, and not in any way more efficient than just a 2nd order SDM. Because of all these downsides, no further tests were done. This section is not omitted and still placed here, in the appendix, because I found it kind of interesting to see how the behaviour of a 2nd order SDM with very limited input range(s) and constant input could be described in the time domain.



(a) SCC Compact (b) Prototype

Fig. 25: Analogue output voltage fluctuation due to LED flashing.

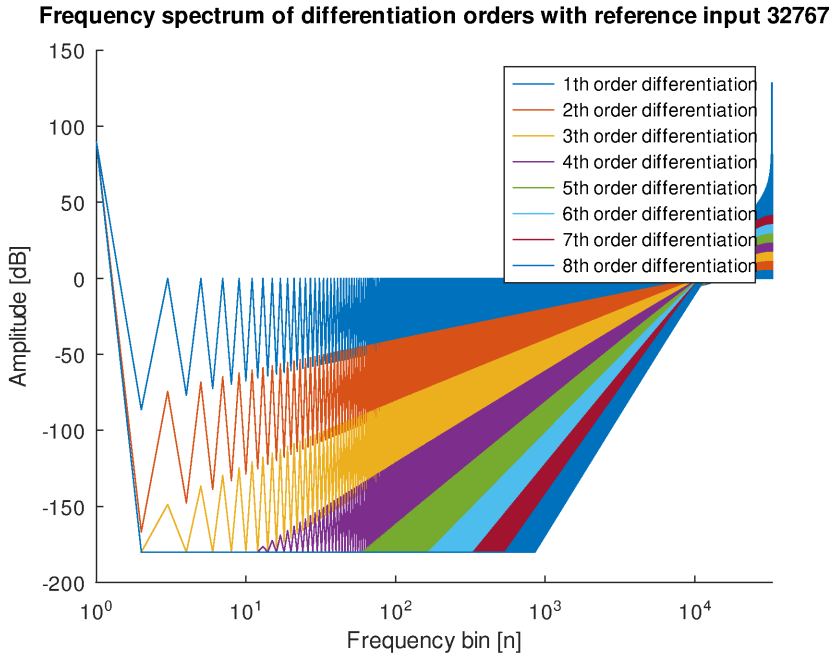


Fig. 26: Frequency spectra of unhelpful outputs to show noise shaping due to differentiation.

During the preparation of this work the author(s) used no artificial intelligence tools.

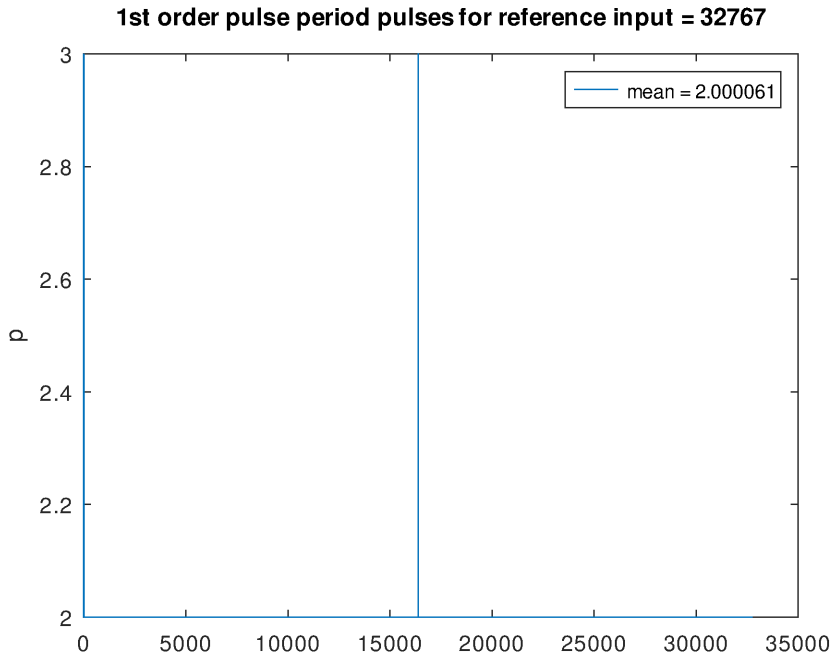
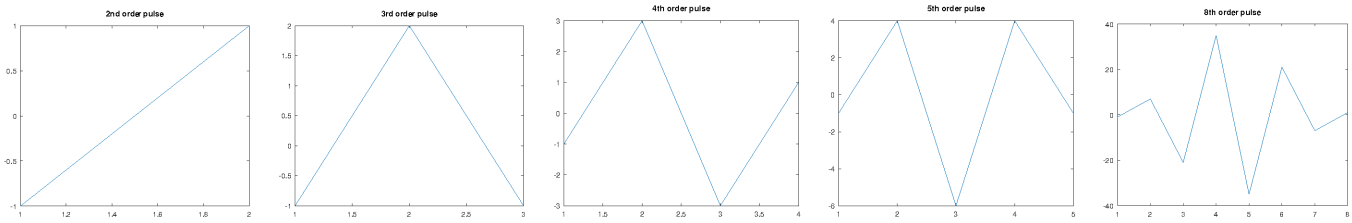
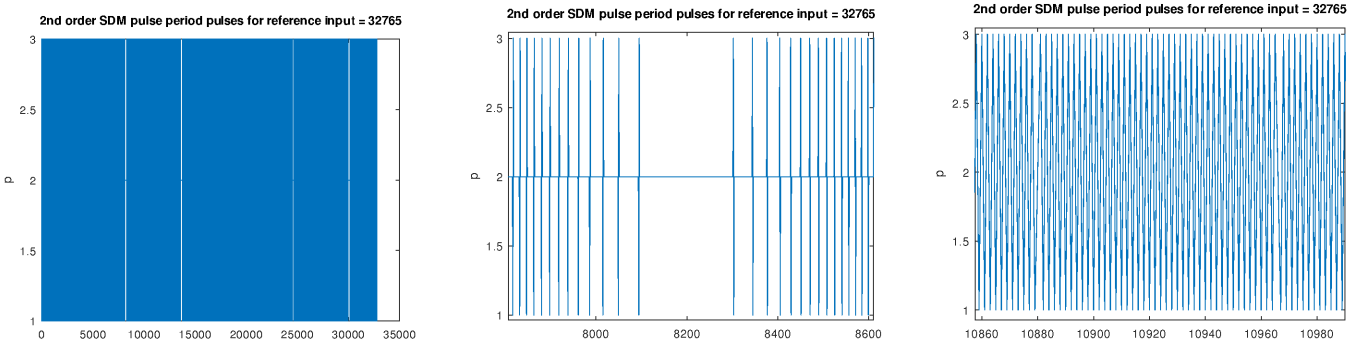


Fig. 27: Pulses in the 1st order pulse period p.



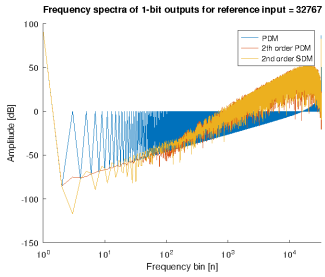
(a) 2nd order pulse (b) 3rd order pulse (c) 4th order pulse (d) 5th order pulse (e) 8th order pulse

Fig. 28: Time domain plot of higher order pulses, where the number of samples of the pulse is always equal to the derivative order.

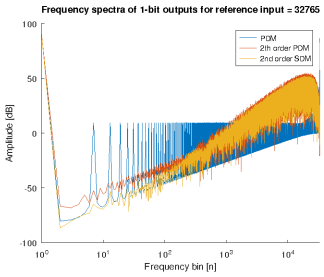


(a) Overview (b) Zoom where pulse polarity inverts (phase shift) (c) Zoom where pulse polarity inverts and half a pulse is snooped away

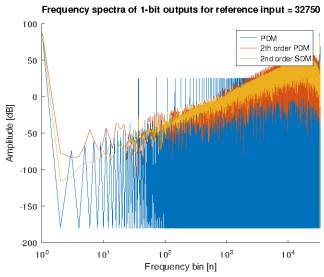
Fig. 29: Time domain plot of 2nd order SDM pulse period p



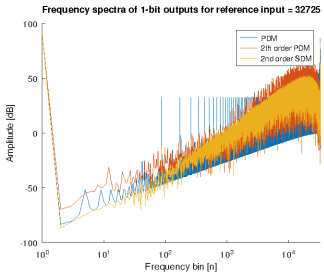
(a) Best possible reference input 32767



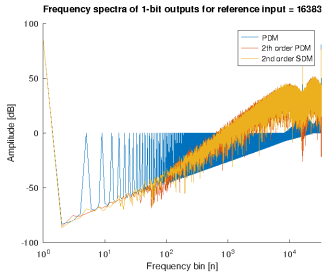
(b) Reference input slightly below a



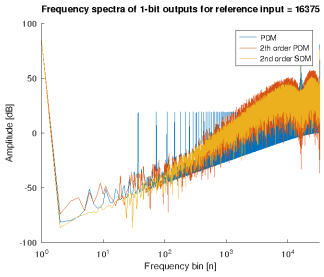
(c) Reference input slightly below b



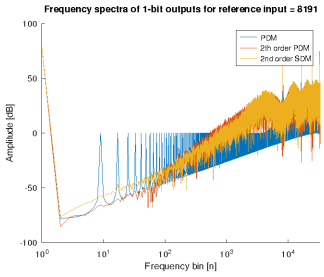
(d) Reference input slightly below c



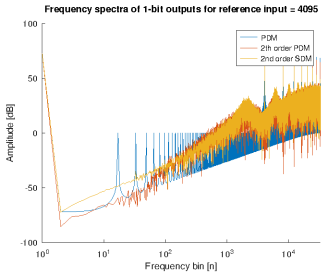
(e) Best possible reference input below $\frac{N}{4}$: 16383



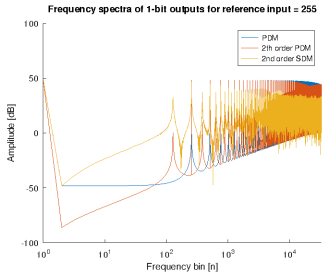
(f) Reference input slightly below e



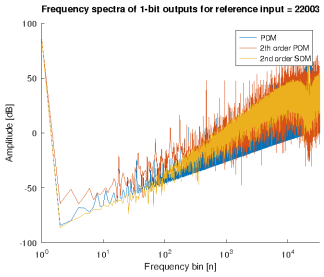
(g) Best possible reference input below $\frac{N}{8}$: 8191



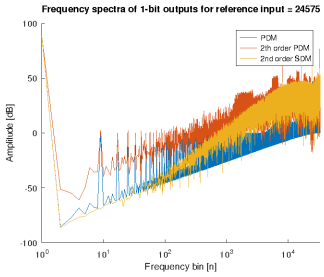
(h) Best possible reference input below $\frac{N}{16}$: 4095



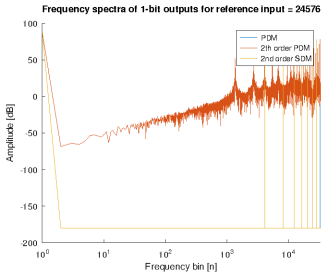
(i) Best possible reference input below $\frac{N}{256}$: 255



(j) Reference input 22003



(k) Reference input 24575



(l) Reference input 24576, 1st order PDM and 2nd order SDM frequency spectra up until the pulse frequency is almost completely empty

Fig. 30: Comparison of the frequency spectra of the outputs of a 2nd order SDM, 1st order PDM and 2nd order PDM.

REFERENCES

[1] "Smart Cylinder Control - Brunelco," Aug. 2021, [Online; accessed 9. Apr. 2025]. [Online]. Available: https://www.brunelco.nl/en/projecten/smart-cylinder-control

[2] "NEN Connect - IEC 61131-2:2017 en," Feb. 2025, [Online; accessed 6. Feb. 2025]. [Online]. Available: https://connect.nen.nl/Standard/Detail/3532153?compId=16755&collectionId=0

[3] "Analog Front-End," Jan. 2025, [Online; accessed 3. Feb. 2025]. [Online]. Available: https://www.nxp.com/products/analog-and-mixed-signal/analog-front-end:ANALOG-FRONT-END

[4] "Enerpac-SccIoExtenders-Hardware - Revision 82: /trunk/Documenten/Handleidingen & specs," May 2018, [Online; accessed 10. Feb. 2025]. [Online]. Available: https://brunux:8443/svn/Enerpac-SccIoExtenders-Hardware/trunk/Documenten/Handleidingen%20&%20specs/SCC%20IO%20Extenders%20-%20Specifications%20&%20Installation%20manual%20v2.pdf

[5] "Enerpac-SccIoExtenders-Hardware - Revision 82: /trunk/Hardware/SCC Analog input extender," Oct. 2024, [Online; accessed 10. Feb. 2025]. [Online]. Available: https://brunux:8443/svn/Enerpac-SccIoExtenders-Hardware/trunk/Hardware/SCC%20Analog%20input%20extender

[6] "NEN Connect - IEC 61000-4-2:2025 en;fr," Apr. 2025, [Online; accessed 9. Apr. 2025]. [Online]. Available: https://connect.nen.nl/Standard/Detail/3719620?compId=16755&collectionId=0

[7] "NEN Connect - IEC 61000-4-5:2014+A1:2017-CSV en;fr," Apr. 2025, [Online; accessed 9. Apr. 2025]. [Online]. Available: https://connect.nen.nl/Standard/Detail/3531149?compId=16755&collectionId=0

[8] "NEN Connect - CLC Guide 34:2024 en," Apr. 2025, [Online; accessed 18. Apr. 2025]. [Online]. Available: https://connect.nen.nl/Standard/Detail/3702870?compId=16755&collectionId=0

[9] "NEN Connect - NEN-EN-IEC 61000-6-2:2019 en," Apr. 2025, [Online; accessed 18. Apr. 2025]. [Online]. Available: https://connect.nen.nl/Standard/Detail/3579492?compId=16755&collectionId=0

[10] "STM32F103 - PDF Documentation," Apr. 2025, [Online; accessed 9. Apr. 2025]. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32f103/documentation.html

[11] "Enerpac-SccIoExtenders-Hardware - Revision 82: /trunk/Hardware/SCC Analog output extender," Dec. 2024, [Online; accessed 10. Feb. 2025]. [Online]. Available: https://brunux:8443/svn/Enerpac-SccIoExtenders-Hardware/trunk/Hardware/SCC%20Analog%20output%20extender

[12] "Enerpac-SccIoExtenders-Hardware - Revision 82: /trunk/Hardware/SCC Digital input extender," Dec. 2024, [Online; accessed 10. Feb. 2025]. [Online]. Available: https://brunux:8443/svn/Enerpac-SccIoExtenders-Hardware/trunk/Hardware/SCC%20Digital%20input%20extender

[13] "Enerpac-SccIoExtenders-Hardware - Revision 82: /trunk/Hardware/SCC Digital output extender," Dec. 2024, [Online; accessed 10. Feb. 2025]. [Online]. Available: https://brunux:8443/svn/Enerpac-SccIoExtenders-Hardware/trunk/Hardware/SCC%20Digital%20output%20extender

[14] "High-Speed CAN Transceiver with Standby Mode," Apr. 2025, [Online; accessed 9. Apr. 2025]. [Online]. Available: https://www.nxp.com/products/TJA1042

[15] "STM32G0x1 - PDF Documentation," Mar. 2025, [Online; accessed 18. Mar. 2025]. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32g0x1/documentation.html

[16] "STM32G0 Series," Apr. 2025, [Online; accessed 18. Apr. 2025]. [Online]. Available: https://products.psacertified.org/products/stm32g0-series

[17] "BF556A," Feb. 2025, [Online; accessed 10. Feb. 2025]. [Online]. Available: https://www.nxp.com/part/BF556A

[18] I. T. AG, "PROFET™+: Highest Design Flexibility in the Market," *Infineon Technologies AG*, Feb. 2025. [Online]. Available: https://www.infineon.com/cms/en/product/power/smart-power-switches/high-side-switches/classic-profet-24v-automotive-smart-high-side-switch/bts5210l

[19] "LTC1863 Datasheet and Product Info | Analog Devices," Feb. 2025, [Online; accessed 10. Feb. 2025]. [Online]. Available: https://www.analog.com/en/products/ltc1863.html

[20] "NEN Connect - IEC 61131-2:2007 en," Feb. 2025, [Online; accessed 6. Feb. 2025]. [Online]. Available: https://connect.nen.nl/Standard/Detail/118887?compId=0&collectionId=0

[21] V. Friedman, "The structure of the limit cycles in sigma delta modulation," *IEEE Trans. Commun.*, vol. 36, no. 8, pp. 972–979, Aug. 2002.

[22] V. Medina, P. Rombouts, and L. H. Corporales, "A Different View of Sigma-Delta Modulators Under the Lens of Pulse Frequency Modulation," *Authorea Preprints*, Jan. 2024.

[23] L. Hernandez and E. Gutierrez, "Analytical Evaluation of VCO-ADC Quantization Noise Spectrum Using Pulse Frequency Modulation," *IEEE Signal Process. Lett.*, vol. 22, no. 2, pp. 249–253, Sep. 2014.

[24] T. Sutton, S. Gregory, J. T. Waltman, K. W. White, and Q. Inc, "Pulse density modulation circuit (parallel to serial) comparing in a nonsequential bit order," Feb. 1993, [Online; accessed 18. Mar. 2025]. [Online]. Available: https://patents.google.com/patent/US5337338A/en

[25] D. C. Richardson and T. I. Inc, "Digital integrator for pulse-density modulation using an adder carry or an integrator overflow," Sep. 1997, [Online; accessed 18. Mar. 2025]. [Online]. Available: https://patents.google.com/patent/US5995546A/en

[26] Arm, "Armv6-M Architecture Reference Manual," *Arm*, Jan. 2025. [Online]. Available: https://developer.arm.com/documentation/ddi0419/latest

[27] A. Ltd., "Fixed Virtual Platforms – Pre-configured Simulation Models," Apr. 2025, [Online; accessed 11. Apr. 2025]. [Online]. Available: https://www.arm.com/products/development-tools/simulation/fixed-virtual-platforms

[28] "Buy PIC, ARM, AVR, 8051, MSP430 Microcontroller Simulators - Proteus VSM," Apr. 2025, [Online; accessed 11. Apr. 2025]. [Online]. Available: https://www.labcenter.com/buy-vsm

[29] "Welcome to QEMU's documentation! — QEMU documentation," Apr. 2025, [Online; accessed 11. Apr. 2025]. [Online]. Available: https://www.qemu.org/docs/master

[30] "Renode - documentation," Apr. 2025, [Online; accessed 11. Apr. 2025]. [Online]. Available: https://renode.readthedocs.io/en/latest

[31] "Time framework - Renode - documentation," Apr. 2025, [Online; accessed 11. Apr. 2025]. [Online]. Available: https://renode.readthedocs.io/en/latest/advanced/time_framework.html

[32] "TCG Instruction Counting — QEMU documentation," Mar. 2025, [Online; accessed 18. Mar. 2025]. [Online]. Available: https://www.qemu.org/docs/master/devel/tcg-icount.html

[33] "Cortex-M0+ Technical Reference Manual r0p1," Mar. 2025, [Online; accessed 18. Mar. 2025]. [Online]. Available: https://developer.arm.com/documentation/ddi0484/c/CHDCICDF

[34] "Arm Cortex-M0+ - Microcontrollers - STMicroelectronics," Apr. 2025, [Online; accessed 12. Apr. 2025]. [Online]. Available: https://www.st.com/content/st_com/en/arm-32-bit-microcontrollers/arm-cortex-m0-plus.html

[35] "Stellaris boards (lm3s6965evb, lm3s811evb) — QEMU documentation," Apr. 2025, [Online; accessed 12. Apr. 2025]. [Online]. Available: https://www.qemu.org/docs/master/system/arm/stellaris.html

[36] "MAX22000 Datasheet and Product Info | Analog Devices," Apr. 2025, [Online; accessed 15. Apr. 2025]. [Online]. Available: https://www.analog.com/en/products/max22000.html

[37] "Military Specification," Apr. 2025, [Online; accessed 15. Apr. 2025]. [Online]. Available: https://landandmaritimeapps.dla.mil/programs/MilSpec/ListDocs.aspx?BasicDoc=MIL-STD-883

[38] "Software Configurable Input and Output Analog Front End Family," Feb. 2025, [Online; accessed 4. Feb. 2025]. [Online]. Available: https://www.nxp.com/products/NAFE33352

[39] "AD7606 Datasheet and Product Info | Analog Devices," Apr. 2025, [Online; accessed 15. Apr. 2025]. [Online]. Available: https://www.analog.com/en/products/ad7606.html

[40] “ADS9815 data sheet, product information and support | TI.com,” Apr. 2025, [Online; accessed 15. Apr. 2025]. [Online]. Available: https://www.ti.com/product/ADS9815
[41] “MAX1300 Datasheet and Product Info | Analog Devices,” Apr. 2025, [Online; accessed 15. Apr. 2025]. [Online]. Available: https://www.analog.com/en/products/max1300.html
[42] “NAFE13388B40BS,” Apr. 2025, [Online; accessed 15. Apr. 2025]. [Online]. Available: https://www.nxp.com/part/NAFE13388B40BS#
[43] “ISOW1044 data sheet, product information and support | TI.com,” Apr. 2025, [Online; accessed 15. Apr. 2025]. [Online]. Available: https://www.ti.com/product/ISOW1044
[44] “ISO1042 data sheet, product information and support | TI.com,” Apr. 2025, [Online; accessed 15. Apr. 2025]. [Online]. Available: https://www.ti.com/product/ISO1042
[45] “ISO1050 data sheet, product information and support | TI.com,” Apr. 2025, [Online; accessed 15. Apr. 2025]. [Online]. Available: https://www.ti.com/product/ISO1050
[46] “NUCLEO-G0B1RE - STMicroelectronics,” Apr. 2025, [Online; accessed 15. Apr. 2025]. [Online]. Available: https://www.st.com/en/evaluation-tools/nucleo-g0b1re.html#documentation
[47] “NAFE13388-UIM 8-Channel Universal Input AFE Arduino® Shield Board,” Apr. 2025, [Online; accessed 15. Apr. 2025]. [Online]. Available: https://www.nxp.com/design/design-center/development-boards-and-designs/NAFE13388-UIM
[48] “XTR300 data sheet, product information and support | TI.com,” Apr. 2025, [Online; accessed 15. Apr. 2025]. [Online]. Available: https://www.ti.com/product/XTR300#tech-docs
[49] “XTR305,” Apr. 2025, [Online; accessed 15. Apr. 2025]. [Online]. Available: https://www.ti.com/product/XTR305#tech-docs
[50] “XTR300EVM Evaluation board | TI.com,” Apr. 2025, [Online; accessed 15. Apr. 2025]. [Online]. Available: https://www.ti.com/tool/XTR300EVM
[51] Vishay, “VO1400AEF Solid-State Relays | Vishay,” Apr. 2025, [Online; accessed 16. Apr. 2025]. [Online]. Available: https://www.vishay.com/en/product/84724
[52] S. W. Smith, *The scientist and engineer's guide to digital signal processing*. USA: California Technical Publishing, 1997.
[53] “Introduction to DMA controller for STM32 MCUs - Application note,” Mar. 2025, [Online; accessed 20. Mar. 2025]. [Online]. Available: https://www.st.com/resource/en/application_note/an2548-introduction-to-dma-controller-for-stm32-mcus-stmicroelectronics.pdf
[54] “sourceware.org Git - newlib-cygwin.git/blob - newlib/libc/string/memset.c,” Apr. 2025, [Online; accessed 23. Apr. 2025]. [Online]. Available: https://sourceware.org/git/?p=newlib-cygwin.git;a=blob;f=newlib/libc/string/memset.c
[55] “sourceware.org Git - newlib-cygwin.git/blob - newlib/libc/string/memcpy.c,” Mar. 2025, [Online; accessed 21. Mar. 2025]. [Online]. Available: https://sourceware.org/git/?p=newlib-cygwin.git;a=blob;f=newlib/libc/string/memcpy.c
[56] E. Gutierrez, L. Hernandez, F. Cardes, and P. Rombouts, “A Pulse Frequency Modulation Interpretation of VCOs Enabling VCO-ADC Architectures With Extended Noise Shaping,” *IEEE Trans. Circ. Syst. I*, vol. 65, no. 2, pp. 444–457, Aug. 2017.